POLITECNICO DI TORINO

DIPARTIMENTO DI AUTOMATICA E INFORMATICA

Corso di Laurea Magistrale in Ingegneria Informatica

Tesi di Laurea Magistrale

Ottimizzazione di un sistema di raccomandazione e profilazione clienti

Relatore
Prof. Paolo Garza

Candidato
Giuseppe Giametta

Torino Dicembre 2019

I hereby declare that, the contents and organisation of this thesis constitute my own original work and does not compromise in any way the rights of third parties, including those relating to the security of personal data.

. . . . . .

Giuseppe Giametta
Turin, October 09, 2019

Sommario

Negli ultimi anni le offerte personalizzate sul cliente sono diventate un aspetto cruciale, cui un'azienda deve tener conto al fine di mantenere un buon livello di competitività. Questa esigenza ha contribuito alla formazione di un panorama variegato di sistemi di profilazione atti all'automatizzazione del processo di raccomandazione. È in questo contesto che la presente tesi si pone come obiettivo quello di fornire una descrizione accurata del processo di ottimizzazione di un sistema attualmente in funzione presso l'azienda Reply, che si occupa di presentare al cliente in ambito bancario un'offerta personalizzata attraverso un'opportuna profilazione.

Ringraziamenti

Desidero ringraziare il mio relatore, il Professor Paolo Garza, uno dei professori più preparati e corretti con cui abbia avuto il piacere di collaborare.

Dedico questa tesi alla mia famiglia che mi è sempre stata accanto nonostante le difficoltà e i sacrifici, e che hanno saputo darmi sostegno per raggiungere questo traguardo. Ringrazio Giuseppe Cannizzaro e la sua famiglia, che costituiscono a tutti gli effetti la mia seconda famiglia.

Ringrazio Donato che con la sua bravura e caparbietà mi ha guidato durante questo percorso e ringrazio anche i ragazzi di Reply che con la loro simpatia e disponibilità hanno saputo riempire le mie giornate.

Un grazie sincero a quei bravi ragazzi di "Quelli della Montagna" con cui ho condiviso gran parte se non tutta la mia esperienza a Torino. Grazie Ragazzi per avermi sostenuto in ogni momento!

Ringrazio anche i colleghi di Palermo con cui non ho mai perso i rapporti e continuo tuttora a condividere momenti belli.

Concludo ringraziando anche Casa Crudo dove ho trascorso gran parte del mio tempo, diventando a tutti gli effetti il settimo coinquilino.

Grazie a tutti!

Indice

Elenco delle tabelle

Elenco delle figure

Capitolo 1

Introduzione

Nella società odierna, la diffusione pervasiva delle tecnologie offre la possibilità di creare sistemi molto complessi che si adattino al meglio alle esigenze delle aziende. Si ricerca costantemente una gestione modulare dei vari applicativi che possa facilitarne lo sviluppo di ogni componente nelle varie fasi del progetto. Di conseguenza, è importante stare al passo con i tempi fornendo sempre il miglior prodotto nel minor tempo possibile. In ambito bancario, riscontrano particolare richiesta i sistemi per la raccomandazione di offerte mirate alla vendita dei prodotti, in accordo con le esigenze dei clienti. I sistemi di raccomandazione si occupano di fornire al cliente delle offerte personalizzate per le quali potrebbe mostrare particolare interesse, sulla base di alcuni indicatori.

La presente tesi mira a fornire un sistema di raccomandazione basato sul trattamento dei dati in *real-time* e predilige una struttura modulare che consente un'implementazione facilitata di nuove specifiche di progetto. Tale raccomandazione riguarda la selezione di opportuni banner pubblicitari o la vendita di prodotti bancari, per cui il cliente potrebbe mostrare particolare interesse.

Il presente lavoro è stato commissionato dalla società di consulenza **Reply**. Questa azienda vanta di numerosi progetti in funzione. Tra questi, vi è anche un sistema di raccomandazione denominato SNAP. Quest'ultimo è stato richiesto da **Cedacri**, una delle maggiori aziende italiane specializzate in servizi informatici per il settore bancario.

