Università Politecnica delle Marche

Facoltà di Ingegneria

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione

---

Tesi di Laurea

Progettazione e implementazione di un sistema per il monitoring dell'operatività delle aziende clienti di una banca

Design and implementation of a system for monitoring the operations of Bank's Corporate Clients

Relatore

Prof. Domenico Ursino

Candidato

Mattia Ippoliti

Anno Accademico 2020-2021

Indice

Elenco delle figure

Elenco delle tabelle

Elenco dei listati

Introduzione

Il progetto "New Banking Platform" è un'idea che nasce dall'esigenza di creare una strategia digitale integrata con la strategia di business, in un mondo sempre più digitalizzato. Il progetto si è posto l'ambizioso obiettivo di offrire un modello architetturale orientato al cambiamento veloce e interamente disegnato sul cliente (retail, corporate), attraverso l'utilizzo dei dati.

Nel corso del tempo, i sistemi informativi di organizzazioni pubbliche e private hanno accumulato un enorme quantità di dati, generati in parte da transazioni interne fra le divisioni amministrative, logistiche e commerciali, e in parte da fonti esterne all'organizzazione. Anche se sono stati raccolti e salvati in maniera strutturata, questi dati non possono essere acceduti e utilizzati direttamente a scopo decisionale. Occorre processarli, attraverso appositi strumenti, adottando un approccio analitico, in maniera tale da trasformarli in informazioni e conoscenza che possano essere utilizzate dagli organi decisionali.

La banca, in questo senso, ha avviato un percorso di trasformazione organizzativa e tecnologica per rispondere alle richieste del business e del regolare finanziario, in termini di qualità, completezza e tempestività dei flussi informativi. Inoltre, per rafforzare la capacità di governo dei dati da parte dell'IT, all'interno del Gruppo BO, è stato creato il "Data Technology Office". La capacità di gestire in modo evoluto dati e informazioni, da trattare come asset, è un elemento fondante della trasformazione digitale dei servizi finanziari.

Il Data Technology Office vuole essere fattore abilitante per il successo e lo sviluppo della banca (non solo ente di controllo e ottimizzazione), supportando la strategia di ricerca di nuove opportunità, nuovi mercati da acquisire con l'utilizzo delle migliori tecnologie, metodi agili che consentano di adattare rapidamente le applicazioni aziendali ai cambiamenti del mercato.

L'obiettivo è, quindi, quello di riscrivere gran parte del Sistema Informativo di Intesa, con una logica più veloce, meno costosa, mantenendo la centralità del cliente ed una vista univoca del medesimo. Il progetto si declina in quattro macro-aree progettuali:

- *Agilità*, adozione di soluzioni agili in grado di permettere l'integrazione di nuove componenti custom o di prodotti di mercato, in modo semplice, veloce e governato. La soluzione dovrà garantire la continuità nel processo di innovazione, nonchè nel garantire gli aspetti normativi, in modo efficace e semplice.

- *Always-ON*, applicazioni e servizi sempre "disponibili" per offrire in orario esteso servizi oggi erogati solo in orario diurno. I servizi e i prodotti dovranno sempre più essere disponibili ai clienti sui diversi canali (ad esempio filiale, Bancomat evoluti, Internet, phone banking, etc.) e in orario esteso, ovvero tutti i giorni della settimana e in real-time.
- *Hub dei Servizi*, ovvero la creazione di un "servizio dei servizi", per erogare servizi al gruppo e fuori gruppo, anche alle banche e alle filiali estere.
- *Customer Centricity*, vista completa del cliente, sia esso retail o corporate, in modo centralizzato, univoco e aggiornato.

Tra le varie progettualità che hanno interessato la New Banking Platform, una molto importante queste è il "Motore Calo Operatività", ovvero il modello che monitora l'andamento delle attività transazionali dei clienti di tipo Aziende e Persone Giuridiche.

L'obiettivo di questa tesi è descrivere l'evolversi di questa progettualità, partendo dagli strumenti, dalle tecniche e dalle metodologie di Business Intelligence (BI), documentandone l'applicazione pratica nel trattamento dei Big Data di una realtà leader nel proprio settore.

L'iniziativa fa parte di un ampio programma all'interno della New Banking Platform, rappresentato dal nome di BFD (Big Financial Data), di innovazione della divisione IT del Gruppo. Infatti la banca mantiene da sempre il proprio sistema informativo all'avanguardia. Col progredire delle tecnologie di Business Intelligence e analytics, è stato dunque naturale voler fare leva sull'enorme patrimonio informativo custodito nei sistemi della banca. Così, nel 2018, viene avviata l'iniziativa BFD con l'obiettivo di trarre vantaggio competitivo da questa preziosa fonte di informazioni. La tesi descrive l'evolversi del progetto Big Financial Data, in particolare della progettualità relativa al modello del calo dell'Operatività. Viene evidenziato l'ampio riscontro che la letteratura sulla Business Intelligence trova all'interno di questo contesto aziendale.

Si parte da un'architettura distribuita per l'integrazione e l'analisi dei dati, che prende il nome di "Data Lake", ed ha l'obiettivo di concentrare in un unico repository l'intero potenziale informativo racchiuso nei sistemi IT della banca. Tale Lake alimenta, a sua volta, i motori analitici e predittivi come il "Motore Calo Operatività".

Il progetto in questione prevede tre fasi principali, ovvero:

- *il modulo di data preparation*, finalizzato al recupero di tutti i dati necessari al calcolo dei KPI (sviluppato in tecnologia *SAS*);
- *il modulo del calo di operatività*, finalizzato al calcolo dello score finale, sulla base delle regole attualmente richieste dagli utenti (sviluppato in tecnologia *Pyhton*);
- *la pubblicazione dell'output Report Completo* e *Ridotto* sul database Oracle, accessibili in autonomia dall'utente tramite laboratorio *SAS*, attraverso la configurazione di una libreria *SAS* ad-hoc che permette tale accesso.

La tesi è organizzata come di seguito specificato:

- Il Capitolo 1 tratta dell'importanza dei dati all'interno della banca e il contesto normativo che fa da panorama, contestualizzando i criteri di qualità del dato;

- Il Capitolo 2 presenta una panoramica sul Gruppo Intesa San Paolo, con un focus particolare alla Direzione Sistemi Informativi (DSI). Esso descrive inoltre, lo stato dell'arte riguardo le tecniche di Business Intelligence, Big Data e le tecnologie utilizzate.
- Il Capitolo 3 contiene la raccolta dei requisiti per il progetto in esame.
- Il Capitolo 4 tratta della progettazione del modello calo operatività delle aziende clienti del Gruppo.
- Il Capitolo 5 dettaglia l'implementazione vera e propria di tale motore, focalizzandosi sul codice del progetto.
- Il Capitolo 6 contiene una fase di testing e tuning del codice in oggetto.
- Il Capitolo 7 contiene delle discussioni sulle lezioni apprese nello sviluppo di tale progettualità;
- Infine il Capitolo 8 presenta le conclusioni e gli eventuali sviluppi futuri.

1

Ruolo dei dati in una banca

In questo primo capitolo verrà descritta l'importanza che ricoprono i dati all'interno di Intesa San Paolo, realtà leader all'interno del settore finanziario. Successivamente si passerà ad introdurre il contesto normativo che sta dietro a determinate politiche aziendali. Infine verranno presentati i principi di riferimento alla base del patrimonio informativo.

1.1 Dati, Informazione e conoscenza

L'evoluzione dei sistemi hardware ha fornito sempre più affinate metodologie di acquisizione e conservazione del dato, consentendo la gestione di enormi moli di dati a costi contenuti. Però tale capacità di registrare ed archiviare dati di qualunque tipo ha finito per ridurre notevolmente la possibilità e la capacità vera e propria di rendere leggibili i dati, e, quindi, di trasformare i dati in informazioni. Si può definire una distinzione fra dati, informazioni e conoscenze come segue [6]:

- *Dati*: In genere i dati descrivono un attributo di una singola entità primaria oppure caratterizzano una transazione che coinvolge due o più entità primarie. Ad esempio, per una banca, i dati si riferiscono a entità primarie, quali clienti, punti vendita e denaro, mentre le ricevute rappresentano una transazione commerciale che coinvolge almeno due di essi.
- *Informazione*: L'informazione è il risultato di estrazioni ed elaborazioni di dati, così da apparire significativa per chi la riceve in uno specifico contesto. Per esempio, per il direttore delle vendite, la percentuale di ricevute superiori a 100€ registrate in una settimana, o il numero di clienti possessori di una carta fedeltà che hanno ridotto del 50% le loro spese nell'ultimo mese rappresentano informazioni importanti che possono essere estratte dai dati grezzi.
- *Conoscenza*: L'informazione diviene conoscenza quando è usata per prendere una decisione e dare seguito ad azioni conseguenti. Dunque, possiamo pensare alla conoscenza come un insieme di informazioni contestualizzate in un certo ambiente, arricchite dall'esperienza e competenza degli organi decisionali, con l'obiettivo di risolvere problemi complessi. Un'analisi delle vendite di una compagnia può rivelare che un gruppo di consumatori residenti in un'area in

cui recentemente un concorrente ha aperto un negozio, hanno ridotto il loro ammontare di spesa abituale. La conoscenza estratta in questa maniera potrà portare ad azioni volte a risolvere il problema, ad esempio introducendo un nuovo sistema di consegna a domicilio gratuito.

È importante sottolineare che la conoscenza può essere estratta dai dati sia in maniera passiva, cioè tramite criteri di analisi ideati dagli organi decisionali, sia in maniera attiva, ovvero tramite l'applicazione di modelli matematici prestabiliti, ai quali non occorre alcuna conoscenza della natura del dato. Il secondo approccio è alla base della Business Intelligence (nel seguito, BI) ed è ciò che si intende descrivere nei prossimi capitoli.

1.1.1 Dati come asset cardine nelle aziende

Negli ultimi anni molte organizzazioni, pubbliche e private, hanno sviluppato sistemi per raccogliere, immagazzinare e condividere il loro patrimonio di conoscenze, divenuto col tempo un vero e proprio asset aziendale dal valore inestimabile. Fornire supporto agli organi decisionali attraverso l'integrazione di processi, facendo leva su tecnologie informatiche, è un'attività che prende il nome di *knowledge management*.

Appare evidente che la Business Intelligence e il knowledge management condividano un obiettivo: lo scopo principale di entrambe le discipline è di sviluppare ambienti che possano supportare chi è chiamato a prendere decisioni e risolvere problemi complessi sulla base di conoscenze racchiuse nei dati in possesso dell'organizzazione che egli rappresenta. Per tracciare un confine fra i due approcci si può osserva che il knowledge management si basa sul trattamento di informazioni solitamente non strutturate, senza particolari riferimenti temporali, conservate in documenti, conversazioni ed esperienze. Al contrario la BI si basa su sistemi che organizzano in maniera strutturata l'informazione, solitamente quantitativa, ad esempio tramite architetture di database.

1.1.2 Ruolo dei dati in Intesa San Paolo

Il "Golden asset" all'interno di una banca sono i dati stessi, così che la capacità di gestire in modo evoluto dati e informazioni, da trattare come asset, è un elemento fondante della trasformazione digitale dei servizi finanziari. La Banca, per questa ragione ha avviato un percorso di trasformazione organizzativa e tecnologica per rispondere alle richieste del Business e del Regolatore in termini di qualità, completezza e tempestività dei flussi informativi. Inoltre, per rafforzare la capacità di governo dei dati da parte dell'IT, all'interno del Gruppo Intesa Sanpaolo, è stato creato il ≪Data Technology Office≫. Gli obiettivi principali di questa trasformazione sono:

1. *Presidio dei dati nell'IT*: garantire la capacità di pieno governo e gestione dei dati, stabilendo una chiara ownership.
2. *Qualità dati e miglioramento*: ridurre le cause di bassa qualità dei dati, assicurandone una tempestiva disponibilità e miglioramento.
3. *Supporto organizzativo*: assicurare pieno supporto, integrazione e allineamento con funzioni di governo dati utente (Data Office).

4. *Compliance in ambito dati*: supportare attivamente il rispetto della compliance normativa in ambito gestione dati lato IT.

Il modello di funzionamento previsto per il raggiungimento dei suddetti obiettivi è così costituito:

(A) *Organizzazione e Governo*: garantire la diffusione e applicazione delle migliori pratiche di gestione dati all'interno dell'IT, stabilendo una chiara ownership rispetto agli asset informativi IT.
(B) *Processi e Capability*: mettere in atto processi operativi allineati alle migliori pratiche di gestione dati, assicurando massima integrazione con il modello operativo IT in essere.
(C) *Strumenti e Tecnologie*: assicurare l'adozione di adeguati strumenti a supporto dei processi di gestione dati, favorendo il riuso degli asset esistenti e l'innovazione continua.
(D) *Change Management*: promuovere i valori chiave di una cultura del dato, garantendo la disponibilità adeguate competenze tecniche, organizzative e digitali in ambito di gestione di dati.

1.1.3 New Banking Platform

Il progetto New Banking Platform è strutturato su tre livelli, detti layer:

- Il *"layer di presentation"* utilizza la user experience per fornire un contatto gradevole con il cliente e consentire una rapida trasformazione. Questo layer è incaricato di presentare le applicazioni sui vari "canali" (filiale, contact unit, Internet banking, mobile, video, ATM, SMS,etc.) adattandosi alle peculiarità sia dei dispositivi che degli utenti. I canali rappresentano lo strato di accesso ai servizi della Banca, dove sia i clienti che i dipendenti accedono attraverso modalità tradizionali o innovative (mobile, social,etc.).
- Il *"layer di agilità"*, contenente la business logic centrale (visibilità dei dati, process engine, orchestratore dei servizi, gestione centralizzata prodotti commerciali), per la realizzazione dei servizi e dei processi di business, attraverso motori di regole per abilitare cambiamenti in tempo reale. Nell'agility layer vengono implementati processi di Business di lunga durata, provvisti di stato. Esso consente, infatti, la progettazione, l'esecuzione, e il monitoraggio dei processi end-to-end, e permette la flessibilità nella creazione di nuovi prodotti e nella revisione di quelli esistenti.
- Il *"core banking layer"* costituisce un importante elemento nella nuova architettura e rappresenta la revisione dei sistemi di governo della Banca al fine di assicurare coerenza e tempestività nella messa a disposizione delle informazioni per le filiere verticali dei singoli stakeholder. È il motore dei singoli servizi applicativi elementari. È la "cassaforte di dati" della banca (Anagrafe, Conti Correnti, Pensioni, Carte,etc.).

La Figura 1.1 riporta uno schema che permette di comprendere come tali layer siano tra loro strutturati.

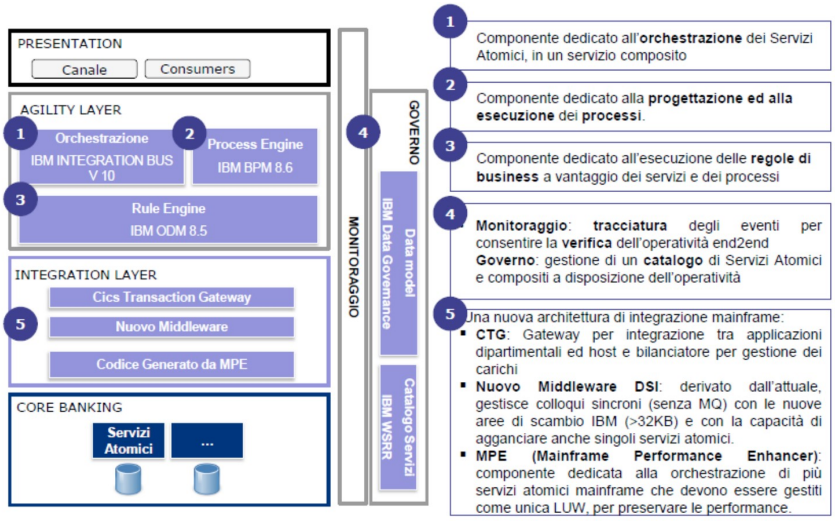

Figura 1.1. Struttura dei layer della New Banking Platform

1.2 Il contesto normativo

A partire dal secondo semestre 2007 il sistema finanziario internazionale è stato interessato da una crisi sistemica che si è andata progressivamente accentuando fino ad avere evidenti ripercussioni sulla congiuntura reale. Quella del 2007 è stata chiamata la crisi dei subprime - cioè dei mutui erogati a categorie di clienti particolarmente rischiosi per la sostanziale inconsistenza delle loro garanzie patrimoniali e reddituali - perché essa si è innescata nel settore immobiliare quando, a seguito dei rialzi dei tassi decisi dalla Fed per fronteggiare attese di crescita dell'inflazione, e dopo un prolungato trend di crescita dei prezzi delle case – una vera e propria "bolla" - gran parte delle famiglie statunitensi beneficiarie, appunto, dei mutui subprime sono risultate non più in grado di pagare le relative rate.

La crisi dei mutui subprime ha evidenziato come i sistemi informativi e le architetture delle banche fossero inadeguate a rappresentare i rischi finanziari nel loro complesso, in modo rapido ed accurato. A livello di provvedimenti in Europa il Comitato di Basilea 2 ha emanato delle linee guida supplementari con l'obiettivo di migliorare la capacità delle banche di individuare e gestire i rischi a livello di intero istituto.

La presenza di un sistema strutturato di gestione dei dati, unitamente ad un adeguato governo della qualità degli stessi, rappresenta un necessario prerequisito all'adempimento di diverse norme di legge e regolamenti di settore. Nel seguito si fornisce un breve cenno alle normative più direttamente rilevanti ai fini del governo dei dati prodotti dal Gruppo Intesa Sanpaolo. Tra queste si evidenziano:

- Gli indirizzi formulati dal Comitato di Basilea per la vigilanza bancaria con il documento "Principi per un'efficace aggregazione e reportistica dei dati di rischio", emanato nel Gennaio 2013 e finalizzato al miglioramento delle procedure di gestione del rischio [5].

- Le "Disposizioni di vigilanza per le banche", emanate dalla Banca d'Italia pochi mesi dopo le direttive Europee [8]. Tali disposizioni riguardano il sistema di gestione dei dati da parte delle banche nazionali.
- I principi richiamati dal Regolamento Europeo n. 679/2016, relativo alla protezione delle persone fisiche con riguardo al trattamento dei dati personali, nonché alla libera circolazione di tali dati [12].
- Le norme volte ad assicurare la qualità dell'informativa contabile e finanziaria prodotta dal Gruppo (art. 154-bis) [16].
- Le norme in materia di responsabilità amministrativa delle società (D.Lgs. 231/01) [34].

1.2.1 Criteri di qualità del dato

Una possibile definizione di Data Quality è quella riportata nello standard ISO 8402 [1] che la definisce come: "The totality of characteristics of an entity that bear on its ability to satisfy stated or implied needs".

Una definizione di questo tipo ci dice che la qualità del dato non dipende solo dalle caratteristiche del dato stesso ma anche dal contesto di business in cui è utilizzato. La qualità del dato è un componente critico dell'organizzazione: non implementare una strategia di valutazione e controllo della qualità dei dati che si possiedono può avere effetti disastrosi.

La presenza di una scarsa qualità dei dati non è un problema teorico, ma un reale problema di business, che incide negativamente sull'efficacia delle decisioni critiche prese dall'azienda. Oggi una delle principali aree di investimento per un'azienda dovrebbe essere una strategia di supporto, verifica e miglioramento della qualità del dato.

In particolare, a partire dal Comitato di Basilea per la vigilanza bancaria [5], è stato richiesto alle banche di definire i criteri di qualità dei dati. Questi sono:

- *accuratezza*: assenza di distorsioni nei processi di registrazione, raccolta e successivo trattamento dei dati;
- *completezza*: registrazione di tutti gli eventi, operazioni e informazioni con i pertinenti attributi necessari per le elaborazioni;
- *tempestività*: disponibilità dei dati prodotti dal sistema informativo nei tempi richiesti dall'utente finale;
- *integrità*: assenza di alterazioni o manipolazioni non autorizzate del dato, in grado di comprometterne l'accuratezza e la completezza.

Ad essi si affianca lo standard aziendale di Data Governance, che individua ruoli e responsabilità nelle funzioni coinvolte nell'utilizzo e nel trattamento delle informazioni aziendali.

In particolare, l'adozione di un modello di Data Governance richiede la definizione di processi ad-hoc in grado di assicurare il raggiungimento degli obiettivi previsti da esso. Ogni processo di Data Governance può essere ricondotto ad uno dei seguenti macro processi:

- Presidio della conoscenza e la documentazione dei dati.
- Presidio dell'affidabilità e della qualità dei dati.

1.2.2 Presidio della conoscenza e la documentazione dei dati

Le fasi rilevanti del macro processo in oggetto sono le seguenti:

1. *Raccolta della documentazione.* Il Data Owner reperisce tutta la documentazione inerente al patrimonio informativo di competenza sia attraverso la collaborazione dei Data Technology Owner (i quali mettono a disposizione, ad esempio, le informazioni e/o la documentazione degli archivi in cui sono disponibili i dati), sia attraverso la collaborazione dei Data User (i quali mettono a disposizione, ad esempio, le informazioni e/o la documentazione degli utilizzi dei dati). Nell'ambito delle progettualità aventi impatto sui dati, l'applicazione delle Data Governance richiede che, in funzione del perimetro dei dati in questione, si proceda prioritariamente al coinvolgimento degli attori competenti, secondo le responsabilità definite nell'ambito del modello di Data Governance.
2. *Analisi del patrimonio informativo di competenza e pianificazione della attività di documentazione dei dati.* Il Data Owner e il Data Technology Owner, analizzati la documentazione preesistente e il contesto di riferimento in cui vengono utilizzati i dati, condividono il piano per la documentazione dei termini e dei relativi attribuiti all'interno dell'apposito strumento aziendale. Nell'ambito delle progettualità aventi impatto sui dati, la definizione del piano delle attività è guidata dallo Sponsor di Progetto con il supporto, ove presente e se previsto nell'ambito dell'iniziativa progettuale, del Digital Business Partner.
3. *Documentazione dei dati.* Il Data Owner e il Data Technology Owner, ciascuno per le attività di competenza, provvedono a documentare i dati attraverso la produzione degli standard di data governance (deliverable) inerenti la documentazione dei dati.
4. *Verifica.* Il Data Office, anche attraverso controlli a campione, verifica l'adeguatezza dei deliverable prodotti (ad esempio, in presenza di una progettualità avente impatto sui dati, verifica che i deliverable siano in linea con gli obiettivi attesi dichiarati in sede di validazione del piano di progetto). Qualora si rilevi una problematica, esso provvede a richiedere un approfondimento agli attori competenti allo scopo di attivare gli opportuni interventi correttivi/integrativi.

1.2.3 Presidio dell'affidabilità e della qualità dei dati

Le fasi rilevanti del macro processo in oggetto sono:

1. *Effettuare l'assessment dei rischi connessi alla qualità dei dati.* Il Data Technology Owner, per i controlli di natura tecnica, e il Data Owner, per i controlli di merito, con il supporto e la collaborazione dei Data User, identificano i potenziali rischi connessi alla qualità dei dati avendo cura di reperire informazioni riguardo le eventuali anomalie che si sono già manifestate in passato.
2. *Implementare un sistema di controlli di data quality.* Il Data Technology Owner e il Data Owner curano, ciascuno per i controlli di competenza, l'implementazione di un sistema di controlli il più possibile automatizzato (i controlli manuali sono da intendersi come una categoria a cui si ricorre al fine di garantire comunque il presidio laddove l'implementazione di un controllo automatico non sia possibile; è tuttavia sempre auspicabile valutarne l'automatizzazione in considerazione della maggior adeguatezza garantita) attraverso il presidio delle fasi

di sviluppo, test e la successiva messa in produzione. Periodicamente loro sono tenuti ad effettuare una valutazione circa l'efficienza ed efficacia del proprio parco controlli.

3. *Presidiare l'esecuzione dei controlli automatici ed eseguire i controlli manuali.* Il Data Technology Owner presidia le fasi di esecuzione dei controlli automatici relativi al patrimonio informativo di competenza. Vengono inoltre eseguiti i controlli manuali secondo la periodicità prevista/definita.
4. *Presidiare gli esiti dei controlli e, in presenza di anomalie, avviare l'iter di remediation* Il Data Owner e il Data Technology Owner, ciascuno per gli ambiti di propria competenza, effettuano il presidio degli esiti dei controlli secondo la periodicità con la quale essi vengono prodotti allo scopo di individuare le potenziali anomalie. A fronte di anomalie o presunte tali (ad esempio, verifiche libere sui dati), si provvede ad effettuare una segnalazione (ove possibile tramite l'apposito applicativo) ove, a valle di un'analisi, se confermata la presenza di un problema, si provvede ad avviare l'iter di remediation.
5. *Assicurare la risoluzione delle anomalie.* Viene monitorato l'iter di remediation e, allo scopo di assicurare la risoluzione del problema, nei casi in cui le attività di remediation previste e/o attuate non risultino adeguate (sia in termini di contenuti sia in termini di tempistiche) per il raggiungimento dell'obiettivo desiderato, si procede con l'avvio della procedura di escalation. Inoltre, qualora si rilevi la necessità di effettuare interventi strutturali allo scopo di sanare definitivamente le criticità e i problemi che impattano in modo ricorrente e/o significativo la qualità del dato (e sia stato accertato che il problema non possa essere definitivamente risolto nell'ambito delle attività di remediation ordinarie), occorre attivare apposite iniziative progettuali denominate "Aree di miglioramento".
6. *Implementare e configurare indicatori di sintesi (Key Quality Indicator).* Il Data Office predispone un sistema di Key Quality Indicators (KQI), ovvero indicatori di sintesi che rappresentano il livello di qualità del patrimonio informativo e dei processi che, modulati a differenti livelli di sintesi e di dettaglio, possono rispondere sia a finalità strategiche che operative. Tali indicatori devono essere, poi, opportunamente configurati allo scopo di essere applicati a ciascun contesto (ad esempio configurando un perimetro, una periodicità di calcolo, eventuali soglie di tolleranza).
7. *Monitorare la qualità dei dati e predisporre la reportistica.* Il governo della Data Governance prevede la presenza di una periodica ed adeguata relazione di Data Quality basata sulla reportistica dei KQI; tale reportistica ha il suo principale scopo nella guida delle decisioni e nel favorire la celere pianificazione degli interventi più urgenti e/o importanti.

1.3 Principi di riferimento

Il Gruppo considera il patrimonio informativo di primaria importanza per la propria organizzazione e per il conseguimento degli obiettivi strategici e di business; a tale riguardo, il presente documento si propone di delineare un sistema di governo dei dati atto a garantirne un elevato livello di qualità.

L'intento è quello di garantire un adeguato livello di presidio nell'arco dell'intero processo di produzione/trasformazione del dato, a partire dalla generazione attraverso le successive trasformazioni e fino all'alimentazione dei sistemi di destinazione (tra cui i Sistemi di Governo , di segnalazione e di reporting). A tale scopo, il Gruppo ha attivato un Modello di Data Governance al fine di rilevare e risolvere eventuali anomalie attraverso le opportune azioni correttive.

In particolare, il sistema di Data Governance è definito in modo da garantire il rispetto dei seguenti requisiti:

- *Governabilità*: il dato, le modalità di aggregazione e perimetrazione, le procedure di estrazione, registrazione, trasformazione e caricamento dati negli archivi sono documentati e classificati al fine di garantirne la tracciabilità e il monitoraggio.
- *Fruibilità*: viene garantita l'accessibilità delle informazioni da parte degli utenti attraverso strumenti adeguati alle funzioni svolte.
- *Integrità e riservatezza*: viene assicurata la protezione dei dati da accessi non autorizzati, tutelandone l'accuratezza, la completezza e l'assenza di manipolazioni nel rispetto delle normative interne ed esterne in materia.
- *Disponibilità*: vengono garantite la disponibilità dei dati quando richiesto dai processi aziendali e la disponibilità delle risorse necessarie a tale scopo.
- *Conservazione e storicizzazione*: sono previste modalità di conservazione specifiche per ciascuna categoria di dati assicurando la disponibilità delle informazioni nel rispetto della normativa vigente e degli eventuali requisiti specifici espressi dagli utenti;
- *Adattabilità*: viene garantita la possibilità di generare e aggregare i dati in modo da consentire risposte adeguate alle esigenze informative interne ed esterne nonché alle evoluzioni aziendali.

2

Contesto di riferimento e tecnologie usate

Nel capitolo corrente si fornirà una panoramica sul funzionamento del Gruppo Intesa San Paolo, focalizzandosi, in particolare, sulla Direzione Sistemi Informativi (DSI) e sul ruolo ricoperto dallo studente in questo contesto. Successivamente, verrà descritto lo stato dell'arte riguardo le tecnologie Big Data utilizzate da un'azienda finanziaria, dei sistemi legacy alimentanti i server dei motori analitici e predittivi, nonchè l'architettura di riferimento.

2.1 Descrizione del ruolo dello studente all'interno dell'azienda

Il gruppo Intesa Sanpaolo è uno dei principali gruppi bancari in Europa, con una capitalizzazione di mercato di 45,3 miliardi di euro, ed è impegnato a sostenere l'economia nei Paesi in cui opera, in particolare in Italia, dove punta a diventare un punto di riferimento in termini di sostenibilità e responsabilità sociale e culturale. Intesa Sanpaolo è leader in Italia in tutti i settori di attività (retail, corporate e wealth management) ed offre i propri servizi a 13,5 milioni di clienti [25].

Lo studente ricopre il ruolo di Data Scientist all'interno della Direzione Sistemi Informativi (DSI) di Intesa Sanpaolo. Nello specifico, opera all'interno del Data Service Hub (DSH). Il lavoro svolto quotidianamente consiste nel progettare e realizzare prodotti di Business Intelligence con scopi analitici e predittivi, assicurando l'adeguatezza delle soluzioni alle esigenze applicative e alle necessità infrastrutturali del gruppo, anche in termini di Sicurezza, Disaster Recovery e di Business Continuity. In questo contesto le innovazioni e i progetti più importanti sono organizzati da un programma denominato "Big Financial Data (BFD)" che mira a ridisegnare l'infrastruttura e i processi, entro la data target del 2021, così da garantire sempre maggiore efficienza e resilienza del sistema IT in relazione agli obiettivi del core business.

All'interno del "Big Financial Data (BFD)" non possono non figurare trend informatici importanti, quali l'introduzione di Dev-Ops, l'utilizzo del Cloud, lo sviluppo di tecniche di Intelligenza Artificiale e, naturalmente, l'analisi dei Big Data tramite la Business Intelligence aziendale. Quest'ultimo aspetto è trattato in maniera approfondita dal Competence Center DWH e BI, con il quale lo studente ha

collaborato per la stesura di questa tesi di laurea. L'ufficio è referente tecnologico per tutti i sistemi di Business Intelligence all'interno dell'IT, fra i quali figurano, ad esempio, SAS, Python e Microsoft BI.

Il Data Service Hub (DSH), dove lo studente si è inserito, rappresenta una fusione di tecnologie, piattaforme dati e funzionalità che permettono di estendere a livello enterprise la Data Governance. L'architettura concettuale target è mostrata in Figura 2.1 ed evidenzia quali sono le Data Sources che alimentano il DSH e le relative filiere a valle che vengono alimentate da quest'ultimo, sfruttando i dati per creare report.

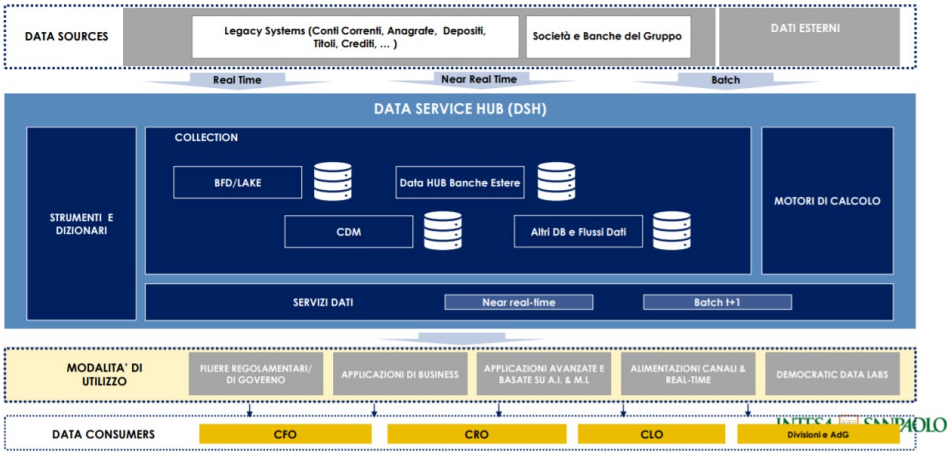

Figura 2.1. Architettura concettuale del posizionamento del Data Service Hub all'interno del Gruppo.

Inoltre lo studente è responsabile della progettazione e del supporto di microservizi riguardanti motori analitici e predittivi e segue le problematiche relative alle schedulazioni giornaliere, settimanali e mensili, grazie a due macchine server all'interno delle quali girano codici scritti rispettivamente in linguaggio SAS e Python, che alimentano tutta la reportistica di gruppi di lavoro business a valle.

Pur svolgendo quotidianamente una mansione distante da quelli che sono i temi chiave dell'adozione della BI in azienda, lo studente si è inserito nell'iniziativa progettuale denominata Big Financial Data per poter raccogliere informazioni e documentare il modo in cui Intesa Sanpaolo ha deciso di sfruttare la grande evoluzione che in questi ultimi anni ha interessato il mondo dell'analytics, sviluppando, insieme al gruppo di lavoro, microservizi che sfruttano proprio queste tecnologie.

2.2 Contesto di riferimento

2.2.1 Stato dell'arte della Business Intelligence

Il termine "Business Intelligece" è stato utilizzato formalmente per la prima volta nel 1989 da Howard Dresner [35] per descrivere l'insieme dei concetti e delle metodologie

utilizzate per migliorare il processo di decisione aziendale, facendo leva su fatti e informazioni raccolti in sistemi di supporto.

In seguito il termine si è evoluto ed è stato rivisto sotto diverse prospettive. In chiave più moderna si possono riportare definizioni che si focalizzano maggiormente sulla BI come strumento utile in termini competitivi per le organizzazioni che operano in un dato mercato e mirano a obiettivi di business sempre più elevati:

"Business Intelligence is all about capturing, accessing, understanding, analyzing and converting one of the fundamental and most precious assets of the company, represented by the raw data, into active information in order to improve business" [4]

"Business Intelligence is the capability of the organization or company to explain, plan, predict, solve problems, think in an abstract way, understand, invent, and learn in order to increase organizational knowledge, provide information to the decision process, enable effective actions, and support establishing and achieving business goals" [7]

Questi concetti prendono forma nelle organizzazioni e nelle aziende tramite l'implementazione di sistemi informativi e di processi aziendali che permettono di raccogliere, organizzare, elaborare, e infine, rappresentare dati e informazioni, provenienti da fonti interne ed esterne, storiche e istantanee, per fornire supporto alle decisioni nei diversi ambiti in cui la società opera.

È possibile rappresentare un'architettura di BI caratterizzante l'applicazione di questa disciplina in un contesto aziendale come riportato nella Figura 2.2.

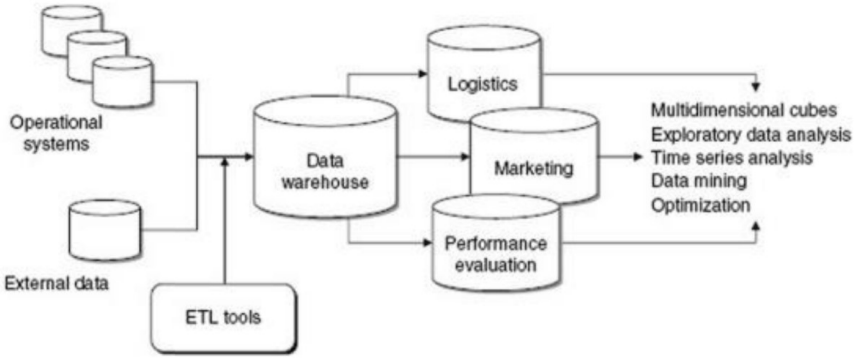

Figura 2.2. Architettura di Business Intelligence

Questo modello, scomponibile nelle macro-aree che seguono, trova ampio riscontro nei modelli adottati da Intesa Sanpaolo, come verrà descritto più avanti. I componenti che caratterizzano tale modello sono i seguenti:

- *Data source*. Il primo stadio consiste nel raccogliere e integrare dati provenienti da fonti eterogenee, interne ed esterne all'organizzazione, indipendentemente dal loro formato.
- *Data Warehouses*. Attraveso processi di Extract, Transform, Load (ETL), i dati raccolti durante la prima fase vengono salvati all'interno di databese concepiti

appositamente per supportare la BI aziendale. Come descritto nel quarto capitolo, esistono diverse modalità di organizzare i dati raccolti, creando, a seconda dell'approccio scelto, uno o più Data Mart, Data Warehouse o Data Lake
- *Business intelligence methodologies.* I dati immagazzinati vengono utilizzati per alimentare modelli matematici o algoritmi di data mining, i cui risultati sono di supporto agli organi decisionali.

- *Tecnologie.* Hardware e software sono fattori di successo rilevanti nell'implementazione di sistemi di BI all'interno di grandi aziende. Durante gli ultimi vent'anni si è osservato che la capacità di calcolo dei processori è raddoppiata ogni 18 mesi, con una costante riduzione dei prezzi. Questo trend ha consentito l'implementazione di algoritmi di analisi dei dati sempre più avanzati mantenendo i tempi di esecuzione ragionevoli. Un secondo fenomeno rilevante è la diminuzione del costo delle memorie di archiviazione di massa, che ha reso più conveniente il salvataggio di ingenti quantità di dati, dell'ordine di grandezza del Petabyte. Lo sviluppo della connettività di rete, sia interna all'azienda che esterna, ha consentito il rapido diffondersi delle conoscenza, seppur voluminosa, estratta da un'analisi di BI. Infine, la maggiore integrazione del mercato tecnologico attraverso standard ufficiali e de-facto, è l'ultimo tassello che completa il quadro di una così vasta diffusione degli strumenti di BI.
- *Analytics.* Come già accennato, i modelli matematici e le metodologie di analisi giocano un ruolo chiave nell'abilitare l'estrazione di informazioni sconosciute da dati di per sé già disponibili a molte organizzazioni. La visualizzazione di dati contestualizzati in un dato istante temporale o in termini di obiettivi aziendali, agevola il processo decisionale, anche se consiste in un supporto passivo. L'introduzione di metodi induttivi di analisi, capaci di rivelare conoscenze fino a quel momento nascoste, consiste, invece, in un supporto attivo ai processi decisionali.
- *Persone.* Le persone che lavorano in un'organizzazione raccolgono il know-how necessario a operare nel contesto della BI. Esse costituiscono la cultura aziendale in questo ambito, e ciò va considerato come un vero e proprio asset dell'organizzazione. L'abilità di raccogliere informazioni e tradurle in atti applicabili in uno specifico contesto ha, di fatto, un impatto molto significativo nel processo decisionale. Pur disponendo di un sistema di BI funzionante, senza il lavoro svolto dalle persone responsabili delle analisi, che raccolgono e interpretano i risultati, non è possibile arrivare a definire action-plan applicabili in una specifica realtà. Dunque, il fattore umano introduce una flessibilità che può rappresentare un vero e proprio vantaggio competitivo fra aziende concorrenti che si avvalgono di strumenti di BI di per sé equivalenti.

Come si evince da quanto appena esposto, il progresso della BI nel mondo IT moderno va di pari passo con lo sviluppo delle metodologie e degli approcci impiegati per analizzare il patrimonio informativo di cui molte organizzazioni dispongono, ma sul quale non hanno ancora fatto leva per trarne vantaggio competitivo. Tale preziosa fonte di conoscenza è, spesso, rimasta inutilizzata perché troppo vasta e difficile da comprendere, almeno fino agli anni recenti, durante i quali sono state sviluppate apposite metodologie per il trattamento dei Big Data.

2.2.2 Stato dell'arte dei Big Data

Anche se oggigiorno è comune sentire parlare di "Big Data", l'origine di questo termine è incerta. Esso diventa di uso comune durante il 2012, interessando sia la comunità scientifica che quella economica. Nel mondo accademico si può citare l'articolo "Big Data Special Sue" [24], mentre, per quanto riguarda imprese e società, sono testimonianza di questo interesse i numerosi report e approfondimenti editi presso giornali di rilievo, come "How Big Data Became so Big" pubblicato sul New York Times il 12 agosto del 2012 [21].

Il termine ha, probabilmente, origini più datate, ma è stato impiegato solo in maniera marginale fin quando non è divenuto celebre grazie ai grandi player del mercato IT, come ad esempio IBM, desiderosi di creare una nuova nicchia di mercato incentrata sull'analytics. La Figura 2.3 che segue mostra proprio la crescita esponenziale dell'uso di questo termine nel 2011-12.

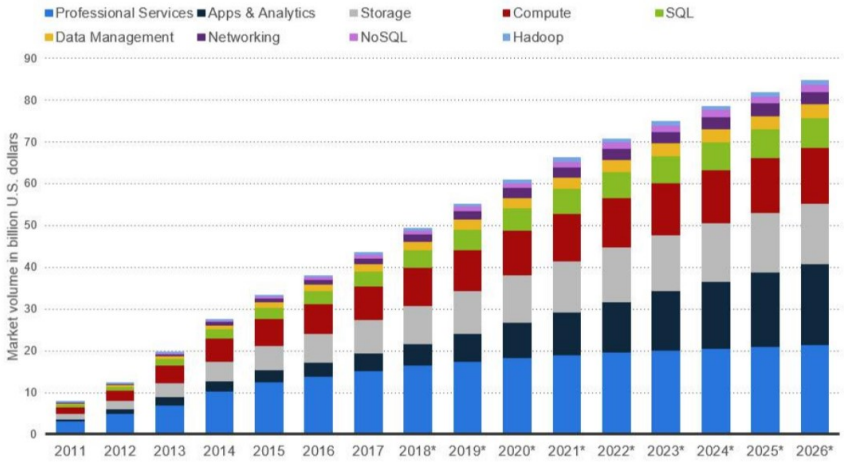

Figura 2.3. Previsione di crescita del mercato dei Big Data

Volendone dare una definizione moderna, si può dire che per Big Data si intende l'insieme di tutte le tecnologie e metodologie di analisi massive di dati. Si parla, cioè, della capacità di estrapolare, analizzare e mettere in relazione un'enorme mole di dati eterogenei, strutturati e non strutturati, per scoprire i legami tra fenomeni diversi e, dunque, prevederne il ripetersi in futuro. Un sistema di Big Data, invece, è definibile tale se è in grado di gestire una quantità di dati che eccedono la normale capacità dei sistemi hardware (HW) e software (SW) usati comunemente per catturare, gestire ed elaborare informazioni in un lasso di tempo ragionevole. È naturale che, come avvenuto per la BI, l'enorme progresso di tutte le tecnologie HW e SW dell'ultimo decennio abbia incidentalmente permesso anche il progresso nella gestione dei Big Data. Per capire meglio la natura di questa entità è utile confrontarla con un tradizionale database relazionale (RDBMS), come riportato in tabella 2.1.

RDMBS	BIG DATA
Dati Relativi ad un certo istante	Dati storici
Transazioni istantanee	Interrogazioni complesse che possono durare diverse ore
Dato sempre consistente	Dato non consistente durante la finestra di caricamento
Modello di dati relazionali, operazioni strutturate e ripetitive	Modello di dati e interrogazioni studiate all'occorrenza
Maggioranza di accessi in lettura sequenziale	Lettura di milioni di record, se possibile in parallelo

Tabella 2.1. Confronto tra database tradizionali (RDMBS) e Big Data

Appare evidente che l'implementazione dei comuni database non sarebbe efficace nel trattamento dei Big Data. Analisi tradizionali, per esempio tramite query in linguaggio SQL, su vaste moli di dati determinerebbero un numero di computazioni così elevato che si tradurrebbe in tempi e costi non sostenibili per qualsiasi organizzazione [27]. Dunque, sono state sviluppate implementazioni ottimizzate per questo genere di applicazione.

La stessa Intesa Sanpaolo, a partire dal 2012, si è interrogata su come convenisse approcciare la tematica. Le seguenti sottosezioni contengono le nozioni basilari sulle quali ci si è basati per individuare la miglior soluzione possibile.

Moderne architetture per il trattamento dei Big Data

Un Data Warehouse (DW), letteralmente magazzino di dati, ad alto livello può essere definito come una collezione di informazioni e di sistemi per rappresentarle. Al suo interno i dati sono strutturati in maniera tale da ottimizzare le query su cui si opererà. Di fatto, si tratta di un database in cui alcune informazioni sono ridondate al fine di evitare computazioni particolarmente pesanti. Lo si può collocare, come in Figura 2.4, a metà fra diverse fonti dati grezze e gli organi decisionali che rappresentano gli utenti di questo sistema di BI. Dunque, un DW, consente la

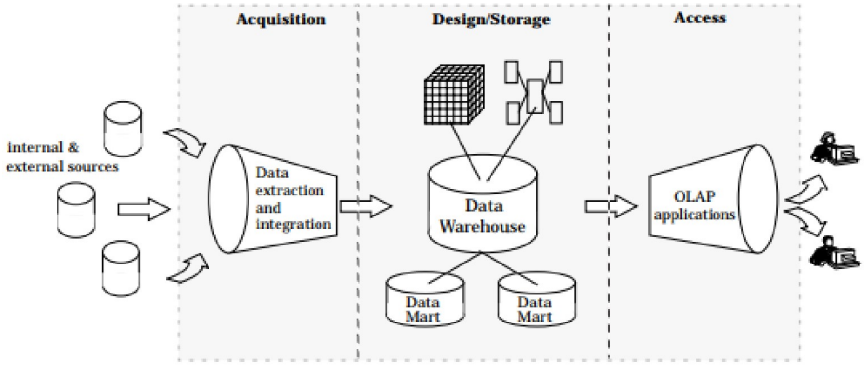

Figura 2.4. Collocazione di un Data Warehouse nell'IT di un'organizzazione [30]

riconciliazione di dati provenienti da più fonti permettendo, così, di mostrare informazioni estraibili solo tramite l'unione delle differenti basi dati. Inoltre, in virtù dell'alta ottimizzazione e personalizzazione dei dati inseriti, le performance delle query sono molto elevate, rendendo tali elaborazioni ripetibili nel tempo.

Di contro, ciò appesantisce la voluminosità del DW stesso, e questo è uno dei fattori che ne possono limitare l'applicabilità in molti contesti. Inoltre, la sua struttura di schema-on-write, ovvero i vincoli forti sull'organizzazione del dato che vi viene inserito, comporta parecchie limitazioni: i DW non sono progettati per la gestione dei dati non strutturati e soffrono di elevata latenza poiché il dato, prima di essere disponibile all'analisi, deve passare da complesse fasi di estrazione e pulizia. Infine, ciò implica che la manutenzione sia molto onerosa, poiché ogni modifica comporta la riorganizzazione di tutta l'implementazione.

Siccome oggigiorno le fonti di dati sono molteplici e in continuo aumento è veramente difficile prevedere a priori uno schema che ne consenta l'immagazzinamento e che resti valido per un lungo periodo. Di contro, non collezionare tali informazioni consisterebbe nella perdita di conoscenze potenzialmente preziose. Per gestire questa eterogeneità, viste le precedenti limitazioni dei Data Warehouse, si è dovuto pensare a nuovi approcci. Nasce, così, il Data Lake [29] (DL). Il termine è apparso per la prima volta a ottobre 2010 sul blog di James Dixon. Il CTO di Pentaho, azienda specializzata in Business Intelligence, lo definiva così:

> *"Se si pensa a un datamart come a un negozio di acqua in bottiglia – depurata, confezionata e strutturata per essere consumata con facilità – il Data Lake è una grande distesa d'acqua in uno stato più naturale. Il contenuto del Data Lake affluisce da sorgenti di dati e vari utenti del lake possono esaminarlo, immergervisi o prelevare campioni." [9]*

Il DL è dunque un repository in cui è possibile salvare svariate tipologie di dati, da quelli perfettamente strutturati a quelli grezzi e destrutturati. Esso segue un approccio di "schema-on-read", per cui solo nel momento dell'analisi si crea un modello, senza vincolare la forma con cui le conoscenze devono essere racchiuse al suo interno.

Una delle conseguenze più dirette di ciò è la possibilità di poter immagazzinare dati in real-time, abbattendo le latenze introdotte dai processi di ETL. Il Data Lake, inoltre, consente analisi non note a priori, proprio perché, non filtrando nulla, nessuna informazione viene persa.

Per chi si occupa di organizzare e analizzare il contenuto del Data Lake, ovvero le figure professionali che prendono il nome di data scientist [13], è possibile attingere a dati alla massima granularità. Ciò non avviene se si adotta un'architettura Data Warehouse poiché il dato che vi viene inserito spesso subisce delle elaborazioni, come filtri o raggruppamenti. Usando i sistemi tradizionali, dunque, potrebbe non essere possibile trovare alcuni pattern nascosti nei dati che vengono esclusi dal collezionmento. Si può osservare che questa mancanza di flessibilità dei DW ha disatteso uno dei principali obiettivi per cui erano stati concepiti: integrare dati provenienti da fonti diverse, eliminando i "silos" di informazioni specifiche, non accessibili da applicazioni diverse da quelle per cui sono stati creati. Di fatto, l'informazione contenuta in un DW rischia di rimanere frammentata, e dunque potrebbe non apparire omogenea la visione dei fatti da parte di utenti differenti.

Un altro fattore che ha determinato il diffondersi di architetture di Data Lake è stato il crescente desiderio, da parte degli utenti, di fare analisi in autonomia, con approccio self-service [15], possibilità garantita dai moderni strumenti di data mining. Inoltre, raggruppare i dati in un solo repository permette di creare un unico punto di accesso alle informazioni aziendali, facilmente controllabile, senza l'esigenza di creare molteplici credenziali e policy di sicurezza all'interno dell'azienda.

Occorre dire, però, che il proliferare dei tool di analisi, ognuno con le sue specificità, porta alla creazione di differenti layer all'interno del DL, e ciò ne rende complessi lo sviluppo e l'amministrazione. Tuttavia, negli ultimi anni, vi è stato un grande avanzamento di tali strumenti, anche in termini di semplicità d'uso e integrazione, favorendo il superamento di queste disomogeneità.

In conclusione, è bene osservare che entrambe le architetture, DW e DL, presentano sia punti di forza che debolezze, e non necessariamente sono sostitute l'una dell'altra, ma anzi, spesso, è bene integrarle in base alle molteplici esigenze che anche un singolo contesto aziendale può avere. La Figura 2.5 riassume le peculiarità delle architetture sopra descritte [28]. Se le performance di analisi e le sorgenti dati

DATAWAREHOUSE	vs.	DATA LAKE
strutturato, processato	DATA	strutturato, destrutturato, grezzo
schema-on-write	PROCESSING	schema-on-read
costoso per grossi volumi	STORAGE	disegnato per abbattere i costi
meno agile, configurazione fissa	AGILITY	molto agile
maturo	SECURITY	sta maturando
professionisti di business	USERS	data scientists

Figura 2.5. Confronto Data Warahouse vs Data Lake

sono particolarmente statiche, il DW è la scelta migliore. Se, invece, si vuole creare un sistema dove poter organizzare ogni tipo di dato, persino in forma non strutturata, il DL è l'architettura consigliata. È comunque certo che un contesto aziendale di grandi dimensioni, come Intesa SanPaolo, abbia esigenze che ricadono in entrambe le categorie (ad esempio, in diverse Business Unit), per cui converrebbe adottare un approccio integrato utilizzando il DL come primo strato di raccolta e collazionamento di molteplici informazioni da inserire in diversi DW, creati a seconda delle esigenze, in un secondo momento.

2.3 Progetto Big Financial Data (BFD)

Il termine *Big Financial Data (BFD)* si riferisce al programma con cui Intesa Sanpaolo ha adottato un'evoluzione tecnologica e organizzativa atta a valorizzare i dati come asset cardine per la Banca. A tal fine, è stata disegnata un'architettura tecnologica che permette di memorizzare, in modo univoco e centralizzato, tutti i dati del Gruppo provenienti dai sistemi legacy che devono superare rigorosi controlli di qualità. Il punto d'accesso così costituito è fonte unica dei dati per tutte le applicazioni del gruppo e per gli enti regolatori e garantisce:

1. coerenza e completezza;
2. tempestività;
3. facilità di accesso.

Gli obiettivi perseguiti sono perciò stati quelli di:

- costruire una piattaforma tecnologica per la banca del futuro;
- alimentare le filiere della nuova architettura BFD;
- abilitare applicazioni di business basate su “advanced analytics”;
- contribuire ad assicurare la compliance regolamentare attraverso il rafforzamento della data governance ed il miglioramento della data quality.

È stata perciò messa in discussione l’architettura tradizionale, costituita da numerosi “silos” contenenti informazioni utili a una specifica applicazione, ma non condivisi fra reparti o divisioni differenti. Tali inefficienze sono state evidenziate ed è stato proposto proposto di porvi rimedio tramite l’adozione di un unico repository (Figura 2.3.

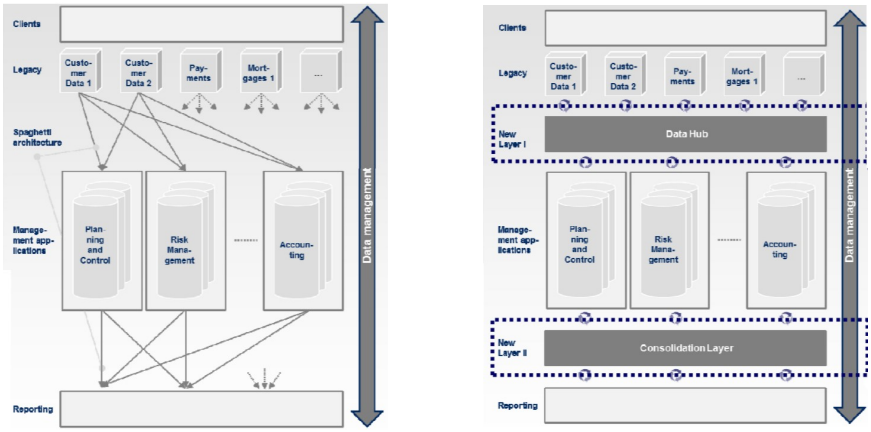

Figura 2.6. Nuova architettura basata su un unico repository di dati (Data Lake di Intesa Sanpaolo)

Nella schematizzazione soprastante si può apprezzare come l’architettura tradizionale obbligasse ogni diversa funzione aziendale (pianificazione e controllo, CRM, amministrazione economica, etc.) ad attingere a basi dati differenti. Ogni interazione con questi database comportava in un dispendio di tempo e risorse IT, e spesso andava reiterata per raggruppare ed elaborare tutte le informazioni appartenenti ai differenti silos. Di contro la nuova architettura basata sul Data Lake abilita l’elaborazione di dati già organizzati in un unico contenitore logico, e non depurati di alcuna informazione. Pertanto, in linea teorica, ciascun reparto è in grado di produrre l’output desiderato con una sola elaborazione dati, attraverso gli strumenti di analisi messi a disposizione per l’interrogazione del DL. Appare evidente che, fin dal principio, il programma Big Financial Data ha sposato le linee guida e le direttive relative al trattamento dei Big Data, proprio come riportate nella letteratura relativa alla BI.

Le fonti individuate che costituiscono quelle di interesse sono i sistemi legacy, essi rappresentano tutte le applicazioni core che gestiscono i partitari, questi sono archivi dove vengono registrate le informazioni delle varie transizioni, come, ad esempio, l'anagrafe e conti correnti. Grazie a BFD (Big Financial Data) vengono copiati tutti i dati di business di tali applicazioni su un infrastruttura centralizzata che raccoglie i dati di business dei vari sistemi legacy: *Data Lake*. Questi dati sono poi fruibili dalle applicazioni a valle, tramite il business layer. Tuttavia *non* è detto che *tutti* i dati di cui ha bisogno siano già su BFD.

2.3.1 Architettura dei dati

L'architettura dati rappresenta il modo in cui i dati sono organizzati e quali sono le interazioni tra i diversi sistemi. Essa permette di standardizzare e centralizzare lo scambio dei dati, gestendone trasversalmente il governo, al fine di supportare il business.

L'architettura è rappresentata secondo la seguente lista:

- Con *Legacy* si intende l'insieme di quelle applicazioni verticali (ad esempio "Conti Correnti", "Mutui", "Carte", "Anagrafe", etc.), che rappresentano la fonte primaria di origine dei dati di ciascun ambito.
- Il termine *Data Lake* si riferisce alla piattaforma tecnologica su cui vengono memorizzati in modo centralizzato i dati provenienti dai sistemi legacy. I sistemi fonte possono valutare la possibilità di copiare sul Data Lake solo un sottoinsieme dei dati di cui sono possessori, escludendo quelle informazioni "tecniche" prive di valore di business. La ownership dei dati sul Data Lake è dei Data Owner/Data Technology Owner dei sistemi di partenza.
- Il *Data Service Layer* è quell'insieme di tecnologie, come Teradata 2.4.2 ed Hadoop 2.4.1, con le quali i dati presenti sul Data Lake e/o le loro aggregazioni vengono rese fruibili alle filiere, ai sistemi di reportistica e di advanced analytics.
- Con *Filiere* s'intende l'insieme di applicazioni che utilizzano i dati del Data Lake per raggiungere precisi scopi di business, come, ad esempio, segnalazioni di vigilanza, reportistica, profitability, etc.
- Gli *Strumenti di Data Governance* sono quegli applicativi che concorrono a garantire l'efficiente funzionamento dell'architettura dati. Tra questi ricordiamo il Dizionario Dati (luogo in cui vengono censiti tutti i dati che si trovano sul Data Lake e corredati di definizioni di business) e gli strumenti di controllo della qualità dei dati (che assicurano il salvataggio corretto e tempestivo dei dati dei sistemi legacy sul Data Lake).

2.3.2 Architettura a Microservizi

Il progetto Big Financial Data ha inoltre l'obiettivo di progettare applicazioni che siano multilingue, facilmente scalabili, facili da gestire e distribuire, altamente disponibili e che minimizzino gli errori.

Nell'origine dello sviluppo applicativo, anche un cambiamento minimo a un software esistente imponeva un aggiornamento completo e un ciclo di controllo sulla qualità (QA, Quality Assurance) a sé, che rischiava di rallentare il lavoro di tutti

i team coivolti da quel determinato applicativo di una data filiera. Tale approccio viene, spesso, definito *"monolitico"*, perché il codice sorgente dell'intera applicazione veniva compilato in una singola unità di deployment. Se gli aggiornamenti a una parte del software provocavano errori, era necessario disconnettere tutto, ed effettuare un rollback totale del software. Tale approccio è ancora applicabile alle piccole applicazioni, ma aziende come Intesa SanPaolo non possono permettersi tempi di inattività e procurare disservizi al cliente.

Cosa sono i microservizi

I microservizi [3] sono un approccio per sviluppare e organizzare l'architettura dei software in modo tale che quest'utlimo era composto da servizi indipendenti di piccole dimensioni che comunicano tra loro tramite API ben definite. Questi servizi sono controllati da piccoli team autonomi. Le architetture dei microservizi permettono di scalare e sviluppare le applicazioni in modo più rapido e semplice, consentendo di promuovere l'innovazione e di accelerare il time-to-market di nuove funzionalità.

In un'architettura di microservizi, ogni microservizio è proprietario di un'attività semplice e comunica con i client o con altri microservizi, con il vantaggio che un'applicazione è realizzata da componenti indipendenti che eseguono ciascun processo applicativo come un servizio. I servizi sono realizzati per le funzioni aziendali e ogni servizio esegue una sola funzione. La Figura 2.7 mostra l'architettura di un'applicazione composta da più microservizi rispetto un'architettura di tipo monolitico.

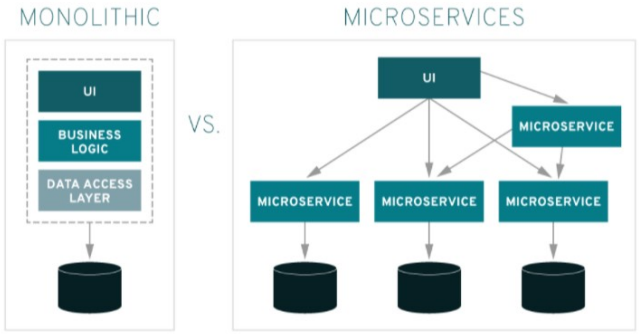

Figura 2.7. Confronto tra un'architettura monolitica ed una basata sui microservizi

Caratteristiche dei microservizi

I microservizi devono avere le seguenti caratteristiche:

- *Autonomi*: ciascun servizio può essere sviluppato, distribuito, eseguito e ridimensionato senza influenzare il funzionamento degli altri componenti. I servizi non devono condividere alcun codice o implementazione con gli altri. Qualsiasi comunicazione tra i componenti individuali avviene attraverso API ben definite.

- *Specializzati*: ciascun servizio si concentra sulla risoluzione di un problema specifico. Se, nel tempo, gli sviluppatori aggiungono del codice a un servizio, rendendolo più complesso, quest'ultimo può essere scomposto in servizi più piccoli.
- *Agili*: i microservizi promuovono le organizzazioni di team indipendenti di dimensioni ridotte, che possano lavorare in modo autonomo e rapido. Ciò riduce i tempi del ciclo di sviluppo.
- *Scalabili e flessibili*: i microservizi consentono di scalare ciascun servizio in modo indipendente; ciò permette ai team di ridimensionare in modo corretto l'infrastruttura in base alle necessità, misurando in modo accurato i costi di una funzionalità.
- *Semplicità di distribuzione*: i microservizi supportano l'integrazione continua e la distribuzione continua, così da poter testare nuove funzionalità, e, ripristinare versioni precedenti quando qualcosa non funziona. Gli errori, dunque, influiscono di meno sul costo delle operazioni, permettendo di sperimentare, aggiornare il codice in modo più semplice e accelerare il "time-to-market" delle nuove funzionalità.
- *Libertà tecnologica*: il team ha la libertà di scegliere gli strumenti migliori per risolvere i problemi specifici. Di conseguenza, il team che costruisce il microservizio può scegliere il miglior strumento per ciascuna applicazione.
- *Codice riutilizzabile*: dividere il software in moduli piccoli e ben definiti permette ai team di utilizzare funzioni per più scopi. Un servizio scritto per una certa funzione, può essere utilizzato come blocco costruttivo per un'altra funzione. Ciò permette all'applicazione di effettuare il "bootstrap" in modo indipendente, poiché gli sviluppatori possono creare nuove funzionalità senza dover scrivere del codice da zero.
- *Gestione degli errori*: l'indipendenza dei servizi aumenta la resilienza di un'applicazione in caso di errori. In un'architettura monolitica, un errore in un unico componente potrebbe avere ripercussioni sull'intera applicazione. Con l'uso dei microservizi, le applicazioni possono gestire completamente gli errori di un servizio, isolando la funzionalità senza bloccare l'intera applicazione.

2.3.3 Approccio DevOps

Il progetto Big Financial Data (BFD) si basa sull'utilizzo di un approccio DevOps. DevOps è un modello secondo cui team di sviluppo interfunzionali programmano in maniera più rapida, rilasciando soluzioni di qualità e sicure per ambienti tradizionali e in cloud.

Il termine "DevOps" [14] è una combinazione di "Development" (sviluppo) e "Operations" (operazioni).

Tale modello influenza il ciclo di vita dell'applicazione nelle fasi di pianificazione, sviluppo e produzione. Come si può notare dalla Figura 2.9, ogni fase è basata sulle altre e a nessuna fase è assegnato un ruolo specifico.

In particolare, tutti i progetti all'interno di Big Financial Data (BFD) devono seguire la seguente suddivisione:

- *Pianificazione*: durante la fase di pianificazione i team DevOps concepiscono, definiscono e descrivono le funzionalità delle applicazioni e dei sistemi da creare.

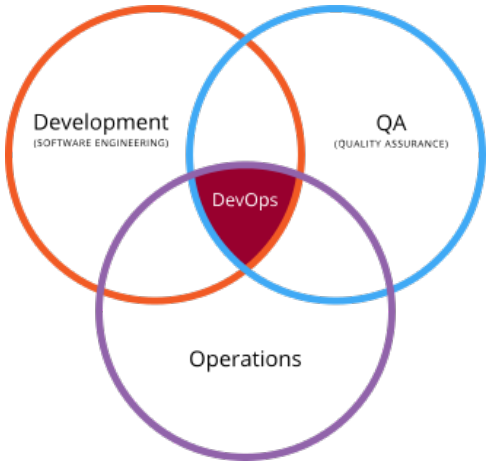

Figura 2.8. DevOps come interSezione di Sviluppo (software engineering), technology operations e garanzia di qualità (Quality Assurance)

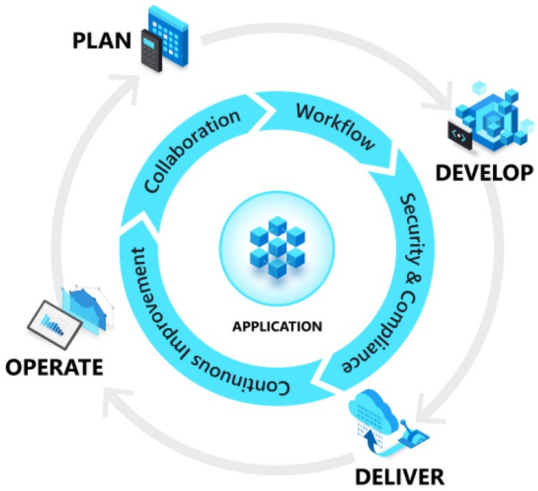

Figura 2.9. Ciclo di vita dell'applicazione

Essi tengono traccia dell'avanzamento a livelli ridotti ed elevati di granularità, dalle attività di un singolo prodotto a quelle relative a portfolio o a più prodotti. La creazione di backlog, la verifica dei bug, la gestione di Agile Software Development con Scrum, l'uso delle lavagne Kanban e la visualizzazione dello stato con i dashboard sono alcuni dei modi in cui i team DevOps pianificano con flessibilità e visibilità.

- *Sviluppo*: la fase di sviluppo include tutti gli aspetti della codifica, tra cui scrittura, test, revisione e integrazione del codice da parte dei membri del team, oltre all'inserimento del codice in artefatti della compilazione che possono essere distribuiti in diversi ambienti. I team DevOps si impegnano per innovare

rapidamente senza sacrificare la qualità, la stabilità e la produttività. A tale scopo usano strumenti a produttività elevata, automatizzano i passaggi ripetitivi e manuali ed eseguono l'iterazione in piccoli incrementi tramite test automatizzati e integrazione continua.

- *Distribuzione*: il recapito è il processo di distribuzione di applicazioni negli ambienti di produzione in modo coerente e affidabile. La fase di recapito include anche la distribuzione e la configurazione dell'infrastruttura di base completamente regolamentata che costituisce tali ambienti. Durante la fase di recapito i team definiscono un processo di gestione del rilascio con fasi di approvazione manuale chiare. Essi configurano, anche, attività di controllo automatizzate che spostano le applicazioni da una fase all'altra fino alla disponibilità per i clienti. L'automazione di questi processi li rende scalabili, ripetibili e controllati. In questo modo i team che adottano DevOps possono eseguire frequentemente la distribuzione con facilità e in tutta sicurezza.
- *Operazioni*: la fase operativa prevede la manutenzione, il monitoraggio e la risoluzione dei problemi delle operazioni negli ambienti di produzione. Durante l'adozione delle procedure DevOps i team si impegnano per assicurare l'affidabilità del sistema e la disponibilità elevata e cercano di ridurre a zero il tempo di inattività, rafforzando, al tempo stesso, la sicurezza e la governance. I team DevOps cercano di identificare i problemi prima che influiscano sull'esperienza dei clienti e di attenuare rapidamente i problemi quando si verificano. Questo livello di vigilanza richiede telemetria avanzata, avvisi di utilità pratica e visibilità completa delle applicazioni e del sistema sottostante.

2.3.4 Archetipi DevOps

All'interno del Gruppo l'adozione dell'approccio DevOps si divide principalmente in 3 archetipi presentati nella Tabella 2.2:

Continous delivery

Il continuous delivery è un approccio in cui i team producono software in cicli brevi, assicurando che il software possa essere rilasciato in modo affidabile in qualsiasi momento; quando si rilascia il software, lo si fa manualmente. Esso mira a costruire, testare e rilasciare software con maggiore velocità e frequenza.

Esso prevede, inoltre, l'automazione del deployment del SW in tutti gli ambienti previsti dalla Delivery Pipeline, dall'ambiente di Sviluppo fino alla Produzione. Il processo copre, anche, le esigenze di gestione delle configurazioni, il refresh dei dati (per gli ambienti di test) e l' esecuzione automatica dei test. Esso viene considerato il processo più critico e fondante per l'adozione delle logiche DevOps. In Figura 2.10 viene riportato lo schema di riferimento del Continuous Delivery.

Continous Integration

Continuous Integration (CI) è una pratica di sviluppo del gruppo ISP che richiede agli sviluppatori di integrare il codice in un repository condiviso più volte al giorno.

Archetipo	*Caratteristiche*	*Fattori Abilitanti*
Basic	Il modello comporta l'adozione della Continuous Integration	Versioning del software, Build centralizzata automatizzata e Deploy automatizzato in ambiente di Application Test
Intermediate	Il modello include le caratteristiche del modello Basic e prevede l'adozione dello Shift Left (anticipo del monitoraggio della qualità del software)	Analisi statica della qualità/sicurezza del software via Sonarqube, Esecuzione degli Unit Test e calcolo della Code Coverage
Full	Il modello include le caratteristiche dei modelli Basic e Intermediate e prevede l'esecuzione dei Test di Non Regressione (TNR), inclusi gli Smoke Test, nonché della Service Virtualization	Predisposizione e automazione dei TNR e degli Smoke Test

Tabella 2.2. Tre archetipi adottati dal gruppo per l'adozione all'approccio DevOps

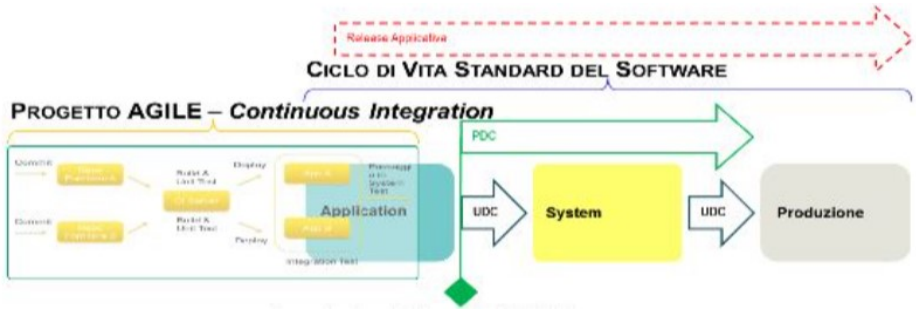

Figura 2.10. Schema del Continuous Delivery

Ogni check-in viene quindi verificato da una build automatizzata, consentendo ai team di rilevare i problemi in anticipo.

Essa, quindi, consente ai diversi team di sviluppatori di lavorare in maniera condivisa, agile e consistente su un medesimo progetto software. Essa permette con un elevato livello di automazione di:

- compilare il software centralmente su un ambiente controllato da Intesa Sanpaolo;
- controllare le regole di buona programmazione con cui è stato scritto il software attraverso un' analisi software su *SonarQube*, evidenziando anomalie agli standard definiti da Intesa Sanpaolo;
- eseguire gli unit test scritti dal team di sviluppo;

- controllare la Code Coverage (percentuale di copertura degli unit test sul totale del software rilasciato).
- rilasciare il software sull'ambiente di Application e riavviare i servizi applicativi (aggancio alla fase di Continuous Delivery per l'ambiente di Application).

Il processo di Continuous Integration permette un' evidenza sulla qualità del software rilasciato in ambiente di Application.

Gli strumenti maggiormente utilizzati in questa fase sono *Bitbucket*, per il versionamento GIT, e *Jenkis* (Sezione 2.4.8). Al ogni rilascio del software sul branch di integrazione (push) viene lanciata automaticamente la pipeline di Continuous Integration su Jenkins.

Continous Monitoring

Il Continuous Monitoring è una delle pratiche DevOps che ha l'obiettivo effettuare il monitoraggio, recependo dati e metriche dai vari processi tecnologici interessati, relativi a:

- l'avanzamento dell'adozione DevOps da parte dei singoli dipartimenti aziendali;
- il loro comportamento in merito al mantenimento della compliance rispetto ai requisiti previsti per ogni livello di adozione;
- le funzionalità, le modalità d'uso e le performance dei servizi DevOps.

Lo strumento utilizzato, per esporre i dati aggregati in forma di dashboard di sintesi è Kibana, a cui si affiancano strumenti specifici per indagini di maggiore dettaglio (es. Change Console, Sonar, Dashboard Kpi Aggregati, etc.).

Continous Testing

Il continuous Testing è il processo di esecuzione di test automatizzati come parte della pipeline di consegna del software per ottenere un riscontro immediato sui rischi aziendali associati ad una release candidata del software. Esso consente quindi l'esecuzione di test predefiniti (funzionali, di performance, di sicurezza, etc.) in un ambiente con frequenza elevata o continuativa, mano a mano che il software viene sviluppato e/o rilasciato (ad esempio, possono esser lanciati tutti i test ad ogni commit). Il processo ha l'obiettivo di ridurre i tempi ed i costi relativi all'esecuzione dei test; per farlo, offre i seguenti servizi:

- Service Virtualization per servizi esterni alle applicazioni;
- Test Automation.

I principali benefici sono:

- aumentare la qualità del software rilasciato;
- ridurre i ricicli tra gli ambienti (e, di conseguenza, i tempi di test)

Gli ambienti impattati sono Application, Integration, System, Independent e Collaudo Utente.

2.3.5 Container e Docker

Nella gestione della struttura a microservizi, la principale tecnologia utilizzata è Docker.

Docker [10] è una piattaforma di container management, che automatizza il deployment di applicazioni all'interno di cointainer, ovvero pacchetti software eseguibili ed autoconsistenti. Questi, infatti, contengono al loro interno il codice e tutte le sue dipendenze, permettendo, così, allo sviluppatore di replicare l'applicazione ovunque, indipendentemente dal sistema operativo dell'host. Gli strumenti per la creazione di container, come Docker, consentono il deployment a partire da un'immagine. Ciò semplifica la condivisione di un'applicazione o di un insieme di servizi, con tutte le loro dipendenze, nei vari ambienti. Docker automatizza anche la distribuzione dell'applicazione (o dei processi che compongono un'applicazione) all'interno dell'ambiente containerizzato.

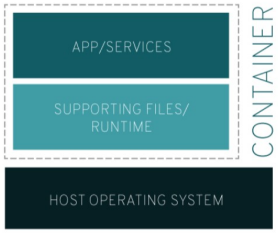

Figura 2.11. Schema di un container

Gli strumenti sviluppati partendo dai container Linux, responsabili dell'unicità e della semplicità di utilizzo di Docker, offrono agli utenti accesso alle applicazioni, capacità di eseguire un deployment rapido, e controllo sulla distribuzione di nuove versioni.

Differenza tra la tecnologia Docker e i container Linux tradizionali

La tecnologia Docker è stata sviluppata partendo dalla tecnologia LXC, che era una soluzione di virtualizzazione leggera, ma non offriva un'esperienza ottimale a sviluppatori e utenti. Docker non fornisce solo capacità di eseguire i container. Essa consente, ad esempio, di semplificare il processo di creazione e costruzione dei container, di invio di immagini e di versioning delle immagini. Al contrario i container Linux tradizionali adottano un sistema init in grado di gestire più processi, consentendo di eseguire più applicazioni come una singola entità. La tecnologia Docker agevola la suddivisione delle applicazioni nei loro vari processi e mette a disposizione gli strumenti per farlo. Questo approccio granulare ha svariati vantaggi. Nella Figura 2.12 vengono rappresentate le principali differenze tra i due.

Kubernetes

Il vantaggio dei container rispetto alle Virtual Machine risiede nella virtualizzazione delle applicazioni software e non, come avviene nelle macchine virtuali, dell'intera

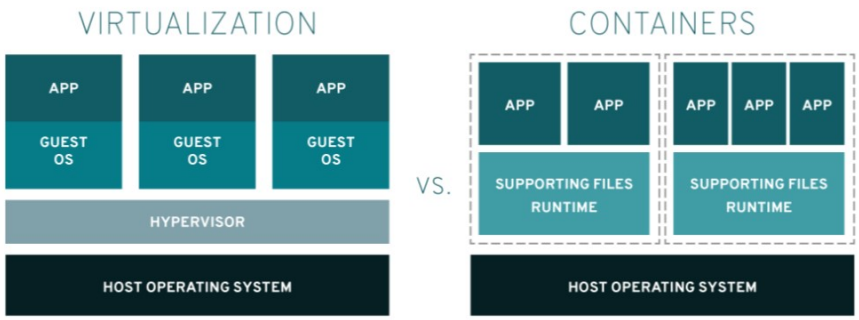

Figura 2.12. Differrenza tra container tradizionali e tecnlogia Docker

infrastruttura informatica. Poiché i container sfruttano il sistema operativo dell'host, invece del proprio, ne consegue che necessitano di risorse di elaborazione minime, oltre a essere più semplici da installare. Gli sviluppatori, in sostanza, non devono intervenire per modificare tutta l'infrastruttura, ma su ciascun singolo componente applicativo. Da qui l'esigenza di un modello per governare automaticamente l'orchestrazione dei container distribuiti in cluster e su più server host. Tale esigenza è, ad oggi, soddisfatta nella realtà dell'azienda il Kubernetes, piattaforma open source che automatizza le operazioni dei container Linux.

L'architettura e gli elementi chiave di Kubernetes, sono i seguenti: master, nodi, kubelet, pod.

- *Master*: si tratta della macchina che controlla i nodi e consente di eseguire un servizio di schedulazione in grado di automatizzare la distribuzione dei container in base ai requisiti impostati dallo sviluppatore e alla capacità di calcolo disponibile.
- *Nodi*: i cluster sono costituiti da nodi, ciascuno dei quali rappresenta un singolo host di calcolo, cioè di macchina virtuale o fisica. I nodi eseguono le attività loro affidate sotto la guida del nodo master che li controlla.
- *Kubelet*: è l'agente software che riceve ed esegue gli ordini dal nodo master, facendo in modo che i container definiti siano avviati ed eseguiti.
- *Pod*: sono gruppi di container che condividono le stesse risorse di calcolo e la stessa rete. Rappresentano anche l'unità di scalabilità di Kubernetes e, quindi, se un container in un pod riceve più traffico di quello che può gestire, Kubernetes replica il pod su altri nodi del cluster.

Pod

All'interno dell'ambiente Openshift, Docker viene utilizzato per l'orchestrazione dei container, dove questi ultimi vengono aggregati in unità dette Pod. Openshift supera molte delle difficoltà comuni legate alla proliferazione dei container, raggruppandoli in un "pod". Un Pod è un gruppo di una o più applicazioni (ovvero, uno o più container) che operano in un environment condiviso. Tale approccio permette di aggregare e orchestrare (anche e soprattutto dal punto di vista del networking) i container eseguiti da Docker.

I pod aggiungono, quindi, un livello di astrazione ai cluster di container, aiutando a programmare i carichi di lavoro e a fornire i servizi necessari, tra cui rete e storage, ai container stessi. Kubernetes agevola, inoltre, il bilanciamento del carico all'interno dei pod e garantisce l'utilizzo di un numero di container adeguato per supportare il lavoro di quella determinata applicazione.

Vantaggi dei contaier Docker

Il gruppo ha, quindi, scelto tale tecnologia per i seguenti motivi:

- *Modularità*: l'approccio Docker alla containerizzazione si basa sulla capacità di estrarre i singoli componenti di un'applicazione, da aggiornare o riparare. Oltre a questo approccio basato sui microservizi, è possibile condividere i processi tra più applicazioni; ciò è fondamentale per far comunicare i diversi motori sviluppati con architettura a microservizi all'interno del Data Service Hub (DSH).
- *Controllo delle versioni delle immagini*: ogni file immagine Docker è composto da più strati, che insieme costituiscono una singola immagine. Uno strato viene creato ad ogni modifica dell'immagine. Ogni volta in cui un utente specifica un comando (ad esempio run o copy), viene creato un nuovo strato. Docker riutilizza questi strati per velocizzare il processo di creazione dei container. Le modifiche sono condivise tra le immagini, migliorando ulteriormente la velocità, la dimensione e l'efficienza. Il controllo delle versioni fa parte del processo di stratificazione. Ogni volta che viene apportata una modifica, il registro delle modifiche offre il controllo totale sulle immagini containerizzate.
- *Rollback*: uno dei maggiori vantaggi della stratificazione è la capacità di eseguire il rollback. Ogni immagine è composta da strati. Se l'iterazione di un'immagine non è soddisfacente, è possibile riportala alla versione precedente. Ciò consente uno sviluppo agile e aiuta a ottenere l'integrazione e il deployment continui (CI/CD).
- *Deployment rapido*: in passato, la configurazione, l'esecuzione e il provisioning di un nuovo hardware all'interno del Gruppo Intesa richiedevano giorni e notevoli investimenti. I container basati su Docker possono ridurre il deployment a pochi secondi. Creando un container per ogni processo, è possibile condividere con rapidità i processi simili con le nuove applicazioni. Poiché non è necessario riavviare un sistema operativo per aggiungere o spostare un container, i tempi per il deployment sono sostanzialmente più brevi. Inoltre, grazie alla velocità del deployment, attraverso i container è possibile creare ed eliminare dati in modo sicuro, semplice ed economico.
  Dunque, la tecnologia Docker è caratterizzata da un approccio basato sui microservizi granulare e controllabile, per un ambiente IT più efficiente.