In passato, Cedacri ha richiesto un sistema di raccomandazione basato su eventi in ambito bancario. Gli eventi sono rappresentati dalle azioni che i

clienti possono compiere tramite il portale Home Banking che ogni istituto bancario mette a disposizione. Il sistema sfrutta una combinazione tra gli eventi e le informazioni contenute nei database per generare una lista di Banner oppure una lista di prodotti acquistabili dal cliente, entrambi identificati tramite un ID, ritenuti apprezzabili dal cliente. Grazie a questo lavoro di ricerca è stato possibile analizzare il sistema attuale e sostituire le sue componenti con nuove tecnologie che portano molteplici vantaggi.

Capitolo 2

Architettura attuale: SNAP

2.1 Descrizione Generale

Il sistema SNAP è costituito da differenti unità logiche che insieme formano il sistema di raccomandazione che fornisce offerte commerciali mirate ad un cliente. In particolare, si utilizza un Load Balancer che si occupa di ricevere richieste HTTP, inviate dall'app dei clienti che usufruiscono del servizio di Home Banking, e smistarle nei nodi meno carichi del Front-end. Il funzionamento di questo meccanismo è trasparente al sistema di raccomandazione a cui va il compito di processare le richieste in arrivo e fornire delle risposte. Le richieste sono in formato JSON e contengono alcuni parametri che identificano univocamente il cliente, la banca e l'evento scatenante. L'identificativo del cliente è il *Numero Direzione Generale (ndg)*, mentre la banca viene identificata dal codice *Associazione Bancaria Italiana (abi)*. In Figura 2.1 possiamo vedere due esempi di richieste.

```
{
   "sessionID": "TEST_VP0000099398",
   "ndg": "000000000123",
   "abi": "22234",
   "interactionPoint": "IP_HB_HP1"
}
```


(a) *Richiesta per la lista dei Banner*


```
{
   "sessionID": "TEST000",
   "ndg": "000000000080",
   "abi": "03127",
   "feedbackName": "Visto",
   "parameters": {"bannerID":
   "69077B32FECC002EE0530A6408DEC179"}
}
```


(b) *Richiesta di invio Feedback*

Figura 2.1: Esempio di richieste in formato JSON

Il sistema prevede quattro server fisici sulla quale sono implementate le varie unità logiche necessarie per soddisfare la fase di processing delle richieste in arrivo. Queste unità sono:

- **Front-End (FE)**.
- **Oracle Event Processing (OEP)**.
- **Oracle Real-time Decisions (RTD)**.

Nella Figura 2.2 è possibile vedere una rappresentazione grafica delle varie unità.