2.3.6 Layer Batch: Schedulazioni TWS

Un sistema batch permette di automatizzare alcune procedure informatiche. I lavori sono impostati in modo che non sia necessario in alcun caso l'intervento dell'uomo. Tutti i parametri sono predefiniti e specificati attraverso appositi script, comandi o

file di controllo (i file batch) che si occupano di verificare che la procedura avvenga come previsto. Una tale modalità operativa differisce dal funzionamento in linea (inline) dei programmi, che richiede l'interazione diretta e continua, solitamente effettuata via console, tra una macchina e un operatore umano che, di conseguenza, deve essere sempre presente per guidare la procedura digitando le nuove istruzioni mano a mano che queste si rendono necessarie. Più in generale, un sistema batch prevede l'esecuzione di più istruzioni o programmi (chiamati anche "job") senza che sia necessario l'intervento di un operatore umano.

Grazie al batch processing, infatti, un amministratore di sistema può decidere a tavolino come suddividere i tempi di utilizzo delle risorse informatiche tra i vari programmi, evitando, così, che ci siano tempi morti nell'utilizzo del processore e delle altre componenti hardware. Mantenendo elevati livelli di utilizzo della macchina, è, dunque, possibile ammortizzarne i costi sul breve-medio periodo, rendendo l'intero sistema più produttivo, e quindi più efficiente, anche dal punto di vista economico.

In particolare, all'interno della realtà Intesa SanPaolo, viene utilizzato TWS (Tivoli Workload Scheduler) della IBM, che è sistema di pianificazione lavoro batch completamente automatizzato, grazie al quale è possibile la "Business Process Automation".

Lo schedulatore TWS, attraverso elaborazioni batch, verrà utilizzato in output dei motori scritti in SAS e in Python per l'invio automatizzato dei file verso i dipartimenti business a valle, tramite IXP.

Invii IXP-File Transfer

Gli invii dei file avvengono tramite infrastruttura di rete IXP, che consente l'interconnessione tra più di due Autonomous System indipendenti, principalmente allo scopo di facilitare lo scambio di traffico Internet.

Tipicamente un IXP è basato su una LAN Ethernet a disposizione degli Internet Service Providers che, connettendosi, possono scambiare traffico IP con gli altri ISP presenti sull'IXP. Lo scambio di traffico – il cosiddetto peering – realizzato tramite un IXP, è generalmente realizzato su VLAN condivise da tutti gli ISP collegati (peering pubblico) o su VLAN dedicate allo scambio di traffico tra coppie di operatori (peering privato). La connessione agli IXP consente a ogni operatore di utilizzare un unico flusso geografico (circuito fisico) per interconnettersi a una molteplicità di reti di altri operatori (Autonomous System – AS), evitando di dover realizzare tante connessioni quanti sono gli AS con i quali si desidera scambiare traffico.

2.4 Tecnologie utilizzate

Dato il ruolo centrale che il Data Lake gioca nel progetto BFD, Intesa Sanpaolo ha condotto uno scouting tecnologico particolarmente esteso ed accurato riguardo i prodotti di mercato più adatti a implementarlo, determinando i migliori secondo tre fattori principali:

- *Costi*: l'elevato ammontare di dati da immagazzinare, determina necessariamente ampi costi di risorse hardware (storage, CPU, RAM, etc) e di licenze e sottoscrizioni. I prodotti scelti vanno nella direzione di minimizzare tali spese.
- *Maturità*: le tecnologie a supporto di un così ambizioso programma devono avere necessariamente un grado di maturità tale da non mettere a rischio la riuscita dello stesso. Inoltre la presenza di bug, o generiche instabilità, si tradurrebbe in una percezione di inaffidabilità prima ancora del completamento del progetto, motivo per cui la maturità del prodotto è un elemento chiave nella scelta. Infine, una maggiore maturità spesso significa anche una più vasta gamma di integrazioni con altri prodotti, fattore che potrebbe agevolare la transizione dalle architetture legacy già presenti in Intesa Sanpaolo.
- *Efficacia*: non tutte le tecnologie, a parità di algoritmi e specifiche, portano ai medesimi risultati in termini di business. Alcuni strumenti possono apparire più efficaci di altri perché dispongono di una migliore interfaccia utente o perché godono di un supporto professionale di maggiore qualità. Ciò che si è cercato di valutare è, dunque, il potenziale impatto positivo, in termini di risultati attesi, generato dalla scelta di una tecnologia piuttosto che un'altra.

Negli anni antecedenti il lancio del programma BFD del 2015, Intesa Sanpaolo ha individuato, sulla base dei sopracitati criteri, svariate tecnologie per la sua implementazione. Quelle sotto riportate sono senza dubbio le più importanti e significative.

- *Hadoop* [2] e *Teradata* [31]: contengono e organizzano i dati di cui è composto il Data Lake e i vari Data Warehouse che da esso hanno origine.
- *InfoSphere DataStage* [19]: è il tool di ETL che alimenta il Data Lake attingendo dalle informazioni contenute nei database legacy di Intesa Sanpaolo.
- *SAS* [17]: è il principale strumento di analisi del contenuto del Data Lake.
- *Python* [26]: è lo strumento che, partendo dai dati elaborati da SAS, implementa microservizi ed ha il vantaggio di avere librerie, come tensorflow, utili per il machine learning.
- *Jenkins* [20]: consente di automatizzare le diverse fasi del ciclo di vita del software, dalla compilazione al test alla distribuzione.
- *Red Hat OpenShift* [18]: è una piattaforma PaaS per applicazioni cloud che automatizza l'hosting, la configurazione, l'implementazione e l'amministrazione degli stack di applicazioni in un ambiente cloud flessibile.

Alle sopracitate tecnologie se ne affiancano alcune altre, anche per via dell'enorme offerta del mercato dell'IT per l'analytics. Esse, però, hanno sicuramente meno rilevanza in Intesa Sanpaolo e pertanto non vengono approfondite in questa tesi.

2.4.1 Apache Hadoop

La sfida che ogni società si pone nello sfruttare il potenziale dei Big Data in suo possesso è analizzare queste grandi moli di dati in tempi accettabili per il business aziendale. Apache Hadoop [2] è un framework open-source nato con l'obiettivo di risolvere quest'esigenza grazie a un'architettura di collezionamento e analisi di dati scalabile orizzontalmente. Il concetto della scalabilità orizzontale (scale-out) consiste nell'affiancare risorse informatiche a basso costo, dette commodity hardware,

per raggiungere performance e capacità molto elevate. Ciò si contrappone alla più tradizionale scalabilità verticale (scale-up) che prevedeva un innalzamento di questi parametri su una singola risorsa. I costi derivanti da questo secondo approccio crescono inevitabilmente in maniera esponenziale, come rappresentato nella Figura 2.13.

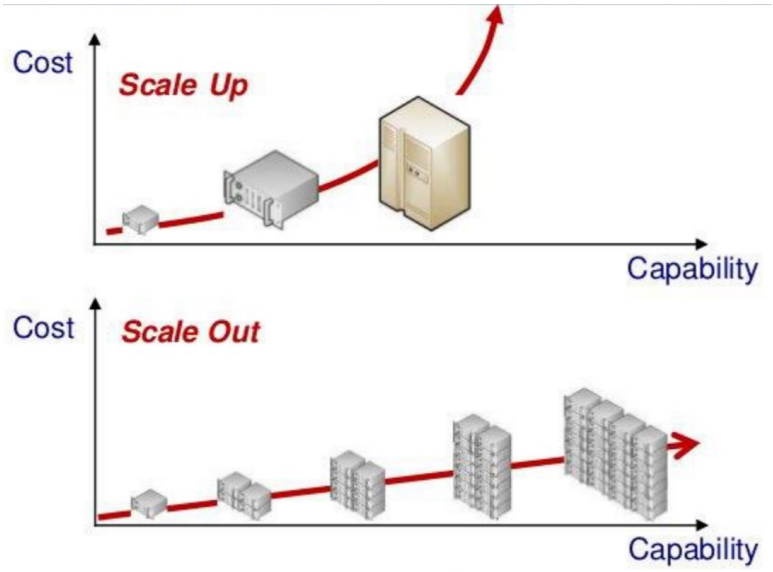

Figura 2.13. Confronto Scale Up vs Scale Down.

Hadoop è oggi uno standard de-facto per la BI e l' analisi di Big Data, grazie alla sua elevata scalabilità. La sua diffusione è stata favorita dal paradigma open-source che ha da sempre caratterizzato il prodotto: le librerie di alto livello sono messe a disposizione di qualsiasi programmatore e risultano di facile utilizzo.

Hadoop è un ottimo strumento per importare nel Data Lake le informazioni contenute nei numerosi database "legacy" di Intesa Sanpaolo, abilitandone l'elaborazione distribuita con vantaggi in termini di costi e performance, oltre che di condivisione dell'informazione fra le varie divisioni aziendali.

Come rappresenta la Figura 2.1, il Data Service Hub (DSH) sfrutta tali dati importati nel Data Lake grazie ad Hadoop, che alimentano a loro volta i singoli microservizi.

La Figura 2.14 mostra come su un unico vasto file system di tipo HDFS possano poggiare svariate metodologie di analisi, ciascuna delle quali si basa su una tecnologia differente e ottimizzata per un determinato obiettivo.

La natura open-source di Hadoop non deve, però, far pensare che l'utilizzo del prodotto sia completamente gratuito. Oltre ai costi dell'hardware occorre mettere in conto un prezzo di supporto professionale della soluzione. In caso di problemi sulla piattaforma Hadoop, un'azienda del calibro di Intesa Sanpaolo non potrebbe affidarsi soltato alle informazioni disponibili attraverso la community. Esistono infatti numerose società specializzate su questa suite (ad esempio, Replay, Accenture,

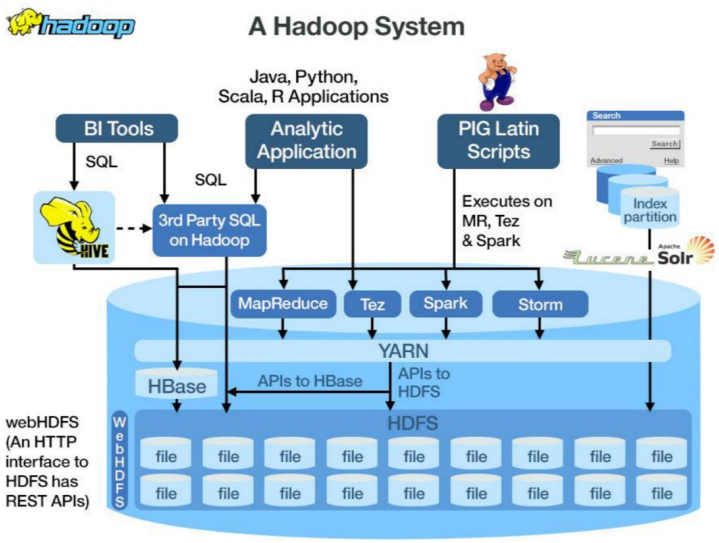

Figura 2.14. Esempio di ecosistema integrato con Hadoop

etc.) alle quali Intesa Sanpaolo si affida per avere supporto nell'implementazione, nella gestione e nel troubleshooting della piattaforma.

2.4.2 Teradata

Teradata è una soluzione proprietaria per prodotti e servizi del mondo analytics per i Big Data. Dunque l'hardware e il software prodotti e venduti da questa società non sono open-source, ed hanno inevitabilmente un costo maggiore, in termini di TCO, rispetto alla soluzione Hadoop. Di contro, si tratta di un prodotto molto più maturo e stabile. Il prodotto offre una vasta gamma di integrazioni con altre soluzioni. La Figura 2.15 fornisce un'idea della varietà di partnership che la riguardano [32].

Figura 2.15. Elenco integrazioni Teradata nel mondo analytics

La soluzione Teradata è basata su appliance, ossia dispositivi costituiti da hardware (server, memoria, storage, etc.) e software pre-configurati (sistema operativo, DBMS, etc.) in grado di gestire carichi di lavoro molto grandi. Fra i vantaggi di adottare un'architettura tecnologica formata da appliance ci sono la garanzia di compatibilità hardware/software, una preconfigurazione che massimizza le performance, nonchè l'elevata facilità di gestione derivante dal fatto che essa è in gran parte demandata al fornitore.

Nel mondo dei database, le appliance si posizionano sul mercato come piattaforme per il data wharehousing. Nell'ottica dei Big Data, possiamo considerare tale soluzione interessante se i dati da trattare hanno un elevato volume e al contempo espongono una struttura tale da poter essere rappresentati in database relazionali, come avviene nel caso di Intesa Sanpaolo, che ha una forte componente legacy. Al contrario non è consigliabile se tale condizione non si verifica, ad esempio nel caso di una start-up che non dispone di un vasto patrimonio informativo creato negli anni.

La peculiarità di Teradata consiste nella massimizzazione del parallelismo delle elaborazioni, anche su database tradizionali. La soluzione permette, infatti, la suddivisione del workload in piccole parti che vengono eseguite in maniera separata su più processori virtuali (dunque parti di software, non hardware) chiamati Access Module Processor (AMP). Più AMP risiedono su una singola appliance (detta nodo) e ognuno di essi è legato a una parte del database. Grazie a questa architettura, Teradata non si affida alla piattaforma hardware per il parallelismo e la scalabilità, bensì tali caratteristiche sono parte integrante del database stesso. Esiste poi un secondo tipo di processore virtuale, il Parsing Engine, che si incarica proprio della suddivisione del workload che dovrà essere eseguito da più AMP.

La Figura 2.16 mostra in maniera piuttosto semplice il design architetturale adottato da Teradata per garantire il parallelismo anche nell'elaborazione di database tradizionali [33].

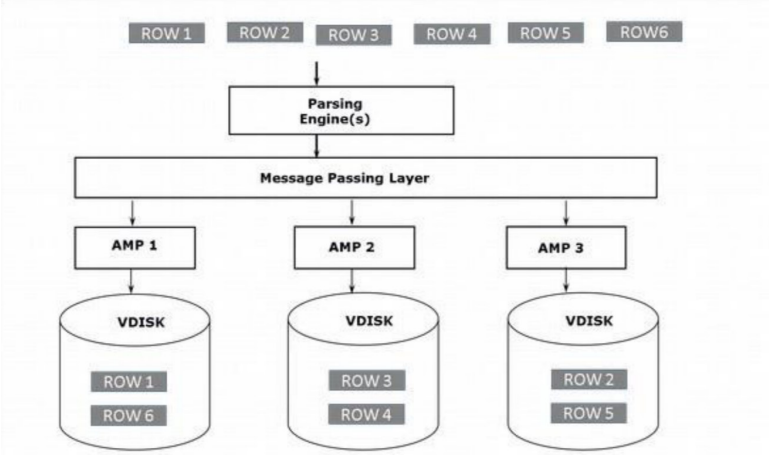

Figura 2.16. Architettura di base di Teradata

Dunque, al contrario di Hadoop, Teradata garantisce una sorta di retrocompatibilità con il mondo dei database legacy, poiché ne mantiene la struttura, pur implementando un sistema che ottimizza le performance rispetto ai workload sequenziali tradizionali. Per tale ragione, Teradata viene classificato in Intesa Sanpaolo come un prodotto ibrido, di congiunzione fra il mondo legacy e l'innovativa tecnologia Hadoop, ma non per questo da scartare, dal momento che alcune filiere si basano tutt'oggi sui database legacy.

2.4.3 DataStage

Dalla teoria riguardo il trattamento dei Big Data si evince la necessita di estrarre ed integrare i dati da sorgenti differenti. Successivamente questi dati possono essere raggruppati affinché vengano sottoposti ad elaborazioni matematiche che ne calcolino valori aggregati, oppure può essere creato un dataset pronto per l'utilizzo da parte di un algoritmo di classificazione. Per effettuare l'estrazione dai vari database legacy di Intesa Sanpaolo è stato scelto IBM InfoSphere DataStage.

IBM InfoSphere DataStage è una piattaforma di ETL che integra i dati di più sistemi eterogenei. Esso sfrutta un'architettura parallela, dunque scalabile, ad alte prestazioni. Può essere installato su server dedicati on-premise (come avviene in Intesa Sanpaolo) o può erogare servizio direttamente dal cloud IBM. In entrambi i casi, ovviamente, deve essere garantito un collegamento agli svariati database in cui risiedono i dati grezzi, cosicché possa estrarli, applicarvi eventuali trasformazioni e, infine, inserire i risultati nel Data Lake, che risiede sulle piattaforme Hadoop e Teradata.

La piattaforma DataStage offre una gestione estesa dei metadati tramite cui permettere di integrare dati eterogenei sia "at rest" che "in motion". Con il termine "data at rest" si intendono i dati salvati in modo persistente in database, file, etc., mentre i "data in motion" sono i dati in fase di elaborazione dalla CPU o salvati in memoria RAM [11]. Nel caso di Intesa Sanpaolo la quasi totalità dei dati integrati tramite DataStage consiste in "data at rest", poiché la banca dispone di un'elevata mole di informazioni storiche consolidate, si pensi, ad esempio, all'anagrafica dei clienti.

Il software DataStage utilizza un'architettura client/server in cui il server gestisce informazioni necessarie per eseguire i processi di ETL, mentre il client permette, mediante l'interfaccia grafica, di definire tali processi. Sono consentite agli utenti elaborazioni sia in modalità batch che in tempo reale. Anche in questo caso la componente statica dei dati è preponderante in Intesa Sanpaolo, che, infatti, adotta la modalità batch, solitamente eseguita durante l'orario notturno per evitare impatti sulle altre applicazioni.

DataStage mette a disposizione le funzioni di trasformazione predefinite che consentono l'integrazione dei dati. Per realizzare i processi ETL si creano dei flussi riutilizzabili in modo che la logica possa essere progettata una sola volta ed eseguita su uno qualsiasi dei server dedicati al software. Il programma eseguibile elementare prende il nome di job e, a seconda della sua tipologia, permette di effettuare l'estrazione, la trasformazione e il caricamento dei dati. Esso è composto al suo interno dagli stage, elementi più di dettaglio che eseguono una funzione dedicata. Per esem-

pio, lo stage di lettura di file consente, come dice il nome stesso, di leggere varie tipologie di file. I job sono costituiti dagli stage e hanno la forma di un flusso.

Il flusso comincia dalla lettura della sorgente di interesse, a cui segue una fase di raggruppamento, per poter passare alla fase successiva di caricamento. Al fine di migliorare le prestazioni, i dati appena letti vengono elaborati dagli stage successivi senza aver ancora completato l'intera lettura della fase precedente. Nella Figura 2.17 viene riportato lo schema di un flusso basilare , che prevede un job di lettura da un generico database, una trasformazione per ciascun record letto, e, alla fine, la creazione di un dataset contenente tutti i record [22].

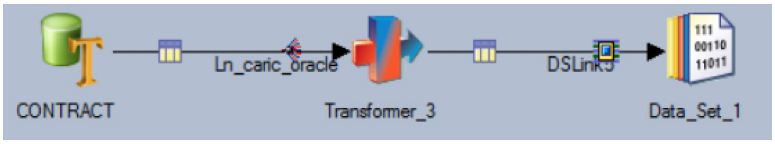

Figura 2.17. Esempio di flusso in InfoSphere Datastage

È possibile suddividere gli stage in due gruppi:

- *Input-output*: chiamati anche passivi, in quanto svolgono meramente la funzione di lettura-scrittura. Questi stage implementano l'operazione di Estrazione e Caricamento in un processo ETL.
- *Trasformer-aggregator*: chiamati anche attivi, si distinguono dai precedenti perché effettuano operazioni di modifica dei dati, conversione, join o altri tipi di trasformazioni.

Tramite flussi più articolati, definiti da specialisti di prodotto, Intesa Sanpaolo è in grado di reperire ed elaborare le informazioni ritenute necessarie. Come già accennato, il processo di ETL avviene ogni notte e si occupa di reperire un determinato set di informazioni ritenuto utile per tutte le funzioni aziendali che poggiano sul Data Lake.

2.4.4 SAS

SAS, acronimo di Statistical Analysis System, è un complesso di prodotti software integrati che permettono a un programmatore di inserire, ricercare e gestire dati, generando su di essi report e grafici basati su analisi statistiche e matematiche. Esso dunque rappresenta lo strumento di front-end con il quale viene analizzato il Data Lake per estrarne conoscenza. SAS consiste in tre principali finestre, come mostrato nella Figura 2.18.

Fra le finestre evidenziate, le più importanti sono, sicuramente, l'editor di programma, per scrivere e inviare il codice di analisi, la finestra di log, per il debugging del codice stesso, e la finestra di output per mostrare i risultati.

Un'interfaccia ancora più avanzata è SAS Enterprise Miner, mostrato in Figura 2.19, che consente la programmazione "a blocchi" per la definizione delle analisi e la rappresentazione dell'output. Il sistema drag and drop permette di trascinare dei cubi rappresentanti una determinata funzione fra quelle possibili. Queste ultime sono:

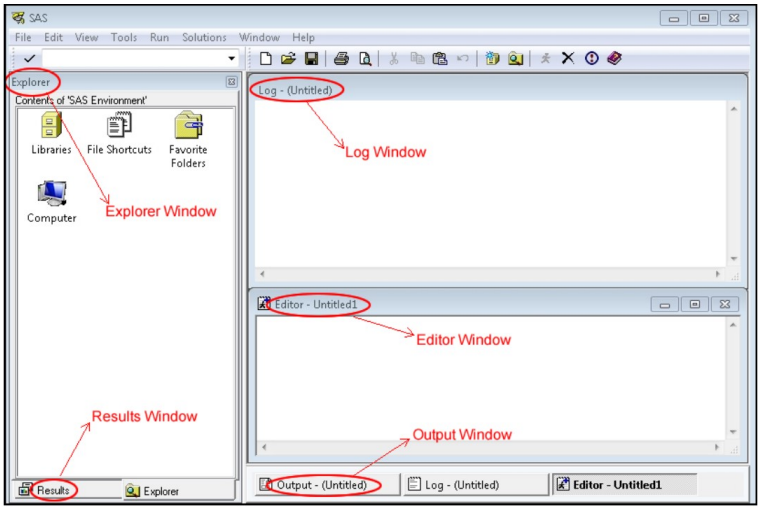

Figura 2.18. SAS Enterprice Guide

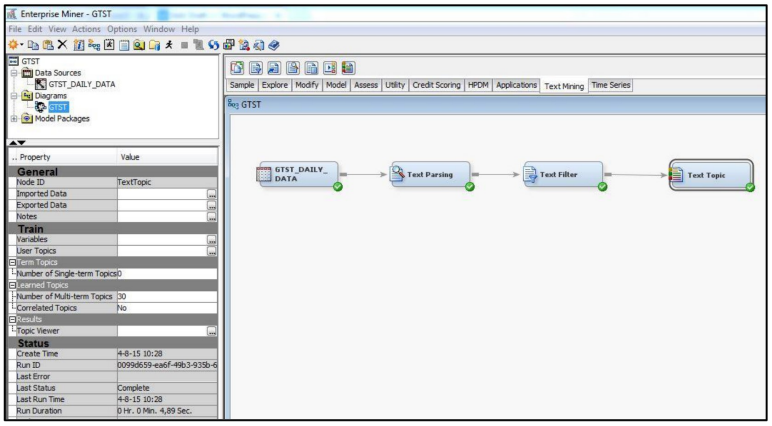

Figura 2.19. SAS Enterprice Miner

- Interrogazione della base dati che, in Intesa Sanpaolo, è il Data Lake.
- Elaborazione, sia essa statistica o matematica, dei dati estratti.
- Rappresentazione dei risultati mediante indici, grafici o tabelle.

Intorno agli anni 2000, in seguito alle numerose richieste dei clienti, SAS introdusse sul mercato quelle che vengono definite "soluzioni verticali", che permettono di progettare apposite applicazioni per risolvere uno specifico problema aziendale contestualizzato in un dato settore. Esse sono principalmente due:

- *SAS Enterprise Miner*: la già descritta interfaccia grafica, facile da utilizzare, per analisi descrittive.
- *SAS Forecast Server*: propone un'interfaccia intuitiva per il forecast, ossia per il processo di previsione di determinati indici utili al funzionamento societario.

I programmi SAS sono costituiti da fasi ben distinte e separate dette "step". Ogni passo viene completato prima di passare a quello successivo. Le fasi dei dati (Data step) sono scritte direttamente dall'utente e sono principalmente usate per la manipolazione dei dati (da cui il nome). Le fasi procedurali (Proc step) sono, invece, delle istruzioni di programmi rese disponibili direttamente da SAS. Uno step inizia con la parola DATA o con la parola PROC e termina con la parola RUN. Attraverso un processo che ha qualche analogia con quanto descritto per DataStage, l'utente (ossia il programmatore SAS), può, dunque, stabilire un flusso per l'analisi dei dati immagazzinati nel Data Lake e l'esposizione dei risultati.

La piattaforma SAS, dunque, offre un elevato potenziale nel campo dell'analytics, ma può apparire ostica per chi non vi opera abitualmente. Tuttavia uno degli obiettivi del Data Lake è ampliare la platea degli utilizzatori delle informazioni in esso contenute. Per questa ragione è bene che le aziende che adottano soluzioni come SAS, così come ha fatto Intesa Sanpaolo, prestino particolare attenzione nel favorire lo sviluppo del know how su queste piattaforme, anche internamente alla propria organizzazione.

2.4.5 Python

Python è un linguaggio di programmazione di più "alto livello" rispetto alla maggior parte degli altri linguaggi, orientato a oggetti, adatto, tra gli altri usi, a sviluppare applicazioni distribuite, scripting, computazione numerica e system testing. È un linguaggio multi-paradigma che ha, tra i principali obiettivi, dinamicità, semplicità e flessibilità. Supporta il paradigma object oriented, la programmazione strutturata, e molte caratteristiche di programmazione funzionale e riflessione.

All'interno dei progetti sviluppati con architettura a microservizi riceve in input le tabelle restituite da SAS, caricate attraverso un database Oracle, ed applica una serie di algoritmi di machine learning.

Le librerie di Python non sono altro che collezioni di metodi e funzioni che permettono di svolgere delle azioni senza scrivere il codice di ogni passaggio. Questo è molto utile perché permette di semplificare enormemente il nostro codice, sgravandolo dall'onere di dover implementare a mano una moltitudine di operazioni. Possiamo, quindi, considerare la libreria come un insieme di moduli. Ogni modulo contiene delle istruzioni e definizioni semplici. L'accorpamento di vari moduli, quindi di codice, costituisce una libreria. Spesso i moduli sono già stati scritti da altri sviluppatori, e non c'è bisogno di ripartire da capo ogni volta. Il loro scopo è quello di semplificare le attività, aiutando gli sviluppatori a scrivere solo poche righe anziché una grande quantità di comandi.

In particolare le librerie fondamentali utilizzate all'interno dei progetti dei vari motori predittivi, come il "Modello Calo Operatività", sono le seguenti:

- *`cx_Oracle`*: Python ha una libreria incorporata, `cx_Oracle`, per la connessione al DBMS Oracle.
- *`Pandas`*: prima di creare e addestrare un modello di machine learning è fondamentale trattare e gestire correttamente i dati. Un modo comune per farlo è avvalersi della libreria `Pandas`, che offre strutture dati e operazioni per manipolare tabelle numeriche e serie temporali.

- *NumPy*: il modulo `NumPy` contiene diverse funzioni e metodo utili per il calcolo scientifico, in particolar modo, per il calcolo vettoriale e matriciale.
- *os*: il modulo `OS` in Python fornisce l'accesso a funzioni specifiche del sistema per gestire il filesystem, i processi e lo scheduler.
- *TensorFlow*: è il framework di machine learning che facilita il processo di acquisizione di dati, modelli di addestramento, elaborazione di previsioni e perfezionamento dei risultati futuri. `TensorFlow` consente agli sviluppatori di creare grafici del flusso di dati; questi sono strutture che descrivono il modo in cui i dati si spostano attraverso un grafico o una serie di nodi di elaborazione. Ogni nodo nel grafico rappresenta un'operazione matematica, e ogni connessione o bordo tra i nodi è un array di dati multidimensionali, o tensore.

2.4.6 Bitbucket

Strettamente collegato a Python (Sezione 2.4.5) è Bitbucket, uno strumento di gestione del codice Git. Bitbucket mette a disposizione dei team uno spazio in cui pianificare i progetti, collaborare su codici, effettuare test e implementarne i risultati. Oltre le potenti librerie, Python ha anche il vantaggio di poter gestire il monitoring del codice; questo è un requisito fondamentale da parte dell'azienda in ottica di DevOps Basic 2.2 per il Continuous delivery (Sezione 2.3.4).

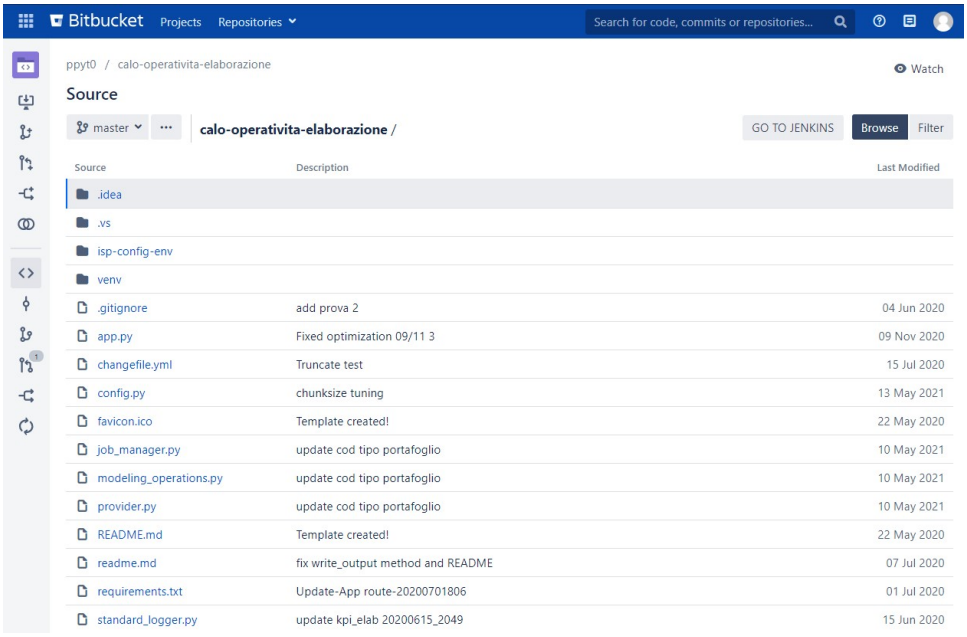

Figura 2.20. Repository del "Modello Calo Operitività" per il versioning Git in Bitbucket

2.4.7 SonarQube

SonarQube è una piattaforma open source per la gestione della qualità del codice. Può essere sintetizzata come un'applicazione web che produce report sul codice duplicato, sugli standard di programmazione, i test di unità, il code coverage, la complessità, i bug potenziali, i commenti, la progettazione e l'architettura. Il censimento di un progetto su SonarQube ha l'obiettivo di mostrare all'utente la prima analisi del codice sorgente del progetto, i cui risultati (segnalazione di eventuali issue e percentuale di coverage raggiunta) concorrono alla formazione del Quality Gate. A seconda del primo censimento del progetto di un acronimo su Sonarqube, arriverà il setting automatico una delle due tipologie di Quality Gate.

Per l'analisi statica del codice sorgente viene utilizzato il prodotto SonarQube che è in grado di analizzare diversi linguaggi di programmazione, fornendo evidenza delle violazioni di diverse regole specifiche per ogni linguaggio. Lo schema logico è rappresentato in Figura 2.21.

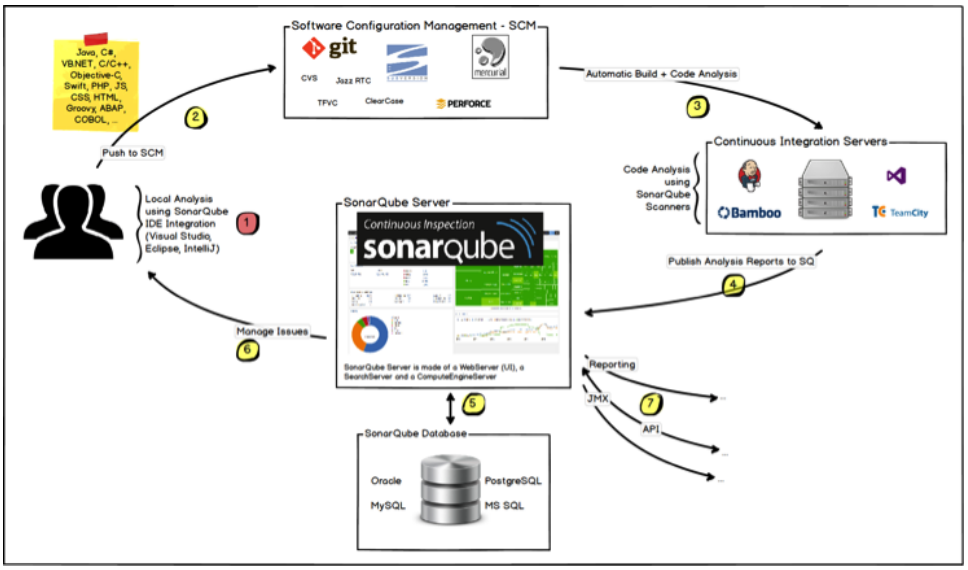

Figura 2.21. Schema logico SonarQube per la gestione della qualità del codice

L'applicazione SonarQube, basandosi sulle regole di buona programmazione, effettua analisi e rende visibili i risultati agli utenti utilizzando delle dashboard grafiche. Allo stato attuale le analisi effettuate sono:

- *Quality Gate differenziati*: fornisce un gate OK/KO per la promozione delle applicazioni verso gli ambienti di System. Componenti con il Quality Gate "Failed" non potranno essere promosse in System a meno di deroghe.
- *Gestione delle Issues*: attraverso filtri specifici che consentono di selezionare velocemente le Issues sulle quali intervenire. Un esempio della dashboard è presente in Figura 2.22

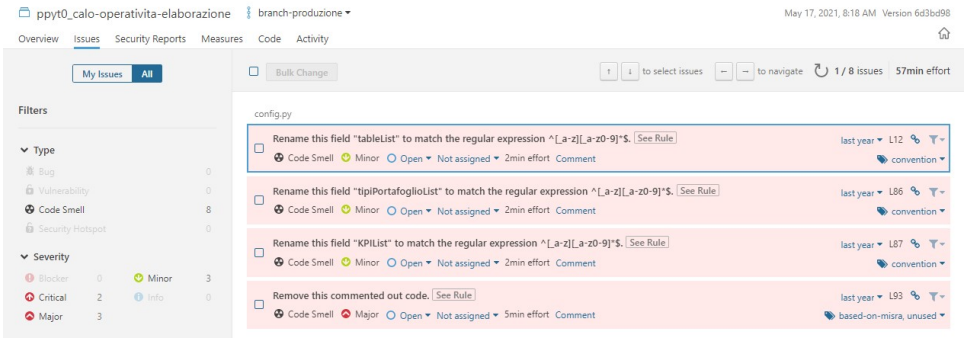

Figura 2.22. Issues per "Modello Calo Operatività"

- *Branching Analysis*: per mantenere lo stato della qualità di diversi branch. L'azienda utilizza su SonarQube principalmente 2 branch:
  - *branch-produzione*: anche chiamato "Master", contiene lo stato della qualità del codice presente in produzione;
  - *branch sviluppo*: contiene lo stato della qualità del codice in fase di sviluppo.
  - *hotfix*: branch dedicato alle emergenze. Utilizzando questo branch, non si bloccano gli sviluppi su env/svil e non si rischia di portare in produzione, a causa di dipendenze, del codice non ancora stabile.

  Un esempio del branching è mostrato in Figura 2.23, dove viene rappresentato il "Master", versione attualmente presente in produzione e il "Branch 5.6" attualmente presente in ambiente di sviluppo.

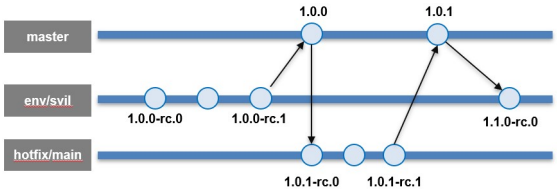

Figura 2.23. Brach per il "Modello Calo Operatività".

- *Governance del parco software*: definisce logiche di aggregazione delle diverse componenti applicative che costituiscono il Portafoglio Applicativo, consentendo una visione aggregata delle metriche.

2.4.8 Jenkins

Jenkins è un server open source di CI/CD (Continuous Integration, e Continuous Deployment), che consente di automatizzare le diverse fasi del ciclo di vita del software, dalla compilazione al test alla distribuzione. In realtà, esiste ancora sotto forma di modulo eseguibile da un application server, ma è anche distribuito come immagine Docker, il che ne semplifica l'uso in ambienti virtualizzati.

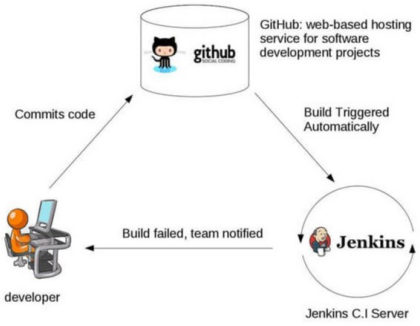

Figura 2.24. Struttura di comunicazione tra Jenkins e Git

È possibile programmare l'esecuzione di determinate attività utilizzando trigger, ad esempio per fare il commit del proprio codice all'interno di un repository, lanciare test ed analizzarne i risultati per avere un feedback immediato sulla sua qualità.

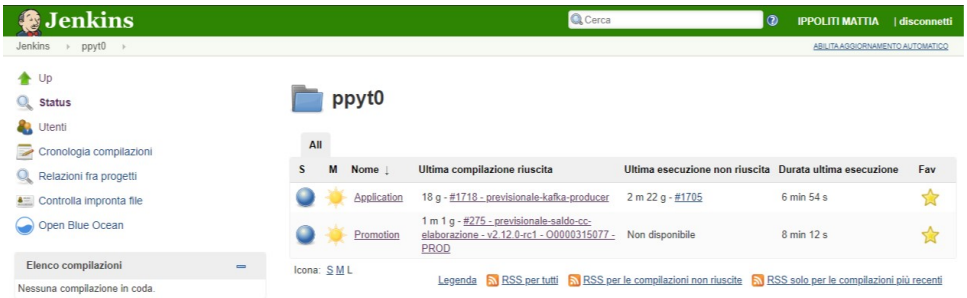

Figura 2.25. Portale Jenkins

2.4.9 PaaS Openshift

Le esigenze e le modalità di sviluppo, test e rilascio del software sono molto cambiate in questi ultimi anni. La progressiva standardizzazione di metodologie di DevOps ha creato una sempre più forte interdipendenza tra il mondo dello sviluppo e quello dell'amministrazione IT. In questo scenario ha trovato terreno fertile lo sviluppo di alcuni dei paradigmi di cloud computing come Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS) e Software-as-a-Service (SaaS).

Differenza tra Infrastruttura "On-Site" e PaaS

La differenza tra IaaS, PaaS e Saas è relativa al controllo sull'infrastruttura sottostante al software. La Figura 2.26 può aiutare meglio a comprendere i livello di astrazione di questi tre modelli:

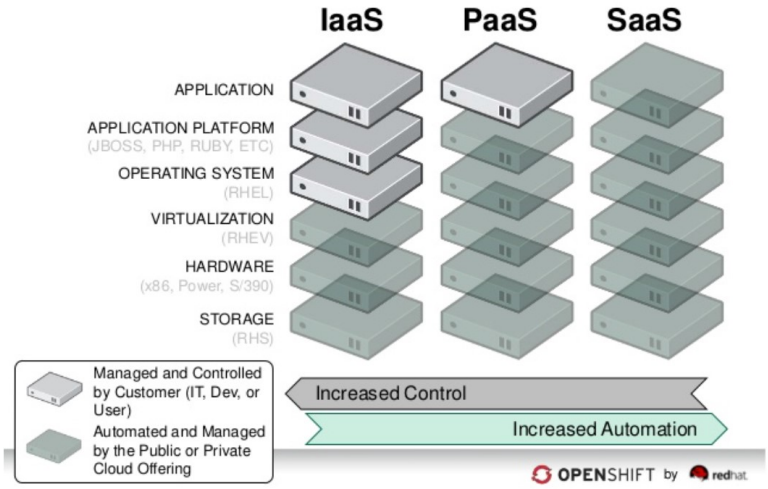

Figura 2.26. Modelli di servizi cloud

- In un ambiente IaaS si avrà il controllo sul sistema operativo, sulla piattaforma applicativa (application server, runtime, etc) e sull'applicazione stessa. Questo significa ovviamente che lo sviluppatore o l'amministratore di sistema avrà anche l'onere di installare, configurare e manutenere non solo il sistema ma anche librerie e runtime necessari all'applicazione. OpenStack è un tipico Infrastructure-as-a-Service.
- In un ambiente SaaS abbiamo solo le applicazioni pubblicate e l'utente non ha possibilità di agire sulla configurazione del software.
- In un ambiente PaaS il sistema e i runtime su cui opera l'applicazione sono forniti e sarà onere dell'utente sviluppare, distribuire e manutenere l'applicazione. Un PaaS fornisce servizi web server, application server, framework, database, messaging, Business Process Management e funzionalità di integrazione. OpenShift Enteprise rientra esattamente in questa definizione.