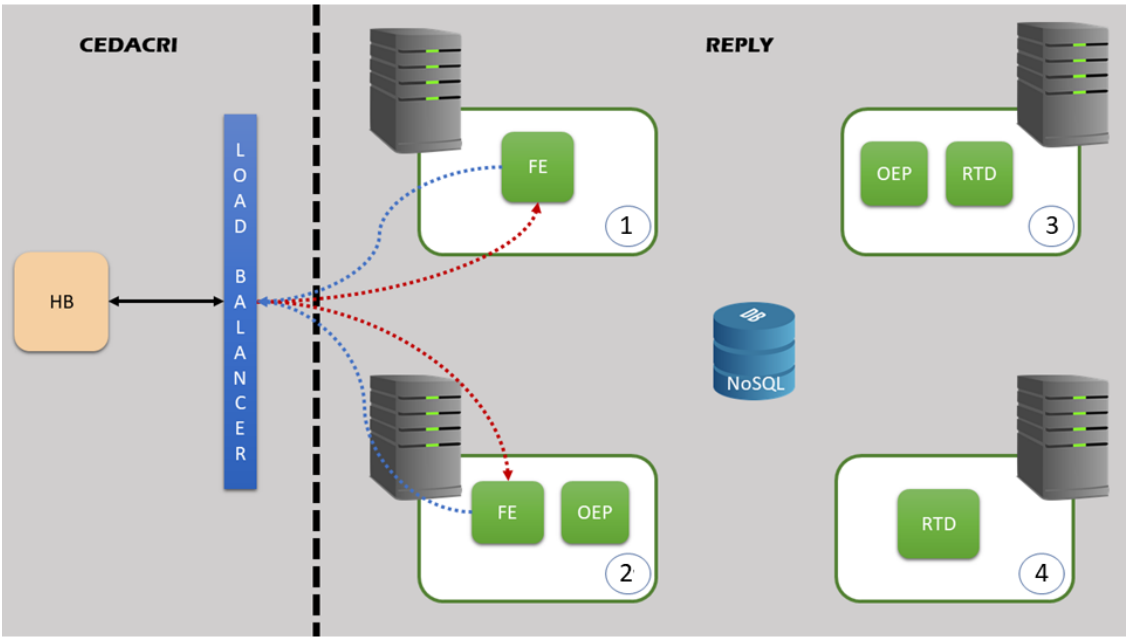


Figura 2.2: Architettura generale di SNAP

Come si evince dalla Figura 2.2, si è scelto di replicare ogni componente nei vari nodi per differenti motivi: scalabilità e tolleranza ai guasti. Il sistema risulta prestante per via della presenza del load balancer che smisterà le richieste sui nodi meno carichi migliorando i tempi di risposta e, inoltre, il servizio verrà sicuramente garantito nel momento in cui un nodo smette di funzionare. In aggiunta, sarà possibile effettuare più facilmente operazioni di manutenzione senza dover necessariamente interrompere il servizio. Per quanto concerne il suo funzionamento, SNAP mette a disposizione dei servizi RESTful, che permettono di fornire a un particolare cliente di una banca una proposta commerciale mirata in base all'evento richiesto. Nel momento in cui arriva una richiesta viene innanzitutto analizzato l'evento, poi in base a quest'ultimo ci sono diversi scenari che si possono attivare.

Uno scenario non è altro che un contesto ben definito che si può attivare al verificarsi di determinate condizioni o al soddisfacimento di ben precise regole. Ad ogni evento (Login, Bonifico, ...) sono legati più scenari che possono

essere ordinati per priorità da ciascuna banca. In questo modo il sistema, alla ricezione di una richiesta da parte di uno specifico cliente e tenendo conto di tale ordinamento, selezionerà il primo scenario per cui risultano soddisfatte tutte le condizioni o regole ad esso legate. Una volta selezionato lo scenario, sulla base di ulteriori regole, come ad esempio indicatori(kpi) precedentemente calcolati, interazioni precedenti dell'utente con le offerte a lui proposte (Feedback sui Banner) oppure all'appartenenza a particolari Target List, viene selezionata l'offerta ritenuta più opportuna. Di seguito sono spiegate le singole componenti applicative facenti parte del sistema, evidenziandone gli aspetti concettuali.

2.1.1 Front-End

L'applicazione del Front-end è chiamata impropriamente così dato che svolge a tutti gli effetti dei servizi di back-end, ma nel complesso del sistema SNAP rappresenta un layer intermedio che si interpone appunto tra il vero sistema di raccomandazione e il mondo esterno, consentendo di esporre i servizi per la ricezione delle richieste. Questa applicazione viene eseguita su un server WebLogic e risiede sulle macchine 1 e 2. Quest'ultimo non è altro che un Application Server multipiattaforma basato su Java EE e fornisce una serie di strumenti indicati per lo sviluppo di applicazioni critiche che devono gestire migliaia di transazioni. Nel dettaglio, è utilizzato Jersey per la creazione di un'applicazione RESTful basata su vari servizi web, ciascuno raggiungibile tramite una URI e dedicato a soddisfare una particolare richiesta.

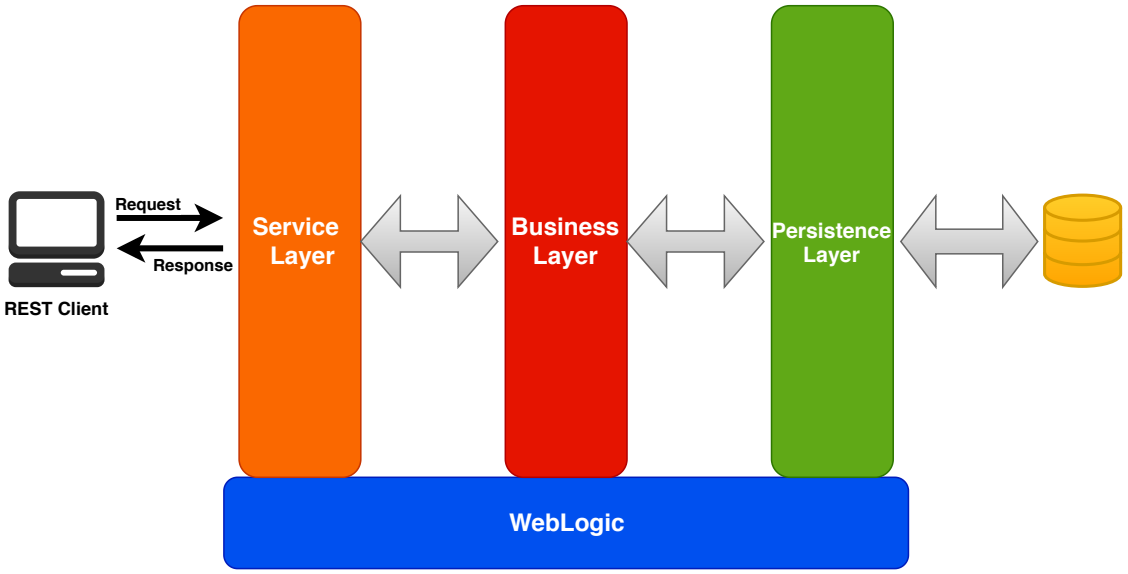


Figura 2.3: Struttura Applicazione RESTful

In Figura 2.3, è possibile visualizzare uno schema rappresentativo di come funziona un'applicazione RESTful. L'intera applicazione si appoggia su un'infrastruttura, che nel caso preso in esame è un server WebLogic. Su questa infrastruttura si va a definire un'architettura basata su più livelli che comunicano tra loro:

- Il primo livello viene detto di *servizio* o di *presentazione* ed è in questo livello che viene usato il framework Jersey. Il suo scopo è quello di fornire un'interfaccia per l'accesso ai servizi web.
- La logica di ciascun servizio viene, invece, inglobata nel secondo livello, detto *business layer*
- L'ultimo livello viene chiamato *persistence layer* e sfrutta il pattern DAO, ovvero fornisce un'interfaccia di accesso ad una particolare struttura persistente (es. NoSql DB).

Ne deriva un sistema robusto, sicuro e scalabile. Questi sono alcuni dei motivi che hanno portato al mantenimento di questa struttura nel nuovo sistema.

Come già descritto in precedenza, SNAP mette a disposizione una serie di servizi utili a fornire offerte commerciali mirate, basate sul comportamento e sulle preferenze espresse dal cliente.

In particolare, sono resi disponibili i seguenti servizi:

- *StartSession.*
- *OfferService.*
- *PostEventService.*
- *PostOutcomeService.*
- *EndSession.*

Sessioni

Il sistema attuale prevede che ogni utente, prima di richiedere un'offerta, debba essere associato ad una sessione. L'utente nel momento in cui si collega su IHB richiede l'apertura di una sessione invocando il servizio attraverso la richiesta al servizio *StartSession*. La chiamata a questo servizio prevede dei parametri obbligatori come l'*abi* della banca del cliente che effettua la richiesta, l'*ndg* del cliente stesso e un identificativo di sessione che viene generato dall'applicativo di Home Banking. Su tali parametri vengono effettuati dei controlli verificando che l'identificativo dell'utente e il codice abi relativo alla banca contengano rispettivamente 12 e 5 caratteri numerici. In caso non siano soddisfatte le condizioni viene inviato un messaggio di errore. Il sistema verifica che l'identificativo di sessione non esista già, sia esso associato allo stesso utente che a uno diverso, e in caso affermativo provvede a registrarlo. Una volta completato il processo di recommendation viene eseguita la chiamata al servizio *EndSession* che si occuperà di memorizzare la chiusura della sessione referenziata dal *session_id*, salvando opportunamente tale informazione.