PaaS

Il Platform as a Service (PaaS) è un servizio di cloud computing in cui l'hardware e la piattaforma software applicativa vengono forniti da terze parti. Ciò offre una piattaforma su cui l'utente può sviluppare, eseguire e gestire applicazioni senza dover creare e gestire l'infrastruttura o la piattaforma normalmente associate a tali processi.

Il servizio PaaS prevede che hardware e software siano ospitati nell'infrastruttura del provider (Microsoft Azure) che distribuisce la piattaforma all'utente come soluzione integrata, stack di soluzioni o servizio erogato tramite una connessione Internet.

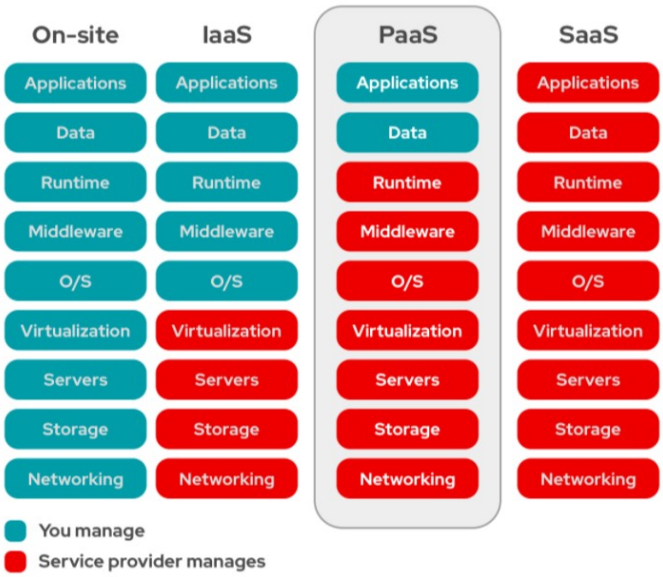

Figura 2.27. Differenze tra IaaS, PaaS e SaaS

Soluzione Platform as a Service di Red Hat: Openshift

Openshift è un *orchestratore di container* (Sezione 2.3.5), indispensabile per gestire sistemi distribuiti che fanno uso di un gran numero di container. In ottica del progetto Big Financial Data (BFD) che ha, alla sua base, l'obiettivo di progettare applicazioni basate sui microservizi (Sezione 2.3.2), Openshift viene utilizzato per la gestione e, la coesione, dei vari microservizi. Tuttavia gestire un gran numero di micro-servizi in un cluster di server è molto complicato senza un orchestratore di container: è necessario risolvere problemi di deployment combinato dei container, gestire il networking multi-host dei container, il service discovery, la presenza di volumi persistenti in modo distribuito, e così via. Openshift permette di gestire in modo semplice lo scaling dei servizi e il load balancing tra le repliche di uno stesso servizio. Questa funzionalità rende semplice ottenere lo scaling selettivo delle singole componenti di un'applicazione a micro-servizi.

Per concludere, il diagramma concettuale delle applicazioni sopracitate, utilizzate nella DSI, sono rappresentate in Figura 2.28. Partendo dal caricamento del codice sulla repository Bitbucket, questo triggera automaticamente Jenkins, software che aiuta il programmatore ad automatizzare il codice. A questo punto, una volta eseguita la build del codice, SonarQube andrà a verificare se il codice buildato rispetta i vincoli di qualità in base agli standard aziendali. Successivamente, passati i test di qualità del codice, si farà, in ambiente di laboratorio, il deploy del codice tramite piattaforma Openshift, creando un container apposito esposto al servizio. A questo punto, una volta eseguiti Unit test automatici nella piattaforma ALM, si passerà in ambiente di sviluppo, andando ad aggiornare il *branch di sviluppo*.

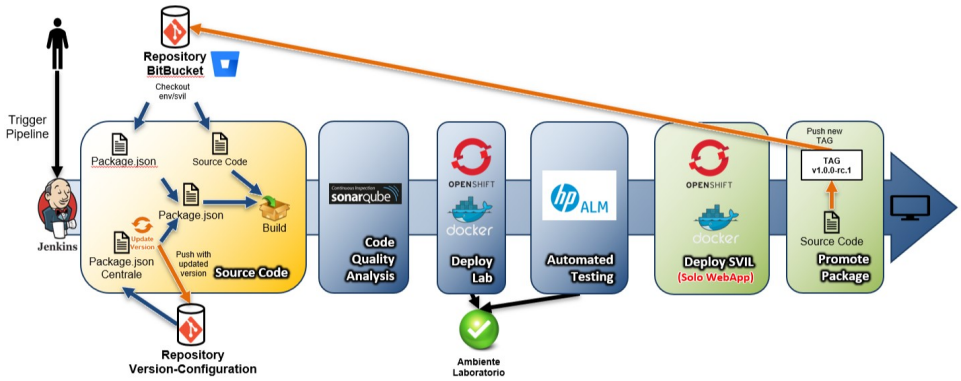

Figura 2.28. Diagramma concettuale della pipeline di build del codice.

3

Analisi dei requisiti

In questo capitolo si tratterà del modo in cui sono state utilizzate le tecnologie discusse nel Capitolo 2 per la progettazione e implementazione di un sistema per il monitoring dell'operatività delle aziende clienti di una banca. Si procederà, poi, all'analisi e raccolta dei requisiti del cliente, owner funzionale dell'applicazione.

3.1 Case Study

Tra le varie progettualità portate avanti all'interno della Data Service Hub (DSH), la presente tesi vuole soffermarsi sul *"motore del calo di intensità dell'operatività"*, finalizzato al monitoring delle attività di aziende clienti.

Alla base delle schedulazioni, effettuate dai *motori di calcolo aziendali*, vi è il server denominato *PPYT0*, avente 32 core, 384 Gb di RAM e 150 TB di archiviazione, il cui schema architetturale è presentato nella Figura 3.1.

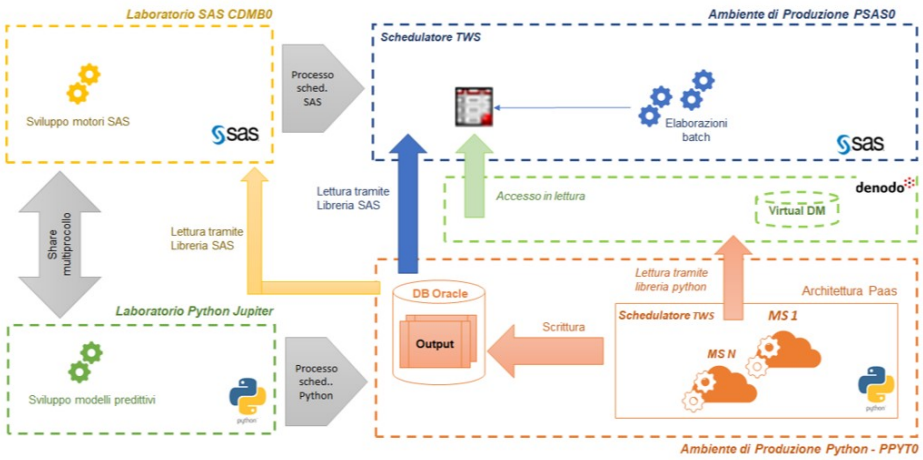

Figura 3.1. Schema di comunicazione architetturale del Server *PPYT0*

Come da schema architetturale, l'Ambiente di Produzione *PPYT0* prende mensilmente le strutture tabellari da un database Oracle, che, a sua volta, legge da *SAS*, attraverso una libreria dedicata. Una volta lette le tabelle in ingresso, il modello vero e proprio viene sviluppato in *Python* e i risultati verranno a loro volta salvati in strutture tabellari all'interno del database Oracle. In Figura 3.2, viene quindi mostrato qual'è il processo per il deploy del codice in Ambiente di Produzione.

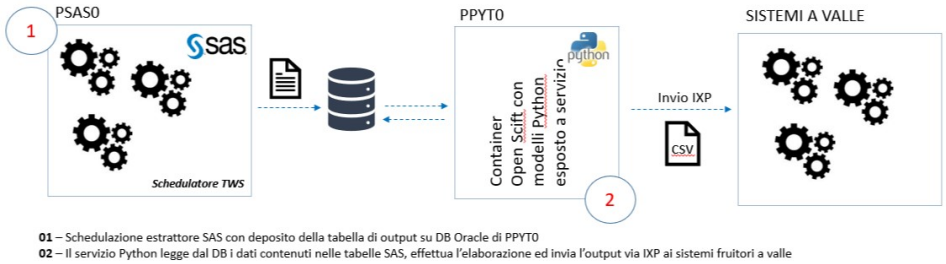

Figura 3.2. Processo di deploy del codice in Ambiente di Produzione.

Inoltre, come illustrato nello schema mostrato in Figura 3.1, in *PSAS0*, oltre che un Ambiente di Produzione e un Ambiente di Sistema di dimensioni ridotte, non è presente un Ambiente di Sviluppo. Al contrario, sul server *PPYT0*, l'architettura è diversa, dal momento che, come si può notare in Figura 3.3, sono presenti 3 ambienti, ovvero:

- *Ambiente di Sviluppo*, dove viene sviluppato il codice. Tale ambiente ha dimensioni più ridotte rispetto all'Ambiente di Produzione
- *Ambiente di Sistema*, all'interno del quale viene testato il codice e si procede alla sua manutenzione.
- *Ambiente di Produzione*, caratterizato da maggior numero di core e RAM. È in questo ambiente che girano le varie applicazioni a microservizi che, schedulate giornalmente o mensilmente, in base alle varie progettualità in esame, eseguono il codice necessario per ottenere le strutture tabellari utilizzate per fare reportistica.

Gli sviluppi in *SAS*, vengono effettuati su Ambiente di Produzione. Gli output, che dovranno essere caricati manualmente sull'DBMS Oracle di sviluppo, dopo la fase di data preparation, sono:

- la Customer Base
- 13 file di output con storico mensile per il calcolo dei KPI

A partire da questi verrà sviluppato, su ambiente *PPYT0* di sviluppo, il modello; quest'ultimo restituirà in output:

- il report completo;
- il report ridotto.

Attualmente il modello considera l'analisi di 13 KPI, mostrati nella Tabella 3.1.

KPI
Bonifici entrata IT
Bonifici uscita IT
Ritiro effetti
SOW CR BT (*)
RIBA RID MAV
F24
Incassi Estero
UT / ACC Banca BT (*)
Pagamenti estero
Pagamenti POS
Altri incassi Italia
Altri pagamenti Italia
Numero enti segnalanti

Tabella 3.1. Elenco dei KPI di riferimento. (*) Il calcolo di tutti i KPI è definito come volumi medi del trimestre, fatta eccezione per questi due KPI che sono calcolati come % media del trimestre.

A questi tre ambienti su *PPYT0* coincidono 3 database Oracle che vengono utilizzati per la lettura e scrittura dei dati, mentre su *PSAS0* si hanno dei file system. La differenza fondamentale tra file system e database sta nel fatto che il file system (disco rigido) gestisce solo l'accesso fisico, mentre il database gestisce sia l'accesso fisico che quello logico. Quest'ultimo è, quindi, progettato per organizzare, archiviare e recuperare facilmente grandi quantità di dati. In Figura 3.3 viene rappresentato il flusso in *Fase di sviluppo*, dove il caricamento da *PSAS0* al database Oracle avviene manualmente.

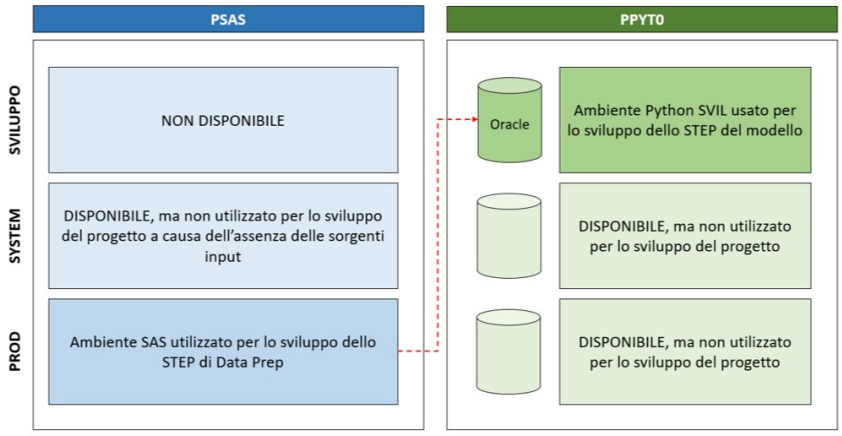

Figura 3.3. Flusso in fase di sviluppo, con caricamento manuale da *PSAS0* al database Oracle di *PPYT0* in Ambiente di Sviluppo.

Al contrario, in Figura 3.4 viene rappresentato il flusso in *Fase di produzione*,

dove il caricamento da *PSAS0* al database Oracle avviene automaticamente.

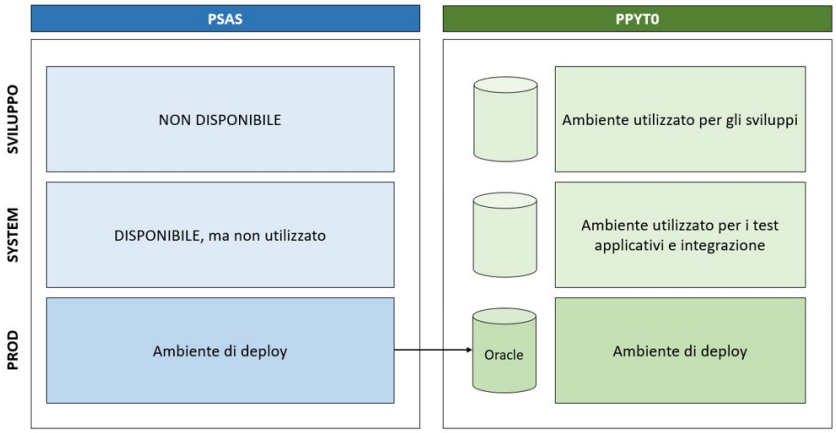

Figura 3.4. Flusso in fase di produzione, con caricamento automatico da *PSAS0* al DBMS Oracle di *PPYT0* in Ambiente di Sviluppo.

Per quanto riguarda il codice *SAS*, sviluppando direttamente in produzione, i test verranno effettuati direttamente in tale ambiente, fornendo il/i codice/i SAS (file `.sas`) come deliverable finale.

- *Ambiente di Sviluppo*: dall'Ambiente di Sviluppo verrà effettuato uno step di push su Git. Esso prevede anche la creazione di un tag associato alla versione precedentemente creata. Secondo il processo automatizzato, l'associazione del tag permette il passaggio in Ambiente di Sistema.
- *Ambiente di Sistema*: in Ambiente di Sistema verranno effettuati tutti i test necessari per il passaggio in produzione. In particolare:
  - *Test di creazione delle tabelle Oracle*: per effettuare questo test sarà necessario utilizzare l'Ambiente di Sistema di PSAS integrato con il database Oracle di tale ambiente.
  - *Test di popolamento delle tabelle Oracle*: in questa fase verrà testato lo step di loading che carica gli output SAS su Oracle.
  - *Test del codice Python.*
  - *Test E2E con la configurazione del microservizio.*
- *Ambiente di Produzione*: la parte di deploy del microservizio verrà effettuata su Openshift. Per la parte SAS verrà fornito il codice (file `.sas`).

3.2 Raccolta dei requisiti

La raccolta dei requisiti necessari all'implementazione delle funzionalità per monitorare i movimenti di aziende clienti e persone giuridiche si è basata principalmente sullo studio della documentazione messa a disposizione dal Business Owner (BO)

che, con il supporto del Digital Business Parter (DBP), ha fornito i requisiti di business al Data Service Hub (DSH), sotto forma del documento di Business Requirement Book (BRB). Il Business Owner di tale progetto è l'ufficio *Advanced Analytics*, che utilizza le tabelle in output per creare della reportistica per la pianificazione commerciale.

Tuttavia, a seguito dell'integrazione UBI, il documento nel BRB ha introdotto nuove funzionalità a seguito di due "Change Request (CR)", che hanno avuto come obiettivo quello di espandere il bacino anagrafico degli utenti. Pertanto, si è dovuto integrare il nuovo requisito, evitando di creare conflitti e ridondanze e, al tempo stesso, cercando di minimizzare l'impatto sui tempi di compilazione del motore. Non è stato necessario creare nuovi codici, ma si è lavorato su quelli già esistenti, effettuando il tuning e modificando alcuni parametri come il *chunk size*.

Un secondo punto di attenzione è la richesta di mascheratura del dato da parte dell'utente; la *CustomerBase*, avendo dei dati sensibili, non potrà effettuare direttamente il caricamento dei dati sui database, ma, al contrario, bisognerà prima mascherare questi ultimi.

3.2.1 Analisi di alto livello

Inizialmente, è stata effettuata un'analisi ad alto livello, partendo dalla documentazione disponibile, al fine di individuare i possibili attori coinvolti nei processi, e delineare, per sommi capi, le operazioni da loro svolte. Da ciò è emerso che il perimetro clienti prevedesse circa 1 milione di soggetti; tale numero può considerarsi scalabile, a meno di eventi particolari. In realtà, a causa dell'integrazione con la banca UBI, sono stati aggiunti circa 300000 nuovi soggetti per le analisi. Con la seconda "change request" per la modifica dei clienti UBI, la fonte alimentante l'anagrafica è stata modificata e il nuovo input è costituito da una tabella differente dalla precedente, poiché quest'ultima è stata dismessa.

3.2.2 Requisiti di dati in Input

Le tabelle che dovranno essere lette in input da *SAS* sono state estratte dal Data Lake aziendale grazie ad Hadoop (Sezione 2.4.1) e Teradata (Sezione 2.4.2). Sarà poi *SAS* stesso a produrre tutte le strutture KPI di input necessarie per l'esecuzione del modello in *Python*. Questa prima fase è chiamata *"modulo di data preparation"*. A questo punto i dati in uscita verranno passati, tramite database Oracle, al modello vero e proprio scritto in linguaggio *Python*.

3.2.3 Deliverable finali

Come requisito in uscita, l'utente ha richiesto tre applicazioni TWS (Sezione 2.3.6), con schedulazione il 15 di ogni mese alle ore 03:00. Esse comprendono:

- job per il perimetro dei clienti;
- job per l'area di analisi;
- job per il livello di loader sul database Oracle;

Inoltre, oltre ai requisiti dell'applicazione in oggetto, l'utente ha richiesto ulteriori deliverable in termini di documentazione. A tal fine il cliente ha richiesto:

- la documentazione funzionale (AFU);
- la documentazione tecnica (ATE);
- il documento di test sui singoli output, composto da:
  - una Customer Base (1 tabella in output);
  - un output con dati mensili relativi alle operazioni per il calcolo dei KPI (13 tabelle in output);
  - un output del modello (2 tabelle in output).
- un file zip contenente i programmi `.sas`;
- un file zip contenente i programmi `.py`;
- la documentazione per schedulazione.

4

Progettazione

Nel capitolo corrente verrà trattata la fase di progettazione delle varie funzionalità, richieste durante l'analisi dei requisiti descritta nel Capitolo 3. Verranno, dapprima, discussi il modulo di data preparation sviluppato in SAS e la relativa l'organizzazione del codice, grazie l'utilizzo di diagrammi di flusso, con particolare attenzione alle tre fasi principali del progetto che compongono questo primo step. Successivamente, verrà trattato il modello vero e proprio sviluppato in Python, oltre che l'organizzazione del codice attraverso l'uso di ulteriori diagrammi di flusso.

4.1 Disegno della soluzione

Il progetto, come da Figura 4.1, prevede tre fasi principali, ovvero:

- *il modulo di data preparation*, finalizzato al recupero di tutti i dati necessari al calcolo dei KPI (sviluppato in tecnologia *SAS*);
- *il modulo del calo di operatività*, finalizzato al calcolo dello score finale, sulla base delle regole attualmente richieste dagli utenti (sviluppato in tecnologia *Pyhton*);
- *la pubblicazione dell'output Report Completo* e *Ridotto* sul database Oracle, accessibili in autonomia dall'utente tramite laboratorio *SAS*, attraverso la configurazione di una libreria *SAS* ad-hoc che permette tale accesso.

Queste tre fasi comprendono dei sotto-moduli, come mostrato in Figura 4.2.

4.2 Modulo data preparation SAS

Il modulo di data preparation, il quale definisce il perimetro degli utenti su cui calcolare i KPI, si divide, a sua volta, in tre fasi principali, ovvero:

- *Il livello di perimetro dei clienti*, dove si definisce sia il perimetro di clienti interessati per il monitoring del calo di operatività, sia l'intervallo temporale interessato da tale analisi. L'output in questa fase è la Customer Base, tabella contenente il perimetro temporale e di utenza, la quale sarà utilizzata da tutti i 13 KPI.

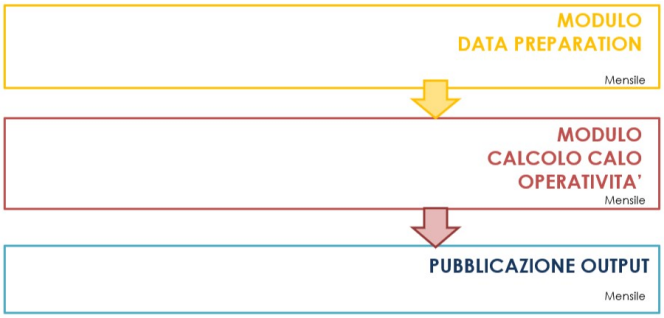

Figura 4.1. Disegno della soluzione per le tre fasi principali del progetto.

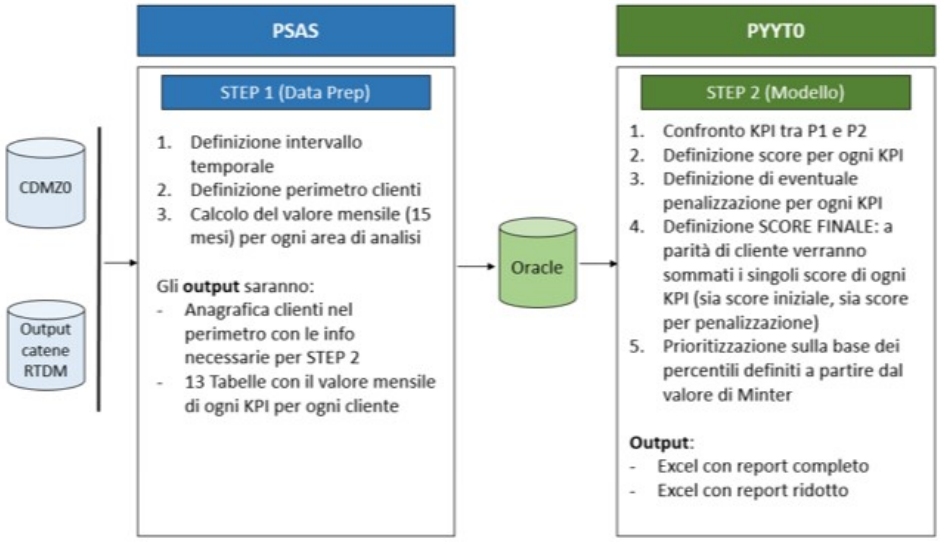

Figura 4.2. Disegno dettagliato delle tre fasi principali dei progetto.

- *Il livello delle aree di analisi*, dove verranno sviluppate delle estrazioni indipendenti finalizzate a calcolare i 13 KPI, a partire da 8 diversi job, a cui corrispondono 8 programmi *SAS* con estensione `.egp`.
- *Il livello di loader*, dove, attraverso una libreria ad-hoc *SAS*, si andranno a caricare sia la Customer Base che le 13 tabelle dei KPI sul database Oracle, per poter essere lette dal modulo Python attraverso la libreria `cx_Oracle`.

In Figura 4.3 viene rappresentato lo schema logico del modulo di data preparation, diviso nelle tre fasi principali sopra riportate.

4.2.1 Livello Perimetro Clienti

In questa prima fase, si sono estratti, dal Data Lake aziendale, varie tabelle di input all'interno dei numerosi database legacy di Intesa SanPaolo, i quali vengono

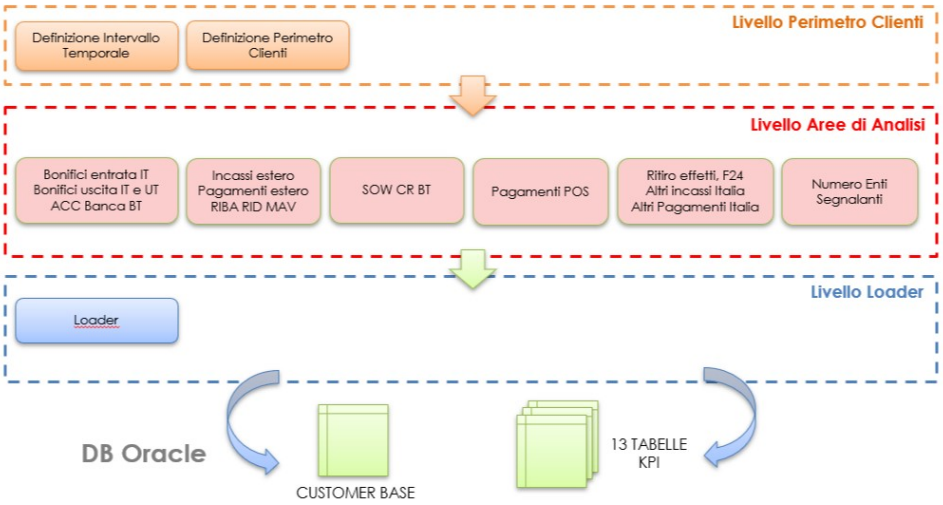

Figura 4.3. Schema della prima fase di *data preparation*

alimentati dal server *CMDZ0*, grazie a tecnologie Hadoop (Sezione 2.4.1) e Teradata (Sezione 2.4.2).

Definizione del perimetro dei clienti

Nella prima fase di data preparation in *SAS*, viene trovato il perimetro degli utenti della Customer Base. Questo task viene effettuato inserendo dei filtri sui `COD_PORTAFOGLIO` (codici di tipo portafoglio). Come da requisiti di progetto, si è interessati unicamente alle aziende retail o ditte individuali, e, a tal fine, vengono definiti i seguenti filtri necessari per l'analisi dei risultati del modello:

- `FLAG_PERIMETRO`, che indica i clienti con un determinato rating;
- `FLAG_NOTIZIE_PREGIUDIZIEVOLI`, che indica la presenza o meno di notizie pregiudizievoli associate al cliente;
- `FLAG_PRODOTTI_MLT`, che indica se il cliente ha almeno un contratto aperto negli ultimi 3 mesi e se esso appartiene ad aziende retail o ditte individuali;
- `FLAG_PRODOTTI_TUTELA`, dove, per ogni cliente, verranno controllati i contratti aperti con `cod_area_informativa` in ("`POL`") negli ultimi tre mesi;
- `FLAG_STARTUP`, che indica se il cliente ha un conto corrente attivo da almeno 15 mesi.

Tutti questi filtri sono parametrizzati in un'unica tabella di configurazione, denominata "`tabconfigurazione.SAS.7bdat`", utilizzata come tabella di input. In output da tale tabella, si avrà che ogni colonna rappresenta un filtro, il quale viene applicato per la creazione della *Customer Base finale*; tali filtri serviranno, poi, a definire le *aree dei singoli KPI*.

Definizione dell'intervallo temporale

L'analisi copre un periodo di 15 mesi a partire dall'ultimo mese consolidato, fatta eccezione per i due KPI *SOW BT* e *enti segnalanti*, per i quali, a causa di una differente disponibilità del dato, è necessario andare indietro di altri due mesi.

Ad esempio, ipotizzando di lanciare il codice nei primi giorni di aprile 2020, i mesi analizzati saranno quelli riportati in Tabella 4.1, a seconda del KPI in esame.

MESE ID	ANNO_MESE	ANNO_MESE per SOW BT e enti segn.
t-0	202003	202001
t-1	202002	201912
t-2	202001	201911
t-3	201912	201910
t-4	201911	201909
t-5	201910	201908
t-6	201909	201907
t-7	201908	201906
t-8	201907	201905
t-9	201906	201904
t-10	201905	201903
t-11	201904	201902
t-12	201903	201901
t-13	201902	201812
t-14	201901	201811

Tabella 4.1. Tabella di esempio della visualizzazione delle mensilità in base al KPI

In particolare, come si può notare dalla Figura 4.4, per tutti i KPI vengono confrontate le numeriche dell'ultimo trimestre (P1), con le numeriche dello stesso trimestre dell'anno precedente (P2). Per i due KPI *SOW BT* e *enti segnalanti*, riportati nell'ultima colonna della Tabella 4.1, a causa di un ritardo nella ricezione dei dati, si va indietro di altri due mesi. Per questi due KPI vengono infatti confrontate le numeriche di t_2, t_3 e t_4 con quelle di t_{14}, t_{15} e t_{16}.

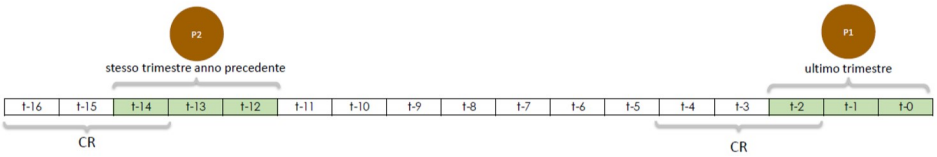

Figura 4.4. Confronto tra ultimo trimestre e stesso trimestre dell'anno precedente.

In questo modo, per ogni cliente viene calcolato il *minter annuale* e *trimestrale* a partire dalla tabella `TS_CDMZ_RPC_MARGN_VOLM_SINTS`. Per *minter* si intende la marginalità annuale, ottenuta confrontando la media del trimestre più recente, con quella del trimestre corrispettivo dell'anno precedente. A partire da questa tabella verranno considerati i mesi:

- t-0, t-1, t-2 per il *margin P1* (`MARG_TRIM_P1`);
- t-12, t-13, t-14 per il *margin P2* (`MARG_TRIM_P2`);
- t-0, t-11 per il *minter annuale*.

Il *minter annuale* corrisponde al campo `IMP_MINTER_ANTE_RETTIFICHE`, dove il *minter* è calcolato su un orizzonte temporale di 12 mesi, come somma dei campi riportati nel Listato 4.1.

```
1    IMP MINTER ANTE RETTIFICHE= sum (t1.IMP INTERESSI EFFETTIVI,t1.IMP INTERESSI FIGURATIVI, t1.IMP BENEFICI DI VALUTA
           , t1.IMP COMMISSIONI ORIGINATION, t1.IMP COMMISSIONI MANTENIMENTO, t1.IMP COMMISSIONI DIVERSE, t1.IMP COSTI
            RICAVI)
```

Listato 4.1. Calcolo del minter annuale.

Al contrario, il *minter trimestrale* è calcolato sull'ultimo trimestre e sullo stesso trimestre dell'anno precedente, corrispondente ai campi `IMP_MARGIN_TRIM_P1` e `IMP_MARGIN_TRIM_P2`. I campi `margin_trim_p1` e `margin_trim_p2` sono quindi calcolati come somma dei campi utilizzati per il calcolo del *minter annuale*, ma su un orizzonte temporale di 3 mesi, rispettivamente degli ultimi tre mesi (rispetto alla data di esecuzione dell'algoritmo) e dei rispettivi tre mesi dell'anno precedente.

Il campo `IMP_DELTA_MARGIN` è definito come differenza tra il margine dell'ultimo trimestre e il trimestre dell'anno precedente.

Il campo `PRC_DELTA_MARGIN` è definito come rapporto tra la differenza dei margini trimestrali e il valore assoluto del margine trimestrale dell'anno precedente, se quest'ultimo è diverso da zero. Questo indice contiene, quindi, il valore percentuale di margine delta, e viene calcolato come:

$$\texttt{prc\_delta\_margin} = \frac{\sum(\texttt{imp\_margin\_trim\_p}_1 - 1 * \texttt{imp\_margin\_trim\_p}_2)}{|\texttt{imp\_margin\_trim\_p}_2|}$$

Customer Base

La *Customer Base finale* filtrata sarà composta dai seguenti campi riportati in Tabella 4.2.

Perciò, tale tabella, serve a definire i clienti che sono nello "scope" dei KPI e, soprattutto, come guida dell' `anno_mese` della Customer Base. Da qui viene definita la tabella `anno_mese_run`, che rappresenta l'orizzonte temporale su cui bisogna effettuare l'analisi.

4.2.2 Livello Aree di Analisi

Se, nella Sezione 4.2.1, l'obbiettivo era delineare il perimetro dei clienti, creando degli appositi filtri su cui poi costruire la Customer Base, questa sezione presenta le aree di analisi su cui costruire gli 8 job *SAS*, a cui corrispondono come output le seguenti tabelle dei KPI:

(i) `BON_A_IT`: tabella contenente i movimenti relativi a bonifici italiani in ingresso aggregati mensilmente per ogni cliente nel perimetro del modello.
(ii) `BON_D_IT`: tabella contenente i movimenti relativi a bonifici italiani in uscita aggregati mensilmente per ogni cliente nel perimetro del modello.

Nomi Tabelle Input	Nomi CustomerBase Finale
anno_mese_t0	NUM_ANNO_MESE
cod_banca	COD_ABI
cod_ndg	COD_NDG
COD_SUPERNSG	COD_SUPERNSG
DT_DESC	DES_DT
cod_area	COD_AREA
AREA_DESC	DES_AREA
cod_filiale	COD_FILIALE
FILIALE_DESC	DES_FILIALE
NOM_ANAGRAFICA_NDG	NOM_ANAGRAFICA_NDG
COD_TIPO_PORTAFOGLIO	COD_TIPO_PORTAFOGLIO
COD_SPECIE_GIURIDICA	COD_SPECIE_GIURIDICA
cod_fiscale	COD_FISCALE
cod_partita_iva	COD_PARTITA_IVA
COD_TIPO_NDG	COD_TIPO_NDG
COD_SEGMNTZ_REGLM_ATTUALE	COD_SEGMNTZ_REGLM_ATTUALE
' '	DES_SEGMNTZ_REGOLAMENTARE
COD_MATRICOLA_GESTORE	COD_MATRICOLA_GESTORE
NOM_NOMINATIVO_GESTORE	NOM_NOMINATIVO_GESTORE
COD_FIGURA_GESTORE	COD_FIGURA_GESTORE
DES_FIGURA_GESTORE	DES_FIGURA_GESTORE
COD_CLASSE_RATING_DEFNT_PD	COD_CLASSE_RATING_DEFNT_PD
COD_STATO_AMMNS_GRUPPO	COD_STATO_AMMNS_GRUPPO
flag_perimetro	FLG_PERIMETRO
flag_startup	FLG_STARTUP
flag_mlt	FLG_PRODOTTI_MLT
flag_tutela	FLG_PRODOTTI_TUTELA
flag_not_preg	FLG_NOTIZIE_PREGIUDIZIEVOLI
IMP_MINTER_ANTE_RETTIFICHE	IMP_MINTER_ANTE_RETTIFICHE
IMP_MARGIN_TRIM_P1	IMP_MARGIN_TRIM_P1
IMP_MARGIN_TRIM_P2	IMP_MARGIN_TRIM_P2
IMP_DELTA_MARGIN	IMP_DELTA_MARGIN
PRC_DELTA_MARGIN	PRC_DELTA_MARGIN

Tabella 4.2. Elenco delle colonne della Customer Base.

(iii) RITIRO_EFFETTI: tabella di input con i movimenti relativi ai pagamenti per ritiro effetti, aggregati mensilmente per ogni cliente nel perimetro del modello. Vengono chiamati effetti bancari i servizi che le banche svolgono per conto e a rischio di privati o di altre banche corrispondenti. Si tratta di un metodo di pagamento sicuro per il soggetto beneficiario. L'effetto bancario viene più comunemente chiamato cambiale, ovvero il titolo rappresentativo il credito esecutivo corrisposto tra le parti.

(iv) SOW_UT_BT: tabella contenente il valore del rapporto tra *utilizzato banca BT* e *utilizzato BT complessivo*, aggregati per mese per ogni cliente nel perimetro del modello.

(v) RID_RIBA_MAV: tabella contenente i movimenti relativi ai pagamenti di RID, MAV e RIBA, aggregati mensilmente per ogni cliente nel perimetro del mo-

dello. La Ri.Ba (Ricevuta Bancaria elettronica) consente alle imprese creditrici di emettere ricevute bancarie fornendo alla banca le informazioni ad esse relative (tramite supporto magnetico, telematico o cartaceo), in sostituzione della materialità dei titoli.
La procedura RID (Rapporti Interbancari Diretti) è un sistema interbancario realizzato per gestire gli incassi mediante l'addebito preautorizzato sul conto corrente del debitore. Il servizio risulta particolarmente indicato per l'incasso di crediti con cadenza periodica rivenienti da attività commerciali e di servizi (ad esempio premi assicurativi, affitti, rate leasing, utenze, ecc.).
Il MAv (Incasso Mediante AVviso) è un prodotto che consente la gestione degli incassi senza fissarne la domiciliazione. Su istruzioni dell'impresa creditrice, la banca predispone ed invia al debitore i bollettini di versamento che possono essere pagati presso qualsiasi sportello bancario o ufficio postale. Il servizio risulta particolarmente indicato per l'incasso di crediti con cadenza periodica o estemporanea (quote associative, acquisti per corrispondenza, ecc.) indirizzati a clientela privata.

(vi) `F23_F24`: tabella di input con i movimenti relativi ai pagamenti F23/F24, aggregati mensilmente per ogni cliente nel perimetro del modello.L'F23 è un modello da utilizzare presso banche, sportelli di Agenzia delle entrate-Riscossione e uffici postali per tutti i versamenti in favore di enti diversi dall'Amministrazione finanziaria (comuni, uffici giudiziari, ecc.) e per il pagamento di alcune imposte indirette (per esempio l'imposta di registro e le imposte ipotecarie e catastali). Al contrario l'F24 è utilizzato per il versamento e la compensazione di gran parte delle imposte e contributi dovuti, a cominciare da quelli risultanti dalla dichiarazione con il modello Unico.

(vii) `INCASSI_ESTERO`: tabella contenente i movimenti relativi ai pagamenti esteri, aggregati mensilmente per ogni cliente nel perimetro del modello.

(viii) `UT_ACC`: tabella contenente il valore del rapporto *utilizzato su accordato*, aggregati per mese per ogni cliente nel perimetro del modello.

(ix) `PAGAMENTI_ESTERO`: tabella contenente i movimenti relativi ai pagamenti esteri, aggregati mensilmente per ogni cliente nel perimetro del modello.

(x) `POS`: tabella contenente i movimenti POS, aggregati mensilmente per ogni cliente nel perimetro del modello.

(xi) `INCASSI_ITALIA`: tabella di input con i movimenti relativi agli incassi in euro dall'Italia, aggregati mensilmente per ogni cliente nel perimetro del modello.

(xii) `PAGAMENTI_ITALIA`: tabella di input con i movimenti relativi ai pagamenti in euro dall'Italia, aggregati mensilmente per ogni cliente nel perimetro del modello.

(xiii) `NUMERO_ENTI_SEGNALANTI`: tabella contenente il numero di enti segnalanti, aggregati mensilmente per ogni cliente nel perimetro del modello.

All'interno dei KPI vi è uno storico di quindici mesi per ogni cliente e, per ogni `anno_mese`, viene riportata la somma degli importi e del numero delle operazioni. In linea generale, per ogni cliente identificato dal `COD_NDG` e dal `cod_supernsg`, verranno visualizzate 15 mensilità. Tuttavia, se in un determinato mese non viene fatto alcun movimento transazionale, allora tale mese non comparirà nella tabella finale. Come si può notare dalla Figura 4.5, il primo cliente, caratterizzato da `COD_NDG`=0000002828000, presenta unicamente 12 righe, a riprova del fatto che nei

tre mesi mancanti non ha svolto alcun movimento. Si è, inoltre, aggiunta un ultima colonna "DESCR_KPI", uguale per tutte le righe, la quale servirà poi nella parte *Python* per specificare il motivo delle segnalazione e, perciò, del perché vi è stato un calo di operatività per quel determinato cliente.

Il job Bonifici_Entrata_Uscita_IT genera in uscita le tabelle Bonfici Dare (in uscita) e Bonifici Avere (in entrata). Tali tabelle hanno quasi tutte la stessa struttura. Preso, ad esempio, il job "Bonifici_Entrata_Uscita_IT", per ogni COD_SUPERNSG e per ogni ANNO_MESE verranno tracciate le informazioni su:

- il numero di operazioni (NUM_MOV);
- l'importo delle operazioni (IMP_MOV).

	NUM_ANNO_MESE	COD_ABI	COD_NDG	COD_SUPERNSG	IMP_MOV	NUM_MOV	DESC_KPI
1	202005	01025	0000002828000	0000000016864417	3025.000	2	BON_ENTRATA_IT
2	202006	01025	0000002828000	0000000016864417	868.420	1	BON_ENTRATA_IT
3	202007	01025	0000002828000	0000000016864417	3125.000	2	BON_ENTRATA_IT
4	202008	01025	0000002828000	0000000016864417	500.000	1	BON_ENTRATA_IT
5	202011	01025	0000002828000	0000000016864417	1125.000	3	BON_ENTRATA_IT
6	202012	01025	0000002828000	0000000016864417	3106.000	2	BON_ENTRATA_IT
7	202102	01025	0000002828000	0000000016864417	400.000	1	BON_ENTRATA_IT
8	202103	01025	0000002828000	0000000016864417	2878.000	2	BON_ENTRATA_IT
9	202104	01025	0000002828000	0000000016864417	2000.000	1	BON_ENTRATA_IT
10	202105	01025	0000002828000	0000000016864417	4725.000	5	BON_ENTRATA_IT
11	202106	01025	0000002828000	0000000016864417	2497.160	2	BON_ENTRATA_IT
12	202107	01025	0000002828000	0000000016864417	2875.000	1	BON_ENTRATA_IT
13	202005	01025	0000047284000	0000000006292540	3351.980	1	BON_ENTRATA_IT
14	202006	01025	0000047284000	0000000006292540	14574.380	2	BON_ENTRATA_IT
15	202007	01025	0000047284000	0000000006292540	3351.980	1	BON_ENTRATA_IT

Figura 4.5. Tabella "ts_cope_inp_bon_a_it.SAS.7bdat" in output dal *SAS* contenente i bonifici in entrata.

L'unica tabella che cambia in output è "ts_cope_inp_enti_seg.SAS.7bdat", relativa al job "num_enti_segnalanti.egp", mostrata in Figura 4.6, dove si ha unicamente il numero di operazioni.

Al contrario, per quanto riguarda i job "SOW_CR_BT.egp" e "UT_ACC_Banca_BT.egp" e i KPI risultati non calcolano il numero di operazioni e l'importo ma, unicamente, la *percentuale dei movimenti*. Sono, perciò, gli unici due KPI che hanno il campo percentuale, come si nota dalla Figura 4.7.

4.2.3 Loader SAS

La comunicazione tra *SAS* e *Python* avviene grazie ad un estrattore specifico "loader *SAS*" che si occupa del caricamento dati sul database Oracle per renderli visibili al microservizio di Python. Le tabelle in uscita sono quelle riportate in Figura 4.8.

4.3 Organizzazione codice SAS

Il codice in *SAS* è organizzato come segue:

	NUM_ANNO_MESE	COD_ABI	COD_NDG	COD_SUPERNSG	num_mov	desc_kpi
1	202007	01025	3920442431000	0000000000005376	2	NUM_ENTI_SEG
2	202008	01025	3920442431000	0000000000005376	2	NUM_ENTI_SEG
3	202009	01025	3920442431000	0000000000005376	2	NUM_ENTI_SEG
4	202010	01025	3920442431000	0000000000005376	2	NUM_ENTI_SEG
5	202011	01025	3920442431000	0000000000005376	2	NUM_ENTI_SEG
6	202012	01025	3920442431000	0000000000005376	2	NUM_ENTI_SEG
7	202101	01025	3920442431000	0000000000005376	2	NUM_ENTI_SEG
8	202102	01025	3920442431000	0000000000005376	2	NUM_ENTI_SEG
9	202103	01025	3920442431000	0000000000005376	2	NUM_ENTI_SEG
10	202104	01025	3920442431000	0000000000005376	2	NUM_ENTI_SEG
11	202105	01025	3920442431000	0000000000005376	2	NUM_ENTI_SEG
12	202106	01025	3920442431000	0000000000005376	2	NUM_ENTI_SEG
13	202004	01025	5706188684000	0000000000005698	2	NUM_ENTI_SEG
14	202005	01025	5706188684000	0000000000005698	2	NUM_ENTI_SEG
15	202006	01025	5706188684000	0000000000005698	2	NUM_ENTI_SEG
16	202007	01025	5706188684000	0000000000005698	2	NUM_ENTI_SEG
17	202008	01025	5706188684000	0000000000005698	2	NUM_ENTI_SEG
18	202009	01025	5706188684000	0000000000005698	2	NUM_ENTI_SEG
19	202010	01025	5706188684000	0000000000005698	2	NUM_ENTI_SEG
20	202106	01025	5706188684000	0000000000005698	1	NUM_ENTI_SEG
21	202101	01025	3920783951000	0000000000005808	1	NUM_ENTI_SEG

Figura 4.6. Tabella "`ts_cope_inp_enti_seg.SAS.7bdat`" in output dal *SAS* contenente il numero di enti affidatanti (legame tra ditta individuale e il titolare), al fine di trovare tutte le ditte individuali che sono state affidate nel periodo di interesse.

	NUM_ANNO_MESE	COD_ABI	COD_NDG	cod_supernsg	prc_mov	desc_kpi
1	202005	01025	5702272312000	000000000000...	0.36700	UT_ACC
2	202006	01025	5702272312000	000000000000...	0.77967	UT_ACC
3	202007	01025	5702272312000	000000000000...	0.72500	UT_ACC
4	202008	01025	5702272312000	000000000000...	0.62267	UT_ACC
5	202009	01025	5702272312000	000000000000...	0.53933	UT_ACC
6	202010	01025	5702272312000	000000000000...	0.00000	UT_ACC
7	202011	01025	5702272312000	000000000000...	0.48467	UT_ACC
8	202012	01025	5702272312000	000000000000...	0.83100	UT_ACC
9	202101	01025	5702272312000	000000000000...	0.91500	UT_ACC
10	202102	01025	5702272312000	000000000000...	0.83533	UT_ACC
11	202103	01025	5702272312000	000000000000...	0.00000	UT_ACC
12	202104	01025	5702272312000	000000000000...	0.00000	UT_ACC
13	202105	01025	5702272312000	000000000000...	1.03433	UT_ACC
14	202106	01025	5702272312000	000000000000...	1.05233	UT_ACC
15	202107	01025	5702272312000	000000000000...	0.39367	UT_ACC
16	202005	01025	3920581968000	000000000000...	1.00000	UT_ACC
17	202006	01025	3920581968000	000000000000...	1.00000	UT_ACC

Figura 4.7. Tabella "`ts_cope_inp_ut_acc_it.SAS.7bdat`" in output da *SAS* contenente la percentuale dei movimenti degli accrediti.

- `COPE_00_Genera_CB.egp`: job che genera la Customer Base che, a partire dall'anagrafica di tutti i clienti della banca, applica dei filtri per delineare il perimetro di utenza interessato. Tale programma è necessario all'esecuzione dei job sottostanti, poichè questi utilizzano tale perimetro di utenza. In Figura 4.9, è inoltre, rappresentato il diagramma che coinvolge le tabelle in input e output al programma *SAS* in esame.
- `COPE_01_Bonifici_Entrata_Uscita.egp`: somma dei movimenti relativi a bonifici italiani in entrata e in uscita nel periodo di analisi di 15 mesi. Tale job

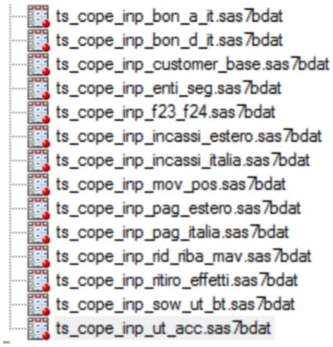

Figura 4.8. Tabelle KPI in output dal loader *SAS*.

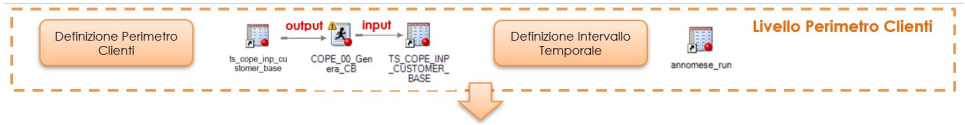

Figura 4.9. Diagramma delle relazioni tra il programma `.egp`, scritto in linguaggio *SAS*, che crea la Customer Base, e le rispettive tabelle di input e output.

genera le seguenti tabelle di KPI:

- `TS_COPE_INP_BON_A_IT`
- `TS_COPE_INP_BON_D_IT`

Ad ogni run viene effettuato il ricalcolo di 15 mesi. In Figura 4.10 è rappresentato il diagramma che coinvolge le tabelle in input e output al programma *SAS*.

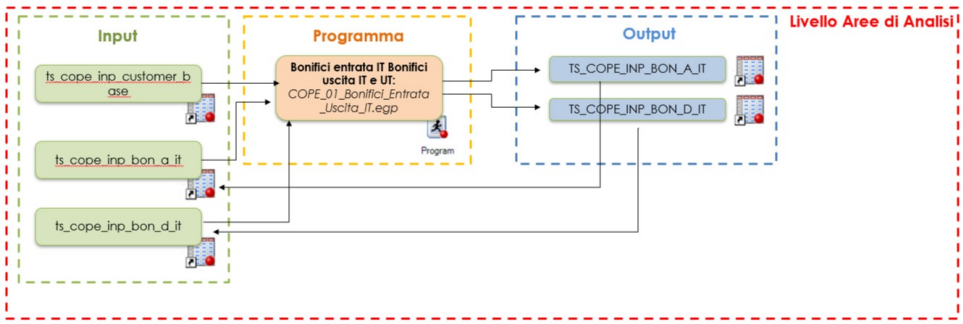

Figura 4.10. Diagramma delle relazioni tra il programma `.egp` scritto in linguaggio *SAS* e le rispettive tabelle di input e output

- `COPE_01_F23F24_incassiIT_pagamentiIT_riteffetti.egp`: questo job genera le seguenti tabelle di KPI:
  - `TS_COPE_INP_F23_F24`
  - `TS_COPE_INP_INCASSI_ITALIA`
  - `TS_COPE_INP_PAG_ITALIA`

– `TS_COPE_INP_RITIRO_EFFETTI`

Ad ogni run viene effettuato il ricalcolo di 13 mesi, mentre i mesi 14 e 15 sono recuperati dagli output generati nel run precedente. In Figura 4.11 è rappresentato il diagramma che coinvolge le tabelle in input e output al programma *SAS* in esame.

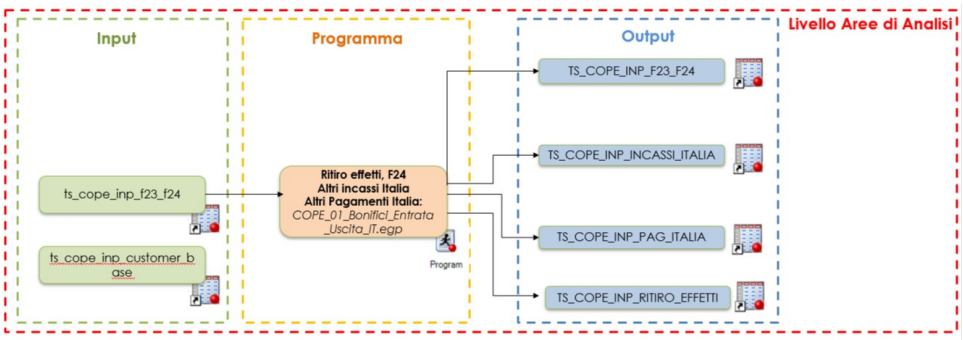

Figura 4.11. Diagramma delle relazioni tra il programma `.egp` scritto in linguaggio *SAS* e le rispettive tabelle di input e output

- `COPE_01_Num_enti_segnalanti.egp`: job che genera la tabella `TS COPE INP ENTI SEG`. Ad ogni run viene effettuato il ricalcolo di 15 mesi. In Figura 4.12 è rappresentato il diagramma che coinvolge le tabelle in input e output al programma *SAS* in esame.

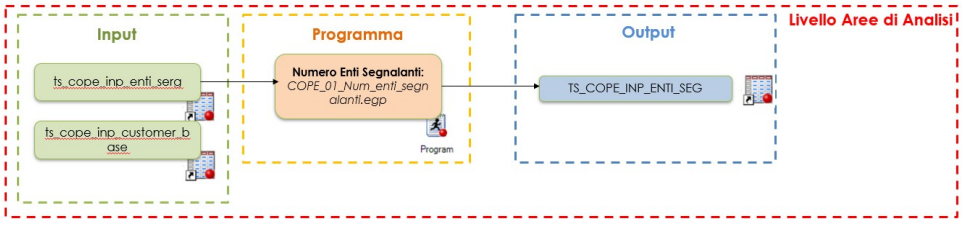

Figura 4.12. Diagramma delle relazioni tra il programma `.egp` scritto in linguaggio *SAS* e le rispettive tabelle di input e output

- `COPE_01_Pagamenti_POS.egp`: job che genera la tabella `TS_COPE_INP_MOV_POS`. Ad ogni run viene effettuato il ricalcolo di 15 mesi. In Figura 4.13 è, inoltre, rappresentato il diagramma che coinvolge le tabelle in input e output al programma *SAS* in esame.
- `COPE_01_RBA_RID_MAV.egp`: job che genera la tabella `TS_COPE_INP_RID_RIBA_MAV`. Ad ogni run viene effettuato il ricalcolo di 15 mesi. In Figura 4.14 è, inoltre, rappresentato il diagramma che coinvolge le tabelle in input e output al programma *SAS* in esame.
- `COPE_01_SOW_CR_BT.egp`: job che genera la tabella `TS_COPE_INP_SOW_UT_BT`. Ad ogni run viene effettuato il ricalcolo di 15 mesi. In Figura 4.15 è, inoltre, rappre-

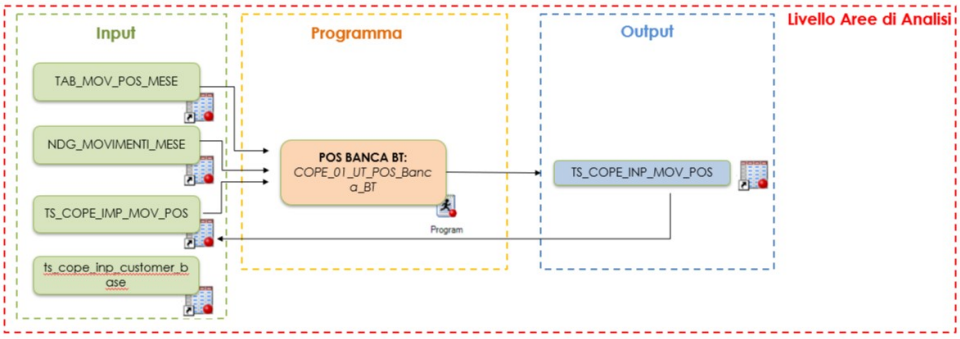

Figura 4.13. Diagramma delle relazioni tra il programma `.egp` scritto in linguaggio *SAS* e le rispettive tabelle di input e output

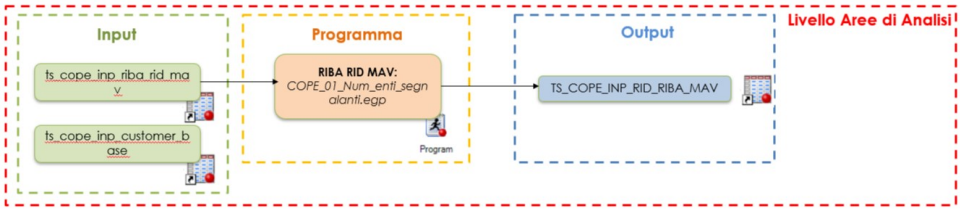

Figura 4.14. Diagramma delle relazioni tra il programma `.egp` scritto in linguaggio *SAS* e le rispettive tabelle di input e output

sentato il diagramma che coinvolge le tabelle in input e output al programma *SAS* in esame.

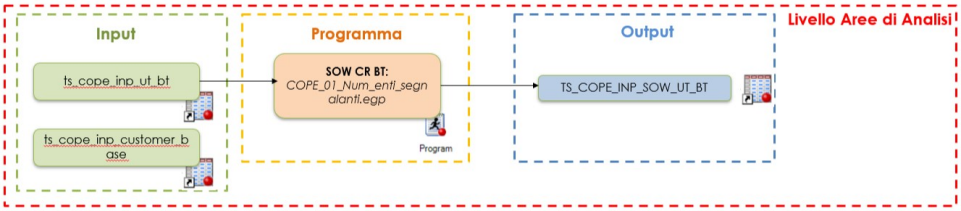

Figura 4.15. Diagramma delle relazioni tra il programma `.egp` scritto in linguaggio *SAS* e le rispettive tabelle di input e output

- `COPE_01_UT_ACC_Banca_BT.egp`: job che genera la tabella `TS_COPE_INP_UT_ACC`. Ad ogni run viene effettuato il ricalcolo di 15 mesi. In Figura 4.16 è, inoltre, rappresentato il diagramma che coinvolge le tabelle in input e output al programma *SAS* in esame.
- `COPE_02_Load_CB_KPI.egp`: job che effettua il caricamento della Customer Base e delle tabelle dei KPI sul DBMS Oracle. Inoltre, in Figura 4.17, è rappresentato il diagramma che coinvolge le tabelle in input e output al programma *SAS* in esame.

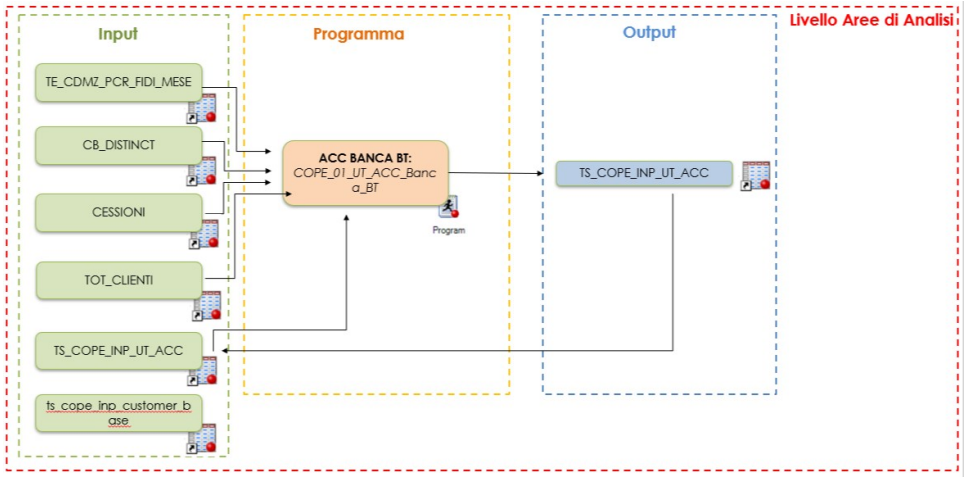

Figura 4.16. Diagramma delle relazioni tra i vari programmi `.egp` scritti in linguaggio *SAS* e le rispettive tabelle di input e output

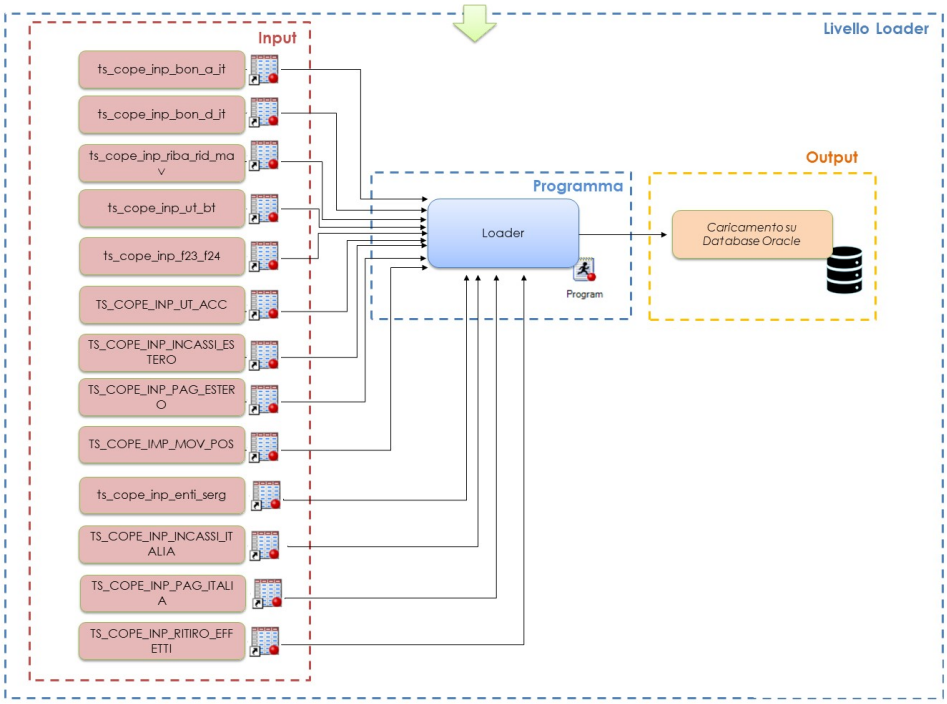

Figura 4.17. Il livello Loader ha un unico programma che effettua il caricamento delle 13 tabelle dei KPI e della Customer Base

4.4 Modulo modello pyhton

Il modello "calo di operatività" è sviluppato sul server dipartimentale *PPYT0*, partendo dalle strutture tabellari create da *SAS*. Di seguito i dettagli degli step effettuati per lo sviluppo del modello, ovvero:

1. il calcolo dello *score* per ogni KPI; questo è un indicatore che riflette i fattori critici di successo per il calcolo dell'operatività dei clienti;
2. il calcolo dello *score* di *penalizzazione* per ogni KPI (fatta eccezione per il `Numero Enti Segnalanti`);
3. il calcolo dello *score totale*;
4. la segmentazione della Customer Base sulla base del *minter annuale*;
5. l'assegnazione della prioritizzazione;
6. la creazione dei report "completo" e "ridotto".

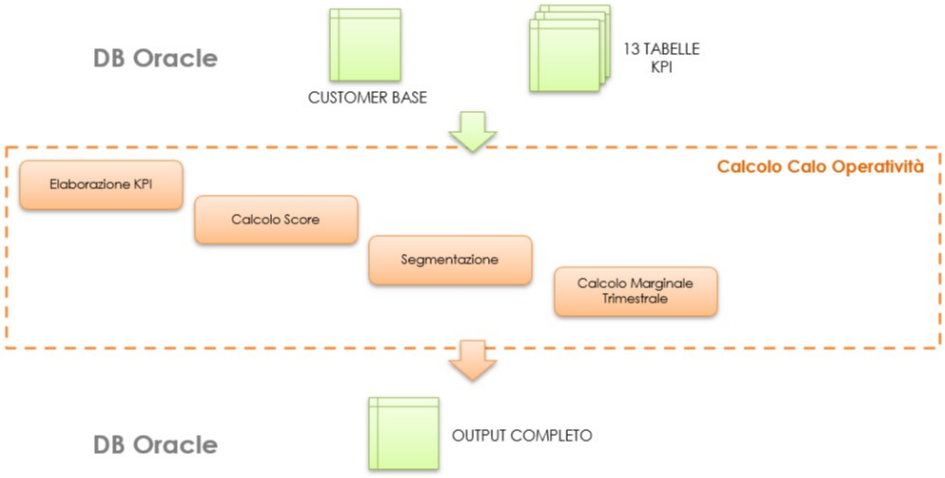

Figura 4.18. Schema del modello Python del "calo di operatività"

4.4.1 Fasi di elaborazione modello Pyhton

Riprendendo più in dettaglio quanto detto nella Sezione 4.4, le fasi principali per lo sviluppo del modello hanno le seguenti caratteristiche:

1. La prima fase consiste nella preparazione delle tabelle di output attraverso due azioni:
   (i) *Truncate* della partizione corrispondente al mese corrente (serve in caso di lanci ripetuti del codice).
   (ii) Eliminazione del *Report Ridotto* nel quale sono presenti dati solo per il mese corrente.

2. Viene letto un chunk di 70.000 record (ogni record corrisponde ad un `COD_NDG`) della Customer Base e vengono calcolati, per ogni record, dei valori che si ripetono per tutti i 13 KPI. Per ciascuno di questi, la profondità storica inserita nella tabella di input del modello è di 15 mesi. Una volta finita l'elaborazione, il chunk con le colonne aggiuntive elaborate viene scritto sulla tabella di output *Report Completo* e *Report Ridotto*.
3. Questa fase corrisponde alla *Segmentazione*. Per prima cosa viene letto il *Report Completo* totale e viene calcolato il `MOD_TOT` sulla base dei campi valorizzati in precedenza. Viene, quindi, effettutato il `RANK_MINTER_ANTE_RETTIFICHE` sulla base dei diversi gruppi di `COD_TIPO_PORTAFOGLIO`.
4. Una volta ottenuto il `MOD_TOT` e il `RANK_MINTER_ANTE_RETTIFICHE` si procede alla prioritizzazione che andrà a valorizzare il campo `DES_TARGET` sulla base dei valori dei due campi precedenti. Vengono, poi, effettuate diverse forzature in base al `RANK` dei margini trimestrali, al `FLAG_STARTUP` e al `COD_SPECIE_GIURIDICA`.
5. Infine si procede all'`UPDATE` del *Report Completo* e del *Report Ridotto*.

4.4.2 Calcolo dello score per ogni KPI

Per ognuno dei 13 KPI, a partire dalle tabelle calcolate secondo le logiche precedentemente descritte, e per ogni `COD_SUPERNSG`, vengono calcolate le seguenti metriche:

- `IMP_P1_KPI` = somma dell'importo (€) del KPI nell'ultimo trimestre disponibile nella tabella calcolata nello step *SAS* di data preparation.
- `IMP_P2_KPI` = somma dell'importo (€) del KPI dello stesso trimestre considerato nel punto precedente ma dell'anno precedente.
- `P1_P2_IMP_KPI` =
  - 9999999: Se `IMP_P1_KPI` > 0 e `IMP_P2_KPI IS MISSING`
  - 0: Se `IMP_P1_KPI` > 0 e `IMP_P2_KPI IS MISSING`
  - `IMP_P1_KPI / IMP_P2_KPI` altrimenti.
- `MOD_IMP_KPI` =
  - -1: Se `P1_P2_IMP_KPI` < 0.99
  - 1: Se `P1_P2_IMP_KPI` > 1.01
  - 0: altrimenti.

  Questo campo viene "valorizzato" in maniera opposta nel caso del KPI relativo a *Numero Enti Segnalanti*. In questo caso infatti:
  - 1: Se `P1_P2_IMP_KPI` < 0.99
  - -1: Se `P1_P2_IMP_KPI` > 1.01
  - 0: altrimenti.
- `IMP_D_KPI` è calcolato secondo la seguente logica:
  - Se `IMP_P1_KPI` e `IMP_P2_KPI` sono entrambi diversi da "missing", allora

$$\texttt{IMP\_D\_KPI} = (\texttt{IMP\_P1\_KPI} - \texttt{IMP\_P2\_KPI}) \tag{4.1}$$

  - Se `IMP_P1_KPI` è "missing" e `IMP_P2_KPI` è diverso da "missing" allora:

$$\texttt{IMP\_D\_KPI} = -\texttt{IMP\_P2\_KPI} \tag{4.2}$$

  - Se IMP_P1_KPI è diverso da "missing" e IMP_P2_KPI è "missing" allora

$$\texttt{IMP\_D\_KPI} = \texttt{IMP\_P1\_KPI} \tag{4.3}$$

- NUM_D_KPI è calcolato secondo la seguente logica:
  - Se IMP_P2_KPI è uguale a "missing" o uguale a 0 , allora NUM_D_P_KPI = . (ovvero sarà "valorizzato" come "missing")
  - Altrimenti, se IMP_P1_KPI e IMP_P2_KPI sono entrambi diversi da "missing" e IMP_P2_KPI è diverso da 0 allora:

$$\texttt{NUM\_D\_P\_KPI} = (\texttt{IMP\_P1\_KPI} - \texttt{IMP\_P2\_KPI})/\texttt{IMP\_P2\_KPI} \tag{4.4}$$

  - Altrimenti, se IMP_P1_KPI è "missing" e IMP_P2_KPI è diverso da "missing" e diverso da 0 allora: NUM_D_P_KPI = -1

$$\texttt{NUM\_D\_P\_KPI} = -1 \tag{4.5}$$

Queste metriche vengono calcolate per tutti i 13 KPI presentati nella Sezione 4.4.1.

4.4.3 Calcolo dello score di penalizzazione per ogni KPI

Per tutti i 12 KPI, ad eccezione del KPI relativo al numero di enti segnalanti, a partire dalle tabelle calcolate secondo le logiche precedentemente descritte nella Sezione 4.4.2, vengono calcolate le seguenti elaborazioni, al fine di assegnare una penalizzazione aggiuntiva nel caso di operatività continuativa nei mesi precedenti e bloccata nell'ultimo trimestre.

Per ogni COD_SUPERNSG vengono calcolate le seguenti metriche:

- M1_KPI:
  - Per tutti i KPI diversi da "SOW CR BT" e "UT/ACC Banca BT" l'indicatore viene definito come:
    - 1 se l'importo delle operazioni è maggiore di 0 per quel KPI nel mese $t-0$ (ultimo mese disponibile nella tabella costruita nello step di data preparation);
    - 0, altrimenti.
  - Per i KPI "SOW CR BT" e "UT/ACC Banca BT" l'indicatore viene definito come:
    - 1 se il valore del rapporto nel mese $t-0$ è diverso da "missing";
    - 0, altrimenti (se il valore del rapporto nel mese t-0 è "missing").
- M2_KPI:
  - Per tutti i KPI diversi da "SOW CR BT" e "UT/ACC Banca BT" l'indicatore viene definito come:
    - 1 se l'importo delle operazioni è maggiore di 0 per quel KPI nel mese $t-1$ (penultimo mese disponibile nella tabella costruita nello step di Data Prep);
    - 0, altrimenti.
  - Per i KPI "SOW CR BT" e "UT/ACC Banca BT" l'indicatore viene definito come:
    - 1 se il valore del rapporto nel mese $t-1$ è diverso da "missing";

        - 0, altrimenti (se il valore del rapporto nel mese $t-1$ è "missing").
- `M3_KPI`:
  - Per tutti i KPI diversi da "`SOW CR BT`" e "`UT/ACC Banca BT`" l'indicatore viene definito come:
    - 1 se l'importo delle operazioni è maggiore di 0 per quel KPI nel mese $t-2$ (penultimo mese disponibile nella tabella costruita nello step di data preparation);
    - 0, altrimenti.
  - Per i KPI "`SOW CR BT`" e "`UT/ACC Banca BT`" l'indicatore viene definito come:
    - 1 se il valore del rapporto nel mese $t-2$ è diverso da "missing";
    - 0, altrimenti (se il valore del rapporto nel mese $t-2$ è "missing").
- `M_COUNT_KPI`:
  - Per tutti i KPI diversi da "`SOW CR BT`" e "`UT/ACC Banca BT`" l'indicatore viene definito come:
    - numero di mesi (tra $t-14$ e $t-0$) nei quali l'importo delle operazioni è maggiore di 0 per quel determinato KPI.
  - Per i KPI "`SOW CR BT`" e "`UT/ACC Banca BT`" l'indicatore viene definito come:
    - numero di mesi (tra $t-14$ e $t-0$) nei quali il rapporto è diverso da "missing".
- `NUM_PEN_KPI`:
  - -1 se:
    - `M1_KPI` $= 0$ e `M_COUNT_KPI` $= 14$ oppure
    - `M1_KPI` $= 0$ e `M2_KPI` $= 0$ e `M_COUNT_KPI` $= 13$ oppure
    - `M1_KPI` $= 0$ e `M2_KPI` $= 0$ e `M3_KPI` $= 0$ e `M_COUNT_KPI` $= 12$
  - 0 altrimenti

4.4.4 Calcolo dello score finale

Il calcolo dello score finale considera la somma di tutti i `NUM_KPI` e i `NUM_PEN_KPI` calcolati per ciascuno dei 13 KPI.

Di seguito la formula per il calcolo dello score finale:

`MOD_TOT = NUM INCASSI ESTERO + NUM PEN INCASSI ESTERO + NUM PAGAMENTI ESTERO + NUM PEN PAGAMENTI ESTERO + NUM POS + NUM PEN POS + NUM RID RIBA MAV + NUM PEN RID RIBA MAV + NUM SOW UT BT + NUM PEN SOW UT BT + NUM BON A IT + NUM PEN BON A IT + NUM BON D IT + NUM PEN BON D IT + NUM PAGAMENTI ITALIA + NUM PEN PAGAMENTI ITALIA + NUM INCASSI ITALIA + NUM PEN INCASSI ITALIA + NUM F23 F24 + NUM PEN F23 F24 + NUM RITIRO EFFETTI + NUM PEN RITIRO EFFETTI + NUM UT ACC + NUM PEN UT ACC + NUM ENTI SEG`

Nel *Report Completo* finale vengono riportati, anche, le seguenti informazioni:

- `NUM_PUNTI_POSITIVI` = somma dei valori positivi di `NUM_KPI` (numero intero tra 0 e 13);
- `NUM_PUNTI_NEGATIVI` = somma dei valori negativi di `NUM_KPI` (numero intero tra 0 e 13);

- NUM_PENALIZZAZIONI = somma dei valori negativi di NUM_PEN_KPI (numero intero tra 0 e 12, in quanto per il Numero Enti Segnalanti non viene calcolata la penalizzazione);
- NUM_PUNTI_MODELLO = NUM_PUNTI_POSITIVI - NUM_PUNTI_NEGATIVI - NUM_PENALIZZAZIONI

Segmentazione della Customer Base sulla base del minter annuale

La segmentazione, attraverso il calcolo dei percentili viene effettuata sulla base del valore di IMP_MINTER_ANTE_RETTIFICHE disponibile nella Customer Base. La lista di clienti in perimetro viene divisa in due parti a seconda del COD_TIPO_PORTAFOGLIO:

- COD_TIPO_PORTAFOGLIO in ("O", "4");
- COD_TIPO_PORTAFOGLIO in ("U", "G", "I");
- COD_TIPO_PORTAFOGLIO in ("9", "D");
- COD_TIPO_PORTAFOGLIO in ("5", "E").

Le quattro sotto-popolazioni vengono, poi, divise in percentili a seconda dell'importo MINTER_ANTE_RETTIFICHE. Ad ogni cliente viene, quindi, associato un valore di RANK_MINTER_ANTE_RETTIFICHE (valore tra 0 e 100).

Assegnazione prioritizzazione

Per ogni cliente nella Customer Base viene assegnata una prioritizzazione (DES_TARGET) che può assumere i seguenti valori:

- 1_ALTA_P
- 2_MEDI_P
- 3_BASSA_P

La prioritizzazione viene inoltre definita sulla base di due informazioni:

- MOD_TOT
- RANK_MINTER_ANTE_RETTIFICHE

A seconda della valorizzazione di queste due metriche è assegnata la priorità riportata nella Figura 4.19.

		RANK_MINTER_ANTE_RETTIFICHE		
		> 85	(60; 85]	<= 60
MOD_TOT	<= -4	1_ALTA_P	1_ALTA_P	2_MEDIA_P
	(-4; 0)	2_MEDIA_P	3_BASSA_P	3_BASSA_P
	>= 0	3_BASSA_P	3_BASSA_P	3_BASSA_P

Figura 4.19. Matrice di valorizzazione delle priorità

Indipendentemente dalla Tabella 4.19 , definita secondo queste logiche, per alcuni clienti in cui `COD_SPECIE_GIURIDICA = \COND"`, la prioritizzazione verrà forzata con la voce "`3_BASSA_P`".

Inoltre, viene effettuata una forzatura a priorità "`3_BASSA_P`" per i clienti che rispettano le seguenti logiche:

- `RANK_MINTER_ANTE_RETTIFICHE` $<= 60$;
- `NUM_PUNTI_MODELLO` $<= -4$;
- `RANK_MARGIN_TRIM1` $<= 30$;
- `RANK_MARGIN_TRIM2` $<= 30$;
- `FLAG_STARTUP` diverso da 1 (ovvero, valorizzato a 0 o "missing").

dove:

- `RANK_MARGIN_TRIM1` indica il percentile definito sulla base del valore del *minter* dell'ultimo trimestre (campo `MARG_TRIM_P1` della Customer Base);
- `RANK_MARGIN_TRIM2` indica il percentile definito sulla base del valore del *minter* del trimestre dell'anno precedente (campo `MARG_TRIM_P2` della Customer Base).

Nei casi particolari in cui entrambi i margini trimestrali risultano "missing", o un margine trimestrale risulta "missing" e l'altro "valorizzato", viene effettuata la forzatura a priorità "`3_BASSA_P`".

Entrambi i percentili sono calcolati esclusivamente sul gruppo di clienti che rientrano nel quadrante in alto a destra: (`NUM_PUNTI_MODELLO` $<= -4$ e `RANK_MINTER_ANTE_RETTIFICHE` $<= 60$).

4.4.5 Creazione del Report Ridotto e Completo

L'output della parte Python consiste nella pubblicazione di due tabelle su Oracle:

- `TS_COPE_OUT_REPORT_COMPLETO`
- `TS_COPE_OUT_REPORT_RIDOTTO`

Per quanto riguarda il *Report Completo*, la tabella ha una profondità storica di 15 mesi e contiene tutte le informazioni relative ai clienti presenti nella Customer Base. Di seguito, nella Figura 4.20, viene mostrata la struttura della tabella della versione estesa del report, dove i clienti con alta priorità sono mostrati in alto, per proseguire in ordine decrescente. Le prime due colonne della Figura 4.20 sono mascherate, poiché contengono dati sensibili dei clienti.

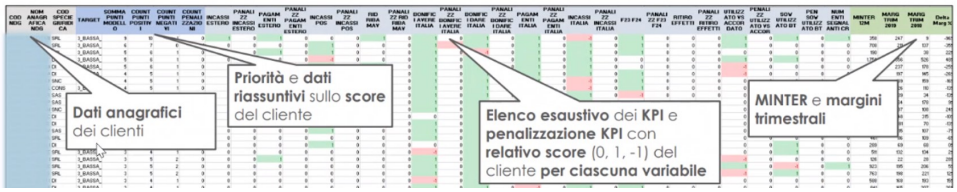

Figura 4.20. Tabella contenente la versione estesa del report in output a *Pyhton*.

Per quanto riguarda il *Report Ridotto*, esso contiene solo le informazioni dell'ultimo run effettuato ed esclusivamente relative ai clienti presenti nella Customer Base con `FLG_PERIMETRO = 1`. Di seguito, nella Figura 4.21, vengono riportate le colonne di interesse del *Report Ridotto*. Le prime due colonne della Figura 4.21 sono mascherate, poiché contengono dati sensibili dei clienti.

Figura 4.21. Tabella contenente la versione ridotta del report in output al *Pyhton*.