Feedback

SNAP non si occupa solo di profilare il cliente ma fornisce anche un meccanismo di interazione con lo stesso cliente basato su feedback. Alcuni scenari possono avere come condizione la presenza di un allarme attivo, ovvero un indicatore composto da una composizione di condizioni. Per questi scenari esiste un'interazione tra l'azione di uno scenario e l'attivazione di un allarme. Per quegli scenari che contemplano un'azione su IHB, verrà considerato il feedback "*interessato*" sul banner come un'azione intrapresa sul cliente con esito positivo, per questo verrà spento l'allarme associato. In caso contrario,

se il cliente mostra un disinteresse/non interesse per tutti i banner proposti dallo scenario, questo verrà considerato come un cliente che ha reagito all'offerta con esito negativo. Questo non comporterà lo spegnimento dell'allarme che ne ha fatto attivare lo scenario. Come riportato sopra, l'interazione di un cliente con un banner (detto feedback) viene memorizzato da SNAP che, in relazione alla tipologia di feedback, stabilisce se disattivare o meno lo scenario e l'allarme associato.
Sono contemplate quattro tipologie di feedback:

- Non interessato.
- Interessato.
- Visto.
- Ricorda dopo.

In relazione alla tipologia di feedback, si innescano tre situazioni differenti:

1. Qualora il cliente non sia interessato al banner, allora lo scenario resta attivo, ma, al successivo evento dello stesso ndg, verrà mostrato un altro banner legato allo scenario, se esistente; in questo modo SNAP presenterà diversi banner cercando di assecondare la preferenza dell'utente. Nel caso in cui lo scenario corrente prevede l'esposizione di un unico banner, al successivo evento si attiverà un nuovo scenario, di priorità minore.
2. Se il cliente è interessato ad un banner oppure non reagisce al banner per più di n visualizzazioni (con n parametrico stabilito dalla banca), lo scenario viene disattivato insieme all'eventuale allarme associato, determinando così la potenziale attivazione di altri scenari la cui priorità è minore rispetto allo scenario corrente, ma che risultano anch'essi attivabili.
3. Nel caso in cui il cliente clicchi su "*ricorda dopo*", SNAP presenterà lo stesso banner al prossimo evento dello stesso utente.

Come spiegato sopra l'interazione viene modellata attraverso i servizi PostEventService (per i banner) e *PostOutcomeService* (per i prodotti), mentre la logica degli scenari viene gestita da OEP. Vi è una sottile differenza da considerare tra i due servizi. Il primo permette l'interazione tra cliente e banner direttamente dall'IHB, mentre il secondo viene richiesto dall'operatore bancario

per suggerire al sistema le preferenze del cliente in modo da poterlo ricalibrare, andando ad attivare o disattivare degli allarmi comportamentali associati al cliente. È necessario specificare che l'operatore bancario utilizza, nel caso dei prodotti, un altro sistema chiamato FEU. Tramite questo sistema l'operatore ha accesso alle informazioni del cliente fornitegli da SNAP che includono una lista di prodotti che gli potrebbero interessare e notifica al sistema di recommendation, sotto suggerimento del cliente, l'interesse per alcuni di essi. Ciò nonostante, entrambi i servizi provvedono a registrare le preferenze dei clienti e registrarli su una struttura persistente, dove potranno essere prelevati e valutati, in modo da poter presentare dei banner coerenti con le preferenze del cliente oppure attivare un allarme comportamentale.

Offerte commerciali

La richiesta di offerte per un cliente è legata necessariamente a uno specifico evento quale *LOGIN*, *BONIFICO* o *VETRINA_PRODOTTI*. Da ogni evento è possibile sfruttare le informazioni che incorpora per poter profilare il cliente. E.g. Tramite dei ripetuti eventi *Bonifico* di trasferimento denaro da un conto ad un altro appartenente ad un altro istituto, è possibile ipotizzare che il cliente stia abbandonando la banca a favore di un nuovo conto. In questo caso il sistema proporrà delle offerte commerciali che possano convincerlo a cambiare idea. Nel momento in cui un utente effettua il login oppure invia un bonifico, dopo aver inizializzato una sessione, viene inviata una richiesta al fine di ottenere un'offerta personalizzata. L'applicazione riceve queste richieste tramite protocollo HTTP in modo da serializzarle e gestirle attraverso un servizio chiamato *OfferService*, che si occupa di verificare che le richieste contengano informazione valide.