Di seguito, nel Listato 4.3, vengono riportati i nomi dei campi presenti all'interno del `TS_COPE_OUT_REPORT_RIDOTTO`:

Nomi Campo
NUM_ANNO_MESE
COD_NDG
COD_SUPERNSG
COD_ABI
COD_MATRICOLA_GESTORE
NOM_NOMINATIVO_GESTORE
NOM_ANAGRAFICA_NDG
COD_SPECIE_GIURIDICA
FLG_NOTIZIE_PREGIUDIZIEVOLI
FLG_PRODOTTI_MLT
FLG_PRODOTTI_TUTELA
FLG_STARTUP
DES_TARGET
DES_MOTIVO_SEGNALAZIONE
IMP_MINTER_ANTE_RETTIFICHE
IMP_MARG_TRIM_P1
IMP_MARG_TRIM_P2
PRC_DELTA_MARGIN

Tabella 4.3. Elenco colonne del *Report Ridotto*

4.5 Organizzazione del codice Python

Il codice è organizzato come segue:

- `app.py`: contiene l'applicazione Flask. Questo è un micro-framework Web scritto in Python, chiamato in questo modo perché ha un nucleo semplice, ma estendibile. La `App.py` mette a disposizione funzionalità fondamentali, quali server di sviluppo, debugger, routing e supporto unit test integrato, mostrati nella Sezione 4.5.2.
- `job_manager.py`: come mostrato nella Figura 4.22, contiene la classe Manager che ha metodi relativi allo stato dell'applicazione, gestisce i processi con i quali vengono eseguite le elaborazioni e la scrittura/update attraverso dei processi in parallelo (*pool*). In particolare:
  - `Start`: attraverso questo metodo viene lanciata l'esecuzione dell'applicazione.
  - `Run`: in questo metodo vengono eseguiti tutti i processi di elaborazione sulla Customer Base, volti alla scrittura del *Report Completo* e *Report Ridotto*.
  - `Altri metodi`: relativi allo stato dell'applicazione (ad esempio `get_status`, `stop`).
- `modeling_operations.py`: contiene la classe *Modeling Operations* che possiede tutti i metodi relativi alla elaborazione del dato. In particolare:
  - `kpi_elaboration`: attraverso questo metodo, per ogni chunk della Customer Base e per ogni KPI, vengono calcolati dei parametri relativi agli importi e al numero di operazioni nel trimestre attuale e nel trimestre dell'anno precedente. Il diagramma di flusso di questa fase è mostrato nella Figura 4.23.
  - `segmentation`: attraverso questo metodo, che segue la scrittura parziale dei report, viene calcolato il `MOD_TOT` attraverso i campi valorizzati dalla *kpi_elaboration* e viene effettuato il ranking sulla base del codice di tipo portafoglio. Il diagramma di flusso di questa fase è mostrato nella Figura 4.24.
  - `prioritization`: serve per calcolare, utilizzando il `MOD_TOT` e il ranking effettuato dalla segmentazione, il `DES_TARGET` e cioè la priorità su ogni `COD_NDG`. Il diagramma di flusso di questa fase è mostrato nella Figura 4.25.
- `provider.py`: contiene la classe *Provider*, nella quale sono contenuti tutti i metodi relativi alla gestione delle tabelle di output (ad esempio `truncate`, `update` e `scrittura`).
- `isp-config-env`: contiene i file `.yml` con le varie configurazioni per i diversi ambienti di rilascio.

4.5.1 Stati del microservizio

Di seguito vengono riassunti in quali stati si può trovare il microservizio (Sezione 2.3.2) dell'applicativo e il loro significato. Questi sono:

- `idle`: è lo stato in cui si trova l'applicativo appena rilasciato e in attesa di essere lanciato. Non è possibile tornare in questo stato da nessuno degli altri.
- `starting`: è il primo stato in cui ci si trova una volta lanciato "`/start`" per avviare l'elaborazione. Durante questa fase vengono svolti i controlli necessari per verificare che sia effettivamente possibile eseguire l'elaborazione del microservizio. Se i controlli vengono superati si passa allo stato `kpi_elaboration`, altrimenti l'applicativo viene fermato e si passa allo stato `failed`.

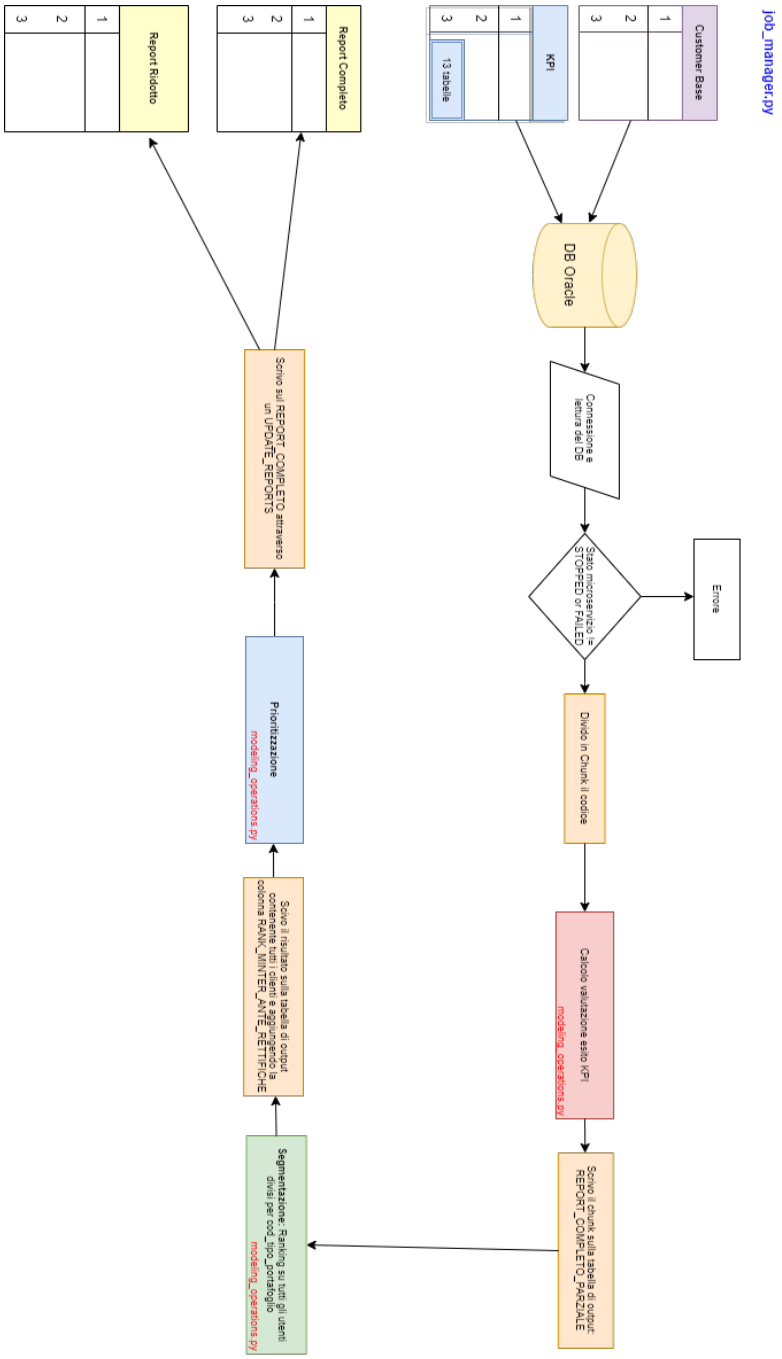

Figura 4.22. Diagramma di flusso del programma `job_manager.py`.

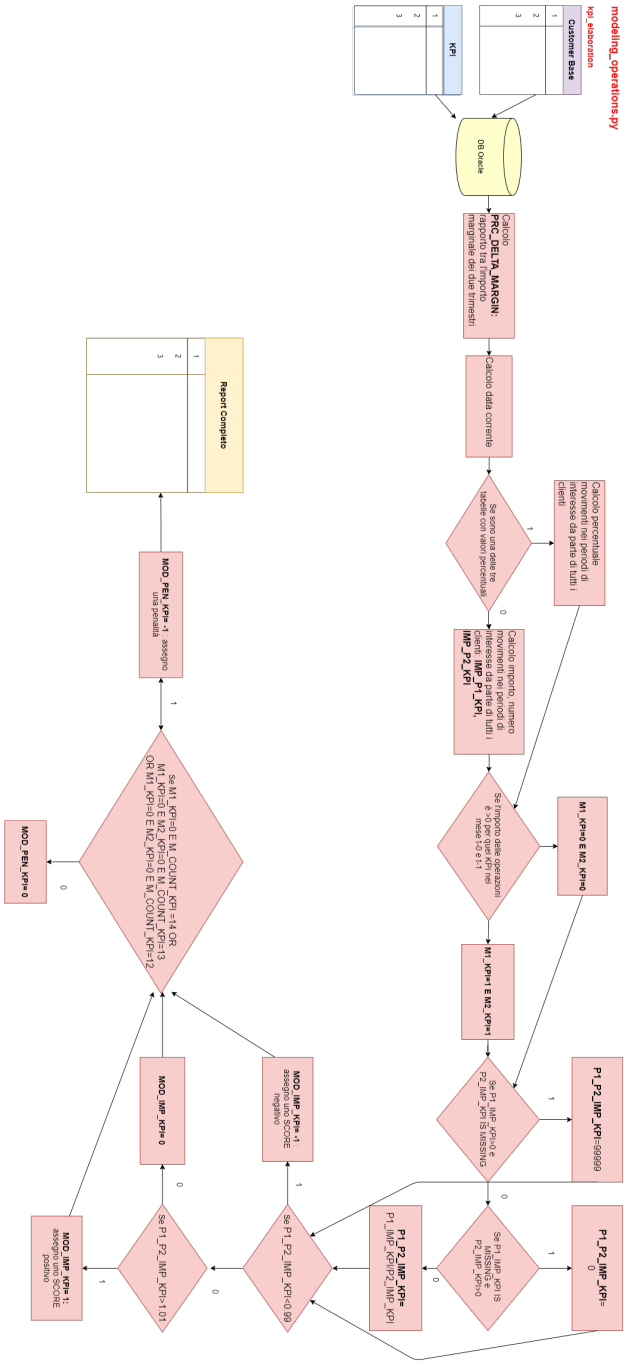

Figura 4.23. Diagramma di flusso del modulo `KPI_elaboration`, all'interno del programma `modeling_operations.py`.

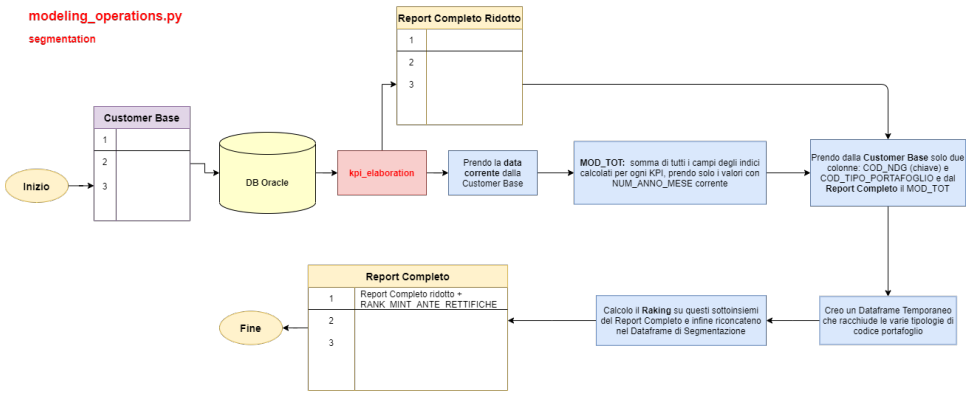

Figura 4.24. Diagramma di flusso del modulo `segmentation`, all'interno del programma `modeling_operations.py`.

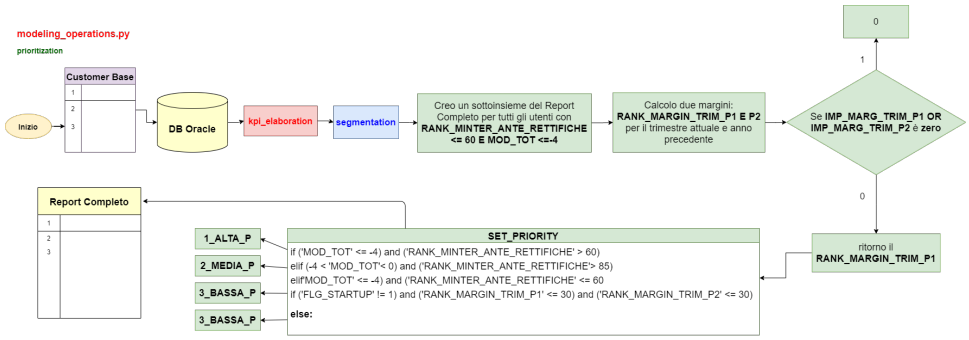

Figura 4.25. Diagramma di flusso del modulo `prioritizzazione`, all'interno del programma `modeling_operations.py`.

- `stopped`: è lo stato in cui si trova l'applicativo una volta stoppata l'esecuzione.
- `segmentation`: è lo stato in cui si trova l'applicativo durante la segmentazione della Customer Base.
- `kpi_elaboration`: è lo stato in cui si trova l'applicativo durante l'elaborazione dei KPI.
- `writing_output`: è lo stato in cui si trova l'applicativo durante la scrittura dei report.
- `prioritization`: è lo stato in cui si trova l'applicativo durante la fase di prioritizzazione.
- `done`: è lo stato in cui ci si trova l'applicativo una volta terminata con successo l'elaborazione; fino quando ci si trova in questo stato non è possibile lanciare di nuovo il comando "`/start`", tale stato viene aggiornato in *expired* il giorno seguente.
- `expired`: è lo stato in cui si passa il giorno seguente alla corretta terminazione della fase di elaborazione, in questo stato è possibile rilanciare il comando "`/start`" per eseguire nuovamente il microservizio.

- `failed`: è lo stato in cui ci si trova quando l'applicativo si ferma a causa di un errore o nel caso in cui non vengano superati i controlli durante l'elaborazione.

4.5.2 Link disponibili `app.py`

I seguenti link di routing servono a gestire e monitorare l'elaborazione del microservizio. Questi sono:

- */*: restituisce lo stato del server;
- */configuration*: restituisce la configurazione corrente dell'applicatione;
- */oratest*: restituisce l'esito del test di connessione al database Oracle;
- */health*: restituisce l'ambiente dove è in esecuzione l'applicativo;
- */status*: restituisce lo stato in cui si trova l'applicativo;
- */jstatus*: stato dell'applicativo con alcune informazioni a contorno (orario di inizio e terminazione, avanzamento dell'elelaborazione);
- */start*: esecuzione del microservizio;
- */stop*: interrruzione dell'esecuzione dell'applicativo.

5

Implementazione

Nel seguente capitolo, verrà approfondita l'implementazione del codice che sta dietro alla progettazione, trattata nel Capitolo 4. In particolare, si scenderà più nel dettaglio sull'organizzazione del codice nei singoli programmi.

5.1 Implementazione SAS

Riprendendo quanto discusso nel Capitolo 4, in questo capitolo si è deciso di ripercorre le stesse tre fasi principali, con un approccio basato sull'implementazione a livello di codice, rispetto che di progettazione.

5.1.1 Livello Perimetro Clienti

Ripercorrendo la Sezione 4.2.1, nel seguito verrà spiegata l'implementazione alla base della *data preparation* per studiare il perimetro dei clienti.

Customer Base

Il calcolo della Customer Base viene effettuato a partire dalla tabella di output `TS_COPE_INP_CUSTOMER_BASE`. Con la "change request" per la modifica dei clienti UBI, la fonte alimentante l'anagrafica è stata cambiata e il nuovo input è costituito dalla tabella `RTD_L2ID.t2_imp_anagrafica`, poiché `T1_anag_clienti` è stata dismessa. Poichè la `T1_anag_clienti` era aggiornata mensilmente, mentre `T2_imp_anagrafica` è aggiornata giornalmente, per mantenere la stessa logica di calcolo dell'orizzonte temporale, `num_anno_mese` della nuova Customer Base è stato calcolato a partire dal `num_anno_mese` della $CustomerBase - 1$, guardando, quindi, al mese precedente.

Con la change request di Aprile 2021, è stato allargato nuovamente il perimetro portafogli per ospitare la nuova clientela relativa alla direzione "Agricoltura e Impact". Sono stati, quindi, inseriti nel filtro della tabella "anagrafica" anche i seguenti codici portafoglio:

- Agribusiness: 9, D

- Impact: 5, E

L'insieme di tali filtri sono mostrati nel Listato 5.1.

```
1   WHERE t1.COD_TIPO_PORTAFOGLIO IN ('O', 'I', 'G', 'U', '4','9','D','5','E');
```

Listato 5.1. Calcolo dei codici di tipo portafoglio interessati all'analisi.

Dalla lista di clienti selezionati, rientrano nello scope del modello solo quelli con almeno un conto corrente (`CC`) attivo sia all'inizio che alla fine del periodo di analisi. Il periodo di analisi considera gli ultimi 15 mesi a partire dal `NUM_ANNO_MESE` presente nella tabella `T1_ANAG_CLIENTI`. Partendo da questa informazione, si avrà:

- T0 = NUM_ANNO_MESE (T1_ANAG_CLIENTI)
- T14 = NUM_ANNO_MESE (T1_ANAG_CLIENTI) – 15 mesi

Ad esempio, $T_0 = 202005$ indica il 20-05-2020, allora $T_{14} = 201903$ sarà il 19-03-20. Per verificare la lista di clienti con almeno un `CC` sempre attivo nel periodo di analisi, viene utilizzata la tabella `TS_CDMZ_RAP_CONTRATTO` con i seguenti filtri:

```
1   WHERE t1.COD_AREA_INFORMATIVA = 'CC'
2   AND (t1.DAT_APERTURA_CONTRATTO < primo giorno del mese t-14)
3   AND (
4   (t1.DAT_CHIUSURA_CONTRATTO IS MISSING) OR(t1.DAT_CHIUSURA_CONTRATTO > ultimo giorno del mese t-0))
5         AND t1.COD_ATTRIBUTO LIKE 'CC%'
6             AND t1.COD_ATTRIBUTO NOT IN (lista di valori contenuti nella colonna COD_ATTRIBUTO della tabella di
              configurazione)
```

Listato 5.2. Calcolo del perimetro dei clienti.

A partire dalla lista di clienti estratti secondo queste logiche, vengono definiti alcuni flag necessari per l'analisi dei risultati del modello:

1. flag perimetro;
2. flag notizie pregiudizievoli;
3. flag prodotti mlt;
4. flag prodotti tutela;
5. flag startup.

Flag Perimetro

Lo step iniziale della fase di Data Preparation è finalizzato alla definizione del perimetro di clienti sul quale verranno calcolati i KPI necessari alla definizione del modello. Rientrano nel perimetro le seguenti tipologie di clienti:

- le persona giuridiche;
- i titolari di conto corrente attivo sia all'inizio che alla fine del periodo (cliente in vita da almeno 15 mesi);
- i clienti collegati a portafogli O, I, G, U;
- clienti che non sono società immobiliari o in liquidazione, ovvero `FALLNC e FALLC`;
- clienti con `SAG` assente, ovvero `XX e IB`.

Il Listato 5.3 definisce tale perimetro.

```
1   WHERE
2   t1.DT_DESC NOT IS MISSING
3   AND t1.COD_TIPO_PORTAFOGLIO IN ('O', 'I', 'G', 'U')
4   AND t1.COD_TIPO_NDG = 'G'
5   AND t1.BANCA_COD = '01025'
6   AND t1.COD_CLASSE_RATING_DEFNT_PD NOT = 'D'
7   AND t1.COD_SPECIE_GIURIDICA NOT IN ('FALC', 'FALNC', 'FALLC','FALLNC');
```

Listato 5.3. Calcolo del perimetro dei clienti.

Inoltre, come già accennato in precedenza, dalla lista di clienti selezionati alla fine di questo step verranno considerati solo quelli con conto corrente attivo sia all'inizio che alla fine del periodo in esame, ovvero tutti i clienti tali per cui la tabella `TS_CDMZ_RAP_CONTRATTO` avrà le caratteristiche riportate nel Listato 5.4.

```
1   WHERE t1.COD_AREA_INFORMATIVA = 'CC'
2   AND (t1.DAT_APERTURA_CONTRATTO < primo giorno del mese t-14)
3   AND (
4   (t1.DAT_CHIUSURA_CONTRATTO IS MISSING) OR      (t1.DAT_CHIUSURA_CONTRATTO > ultimo giorno del mese t-0)
5    )
6        AND t1.COD_ATTRIBUTO LIKE 'CC%';
```

Listato 5.4. Calcolo del perimetro dei clienti, diviso per area informativa.

Dalla lista di clienti selezionati alla fine di questo step verranno considerati solo i clienti con `SAG` assente, `XX` e `IB`, secondo le seguenti logiche:

- Dalla tabella `TS_CDMZ_SAG_ANGRF_GRUPPO` verranno filtrate le righe con `DAT_INIZIO_SAG`, definite sulla base della logica di valorizzazione fornita dagli utenti. Per ogni `COD_SUPERNSG_ATT` associamo il valore di `COD_STATO_AMMNS_GRUPPO_ATT`;
- Dalla tabella `TS_CDMZ_SAG_ANGRF_GRUPPO_STOR` verranno filtrate le righe con `DAT_INIZIO_VALIDITA` e `DAT_FINE_VALIDITA`, secondo la logica di valorizzazione fornita dagli utenti. Per ogni `COD_SUPERNSG`, si associa il valore di `COD_STATO_AMMNS_GRUPPO`, definito come `COD_STATO_AMMNS_GRUPPO_STOR`.
  I due output verranno messi in `FULL JOIN` sulla base di `COD_SUPERNSG_ATT` e `COD_SUPERNSG_STOR` come mostrato nel Listato 5.5.

```
1   COD_SUPERNSG = coalesce(COD_SUPERNSG_ATT, COD_SUPERNSG_STOR)
2
3   COD_STATO_AMMNS_GRUPPO = coalesce(COD_STATO_AMMNS_GRUPPO_ATT, COD_STATO_AMMNS_GRUPPO_STOR)
```

Listato 5.5. Calcolo del perimetro dei clienti, diviso in base al codice identificativo.

L'informazione del campo `COD_STATO_AMMNS_GRUPPO` è associata ad ogni `COD SUPERNSG` della lista di clienti estratti con i primi filtri, al fine di filtrare i casi riportati nel Listato 5.6.

```
1   t1.COD_STATO_AMMNS_GRUPPO IS MISSING OR t1.COD_STATO_AMMNS_GRUPPO IN
2            ('XX', 'IB');
```

Listato 5.6. Calcolo del perimetro dei clienti, filtri aggiuntivi.

Flag Notizie Pregiudizievoli

Si tratta di un indicatore booleano che indica la presenza o meno di notizie pregiudizievoli associate al cliente. Per ogni cliente verrà controllata la presenza o

meno di notizie pregiudizievoli nella tabella NWIRIS_GREZZO, considerando come orizzonte temporale gli ultimi 60 mesi rispetto al campo DATA_ATTO o al campo DATA_REVISIONE, e gli ultimi 36 mesi rispetto al campo DATA_CHIUSURA. Di seguito i Listati 5.7 e 5.8 riportano queste considerazioni.

```
/*flag notizie pregiudizievoli*/
data _null_;
call symput('t5',put((intnx('month',date(),-60,'begin')),Date9.));
call symput('t3',put((intnx('month',date(),-36,'begin')),Date9.));
run;
```

Listato 5.7. Calcolo della presenza o meno di notizie pregiudizievoli associate al cliente.

```
PROC SQL;
   CREATE TABLE WORK.NWIRIS AS
   SELECT DISTINCT t1.COD_ABI,
         CASE WHEN SUBSTR(t1.CF_PIVA,1,5)='00000' THEN SUBSTR(t1.CF_PIVA,6,11)  ELSE  t1.CF_PIVA END AS CF_PIVA,
         t1.COD_TIPO_NOTIZIA,
         t1.DES_NOTIZIA,
         t1.DATA_ELABORAZIONE_FLUSSO,
         t1.DATA_ATTO,
         t1.DATA_CHIUSURA,
         t1.DATA_REVISIONE,
         t1.IMPORTO
      FROM      RTD_FEXT.NWIRIS_GREZZO t1
      WHERE     ( t1.DATA_ATTO >= "&t5."d OR t1.DATA_CHIUSURA >= "&t3."d OR t1.DATA_REVISIONE >= "&t5."d );
QUIT;
```

Listato 5.8. Definizione dell'intervallo della tabella per il calcolo del minter annuale e trimestrale. Questo è necessario nei casi in cui la tabella non è aggiornata con lo stesso anno mese della Customer Base.

Se il cliente è presente nella tabella NWIRIS_GREZZO nell'orizzonte temporale considerato, allora il flag delle notizie pregiudizievoli (flag_not_preg) sarà posto ad 1, altrimenti sarà valorizzato a 0.

Flag Prodotti MLT

Per ogni cliente verranno controllati i contratti aperti con cod_area_informativa in "FML" o "LSG" negli ultimi tre mesi a partire dalla tabella TS_CDMZ_RAP_CONTRATTO, considerando come data di apertura del contratto il campo DAT_APERTURA_CONTRATTO. Tali considerazioni sono riportate nel Listato 5.9.

```
data prodotti_mlt(index=(cod_ndg_contrattuale));
        set cdmz0lib.ts_cdmz_rap_contratto
        (where=(
                cod_area_informativa in (&prodotti_mlt.)
                and cod_abi in (&cod_banca.)
                and datepart(dat_apertura_contratto) <= &eom.
                and dat_apertura_contratto is not missing
                and dat_chiusura_contratto is missing
                and intck('month',datepart(dat_apertura_contratto),&eom.) < &mesi_apertura_contr. ));
        delta = intck('month',datepart(dat_apertura_contratto),&eom.);
run;
```

Listato 5.9. Definizione del flag mlt: clienti che hanno un contratto aperto di tipo mlt da almeno 3 mesi (che non sia stato chiuso al momento dell'esecuzione del modello).

Se il cliente ha almeno un contratto aperto negli ultimi 3 mesi (mesi_apertura_contr) con area FML o LSG e non ancora chiuso al momento dell'esecuzione dell'algoritmo allora il FLAG PRODOTTI MLT verrà posto ad 1, altrimenti verrà posto uguale a 0.

Nello specifico, il passaggio in cui viene definita la distanza in mesi (tra la fine dell'ultimo mese e la data di apertura del contratto) pari a 3 mesi è quello riportato nel Listato 5.10.

```
1    intck('month',datepart(dat_apertura_contratto),&eom.) < &mesi_apertura_contr. ));));
```

Listato 5.10. Calcolo della distanza in mesi per il flag prodotti mlt.

Flag Prodotti Tutela

Per ogni cliente verranno controllati i contratti aperti con `cod_area_informativa` in ("`POL`") negli ultimi tre mesi a partire dalla tabella `TS_CDMZ_RAP_CONTRATTO`, considerando come data di apertura del contratto il campo `DAT_APERTURA_CONTRATTO`. Tali considerazioni sono riportate nel Listato 5.11

```
1    data prodotti_tutela(index=(cod_ndg_contrattuale));
2            set cdmz0lib.ts_cdmz_rap_contratto
3            (where=(cod_area_informativa in (&prodotti_tutela.)
4                    and datepart(dat_apertura_contratto) <= &eom.
5                    and cod_abi in (&cod_banca.)
6                    and dat_apertura_contratto is not missing
7                    and dat_chiusura_contratto is missing
8                    and intck('month',datepart(dat_apertura_contratto),&eom.) < &mesi_apertura_contr.));
9            delta = intck('month',datepart(dat_apertura_contratto),&eom.);
10   run;
```

Listato 5.11. Definizione del flag tutela: clienti che hanno un contratto tutela da almeno 3 mesi (che non sia stato chiuso al momento dell'esecuzione del modello).

Se il cliente ha almeno un contratto aperto negli ultimi 3 mesi (`mesi_apertura_contr`) con area `POL` e non ancora chiuso al momento dell'esecuzione dell'algoritmo allora, il flag `prodotti tutela` verrà posto ad 1, altrimenti verrà posto a 0.

Nello specifico, il passaggio in cui viene definita la distanza in mesi (tra la fine dell'ultimo mese e la data di apertura del contratto) pari a 3 mesi è la medesima dei *prodotti mlt*.

Flag Startup

Per ogni cliente verrà definito il `FLAG STARTUP` a partire dal campo `DAT NASCITA COSTITUZIONE`. La regola verrà definita contando i mesi a partire dal mese di esecuzione per arrivare al mese di costituzione. Se tale distanza, in mesi, è inferiore alla soglia definita dal business, allora questo flag verrà posto ad 1, altrimenti verrà posto a 0. Attualmente, la soglia definita è pari a 24 mesi. Tali considerazioni sono riportate nel Listato 5.12.

```
1    if dat_nascita_costituzione=. then flag_startup=.;
2            else if dat_nascita_costituzione > &eom. then flag_startup=0;
3            else if intck('month',dat_nascita_costituzione,&eom.) le &soglia_startup. then flag_startup=1;
4            else flag_startup=0;
```

Listato 5.12. Definizione del flag startup.

5.1.2 Livello Aree di Analisi

Riprendendo la Sezione 4.2.2, nel seguito verrà spiegato il codice alla base del *calcolo dei KPI*. Inoltre, per ogni singolo KPI, verranno sviluppati dei codici indipendenti finalizzati ad estrarre, da ogni rispettivo input, le numeriche dei mesi di interesse.

Bonifici Entrata/Uscita IT

Come visto nella Sezione 4.3, questo job produce in uscita due tabelle, ovvero:

- `ts_cope_inp_bon_a_it`;
- `ts_cope_inp_bon_b_it`.

Nella prima parte del codice vengono inizializzate tutte le date utili nell'orizzonte temporale di 15 mesi, prendendole dalla `annomese_run`, come mostrato nel Listato 5.13.

```
1   proc sql noprint;
2           select eom, date_15, anno_mese_t0, anno_mese_t14
3           into :eom, :date_15, :anno_mese_t0, :anno_mese_t14
4           from libout.annomese_run;
5   quit;
```

Listato 5.13. Definizione orizzonte temporale.

Successivamente, si trova la massima data di riferimento, all'interno della tabella `ts_cdmz_bon_mov`, poichè, se questa non corrisponde all'ultimo giorno del mese più recente della Customer Base, è necessario ricalcolare 15 mesi indietro, come mostrato nel Listato 5.14.

```
1   /*passaggio per ricalcolare date_15 nel caso la tabella non fosse aggiornata come CB;
2   data in formato sas eom_kpi_sas per ricalcolo e data per oracle eom_kpi*/
3   proc sql noprint;
4           select datepart(max_dt_rif) format ddmmyy10., datepart(max_dt_rif)
5           into :eom_kpi, :eom_kpi_sas
6           from data_bon_mov;
7   quit;
8
9   /*se la tabella è a t-1 rispetto a CB fa ricalcolo di 15 mesi indietro*/
10  %if &eom_kpi. lt &eom_ora. %then
11  %do;
12  data _null_;
13  date_15_ora=put((intnx("month",&eom_kpi_sas.,-14,'B')),ddmmyy10.);
14  call symput('date_15_ora', date_15_ora);
15  run;
16  %end;
```

Listato 5.14. Ricalcolo di 15 mesi indietro.

Dopo di ciò, vengono estratti i movimenti nell'orizzonte temporale di 15 mesi e, allo stesso tempo, vengono filtrati i clienti dalla Customer Base, già calcolati precedentemente. Ciò è mostrato nel Listato 5.15.

```
1   PROC SQL ;
2      CONNECT TO ORACLE as orapass
3       (DB_OBJECTS=ALL READBUFF=10000 PATH=CDMZO authdomain= OraAuth_CDM);
4
5      CREATE TABLE te_cdmz_bon_movimenti AS
6      SELECT *
7         FROM CONNECTION TO orapass (
8         SELECT cod_bonifico,
9                           cod_abi,
10                          dat_riferimento,
11                          imp_bonifico,
12                          cod_tipo_bonifico,
13                          cod_tipo_disposizione,
```

```
                        cod_abi_beneficiario,
                        cod_ndg_beneficiario,
                        cod_iban_beneficiario,
                        cod_abi_ordinante,
                        cod_ndg_ordinante,
                        cod_iban_ordinante
        FROM te_cdmz_bon_movimenti t1
where (dat_riferimento between TO_DATE(%bquote('&date_15_ora. 00:00:00'), 'DD/MM/YYYY HH24:MI:SS')
 and TO_DATE(%bquote('&eom_kpi. 00:00:00'), 'DD/MM/YYYY HH24:MI:SS'))
);

   DISCONNECT FROM orapass;
QUIT;

/*estratto i clienti dalla CB*/
proc sql;
create table cb_distinct as
select distinct cod_abi, cod_ndg, cod_supernsg from libout.ts_cope_inp_customer_base
;
quit;
```

Listato 5.15. Estrazione dei movimenti e dei clienti interessati.

A questo punto, i bonifici in entrata italiani vengono identificati con `COD_NDG_BENEFICIARIO` non "*missing*" e con la prima parte di IBAN che inizia con la voce "IT". Al contrario, i bonifici in uscita italiani, vengono identificati con `COD_NDG_ORDINANTE` diverso da "*missing*" e sempre con la prima parte di IBAN che inizia con la voce "IT". Tali passaggi sono mostrati nel Listato 5.16.

```
data te_cdmz_bon_movimenti_i(drop=estr_ib) te_cdmz_bon_movimenti_o (drop=ib_it);
        set te_cdmz_bon_movimenti;

        num_anno_mese = input((put(year(datepart(dat_riferimento)),vmszn4.))(put(month(datepart(dat_riferimento)),
        vmszn2.)),6.);
        ib_it = substr(cod_iban_ordinante, 1, 2);
        estr_ib = substr(cod_iban_beneficiario, 1, 2);

        if num_anno_mese le &anno_mese_t0 and num_anno_mese ge &anno_mese_t14
        and cod_ndg_beneficiario ne ''
        and (ib_it = 'IT' or ib_it eq '')
        then output te_cdmz_bon_movimenti_i;

        if num_anno_mese le &anno_mese_t0 and num_anno_mese ge &anno_mese_t14
        and cod_ndg_ordinante ne ''
        and (estr_ib = 'IT' or estr_ib eq '') then output te_cdmz_bon_movimenti_o;
run;
```

Listato 5.16. Calcolo dei bonifici in entrata ed uscita.

A questo punto non resta che calcolare le metriche per il numero e l'importo dei movimenti, sia per i bonifici in uscita che quelli in entrata. Tali elaborazioni sono effettuate all'interno del Listato 5.17.

```
/*bonifici uscita IT*/
proc sql;
   create table TS_COPE_INP_BON_D_IT as
   select t1.num_anno_mese as NUM_ANNO_MESE,
                t2.cod_supernsg as COD_SUPERNSG,
          /* importo */
            (sum(t1.imp_bonifico)) format=17.3 as IMP_MOV,
          /* numero */
            (count(t1.cod_abi_ordinante)) as NUM_MOV,
                        'BON_USCITA_IT' as DESC_KPI length=100
      from te_cdmz_bon_movimenti_o t1 inner join tot_clienti t2
          on (t1.cod_abi_ordinante = t2.cod_abi and t1.cod_ndg_ordinante = t2.cod_ndg)
       group by t2.cod_supernsg,
                t1.num_anno_mese;
quit;

/*bonifici entrata IT*/
proc sql;
        create table TS_COPE_INP_BON_A_IT as
        select  t1.num_anno_mese as NUM_ANNO_MESE,
                        t2.cod_supernsg as COD_SUPERNSG,
                (sum(t1.imp_bonifico)) format=17.3 as IMP_MOV,
                (count(t2.cod_supernsg)) as NUM_MOV,
                        'BON_ENTRATA_IT' as DESC_KPI length=100
          from te_cdmz_bon_movimenti_i t1 inner join tot_clienti t2
          on (t1.cod_ndg_beneficiario = t2.cod_ndg and t1.cod_abi_beneficiario = t2.cod_abi)
```

```
27          group by t2.cod_supernsg,
28                   t1.num_anno_mese;
29  quit;
```

Listato 5.17. Calcolo del numero e dell'importo dei moviementi dei bonifici in uscita e in entrata.

Gli output di tale job saranno le due tabelle sopracitate, *ts_cope_inp_bon_a_it* e *ts_cope_inp_bon_b_it*, mostrate nella Sezione 4.2.2, all'interno della Figura 4.5.

F23 F24, Ritiro Effetti, Incassi e Pagamenti IT

Questo job genera le rispettive tabelle dei KPI `F23F24, Incassi Italia, Pagamenti Italia e ritiro effetti`. Esse hanno la stessa struttura della tabella dei bonifici, citata nella Sezione 4.2.2, all'interno della Figura 4.5.

Anche in questo caso, come nella Sezione 5.1.2, vi è una macro che inizializza le date, prese all'interno della tabella `annomese_run`, studiando la massima data dalla tabella dei movimenti. Nel caso in cui tale data non fosse aggiornata come la Customer Base, effettua il ricalcolo dei 15 mesi antecedenti. Il codice che implementa questa elaborazione è mostrato nel Listato 5.18.

```
1   %macro init_date();
2   /*aggiornamento CB*/
3   proc sql noprint;
4           select eom, date_15, anno_mese_t13, anno_mese_t14, intnx("month",eom,-12,'B') as date_13
5           into :eom, :date_15, :anno_mese_t13, :anno_mese_t14, :date_13
6           from libout.annomese_run;
7   quit;
8
9   proc sql noprint;
10          select max(tms_movimento)
11          into :max_tms_mov
12          from RTD_LO_W.t0_cc_movimenti_des_13_m
13          where tms_movimento le &eom.;
14  quit;
15
16  %if &max_tms_mov. lt &eom. %then
17  %do;
18          data _null_;
19          anno_mese_t0_kpi=input(cats (year(&max_tms_mov.) , put(month(&max_tms_mov.),VMSZN2.)), 6.);
20          year = inputn(substr(put(anno_mese_t0_kpi,6.),1,4),4.);
21          month = inputn(substr(put(anno_mese_t0_kpi,6.),5,2),4.);
22          begin_date = mdy(month, 01, year);
23          eom = intnx('month', begin_date, 0, 'E');
24          date_13=intnx("month",eom,-12,'B');
25          call symput ('eom',eom);
26          call symput ('date_13',date_13);
27          do i=0 to 14;
28                  CALL SYMPUTX(cats('anno_mese_t',i),INPUT(CATS(year(intnx("month",eom,-i,'E')),PUT(month(intnx("
            month",eom,-i,'E')),VMSZN2.)),6.));
29          end;
30          run;
31
32  %end;
33
34
35  data _null_;
36  anno_mese_t0_kpi=input(cats (year(&max_tms_mov.) , put(month(&max_tms_mov.),VMSZN2.)), 6.);
37  year = inputn(substr(put(anno_mese_t0_kpi,6.),1,4),4.);
38  month = inputn(substr(put(anno_mese_t0_kpi,6.),5,2),4.);
39  begin_date = mdy(month, 01, year);
40  date_15_mig=intnx("month",begin_date,-14,'B');
41  datetime15mig= DHMS(date_15_mig,0,0,0);
42  datetime15mig2 = put(datetime15mig, datetime20.);
43  call symputx('datetime15mig2',datetime15mig2);
44  run;
```

Listato 5.18. Macro che inizializza le date da utilizzare.

Successivamente, si selezioneranno i contratti dei soli clienti presenti all'interno della Customer Base, poichè, nel passaggio successivo, la chiave alla quale aggan-

ciarsi nella tabella dei movimenti è *codice contratto*. Nel Listato 5.19 viene mostrato il codice che implementa quanto detto.

```
data TS_CDMZ_RAP_CONTRATTO (keep= cod_abi cod_contratto cod_ndg cod_area_informativa);
    set CDMZOLIB.TS_CDMZ_RAP_CONTRATTO (rename=(cod_ndg_contrattuale=cod_ndg) where=(cod_area_informativa='CC'))
    end=eof;
    if _N_=1 then do;
      declare hash x(dataset:"tot_clienti");
      x.defineKey("cod_abi", "cod_ndg");
      x.defineDone();
    end;
    if x.find()=0;
    if eof then x.delete();
run;
```

Listato 5.19. Macro per selezione solo i contratti nella Customer Base.

Infine, dopo aver salvato dentro delle macro variabili i codici causali, se tale codice rientra in uno di quelli presenti nella tabella di configurazione f23 f24 ed ha lo stesso segno di queste ultime, allora questo è il KPI `F23_f24`. Stesso ragionamento per gli altri tre KPI, come si può notare dal Listato 5.20. Tra le varie informazioni che si è deciso di portarsi dietro vi è la somma del numero e quella degli importi dei movimenti.

```
proc sql;
create table tab_causali_recode_mese as
select t1.num_anno_mese,
           t1.cod_abi,
           t1.cod_ndg,
           t1.cod_supernsg,
      /* numero */
        (sum(t1.numero)) as num_mov,
      /* importo */
        (sum(t1.importo)) as imp_mov,
                  case when t1.cod_causale in (&cod_caus_f23_f24.) and t1.segno= &segno_cod_caus_f23_f24. then '
    F23_F24'
                  when t1.cod_causale in (&cod_caus_inc_it.) and t1.segno = &segno_cod_caus_inc_it. then 'INCASSI_IT
    '
                  when t1.cod_causale in (&cod_caus_pag_it.) and t1.segno = &segno_cod_caus_pag_it. then '
    PAGAMENTI_IT'
                  when t1.cod_causale in (&cod_caus_rit_eff.) and t1.segno= &segno_cod_caus_rit_eff. then '
    RITIRO_EFFETTI'
                  else '0'
                  end as DESC_KPI length = 100
      from tab_causali_movimenti t1
      group by  t1.cod_abi,
                                  t1.cod_ndg,
                                  t1.cod_supernsg,
                  (calculated DESC_KPI),
                  t1.num_anno_mese;
quit;
```

Listato 5.20. Divisione dei quattro KPI.

Per concludere, nel passaggio successivo, verranno divisi i vari KPI nelle tabelle di riferimento, come si può notare dal Listato 5.21. In particolare, si moltiplica l'importo per -1 nelle quattro diverse tabelle, poichè gli utenti vogliono vedere gli importi negativi per tali KPI.

```
data libout.ts_cope_inp_f23_f24 libout.ts_cope_inp_incassi_italia
         libout.ts_cope_inp_pag_italia libout.ts_cope_inp_ritiro_effetti;
         set tab_causali_recode_mese;
         if desc_kpi eq 'F23_F24' then
                do;
                        imp_mov=imp_mov*-1;
                        output libout.ts_cope_inp_f23_f24;
                end;
         if desc_kpi eq 'INCASSI_IT' then output libout.ts_cope_inp_incassi_italia;
         if desc_kpi eq 'PAGAMENTI_IT' then
                do;
                        imp_mov=imp_mov*-1;
                        output libout.ts_cope_inp_pag_italia;
                end;
         if desc_kpi eq 'RITIRO_EFFETTI' then
```

```
16                  do;
17                          imp_mov=imp_mov*-1;
18                          output libout.ts_cope_inp_ritiro_effetti;
19                  end;
20                  format imp_mov 15.3;
21  run;
```

Listato 5.21. Divido i vari KPI nelle quattro tabelle.

Incassi Pagamenti Estero

Tale job genera le due tabelle `ts_cope_incassi_estero` e `ts_cope_pagamenti_estero`. Nella prima parte del codice vengono inizializzate le date utili, selezionando da `annomese_run` i primi 15 mesi. Successivamente, si vede qual è il massimo anno mese di riferimento. Questi passaggi sono gli stessi mostrati nel Listato 5.18.

A questo punto, si potranno a filtrare i movimenti nell'orizzonte temporale di riferimento, oltre che effettuare la `JOIN` con il codice statistica filtrato in precedenza. Queste elaborazioni sono mostrate nel Listato 5.22.

```
1   proc sql;
2   create table misure_estrattore_ebs as
3   select t1.cod_abi,
4              t1.cod_contratto_npe,
5              t1.num_anno_mese,
6              t2.cod_stat_segno as segno,
7              (sum(t1.num_statistica)) as sum_of_num_statistica,
8             /* SUM_of_IMP_STATISTICA */
9          (sum(t1.imp_statistica)) format=17.3 as sum_of_imp_statistica
10        from RTD_COIM.t0_imp_ebs_misure_mese t1 inner join tab_configurazione t2
11            on (t1.cod_statistica=t2.cod_stat_rbaridmav)
12        where t1.num_anno_mese between &anno_mese_t0. and &anno_mese_t14. and t1.COD_STATISTICA in (&cod_statistica
            .)
13        and t1.imp_statistica gt 0
14        group by t1.cod_abi,
15                 t1.cod_contratto_npe,
16                 t1.num_anno_mese,
17                 t2.cod_stat_segno;
18  quit;
```

Listato 5.22. Filtro relativo all'orizzonte temporale.

Successivamente, agganciandosi alla `RAP_CONTRATTO`, si prenderanno i contratti dei soli clienti della Customer Base, selezionandoli da `INNER JOINT`. Le corrispettive operazioni sono mostrate nel Listato 5.23.

```
1   proc sql;
2      create table ndg_misure_estrattore as
3      select t2.cod_abi,
4             t2.cod_contratto_npe,
5             t1.cod_ndg,
6             t2.segno,
7             t2.num_anno_mese,
8             t2.sum_of_num_statistica,
9             t2.sum_of_imp_statistica
10       FROM ts_cdmz_rap_contratto t1 inner join misure_estrattore_ebs t2
11           on (t1.cod_abi = t2.cod_abi and t1.cod_contratto = t2.cod_contratto_npe);
12  quit;
```

Listato 5.23. Filtro TRT per pagamenti estero.

A questo punto, si procede ad aggregare per `segno`, per `cod_supernsg` e per `num_anno_mese`, in modo da calcolare il numero di movimenti e l'importo. Queste operazioni sono mostrate nel Listato 5.24.

```
1   proc sql;
```

```
2   create table tab_ebs_tot_1 as
3    select t1.num_anno_mese,
4                   t2.cod_supernsg,
5                   (sum(t1.sum_of_num_statistica)) AS NUM_MOV,
6                   (sum(t1.sum_of_imp_statistica)) FORMAT=17.3 AS IMP_MOV,
7                   (case when segno='A' then 'INC_ESTERO'
8                           when segno='D' then 'PAG_ESTERO'
9                           else ''
10                          end) as DESC_KPI length=100
11        from ndg_misure_estrattore t1, tot_clienti t2
12        where (t1.cod_abi = t2.cod_abi and t1.cod_ndg = t2.cod_ndg) and t2.cod_supernsg not is missing
13        group by t1.segno,
14                 t2.cod_supernsg,
15                 t1.num_anno_mese;
16  quit;
```

Listato 5.24. Calcolo del numero dei movimenti e dell' importo.

Numero di Enti Segnalanti

Tale job genera la tabella `ts_cope_inp_enti_segn`, dove, al contrario delle tabelle precedenti, si ha come output solo il Numero di Enti Segnalanti, e non la colonna riguardante gli import, come mostrato in Figura 4.6, nella Sezione 4.2.2.

In un primo passaggio, vengono inizializzate le date utili, prendendo da `annomese_run` i primi 15 mesi. Successivamente, si vede qual è il massimo anno mese di riferimento. Questi passaggi sono gli stessi mostrati nel Listato 5.18.

Dai clienti della Customer Base vengono prese solo le relazioni presenti all'interno della tabella `rel_pa_ndg` e, successivamente, vengono filtrate le relazioni di tipo `TIT`, come mostrato nel Listato 5.25. Tali relazioni sono quelle che esprimono il legame tra la ditta individuale e il titolare della ditta.

```
1    proc sql ;
2      connect to oracle as orapass
3       (db_objects=all readbuff=10000 path=cdmz0 authdomain= oraauth_cdm);
4
5      create table tr_cdmz_rel_pa_ndg as
6      select *
7         from connection to orapass (
8         select cod_ndg,
9                   cod_abi,
10             cod_tipo_relazione,
11                          flg_referente,
12                          cod_ndg_posizione_anagrafica,
13                          dat_inizio_relazione,
14                          dat_fine_relazione
15          from tr_cdmz_rel_pa_ndg
16                  where dat_inizio_relazione <= to_date(%bquote('&eom_kpi. 00:00:00'), 'dd/mm/yyyy hh24:mi:ss')
17                  and (dat_fine_relazione >=to_date(%bquote('&date_15_ora. 00:00:00'), 'dd/mm/yyyy hh24:mi:ss')
18                          or dat_fine_relazione is null)
19  );
20
21     disconnect from orapass;
22  quit;
23
24  /*prendo solo le relazioni dei clienti presenti nella cb*/
25  data tr_cdmz_rel_pa_ndg_1;
26          set tr_cdmz_rel_pa_ndg (where=(cod_tipo_relazione in ("TIT"))
27                                                          ) end=eof;
28      if _n_=1 then do;
29        declare hash x(dataset:"tot_clienti");
30        x.definekey("cod_abi", "cod_ndg");
31        x.definedone();
32      end;
33      if x.find()=0;
34      if eof then x.delete();
35  num_anno_mese = input((put(year(datepart(dat_inizio_relazione)),vmszn4.))(put(month(datepart(dat_inizio_relazione)
          ),vmszn2.)),6.);
36  run;
```

Listato 5.25. Filtro delle sole relazioni di interesse.

Successivamente, si filtrano i clienti che hanno un `flag_affidato`=1, preso dalla `anagrafica_pg`.

```
data anagrafica_pg (keep= cod_abi cod_ndg cod_supernsg);
length cod_ndg $13;
length cod_abi $5;
set ctimp.anagrafica_pg(where=(flg_affidato=1)) end=eof ;
if _n_=1 then do;
      declare hash x(dataset:"tot_clienti");
      x.definekey("cod_supernsg");
          x.definedata("cod_abi", "cod_ndg");
      x.definedone();
    end;
    if x.find()=0;
    if eof then x.delete();
run;
```

Listato 5.26. Filtro su cliente affidato.

A questo punto vengono, poi, aggiunte le informazioni del Numero di Enti Segnalanti prendendoli dalla tabella `scr_ndg`. Tale campo è quello che, in seguito, va a "valorizza" il Numero di Enti Segnalanti nella tabella finale. Il Listato 5.27 presenta il codice che implementa tale aggiunta.

```
proc sql ;
   connect to oracle as orapass
    (db_objects=all readbuff=10000 path=cdmz0 authdomain= oraauth_cdm);

   create table te_cdmz_scr_ndg as
   select *
      from connection to orapass (
      select distinct cod_ndg,
                                  cod_abi,
                                  num_anno_mese,
                                  num_enti_affidatanti
         from te_cdmz_scr_ndg
                where num_anno_mese between %bquote(&anno_mese_t14.) and %bquote(&anno_mese_t0.)

);

   disconnect from orapass;
quit;
```

Listato 5.27. Filtro su cliente affidato.

Pagamenti POS

Questo KPI genera la tabella di output `ts_cope_imp_mov_pos.sas7bdat`, all'interno della quale vi è sia l'informazione relativa al numero di movimenti che quella relativa all'importo dei movimenti per pagamenti POS.

Vi è un primo passaggio dove si studia la data più recente dalla tabella dei movimenti, per poi estrarre i movimenti dalla `pos_movimenti`, nell'orizzonte temporale di 15 mesi. Questi passaggi sono gli stessi mostrati nei Listati 5.13 e 5.14.

Successivamente, si aggancia l'estrazione nuovamente alla `RAP_CONTRATTO`, prendendo tutti i contratti dei soli clienti della Customer Base, come mostrato nel Listato 5.23.

Infine, si definisce la *label* del KPI `MOV_POS`, aggregando i movimenti per cliente `cod_supernsg` e `annomese`. Come ultimo passaggio, si associa nel codice abilitativo `COD_ABI` sia il `COD_NDG` che il `COD_SUPERNDG` della Customer Base. Quest'ultimo passaggio è mostrato nel Listato 5.28.

```
proc sql;
   create table TS_COPE_INP_MOV_POS as
   select t1.annomese as NUM_ANNO_MESE,
                t2.COD_SUPERNSG,
          (sum(t1.numero)) as NUM_MOV,
          (sum(t1.importo)) format=17.3 as IMP_MOV,
                'POS' as DESC_KPI length=100
```

```
8         from ndg_monimenti_pos t1, tot_clienti t2
9         where (t1.cod_abi = t2.cod_abi and t1.cod_ndg = t2.cod_ndg) and t2.cod_supernsg not is missing
10        group by t2.cod_supernsg,
11                 t1.annomese;
12    quit;
13
14    /*aggancio abi e ndg attuali della cb*/
15    proc sql;
16       create table libout.TS_COPE_INP_MOV_POS as
17       select t1.NUM_ANNO_MESE,
18                  t2.COD_ABI,
19                  t2.COD_NDG,
20                  t2.COD_SUPERNSG,
21              NUM_MOV,
22              IMP_MOV,
23                  DESC_KPI
24          from TS_COPE_INP_MOV_POS t1 inner join cb_distinct t2 on t1.cod_supernsg=t2.cod_supernsg;
25    quit;
```

Listato 5.28. Creazione della tabella finale.

Rid MAV

Questo KPI genera la tabella di output `ts_cope_imp_riba_mav.sas7bdat` all'interno della quale vi è sia l'informazione relativa al numero di movimenti che quella relativa all'importo degli stessi per pagamenti MAV.

Come per i Pagamenti POS, Sezione 5.1.2, vi è un primo passaggio dove si studia la massima data più recente dalla tabella dei movimenti, per poi andare ad estrarre i movimenti dall'unione delle tre tabelle `presn_italia`, `presn_ita_chiuse` e `presn_sepa`, nell'orizzonte temporale di 15 mesi. Questi passaggi sono gli stessi mostrati nei Listati 5.13 e 5.14.

Successivamente, ci si aggancia nuovamente alla `RAP_CONTRATTO`, prendendo tutti i contratti dei soli clienti della Customer Base, come mostrato nel Listato 5.23.

Infine, come già mostrato nel Listato 5.28, si definirà la label del KPI `rid_mav`, aggregando i movimenti per cliente `cod_supernsg` e per `annomese`. Come ultimo passaggio si associerà, attraverso il codice abilitativo `COD_ABI`, sia il `COD_NDG` che il `COD_SUPERNDG` della Customer Base.

SOW BT e UT ACC

Per gli ultimi due KPI, i passaggi sono identici a quelli mostrati nella Rid MAV, Sezione 5.1.2, fatta eccezione per il fatto che i calcoli finali sono effettuati con valore percentuale sulla media del trimestre.

5.1.3 Livello Loader

Questo job effettua il caricamento della Customer Base e dei KPI all'interno del database Oracle e, ad ogni run, effettua la *truncate* di ciò che è già presente nel database. Questi calcoli saranno effettuati ogni volta per ognuno dei 15 mesi. Infine, si effettua una `append` sulle 13 tabelle. Il Listato 5.29 implementa quanto appena detto.

```
1   %macro truncate_table(tabin=);
2   %if %sysfunc(exist(ORAPYT.&tabin.)) eq 1 %then %do;
3       proc sql;
4       connect to oracle (authdomain="OraAuth_PPYT_APP" path='ppyt0');
5       execute(declare ret number := 0; begin ret := PPYT_OWN.Pkg_PPYT_Admin_Partition.Fnd_PPYT_Truncate_Table(%
            bquote('&tabin.')); end; ) by oracle;
```

```
6           execute(commit) by oracle;
7       disconnect from oracle;
8       quit;
9       %end;
10  %mend truncate_table;
11  %truncate_table(tabin = TS_COPE_INP_CUSTOMER_BASE);
12  %truncate_table(tabin = TS_COPE_INP_BON_A_IT);
13  %truncate_table(tabin = TS_COPE_INP_BON_D_IT);
14  %truncate_table(tabin = TS_COPE_INP_ENTI_SEG);
15  %truncate_table(tabin = TS_COPE_INP_F23_F24);
16  %truncate_table(tabin = TS_COPE_INP_INCASSI_ESTERO);
17  %truncate_table(tabin = TS_COPE_INP_INCASSI_ITALIA);
18  %truncate_table(tabin = TS_COPE_INP_MOV_POS);
19  %truncate_table(tabin = TS_COPE_INP_PAG_ESTERO);
20  %truncate_table(tabin = TS_COPE_INP_PAG_ITALIA);
21  %truncate_table(tabin = TS_COPE_INP_RID_RIBA_MAV);
22  %truncate_table(tabin = TS_COPE_INP_RITIRO_EFFETTI);
23  %truncate_table(tabin = TS_COPE_INP_SOW_UT_BT);
24  %truncate_table(tabin = TS_COPE_INP_UT_ACC);
25
26  %macro append_table(tabin=);
27  proc append data=libout.&tabin.
28                          base=ORAPYT.&tabin.;
29  run;
30  %mend append_table;
31  %append_table(tabin = TS_COPE_INP_CUSTOMER_BASE);
32  %append_table(tabin = TS_COPE_INP_BON_A_IT);
33  %append_table(tabin = TS_COPE_INP_BON_D_IT);
34  %append_table(tabin = TS_COPE_INP_ENTI_SEG);
35  %append_table(tabin = TS_COPE_INP_F23_F24);
36  %append_table(tabin = TS_COPE_INP_INCASSI_ESTERO);
37  %append_table(tabin = TS_COPE_INP_INCASSI_ITALIA);
38  %append_table(tabin = TS_COPE_INP_MOV_POS);
39  %append_table(tabin = TS_COPE_INP_PAG_ESTERO);
40  %append_table(tabin = TS_COPE_INP_PAG_ITALIA);
41  %append_table(tabin = TS_COPE_INP_RID_RIBA_MAV);
42  %append_table(tabin = TS_COPE_INP_RITIRO_EFFETTI);
43  %append_table(tabin = TS_COPE_INP_SOW_UT_BT);
44  %append_table(tabin = TS_COPE_INP_UT_ACC);
```

Listato 5.29. Codice relativo al livello di loader.

5.2 Implementazione Python

Per quanto riguarda il microservizio in Python, questo funziona attraverso un semplice rooting, una `Flask App.py`, grazie al quale l'applicazione darà in risposta diversi comportamenti in base al determinato rooting scelto. I più importanti sono quelli che referenziano lo stato del microservizio e, attraverso tali stati, potremmo vedere non solo lo stato attuale di questo, ma anche altre informazioni utili, mostrate nella Sezione 4.5.2.

Come già delineato nella Sezione 4.4.1, si definirà il codice che implementa le varie fasi principali del modello in Python.

Funzione di Start

Nell'operazione di "`/Start`" la prima cosa che si fa è controllare lo stato attuale; il momento in cui questo può iniziare è preceduto da uno stato di `failed`, `idle` (attesa), `stopped` o `expired`. Una volta partito, si registra la data di partenza e lo stato passa in `starting`. In questo preciso momento, verranno creati dei thread per gestire nel modo migliore possibile i processi, dal punto di vista della memoria e di occupazione di risorse della macchina. Quanto appena spiegato è mostrato all'interno del Listato 5.30.

```
1       # start the thread
2       def start(self):
```

```
3           try:
4               # init counters
5               self.get_status()
6               if self.status in [JobStatus.FAILED, JobStatus.IDLE, JobStatus.STOPPED, JobStatus.EXPIRED]:
7                   self.started = datetime.now()
8                   self.status = JobStatus.STARTING
9
10                  # update the db report tables
11                  logger.info('PREPARING OUTPUT TABLES')
12                  self.provider.prepare_output_tables()
13
14                  # start async thread
15                  self.thread = threading.Thread(target=self.run, args=())
16                  self.thread.daemon = True
17                  self.thread.start()
18                  self.thread.join()
19              else:
20                  logger.warning(f'You can\'t start while status is {self.status}')
21          except:
22              self.status = JobStatus.FAILED
23              self.last_run = datetime.now()
24              logger.error(traceback.print_exc())
```

Listato 5.30. Codice della funzione di start.

Funzione di Run

Una volta che il microservizio è partito, si passerà ad una fase di `run`. Durante questa fase, innanzitutto, si avrà una connessione al database Oracle, attraverso utente, password e stringa di collegamento Oracle. L'input principale di questa funzione è la Customer Base, che, come spiegato nella Sezione 5.1.3, è l'output della fase precedente SAS. Al contrario, in uscita dalla funzione, si avranno due tabelle: report completo e report ridotto. La differenza tra i due report è che, in quello completo, oltre che segnalare i motivi per cui vi è stato un calo di operatività da parte dell'utente, vi sono evidenziati anche gli score relativi ai 13 KPI, oltre che dei ranking legati alla segmentazione.

Se, dopo la connessione al database, lo stato del microservizio non è in `stopped` o `failed`, si procederà in *chunck*. Tali *chunck* vengono utilizzati per questioni di risorse, in quanto, in ambiente di produzione, si interroga la Customer Base che è massiva. Portarsi dietro, durante tutta l'elaborazione, l'intera tabella, soprattutto con la presenza di 13 elaborazioni dei KPI e altre operazioni di ranking, è molto usurante dal punto di vista delle risorse della macchina. Perciò, per questo motivo, la Customer Base è stata divisa in *chunck*, con un *chunck size* che, a seguito delle evolutive per l'integrazione di banca UBI, è di circa 70.000 clienti. Il codice che implementa quanto detto è mostrato nel Listato 5.31.

```
1   # run the process
2      def run(self):
3          # Oracle connection
4          try:
5              connection = cx_Oracle.connect(self.config.ora_user, self.config.ora_password,
6                                             self.config.connection_string_cx)
7          except:
8              logger.warning("Unable to connect")
9              logger.error(traceback.format_exc())
10
11         chunk = 1
12         try:
13
14             # fetching chunks from customer_base
15             if self.status not in [JobStatus.STOPPED, JobStatus.FAILED]:
16
17                 # chunked elaboration for the segmentated Customer Base
18                 for customer_base_chunk in pd.read_sql(
19                         "SELECT " + ", ".join(self.config.cust_base_columns_list) + " FROM " + self.config.
        table_input,
20                         connection, chunksize=self.config.chunksize):
21
```

```
22                      logger.info('CHUNK N: ' + str(chunk))
23                      if self.status not in [JobStatus.STOPPED, JobStatus.FAILED]:
```

Listato 5.31. Codice sulla connessione e divisione in *chunck* della funzione di run .

Successivamente, si partirà con l'elaborazione vera e propria, attraverso la `kpi_elaboration`. Il codice che implementa questa fase all'interno della funzione di `run` è mostrato nel Listato 5.32.

```
1                       logger.info('CHUNK N: ' + str(chunk))
2                       if self.status not in [JobStatus.STOPPED, JobStatus.FAILED]:
3
4                           # kpi chunk elaboration process
5                           try:
6                               self.status = JobStatus.KPI_ELABORATION
7                               kpi_elab_process = Process(target=self.modeling.kpi_elaboration(customer_base_chunk))
8                               kpi_elab_process.start()
9                               self.processes.append(kpi_elab_process)
10                              chunk_elab = self.modeling.queue.get()
11                              kpi_elab_process.join()
12                          except:
13                              logger.error('Kpi Elaboration failed')
14                              logger.error(traceback.format_exc())
15                              self.status = JobStatus.FAILED
```

Listato 5.32. Codice che implementa la kpi elaboration all'interno della funzione di run.

5.2.1 KPI Elaboration

La funzione `kpi_elaboration`, presente all'interno della classe `ModelingOperations`, nel file `modeling_operations.py`, ha come prima operazione quella di connessione al database Oracle attraverso la libreria `cx_Oracle`.

A questo punto si prende la data corrente e, attraverso la definizione dei *periodi*, si effettueranno operazioni diverse, mostrate nella Sezione 4.4.3, per i 13 diversi KPI. Per *periodi* si intende il confronto tra il trimestre attuale e il trimestre dell'anno precedente. Tali KPI vengono, poi, elaborati per ogni *chunck*.

Per ogni KPI viene data una valutazione che ha esito positivo o negativo; in base a questo, si decide se segnalare l'utente che ha subito un calo di operatività. Oltre ad eseguire il calcolo per il KPI singolo, viene anche conteggiato il numero di penalità che un utente ottiene su 13 KPI.

La coda del processo dell'elaborazione dei KPI restituirà la Customer Base con l'aggiunta di tutte le colonne create da tale processo al `job_manager.py` e, se non ci sono eccezioni, si procederà a scrivere il risultato. Questo viene mostrato nel listato 5.33. Questa nuova tabella verrà chiamata `report_completo_parziale`.

```
1     # SCRITTURA CAMPI NON UTILI PER LA SEGMENTAZIONE E PRIORITIZZAZIONE
2                           try:
3                               self.status = JobStatus.WRITING_OUTPUT
4                               self.write_output(chunk_elab)
5
6                           except:
7                               logger.error('Writing output failed')
8                               logger.error(traceback.format_exc())
9                               self.status = JobStatus.FAILED
```

Listato 5.33. Codice che implementa la scrittura del risultato.

Da notare che, fino a questo punto, durante i vari passaggi, lo stato è passato da `starting` a `kpi_elaboration` a `writing_output`. Una volta terminata quest'ultima fase, passerà alla `segmentation` e, successivamente, `prioritization`.

Funzione di scrittura dell' output

Questa funzione, contenente i KPI elaborati, ha come parametro il dataframe `df_calo_op` ed esegue, innanzitutto, il conteggio, per ogni utente, dei punti positivi, di quelli negativi e delle penalizzazioni. Questi tre `COUNT` verranno nuovamente sommati per creare la `somma_punti_modello`, che andrà scritta sul report completo.

Successivamente, si definirà il `des_motivo_segnalazione`, che rappresenta una concatenazione dei nomi dei KPI che hanno una valutazione negativa. Questo è stato ideato poiché, leggendo il report completo, il cliente deve capire non solo che ha avuto un calo di operatività rispetto al trimestre dell'anno precedente, ma anche la motivazione, data dalla concatenazione dei KPI che hanno avuto una valutazione negativa. Quanto appena detto è mostrato nel Listato 5.34.

```
1       def write_output(self, df_calo_op):
2           logger.info('writing output')
3           # count NUM_PUNTI, NUM_PENALIZZAZIONI, NUM_PUNTI_MODELLO
4           df_calo_op = self.modeling.counting(df_calo_op)
5           # define MOTIVO_SEGNALAZIONE
6           df_calo_op['DES_MOTIVO_SEGNALAZIONE'] = df_calo_op.apply(
7               lambda r: self.modeling.set_motivo_segnalazione(r), axis=1)
8           # rename columns
9           df_calo_op = df_calo_op.rename(columns={
10                  'NUM_BONIFICI_AVERE_ITA': 'NUM_BONIFICI_AVERE_ITALIA',
11                  'NUM_BONIFICI_DARE_ITA': 'NUM_BONIFICI_DARE_ITALIA',
12                  'NUM_PAGAMENTI_ITA': 'NUM_PAGAMENTI_ITALIA',
13                  'NUM_NUM_ENTI_SEG': 'NUM_ENTI_SEGNALANTI_CR',
14                  'NUM_INCASSI_ITA': 'NUM_INCASSI_ITALIA',
15                  'NUM_PEN_INCASSI_IT': 'NUM_PEN_INCASSI_ITA',
16                  'IMP_MARGIN_TRIM_P1': 'IMP_MARG_TRIM_P1',
17                  'IMP_MARGIN_TRIM_P2': 'IMP_MARG_TRIM_P2'
18          })
```

Listato 5.34. Codice che contiene il conteggio e la concatenazione dei motivi della segnalazione di calo operatività.

Infine, si va a scrivere sul report attraverso un parallellismo, poichè l'operazione di scrittura rallentano di molto l'elaborazione finale. Grazie a questa ottimizzazione, dalle 9 ore iniziali di elaborazione, si è passati a 4 ore. Il parallelismo divide ciascun *chunck* in sotto-categorie che verranno scritte, attraverso una `pool`, in parallelo sulla tabella di output. Il codice che implementa quanto appena detto è rappresentato nel Listato 5.35.

```
1           logger.info('Writing reports')
2           # writing reports
3           c = self.config.chunksize_save
4           p = self.config.processes_save
5
6           with Pool(processes=p) as pool:
7               pool.map(self.provider.write_report_completo,
8                        [df_calo_op[i * c:(i + 1) * c] for i in range((len(df_calo_op) + c - 1) // c)])
9
10              pool.map(self.provider.write_report_ridotto,
11                       [df_calo_op[df_calo_op['FLG_PERIMETRO'] == 1][i * c:(i + 1) * c] for i in
12                        range((len(df_calo_op) + c - 1) // c)])
13
14              pool.close()
15              pool.join()
16          logger.info('writing output:DONE')
```

Listato 5.35. Codice che contiene il parallelismo della scrittura dell'output.

Per fare questo, si è creato il file `provider.py`, il quale contiene tutte quelle operazioni di interazione con Oracle, come la scrittura del report completo, il suo update e la sua truncate. Nel Listato 5.36 viene mostrata la funzione richiamata nel Listato 5.34.

```
1       def write_report_completo(self, df: pd.DataFrame):
2           columns = self.config.report_completo_columns_list
3           table = self.config.table_report_completo
4           try:
5               engine = create_engine('oracle+cx_oracle://' + self.config.connection_string_pd,
6                                      max_identifier_length=128, pool_pre_ping=True)
7
8               df[columns].to_sql(table, engine, index=False, if_exists='append', schema='PPYT_OWN',
9                                  chunksize=100)
10          except:
11              logger.warning('Cannot write report')
12              traceback.print_exc()
```

Listato 5.36. Codice che contiene il conteggio e la concatenazione dei motivi della segnalazione di calo operatività.

5.2.2 Segmentazione

Dopo l'operazione di scrittura nel reporto completo, vi è l'operazione di segmentazione. In questo caso vengono, innanzitutto, estratti non i singoli chunk ma l'intera tabella, per poi effettuare un ranking su tutti gli utenti della Customer Base, dividendoli per codice di tipo portafoglio. Tale operazione è mostrata nel Listato 5.37.

```
1           self.tipiPortafoglioList = [['0', '4'], ['U', 'G', 'I'], ['9', 'D'], ['5', 'E']]
```

Listato 5.37. Divisione dei codici di tipo portafoglio.

Da qui, all'interno del `job_manager.py`, se lo stato non è in `stopped` o `failed`, si cambierà lo stato a `segmentation` e si richiamerà la funzione di segmentazione presente nella `modeling_operations.py`. Tutto ciò è racchiuso nel Listato 5.38.

```
1               # segmentation process
2               if self.status not in [JobStatus.STOPPED, JobStatus.FAILED]:
3                   try:
4                       self.status = JobStatus.SEGMENTATION
5                       segmentation_process = Process(target=self.modeling.segmentation())
6                       segmentation_process.start()
7                       self.processes.append(segmentation_process)
8                       df_segmentation = self.modeling.queue.get()
9                       segmentation_process.join()
10                  except:
11                      logger.error('Segmentation failed')
12                      logger.error(traceback.format_exc())
13                      self.status = JobStatus.FAILED
```

Listato 5.38. Codice che contiene il richiamo della funzione di segmentazione.

A questo punto, viene eseguita la connessione al database, per poi calcolare la data attuale e leggere la Customer Base elaborata, addizionata dell'elaborazione dei KPI.

Perciò, dopo aver estratto tutte le righe della nuova Customer Base, si itererà, attraverso un ciclo `for`, per codice di tipo portafoglio, calcolando un percentile attraverso un *rank*. Il numero derivante da questa operazione verrà, poi, associato a ciascun cliente e servirà per essere "sovrapposto" ai valori che vengono estratti dal calcolo dell'elaborazione dei KPI, come spiegato nella Sezione 4.4.4. Perciò, a questo punto, si avrà una Customer Base che avrà sia le colonne prese dall'elaborazioni `SAS`, sia le informazioni elaborate tramite i 13 KPI, unite alla colonna `RANK_MINTER_ANTE_RETTIFICHE`, calcolata sull'importo ante rettifiche, attraverso

l'operazione di *rank* precedentemente citata. Infine, concatenarà il tutto in un unico dataframe, e la coda del processo restituirà tale dataframe con la segmentazione. Quanto appena detto è mostrato nel Listato 5.39.

```
    # ranking by percentile calculation
    def segmentation(self):
        connection = cx_Oracle.connect(self.config.ora_user, self.config.ora_password,
                                       self.config.connection_string_cx)
        current_date = dt.datetime.strptime(str(
            pd.read_sql("SELECT MAX(" + self.config.table_input + ".NUM_ANNO_MESE) FROM " + self.config.
        table_input,
                        connection).loc[0].item()), '%Y%m')

        df_elab = pd.read_sql("SELECT " + ", ".join(
            self.config.prioritization_columns_list) + ",(" + self.config.final_score_columns_list + ") AS MOD_TOT
         FROM " + self.config.table_report_completo + " WHERE " + self.config.table_report_completo + ".
        NUM_ANNO_MESE = " + current_date.strftime(
            '%Y%m'), connection)

        df_cod_tipo_portafoglio = pd.read_sql(
            "SELECT " + ", ".join(self.config.cod_tipo_por_columns_list) + " FROM " + self.config.table_input,
            connection)

        df_elab = df_elab.merge(df_cod_tipo_portafoglio, on='COD_NDG', how='left')

        df_segmentation = pd.DataFrame([])

        # df division by COD_TIPO_PORTAFOGLIO
        for typo in self.config.tipiPortafoglioList:
            temp = df_elab[df_elab['COD_TIPO_PORTAFOGLIO'].isin(typo)].copy()
            temp = temp.fillna(0)
            # ranking minter
            temp['RANK_MINTER_ANTE_RETTIFICHE'] = temp['IMP_MINTER_ANTE_RETTIFICHE'].rank(pct=True) * 100
            df_segmentation = pd.concat([df_segmentation, temp], ignore_index=True)

        logger.info('Segmentation: DONE')
        self.queue.put(df_segmentation)
```

Listato 5.39. Calcolo della segmentazione attraverso l'operazione di ranking.

5.2.3 Prioritizzazione

La prioritizzazione, come già presentato nella Sezione 4.4.4, è un processo attraverso il quale il ranking uscente dalla segmentazione e il campo `MOD_TOT` uscente dall'elaborazione dei 13 KPI si sovrapporranno in modo che verrà assegnata una priorità ad ogni utente.

Innanzitutto, all'interno del `job_manager.py`, se lo stato non è in `stopped` o `failed`, si cambierà lo stato in `prioritization` e si richiamerà la funzione di segmentazione presente in `modeling_operations.py`. Tutto ciò, similmente a quanto mostrato per la segmentazione, è racchiuso nel Listato 5.40.

```
            # prioritization process
            if self.status not in [JobStatus.STOPPED, JobStatus.FAILED]:
                try:
                    self.status = JobStatus.PRIORITIZATION
                    prioritization_process = Process(target=self.modeling.prioritization(df_segmentation))
                    # prioritization_process = Process(target=modelOperations.prioritization(df_in))
                    prioritization_process.start()
                    self.processes.append(prioritization_process)
                    df_prioritization = self.modeling.queue.get()
                    prioritization_process.join()
                    logger.info('Prioritization: DONE')
                except:
                    logger.error(traceback.format_exc())
                    logger.error('Prioritization failed')
                    self.status = JobStatus.FAILED
```

Listato 5.40. Richiamo della funzione di prioritizzazione all'interno del job manager.

La funzione all'interno della `modeling_operations.py` implementa l'assegnazione delle priorità riportate nella Tabella 4.19, mostrata all'interno della Sezione

4.4.4. Finita la prioritizzazione, il processo restituisce il dataframe prioritizzato. Il codice che implementa quanto di cui detto sopra è mostrato nel Listato 5.41.

```
1       def set_priority(self, row):
2           if row['COD_SPECIE_GIURIDICA'] != 'COND ':
3               if (row['MOD_TOT'] <= -4) and (row['RANK_MINTER_ANTE_RETTIFICHE'] > 60):
4                   return '1_ALTA_P'
5               elif (-4 < row['MOD_TOT'] < 0) and (row['RANK_MINTER_ANTE_RETTIFICHE'] > 85):
6                   return '2_MEDIA_P'
7               elif (row['MOD_TOT'] <= -4) and (row['RANK_MINTER_ANTE_RETTIFICHE'] <= 60):
8                   if (row['FLG_STARTUP'] != 1) and (row['RANK_MARGIN_TRIM_P1'] <= 30) and (
9                           row['RANK_MARGIN_TRIM_P2'] <= 30):
10                      return '3_BASSA_P'
11                  else:
12                      return '2_MEDIA_P'
13              else:
14                  return '3_BASSA_P'
15          else:
16              return '3_BASSA_P'
```

Listato 5.41. Codice che implementa il processo di prioritizzazione.

5.2.4 Update del report completo

Terminata la fase di prioritizzazione, si eseguirà un update del report completo. Questo infatti si completerà con le colonne risultati dall'operazione precedente di prioritizzazione.

Questa fase viene nuovamente eseguita attraverso una parallelizzazione in `pool`, poichè, altrimenti, le fasi di scrittura risultavano essere molto lunghe.

La prima operazione, all'interno della funzione nel file `provieder.py`, è la connessione al database, per poi estrarre la data del mese corrente. Per `COD_NDG`, che rappresenta la chiave univoca per ogni utente, viene eseguito l'update, scrivendo le colonne mancanti della prioritizzazione all'interno del report completo e del report ridotto. Il codice che implementa tutto ciò è mostrato nel Listato 5.42.

```
1       def update_reports(self, df):
2
3           connection = cx_Oracle.connect(self.config.ora_user, self.config.ora_password,
4                                          self.config.connection_string_cx)
5           current_date = dt.datetime.strptime(str(
6               pd.read_sql("SELECT MAX(" + self.config.table_input + ".NUM_ANNO_MESE) FROM " + self.config.
7           table_input,
                            connection).loc[0].item()), '%Y%m')
8           cur = connection.cursor()
9           # UPDATE
10          for cod in df['COD_NDG']:
11              cur.execute("UPDATE TS_COPE_OUT_REPORT_COMPLETO SET DES_TARGET = '" + str(
12                  df[df['COD_NDG'] == cod]['DES_TARGET'].item()) + "' WHERE COD_NDG='" + str(
13                  cod) + "' AND NUM_ANNO_MESE= " + current_date.strftime('%Y%m'))
14          for cod in df[df['FLG_PERIMETRO'] == 1]['COD_NDG']:
15              cur.execute("UPDATE TS_COPE_OUT_REPORT_RIDOTTO SET DES_TARGET = '" + str(
16                  df[df['COD_NDG'] == cod]['DES_TARGET'].item()) + "' WHERE COD_NDG= '" + str(
17                  cod) + "' AND NUM_ANNO_MESE= " + current_date.strftime('%Y%m'))
18
19          connection.commit()
20          cur.close()
```

Listato 5.42. Codice che implementa il processo di update del report completo e ridotto.

All'interno del `job_manager`, l'update del report viene dato a `DONE` e lo stato del microservizio passerà, a sua volta, a `DONE`, come mostrato nel Listato 5.43.

```
1       # write the required reports
2       def write_output(self, df_calo_op):
3           logger.info('writing output')
4
5           # count NUM_PUNTI, NUM_PENALIZZAZIONI, NUM_PUNTI_MODELLO
```

```
6           df_calo_op = self.modeling.counting(df_calo_op)
7
8           # define MOTIVO_SEGNALAZIONE
9           df_calo_op['DES_MOTIVO_SEGNALAZIONE'] = df_calo_op.apply(
10              lambda r: self.modeling.set_motivo_segnalazione(r), axis=1)
11
12          # rename columns
13          df_calo_op = df_calo_op.rename(columns={
14                  'NUM_BONIFICI_AVERE_ITA': 'NUM_BONIFICI_AVERE_ITALIA',
15                  'NUM_BONIFICI_DARE_ITA': 'NUM_BONIFICI_DARE_ITALIA',
16                  'NUM_PAGAMENTI_ITA': 'NUM_PAGAMENTI_ITALIA',
17                  'NUM_NUM_ENTI_SEG': 'NUM_ENTI_SEGNALANTI_CR',
18                  'NUM_INCASSI_ITA': 'NUM_INCASSI_ITALIA',
19                  'NUM_PEN_INCASSI_IT': 'NUM_PEN_INCASSI_ITA',
20                  'IMP_MARGIN_TRIM_P1': 'IMP_MARG_TRIM_P1',
21                  'IMP_MARGIN_TRIM_P2': 'IMP_MARG_TRIM_P2'
22          })
23
24          logger.info('Writing reports')
25          # writing reports
26          c = self.config.chunksize_save
27          p = self.config.processes_save
28
29          with Pool(processes=p) as pool:
30              pool.map(self.provider.write_report_completo,
31                       [df_calo_op[i * c:(i + 1) * c] for i in range((len(df_calo_op) + c - 1) // c)])
32
33              pool.map(self.provider.write_report_ridotto,
34                       [df_calo_op[df_calo_op['FLG_PERIMETRO'] == 1][i * c:(i + 1) * c] for i in
35                        range((len(df_calo_op) + c - 1) // c)])
36
37              pool.close()
38              pool.join()
39          logger.info('writing output:DONE')
```

Listato 5.43. Codice che fa partire la funzione di update dei report e fa passare lo stato a done.

6

Testing e Tuning

In questo capitolo si discuterà l'ultima fase del lavoro finora presentato, ovvero la validazione del progetto. Verranno dapprima analizzate le procedure di test e di rilascio seguite all'interno di Intesa SanPaolo e, successivamente, verrà mostrato come esse siano state utilizzate nel progetto in esame. Infine, è stato presentato lo strumento aziendale che permette di effettuare il tuning per cambiamenti alle logiche implementative, a posteriori dal rilascio in produzione.

6.1 Il processo di test

Il processo di testing definito all'interno dell'azienda prevede i seguenti livelli, ovvero:

- *Unit Test*, livello nel quale sono testati i singoli moduli delle applicazioni; l'ambiente di sviluppo contiene gli strumenti per effettuare tale primo test unitario;
- *Application Test*, per assicurare che siano soddisfatti i requisiti funzionali;
- *System Test*, per verificare che l'applicazione soddisfi anche i requisiti non funzionali come l'integrazione con la sicurezza;
- *User Acceptance Test*, per fornire agli utenti un ambiente stabile in cui testare i requisiti end-to-end;
- *Performance Test*, per garantire che l'applicazione soddisfi i livelli di performance dichiarati, necessari per il soddisfacimento dell'utente.

Per poter testare le funzioni del sistema in modo efficace è necessario affrontare, secondo una modalità a cascata, i seguenti passi, ovvero:

- *pianificazione del test*, per arrivare ad identificare le *risorse* necessarie nella fase di test;
- *progettazione dei casi di test*, per definire a partire dai requisiti di progetto i casi di test ed identificare la copertura rispetto ai requisiti;
- *esecuzione dei test*, attività nella quale vengono eseguiti i casi di test e registrato l'esito.