In genere una richiesta contiene i seguenti valori:

- *Session_id*: identificativo della sessione a cui il cliente partecipa.
- *abi*: codice della banca.
- *ndg*: identificativo del cliente per quella banca.
- *interactionPoint*: codice dalla quale si può desumere il *canale di vendita* e l'*evento*.

Stabilito lo scenario, viene effettuata una chiamata alla componente RTD che si occupa di fornire la proposta migliore per il cliente. Tale proposta può

consistere in un set di banner o di prodotti soddisfacenti le regole di quello scenario.

2.1.2 Oracle Event Processing

Ciascun tipo di richiesta può essere vista come un evento e l'insieme degli *eventi* forma uno *stream*. La sua gestione è interamente a carico di OEP [1]. OEP è una piattaforma specializzata nello sviluppo di applicazioni event-driven. Gli eventi vengono letti dalle sorgenti e tradotti in un linguaggio comprensibile ad OEP attraverso strutture specilizzate chiamate ***adapters***. Come già illustrato in precedenza, le richieste vengono memorizzate in delle code JMS dove vengono prelevate da OEP. Nel dettaglio, sono stati utilizzati dei JMS adapters per ricevere ed inviare eventi all'interno di OEP. In questo modo vengono creati gli eventi. Da qui in poi, gli eventi vengono gestiti da un ***nodo processore*** che applica una sequenza di meccanismi atti a selezionare lo scenario più adatto al cliente sulla base delle sue preferenze e delle priorità definite dai singoli istituti. Il processore sfrutta un particolare linguaggio chiamato CQL che costituisce un'estensione del classico linguaggio SQL. Con esso è possibile applicare delle regole agli eventi per una corretta gestione degli stessi. Inoltre sfrutta degli handle a strutture persistenti per poter arricchire questi dati e filtrarli. Compito principale del nodo processore è, come già detto quello di selezionare uno scenario. La scelta dello scenario si basa su una serie di regole standard decise dalle banche e da alcuni allarmi comportamentali che devono essere attivi per il cliente preso in considerazione. Le regole possono anche essere legate a specifici attributi della richiesta, come ad esempio, per l'evento BONIFICO si potrebbe valutare l'importo del bonifico, il beneficiario o la causale per verificare lo scopo del bonifico. Si può controllare se il bonifico è riferito al pagamento di un affitto verificando la causale oppure si può analizzare il beneficiario del bonifico unito con l'importo dello stesso per identificare il rischio di abbandono dall'istituto. Una volta che lo scenario è stato individuato, si procede con il suo inserimento in un'altra coda JMS, dalla quale verrà prelevato dal Front-end. Di seguito una rappresentazione grafica dell'interazione di OEP e il Front-end.