6.2 Unit Test

Con Unit Test si intendono quei test che vengono effettuati dagli sviluppatori su piccoli moduli di codice, dove per piccoli moduli si intende il minimo componente di un programma dotato di funzionamento autonomo. Questi possono essere eseguiti anche sulla macchina dello sviluppatore, in modalità stand-alone. Le eventuali anomalie vengono individuate e corrette direttamente dallo sviluppatore in locale.

Al fine di semplificare il lavoro di scrittura degli Unit Test è necessario rispettare certi criteri di scrittura del codice:

- *Solid Principles*: La parola è un acronimo che serve a ricordare tali principi (Single responsibility, Open-closed, Liskov substitution, Interface segregation, Dependency inversion);
- *Dependency Injection*: è un design pattern il cui scopo è quello di semplificare lo sviluppo e migliorare la testabilità di software di grandi dimensioni, andando a dichiarare le dipendenze di cui un componente necessita.
- *Inversion of control*: pattern per cui un componente di livello applicativo riceve il controllo da un componente appartenente a un libreria riusabile, in modo da rendere le singole componenti software il più indipendenti e orientate ai microservizi possibili.

Lo scopo dello Unit Test è quello di verificare il corretto funzionamento di parti di programma permettendo così una precoce individuazione dei bug.

In azienda gli Unit Test sono eseguiti sempre in locale, attraverso l'utilizzo in Python della libreria *unittest*. Per tale framework, la directory radice del progetto viene usata per l'individuazione dei test. Questo percorso può essere modificato nella scheda Test in base ai valori specificati dall'utente. Andando a creare il codice `PythonSettings.jsfile`, all'interno della cartella "Impostazioni" locale, bisognerà aggiungere il campo `TestFramework` al file di impostazioni *json* e impostarlo su *unittest*, come mostrato nel Listato 6.1.

```
{
"TestFramework": "unittest",
"UnitTestRootDirectory": "testing",
"UnitTestPattern": "test_*.py"
}
```

Listato 6.1. Codice per impostare gli Unit test.

Successivamente, selezionando il framework all'interno del proprio IDE, viene creato automaticamente un file `test.py` con codice che importa il modulo standard, mostrato nel Listato 6.2.

```
{
import unittest

class Test_test1(unittest.TestCase):
    def test_A(self):
        self.fail("Not implemented")

if __name__ == '__main__':
    unittest.main()
```

Listato 6.2. Classe di test standard per gli Unit test.

6.3 Ambiente di Application Test

In ambiente di Application Test, come mostrato nella Figura 6.1, rientrano i due archetipi *DevOps Basic* e *Intermediate*, già trattati nella Sezione 2.3.3.

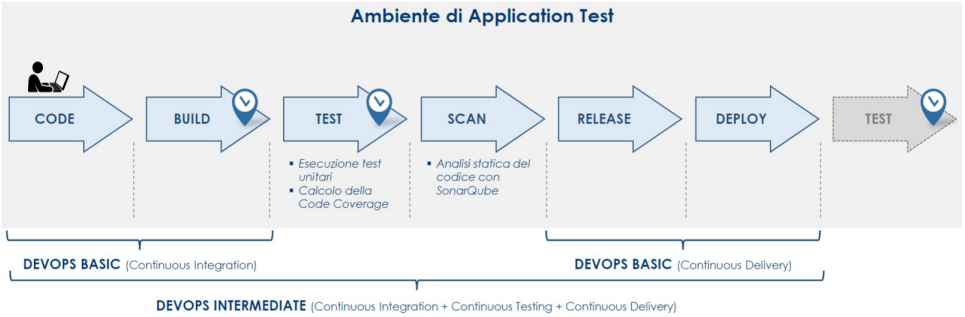

Figura 6.1. Ambiente di Application Test

In particolare, in ottica di *Continous Testing*, i tre principali checkpoint utilizzati all'interno dell'azienda sono i seguenti:

- esecuzione degli Unit Test, visti nella sezione precedente;
- calcolo della code coverage;
- analisi statica del codice con SonarQube;

6.3.1 Quality test Sonarqube

SonarQube, strumento presentato nella Sezione 2.4.7, permette di attuare dei *Quality Test* sul codice in esame. Tramite i controlli derivati da queste analisi, è possibile analizzare l'aderenza del codice sorgente, presente su *GIT*, a determinate regole di buona programmazione.

SonarQube si basa su un set di regole specifiche per diversi linguaggi, ciascuna delle quali in grado di identificare violazioni rispetto a diversi pattern di programmazione. Le regole vengono classificate in base a vari attributi, ovvero:

- *severity* che determina l'importanza della violazione all'interno del modello di qualità che si sta definendo. Le severity, in ordine decrescente di importanza, sono cinque:
  - *blocker*: queste issue potrebbero rendere instabile l'applicazione in produzione;
  - *critical*: queste issue potrebbero portare ad un comportamento imprevisto in produzione senza impatti sull'integrità dell'intera applicazione;
  - *major*: queste issue potrebbero avere un sostanziale impatto sulla produttività;
  - *minor*: queste issue potrebbero avere un impatto minore sulla produttività.
- *info*, cioè le issue non ancora definite in modo formale nel modello di qualità;
- *type*, che identifica la tipologia di rischio che la regola va ad indirizzare;

- *bug*, cioè le regole che indirizzano ad aspetti di robustezza;
- *vulnerabilities*, regole che indirizzano ad aspetti di sicurezza;
- *code smell*, regole che indirizzano ad aspetti di manutenibilità.

Sulla base di tali attributi vengono derivati tre indicatori in grado di fornire insieme un informazione sintetica e attendibile della qualità dell'applicazione analizzata:

- *Reliability Rating*: è l'indicatore che rappresenta il valore della qualità dell'applicazione da un punto di vista della robustezza. Il rating viene fatto sulla base del numero delle violazioni di regole di *Type* = *Bugs* e sulla corrispondente *severity*: si avrà che il codice è di classe A se ha 0 vulnerabilità, fino ad arrivare alla classe E se c'è almeno una vulnerabilità con *severity* di tipo *blocker*.
- *Security Rating*: è l'indicatore che rappresenta il valore di qualità dell'applicazione da un punto di vista della sicurezza. Il rating viene fatto sulla base del numero di violazioni di regole di *Type* = *Vulnerabilities* e sulla corrispondente *severity*: si avrà anche qui che il codice è di classe A se ha 0 vulnerabilità, fino ad arrivare alla classe E se c'è almeno una vulnerabilità con *severity* di tipo *blocker*.
- *Maintainability Rating*: è l'indicatore che rappresenta il valore di qualità dell'applicazione da un punto di vista della Manutenibilità. Il rating viene fatto sulla base del numero di violazioni di regole con *Type* = *Maintainability* e sulla somma dei Technical Debt di ciascuna (per debito tecnico si intende il tempo stimato di fix di una violazione), dividendo per il tempo stimato di sviluppo: si avrà che il codice è di classe A se ha un rapporto tra 0% e 5%, fino ad arrivare alla classe E se c'è un rapporto superiore al 50%.

Per quanto riguarda il motore di calo operatività, i test di code quality sono attualmente tutti superati, mostrando una ottima aderenza del codice sorgente con le regole aziendali di buona programmazione, come si nota dalla Figura 6.2.

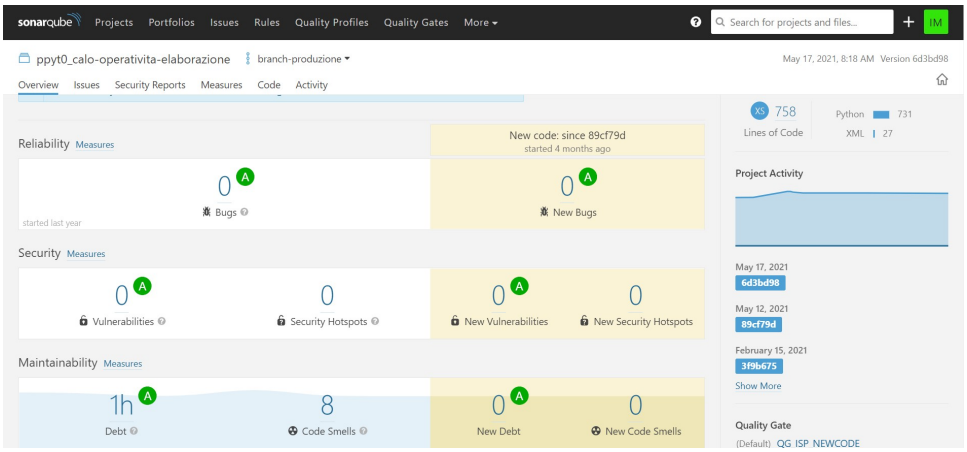

Figura 6.2. Code Quality per il modello calo operatività

Come già visto nella Sezione 2.4.7, i *Quality Gate* forniscono un indicazione OK/-KO per la promozione delle applicazioni verso gli ambienti di System. Componenti con il Quality Gate "failed" non potranno essere promosse in System.

Inoltre, per quanto riguarda la *gestione degli issues*, introdotta nella Sezione 2.4.7, l'unico caso di categoria critical che è stato risolto nel motore di calo operatività a seguito del lancio è il seguente riportato in Figura 6.3. Tale issue suggeriva come, all'interno del file `job_manager.py`, si potesse ridurre la complessità computazionale attraverso un annidamento differente.

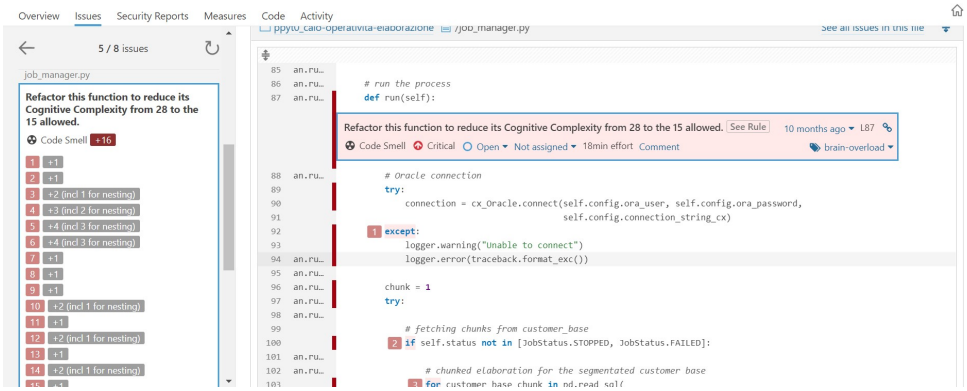

Figura 6.3. Esempio issue di grado critical in SonarQube.

6.3.2 Calcolo della Code Coverage

Il calcolo della Code Coverage è relativo alla percentuale delle linee di codice di un progetto che vengono eseguite durante gli Unit Test, data quindi dal numero di righe di codice effettivamente coperte dagli Unit Test diviso il numero totale di righe che compongono l'intero codice in esame.

All'interno dell'azienda l'obiettivo è raggiungere il 100% di code coverage in modo da poter individuare più facilmente tutti i bug. Si può infatti affermare che se si ha un code coverage del 10%, sicuramente ci troviamo nella situazione di Unit Testing pressochè nullo. L'obiettivo della Code Coverage, quindi, è quello di capire quanto del codice (metodo, classe, ecc.) in analisi è stato testato per rilevare la presenza di eventuali bug.

In Figura 6.4 è mostrata la code coverage di alcune delle prime run del modello calo operatività.

6.4 Ambiente di System Test

In ambiente di System Test, come mostrato nella Figura 6.5, rientrano i due archetipi *DevOps Intermediate* e *Full*, già introdotti nella Sezione 2.3.3. In particolare, in ottica di *Continous Testing*, i due principali checkpoint utilizzati all'interno dell'azienda sono i seguenti:

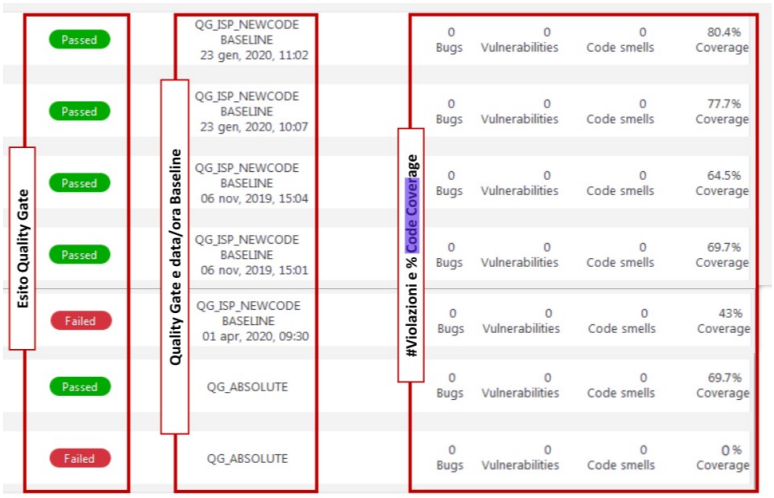

Figura 6.4. Code Coverage del modello calo operatività in SonarQube.

- Smoke Test;
- Test di non regressione automatici.

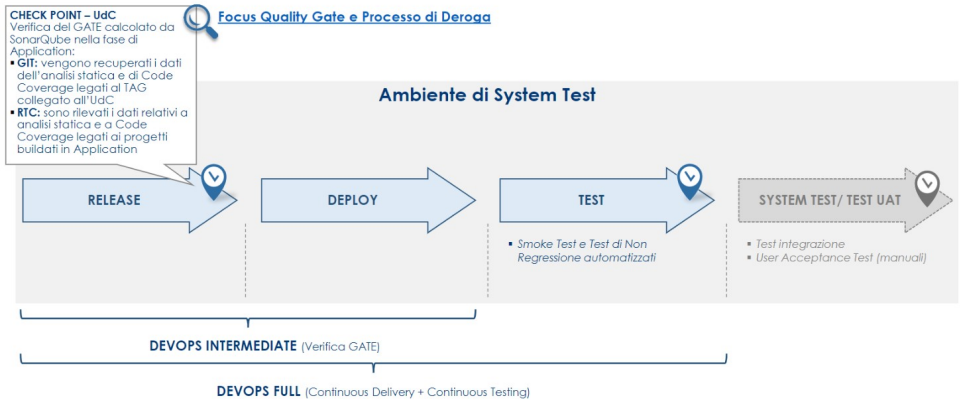

Figura 6.5. Ambiente di System Test

6.4.1 Smoke Test

Gli Smoke test vengono utilizzati per verificare che, a fronte di interventi evolutivi sull'acronimo *PPYT0*, le funzionalità principali dello stesso continuino a funzionare correttamente. Sono quindi un sottoinsieme di tutti i test case definiti e pianificati che coprono la funzionalità principali di un sistema, per accertarsi che le loro funzioni più cruciali funzionino, ma senza preoccuparsi dei dettagli.

Il Test set contiene tutti i test di non regressione di minima (o Smoke test) da eseguire per verificare le funzionalità core del progetto e la stabilità del sistema al passaggio in System Test.

Lo strumento utilizzato all'interno dell'azienda per effettuare Smoke Test è *ALM*. Per poter iniziare a salvare gli Smoke test su *ALM*, è necessario avere su quest'ultimo un'alberatura predisposta l'acronimo in questione. Sotto la cartella *PPYT0* è necessario sia presente l'entità Release con la seguente nomenclatura: `MDC_PPYT0_TDNR_STEP000`.

Un esempio di struttura che verrà creata è riportata nella Figura 6.6.

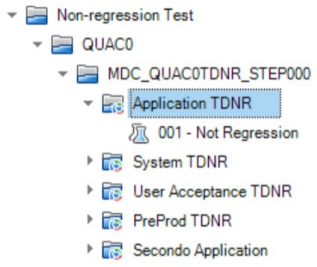

Figura 6.6. Esempio di alberatura per Smoke Test.

In particolare, è possibile configurare gli Smoke Test in modo da ricevere in automatico una mail al termine dell'esecuzione dei test, contenente l'Execution Report e link del log di Jenkins, come mostrato in Figura 6.7.

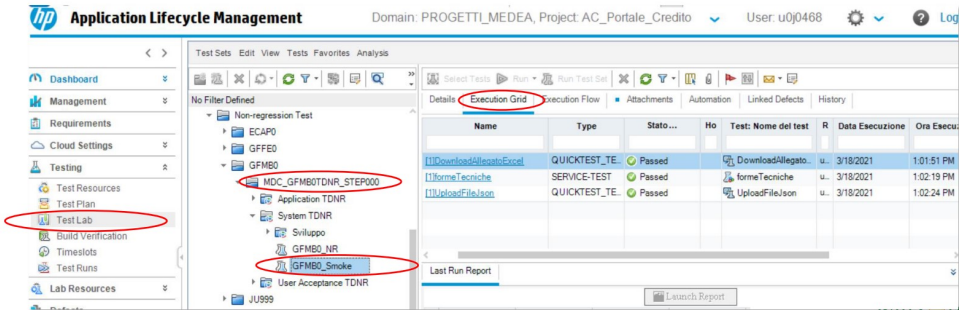

Figura 6.7. Esempio di creazione di Smoke Test automatici.

Secondo le direttive aziendali, almeno l'80% dei ricicli di ogni cambiamento attuato al codice (UDC) deve essere testato. La media "passed" di questi test deve essere pari o superiore al 70% sull'ultima esecuzione del riciclo stesso. È possibile rieseguire più volte i test all'interno dello stesso riciclo per sopperire ad eventuali problematiche di ambiente, dati o gestione dei test.

Al fine di comprendere meglio come venga calcolata la percentuale relativa alla media "passed" dei test su ogni singola Unità di cambiamento (UDC), è mostrata

in Figura 6.8 una UDC del server in esame *PPYT0*, avente una certa *"% Media ALM Passed / Target Mensile"*.

UDC	Ricicli testati su Ricicli CC	% Ricicli testati / Target Mensile	% Media ALM Passed / Target Mensile	Stato finale UDC	Toolchain	Count
O0000112233	5 su 6	83.3%/80.0%	82.2%/70.0%	OK	RTC	1

Figura 6.8. Esempio di percentuale di test superati per una data UDC.

La media percentuale dei test "passed", nel caso specifico pari all'82,2%, è ottenuta attraverso il seguente calcolo: *(Sommatoria delle % relative agli Smoke Test superati su ALM relative a tutti i ricicli della specifica UDC)* diviso *(il numero totale dei ricicli effettuati sulla specifica UDC)*.

Sia il numeratore (*sommatoria delle %*) che il denominatore (*numero totale dei ricicli*), esaminano solo il mese solare corrente. Nel caso venissero lanciati gli Smoke Test per un medesimo riciclo, per il calcolo della sommatoria sarà considerata solo la percentuale relativa all'ultimo lancio su *ALM*.

6.4.2 Test di non regressione

I test di non regressione sono delle verifiche sulle modifiche di un programma precedentemente testato, per assicurare che non siano stati introdotti o non scoperti dei difetti in aree del software che non sono state cambiate, a causa delle modifiche effettuate. Affinché sia considerato "passed" il parametro di lancio dei Test di Non Regressione, è necessario che:

- uno o più test-set siano stati creati su *ALM* e categorizzati di tipo "Non-regression";
- i test-set siano stati lanciati manualmente da *ALM* in seguito alla promozione di tutte le UDC in ambiente di System Test e prima del passaggio in Produzione di almeno una di esse; a fronte del primo lancio manuale, i test-set vengono poi eseguiti sequenzialmente in modo automatico;
- Il 70% dei test di non regressione automatici deve avere esito positivo al momento dell'uscita dal livello di System Test dell'UDC.

In particolare, è stato inserito un semaforo all'interno di *ALM* che effettua una verifica automatica delle soglie relative ai test di non regressione (TNR). Selezionando infatti un progetto nella sezione release, sarà visibile il pulsante "Verifica Gate DevOps Full" che, una volta effettuati i test, sarà necessario cliccare.

Se i test passed supereranno la soglia prevista nel periodo si aprirà il pop-up *2.1*, mostrato nella Figura 6.9, con la percentuale indicata. In questo modo si potranno promuovere le UDC collegate al progetto, sicuro che i successivi controlli saranno superati.

Nel caso non venisse superata la soglia prevista nel periodo, utilizzando la stessa funzione da *ALM*, comparirà un pop-up *2.2*, mostrato in Figura 6.9. In questo caso i test di non regressione dovranno essere ripetuti o corretti per sanare la situazione.

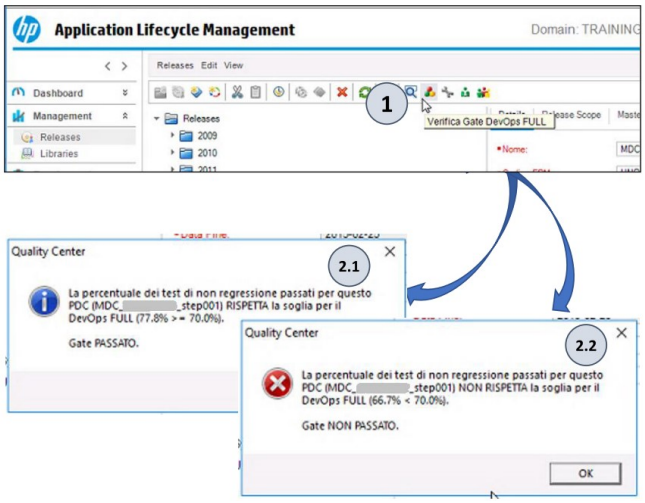

Figura 6.9. Esempio di test di non regressione in *ALM*.

6.5 Unità di cambiamento (UDC)

Relativamente al momento successivo al rilascio del progetto, si è notato come alcune parti del codice potevano essere riviste e migliorate, andando settimanalmente a fare modifiche al codice in ambiente di system test, per poi andare a fare il deploy e portare tale migliorie in produzione. Lo strumento all'interno dell'azienda che permette di far ciò sono le unità di cambiamento (UDC).

Una UDC, dal momento in cui viene richiesta dal referente aziendale di progetto al momento in cui viene attivata in Produzione, deve essere trattata tenendo presente una serie punti di consistenza, denominati "Stati dell'UDC".

Per Ciclo di vita dell'UDC si intende quindi un percorso tecnico e organizzativo che disciplina e controlla la creazione, l'avanzamento, la distribuzione e l'attivazione di una Unità di Cambiamento, qualunque sia la piattaforma tecnologica che ospiterà il software contenuto nell'UDC medesima. I passaggi di stato si distinguono in:

- promozioni, cioé avanzamenti nel ciclo di vita standard che possono essere:
  - manuali, quando ad esempio il referente applicativo dichiara 'chiusa' l'UDC con il passaggio alla fase di distribuzione in system test);
  - automatiche, quando ad esempio la macchina degli stati ha verificato che tutte le condizioni per un determinato avanzamento di stato sono state superate;
- ricicli, che sono regressioni nel ciclo di vita standard, che possono essere:
  - manuali, quando ad esempio il referente applicativo forza il riciclo di una UDC in System test per poter modificare alcuni degli oggetti o inserirne degli altri;
  - automatiche, quando ad esempio la *ALM* ha verificato che alcune delle condizioni per un determinato avanzamento di stato non sono state superate e l'UDC deve essere regredita per correggere una situazione di errore;

Durante l'attività di "Build" del software per il rilascio in ambiente di Application Test verranno eseguiti i seguenti controlli:

- Controllo sull'esito degli Unit Test e calcolo della relativa percentuale di Code Coverage raggiunta (% di copertura degli Unit Test rispetto al nuovo codice rilasciato);
- Controllo sulle regole di buona programmazione e sicurezza da parte di SonarQube.

In presenza di Unit Test in stato "failed", il motore calo operatività non può essere caricato in ambiente di application test (Sezione 6.3). L'obiettivo di questa prima verifica è bloccare la pipeline di Build Jenkins (Sezione 2.4.8), nel caso in cui sia presente uno Unit Test con esito KO, con lo scopo di informare lo sviluppatore di un problema nel codice. Se non sono presenti Unit Test falliti, la pipeline prosegue e si passa allo step successivo in cui viene eseguita l'analisi di SonarQube (Sezione 2.4.7).

6.5.1 Gate sui rilasci in System Test

Si effettua un controllo sull'assenza di violazioni bloccanti ed una percentuale di code coverage, Sezione 6.3.2, superiore a una determinata soglia. I controlli riguarderanno la qualità e sicurezza del codice oggetto di modifica, e candidato al passaggio in ambiente di System Test.

Nel caso in cui il quality gate fallisca, il software non potrà essere promosso e rilasciato in ambiente di System Test.

6.5.2 Gate sui rilasci in Produzione

In questo caso il controllo è sull'assenza di violazioni bloccanti ed una percentuale di code coverage superiore all'80%. I controlli riguarderanno la qualità e sicurezza del codice oggetto di modifica e candidato al passaggio in ambiente di Produzione.

Anche in questo caso, nel caso in cui il quality gate fallisca, il software non potrà essere promosso e rilasciato in ambiente di System Test.

Nella Figura 6.10 viene mostrato il diagramma di flusso di tutti i test discussi fin'ora, utilizzati nel motore calo operatività, per la promozione in ambiente di produzione di una UDC.

Successivamente, verrà raccolto all'interno della Change Console, strumento aziendale per la creazione delle UDC, l'esito dell'analisi Sonar relativa al software associato all'UDC per il passaggio in system test, al fine di verificare il superamento del gate.

Nella promozione dell'UDC in System Test verrà verificato che tutte le build selezionate abbiano soddisfatto i requisiti previsti dal Gate. Se questa condizione è verificata, allora l'UDC verrà attivata in System Test o Produzione. In Figura 6.11 vengono presentati i tre passaggi di stato che una UDC deve compiere prima di arrivare in produzione.

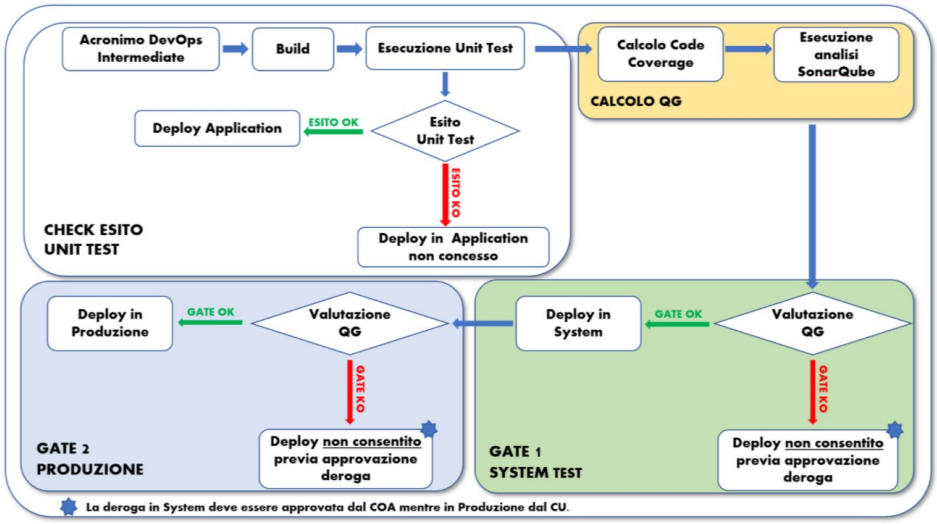

Figura 6.10. Diagramma di flusso per la promozione di un UDC

Figura 6.11. Passaggi di stato e promozioni delle UDC.

7

Discussione e Lezioni Apprese

Nel capitolo corrente verrà proposto un riepilogo di quanto svolto nel periodo di tirocinio, per poi passare a delle considerazioni di carattere personale nonchè all'esame delle best practice apprese in azienda.

7.1 Discussione

Il tirocinio presso Intesa SanPaolo ha avuto una durata complessiva di sei mesi. Durante questo periodo, circa due mesi sono stati impiegati nella formazione, sia teorica che pratica, in merito ai concetti utili allo svolgimento del lavoro in azienda. In particolare, in questa sezione ci si focalizzerà su:

- considerazioni riguardo l'esperienza di tirocinio;
- considerazioni personali.

7.1.1 Considerazioni riguardo l'esperienza di tirocinio

Gli argomenti trattati nel tirocinio sono stati principalmente i seguenti:

- il modello organizzativo delle aziende bancarie e finanziarie;
- le normative che ne regolamentano i dati;
- il modello organizzativo di Intesa SanPaolo;
- le pratiche di analisi, sviluppo e assistenza adottate dall'azienda;
- il framework aziendale "Change Console";
- l'utilizzo dei software *Virtual Studio*, *Toad for SQL* e *SAS*, e di tutti gli altri strumenti utilizzati in fase di sviluppo dall'azienda.

Tali nozioni sono state fornite da soggetti con una notevole esperienza all'interno dell'azienda e sono risultate indispensabili ai fini dell'inserimento nell'ambiente lavorativo.

I mesi successivi sono stati dedicati alle attività discusse nel corso della trattazione, collaborando con i colleghi che ne sono stati coinvolti. Gran parte di questo secondo lasso di tempo è stata spesa nel gestire i problemi quotidiani che si sono dovuti portare affrontare per portare avanti queste progettualità e nella stesura dei

requisiti, a cui hanno fatto seguito, a livello di *effort*, la gestione del team di lavoro formato da fornitori esterni per la realizzazione dei vari software. Sebbene sia stato presentato unicamente il motore calo operatività, nei vari mesi le progettualità da gestire sono state diverse, e in molti casi sono state svolte in parallelo. In esse, si è ricoperto sia un ruolo di supporto alla programmazione che un ruolo di gestione dei costi e delle tempistiche del progetto.

7.1.2 Considerazioni personali

L'argomento di tirocinio e di tesi si è rivelato molto interessante, in quanto parte di una serie di sviluppi del tutto nuovi all'interno del dipartimento. Inoltre, questa prima esperienza in un ambiente lavorativo, è stata per me valorizzante, in quanto mi ha consentito di accrescere *in primis* le conoscenze tecniche legate al mondo dell'ingegneria del software. Oltre a ciò, ho avuto modo di sperimentare cosa significa far parte di un'azienda, sia dal punto di vista lavorativo che dal punto di vista umano.

L'insieme delle attività svolte in questa esperienza formativa mi hanno permesso di comprendere meglio il mondo del lavoro, migliorare le mie conoscenze riguardo gli aspetti progettuali e affrontare i primi casi pratici di gestione di un progetto.

Accanto a questo importante bagaglio di informazioni, utilissimo dal punto di vista formativo e prettamente tecnico, ritengo importante sottolineare l'altro aspetto altrettanto fondamentale, ovvero la possibilità di crescita a livello personale. Relazionarsi con altri colleghi nel mondo del lavoro, venire a contatto con i meccanismi e le norme che regolano la vita aziendale e individuare gli aspetti applicativi del processo produttivo sono, certamente, elementi di grande rilevanza che, nel corso di questa esperienza, ho imparato a valorizzare e, successivamente, a mettere in pratica.

7.2 Lezioni Apprese

Gli aspetti fondamentali della crescita all'interno della azienda hanno, anche, il loro riscontro in alcune best practice che ho sviluppato ed affinato all'interno dell'azienda. Tali best practice verranno analizzate nelle prossime sottosezioni.

7.2.1 Testing del software

All'interno del ciclo di vita di un progetto una delle best practice affinate in azienda è il test continuo del software rilasciato, con l'obiettivo di raggiungere il massimo livello di qualità dello stesso. Il fattore "qualità" è inversamente proporzionale alla quantità di bug o difetti presenti nel prodotto software. Maggiori sono i bug, più bassa sarà la qualità, e viceversa.

Pertanto, l'obiettivo dei vari test effettuati attraverso delle unità di cambiamento (UDC), anche dopo il rilascio in produzione del motore, come spiegato nella Sezione 6.5, riguarda l'eliminazione metodica di bug e difetti. Dopo tutto, i test vengono eseguiti con l'obiettivo di fornire un prodotto software della massima qualità all'utente al fine di soddisfare le sue esigenze e aspettative.

7.2.2 Scalabilità

Progettare un codice con la scalabilità è un requisito fondamentale per l'intera direzione dei sistemi informativi (DSI), dato che, molto spesso, nei progetti aziendali di questo genere, vengono sempre aggiunte nuove caratteristiche o vengono modificate le cadenze per le quali si vuole ricevere il dato. Pertanto, la facilità di aggiungere nuove caratteristiche ad un codice base del software diventa una best pratice fondamentale nella scrittura del software, sia per un aspetto di tempistiche per eventuali evolutive che per l'effort a livello di costo.

7.2.3 Stato Avanzamento Lavori (SAL)

Una lezione appresa nella gestione dei progetti è, senza dubbio, l'utilizzo del SAL come riunione periodica, stabilita con l'obiettivo di garantire e verificare l'avanzamento di un progetto rispetto ai propri obiettivi. I temi trattati solitamente sono:

- stato attuale del progetto (avanzamento rispetto agli obiettivi);
- rispetto dei tempi (in relazione ai deliverable rilasciati);
- analisi delle criticità (problemi e azioni correttive, rischi e azioni di mitigazione);
- approvazione delle nuove azioni e prossime attività.

7.2.4 Reverse Engineering

Molto spesso in azienda si è portati a fare un procedimento di ragionamento inverso, ovvero: un processo di scoperta dei principi di funzionamento di un sistema software attraverso l'analisi del suo funzionamento e dei risultati che esso produce. Molto spesso, infatti, bisogna partire dal risultato ottenuto da un codice di cui non si ha accesso, sviluppato da un'altro dipartimento. Partendo dall'analisi dei log, si procede con un processo di "Reverse Engineering" per scoprire dov'è nascosto l'errore. Si inizia, perciò, con il prodotto finale e si lavora nella direzione opposta per arrivare alle specifiche base di progettazione del prodotto. Durante il processo, vengono scoperte informazioni vitali sui metodi di produzione del codice stesso.

8

Conclusioni

Nell'elaborato corrente è stato presentato lo sviluppo delle funzionalità relative al motore del calo di operatività, di proprietà di Intesa SanPaolo.

Nella prima parte si è esaminata una panoramica sulla direzione sistemi informativi (DSI) del Gruppo Intesa e su come si colloca la seguente tesi in questo contesto. Sono stati, inoltre, descritti l'importanza dei dati e il ruolo nella banca all'interno del progetto New Banking Platform.

In seguito, si è parlato delle tecnologie utilizzate e delle architetture dati alla base dei progetti sviluppati.

Terminata la parte introduttiva, si è passati all'analisi di quello che costituisce l'argomento centrale del lavoro di tesi, ovvero la realizzazione di un sistema per il monitoring dell'operatività delle aziende clienti di una banca. Si è partiti dalla stesura dei requisiti, effettuata secondo la prassi aziendale, per poi proseguire con la progettazione del modulo di data preparation in *SAS* e del modello vero e proprio in *Python*, grazie anche all'ausilio dei diagrammi di flusso. È stata, quindi, descritta l'implementazione di quanto progettato, basata sull'utilizzo del questi due strumenti.

Infine, dopo aver illustrato i meccanismi di test e rilascio adottati all'interno di Intesa SanPaolo, sono state discusse le unità di cambiamento (UDC) che settimanalmente o mensilmente si portano in produzione per migliorare il codice.

Il lavoro esaminato in questa sede rappresenta una piccola parte di un processo ben più ampio, che ha come obiettivo la digitalizzazione dei dati della banca. Ciò rappresenta un progetto ambizioso, sia per il numero di strutture ed enti coinvolti che per l'ammontare degli investimenti

Tuttora il modello del calo di operatività, effettua solo analisi mensili, ma un possibile scenario che si sta valutando è quello di rendere tale analisi settimanale in modo da essere più efficienti nell'analisi dei dati inviati con cadenza maggiore ai reparti business.

In seconda istanza, dal momento che attualmente il perimetro è rappresentato solo da aziende clienti della banca, una possibile evoluzione potrebbe riguardare l'estensione del perimetro di utenza anche a tutti i clienti della banca, fornendo ai reparti business un quadro più completo sull' operatività dei clienti. Già in passato, difatti, per progetti simili si è seguito un iter di evolutive analogo. Lo step finale

dei motori è quello di alimentare, successivamente, la parte back-end delle varie applicazioni all'interno dell'App mobile di Intesa San Paolo.

Riferimenti bibliografici

1. ISO 8402:1994. *Quality management and quality assurance.* Quality management systems, (March 2004).
2. Apache. `https://hadoop.apache.org/`.
3. Amazon Web Services (AWS). `https://aws.amazon.com/it/microservices/`.
4. Ben Azvine. *"Real time business intelligence for the adaptive enterprise".* The 8th IEEE International Conference on and Enterprise Computing, 2006.
5. BCE. *"Principi per un'efficace aggregazione e reportistica dei dati di rischio".* Banca Centrale Europea, 2013.
6. Vercellis Carlo. *Business intelligence: data mining and optimization for decision making.* John Wiley & Sons, 2011.
7. Wells D. Business analytics – Getting the point. `http://b-eyenetwork.com/view/7133`, 10 Settembre 2019.
8. Banca d'Italia. Circolare n. 285 del 17 dicembre 2013. `https://www.bancaditalia.it/compiti/vigilanza/normativa/archivio-norme/circolari/c285/`, 2013.
9. James Dixon. Pentaho, Hadoop, and Data Lakes. `https://jamesdixon.wordpress.com/2010/10/14/pentaho-hadoop-and-data-lakes/`, 2010.
10. Vincenzo Santucci e Sandro Spadaro. *"Docker".* Apogeo, 2020.
11. P. Zikopoulos C. Eaton. *"Understanding big data: Analytics for enterprise class hadoop and streaming data".* Mc Graw Hill, 2015.
12. Parlamento europeo e del Consiglio. Regolamento Generale sulla Protezione dei Dati. `https://www.garanteprivacy.it/documents/10160/0/Regolamento+UE+2016+679.+Arricchito+con+riferimenti+ai+Considerando+Aggiornato+alle+rettifiche+pubblicate+sulla+Gazzetta+Ufficiale++dell%27Unione+europea+127+del+23+maggio+2018`, 2008. [Online; accessed 19-July-2008].
13. Provost Foster and Tom Fawcett. *Data science and its relationship to big data and datadriven decision making.* Big data 1.1, (2013).
14. Emily Freeman. *"DevOps For Dummies".* Wiley, 2019.
15. Alex Gorelik. *The Enterprise Big Data Lake: Delivering on the Promise of Hadoop and Data Science in the Enterprise.* O'Reilly, 2016.
16. Paolo Gualtieri. *Teoria dell'intermediazione finanziaria - IV edizione.* Egea, 2018.
17. SAS Enterprice Guide. `https://www.sas.com/it_it/home.html`.
18. Red Hat. `https://www.redhat.com/it/technologies/cloud-computing/openshift`.
19. IBM. `https://www.ibm.com/us-en/marketplace/datastage`.
20. Jenkins. `https://www.jenkins.io/`.
21. Steve Lohr. How Big Data became So Big. `https://www.nytimes.com/2012/08/12/business/how-big-data-became-so-big-unboxed.html`, 2012.

22. Cagliero Luca and Valentin Popov. *"Sviluppo di un sistema a supporto dei processi di validazione dei finanziamenti basato su tecniche di classificazione."*. Blue Reply, 2018.
23. Capdevila Prat. Marc. *Big data: data analysis and decision making. BS thesis.* Universitat Politècnica de Catalunya, 2017.
24. Nature. *Community cleverness required. Nature 455.* Nature, 03 September 2008.
25. Intesa San Paolo. Il Gruppo Intesa San Paolo. `https://group.intesasanpaolo.com/it/chi-siamo`, 2021.
26. Python. `https://www.python.org/`.
27. Kimbal Ralph and Margy Ross. *The data warehouse toolkit: the complete guide to dimensional modeling.* John Wiley & Sons, 2011.
28. Fabio Rosellini. *"Calcolo di indicatori di performance aziendale in contesto Big Data."*. Tesi UniBo, 2018.
29. Pasopuleti Pradeep Purra Beulah Salome. *Data Lake Development with Big Data.* Packt Pub Ltd, 26 novembre 2015.
30. Paim Fá Rilston Silva and Jaelson Freire Brelaz de Castro. *DWARF: An approach for requirements definition and management of data warehouse systems.* 11th IEEE International, 2003.
31. Teradata. `https://it.teradata.com/`.
32. Teradata. `https://www.datanami.com/2018/10/17/inside-teradatas-audacious-plan-to-consolidateanalytics/`.
33. Architettura Teradata. `https://www.tutorialspoint.com/teradata/teradata_architecture.htm`.
34. Damiano Marinelli Vincenzo Apa Giovanni Caruso Piercarlo Felice˙ *La responsabilità amministrativa degli enti. Guida operativa al D.Lgs. n. 231/2001.* Maggioli Editore, 2019.
35. Grossmann Wilfried and Stefanie Rinderle-Ma. *Fundamentals of business intelligence.* Springer, 2015.