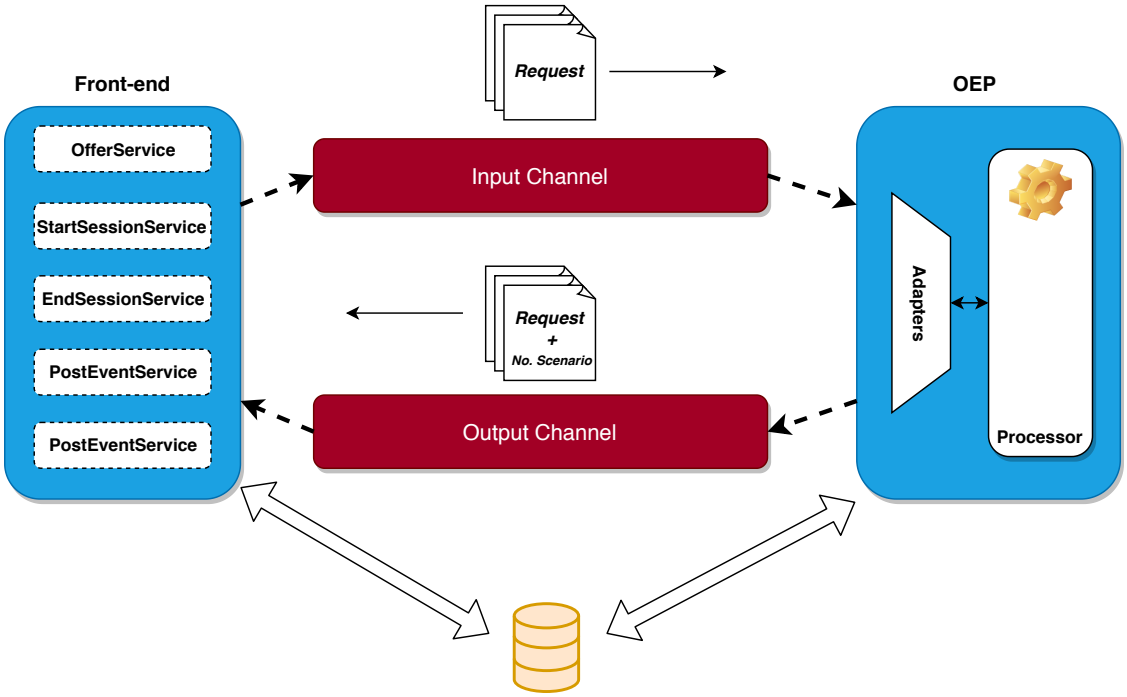


Figura 2.4: Interazione tra FE e OEP

Scenari e Allarmi

Uno **scenario** SNAP rappresenta un determinato caso d'uso che si può verificare al soddisfacimento di specifiche condizioni relative sia all'evento che un cliente innesca su un canale (ad esempio gli eventi LOGIN, BONIFICO o VETRINA PRODOTTI su canale IHB), sia alla specifica profilazione del cliente. Esistono degli scenari la cui attivazione dipende da uno o più **allarmi comportamentali**, definiti da ogni Istituto, e altri che sono invece indipendenti dagli allarmi. Quest'ultimi sono indicatori comportamentali basati su delle condizioni. Possono essere definiti sia dagli operatori bancari oppure possono essere attivati/disattivati dal sistema stesso in base alle azioni intraprese dall'utente. L'attivazione di uno scenario avviene quando una serie di condizioni è soddisfatta da un determinato ndg. Tra queste condizioni, in relazione al caso specifico, vi può essere anche un set di allarmi che si richiede debba essere attivo per quell'*ndg*. Qualora uno scenario sia legato a degli allarmi, affinché un evento effettuato da un *ndg* lo possa fare scattare, è necessario che questi allarmi siano attivi sullo specifico *ndg*. Nel momento in cui uno scenario viene attivato, ad esempio mediante un evento su IHB come un LOGIN o un BONIFICO, viene proposto al cliente un banner sulla pagina IHB che la banca ha

deciso di sponsorizzare. Ci sono anche eventi, come VETRINA_PRODOTTI che determinano, per il corrispettivo scenario, l'esibizione al cliente di una lista prodotti acquistabili. Le condizioni di attivazione di un particolare scenario verranno verificate da SNAP ogni qual volta si verificano determinati eventi: ad es., qualora uno scenario sia attivabile al tempo τ_1, poiché soddisfatte tutte le sue condizioni di attivazione (tra le quali anche il soddisfacimento di un allarme da parte di quell'ndg) allora, al verificarsi di un determinato evento al tempo τ_1, questo scenario si innescherà per l'ndg in esame. Tuttavia, nel caso in cui, da lato front-end, al tempo τ_2, un operatore dovesse decidere di disattivare quell'allarme per l'*ndg* in questione, SNAP prenderà atto che tale scenario non è più attivabile e dunque, a partire da quel momento il medesimo scenario non scatterà al presentarsi di eventi determinati da quell'*ndg*.

2.1.3 Oracle Real-Time Decisions

Una volta che lo scenario è noto si procede con la selezione dei banner inerenti. SNAP prevede l'uso di RTD [2] per il supporto alle decisioni. Questa piattaforma fornisce una serie di strumenti che, uniti a delle business rule, permettono di creare applicazioni decisionali. In generale, questa unità è costituita da varie componenti che permettono di definire una "**inline service**", costituita da metadati che descrivono come l'applicazione gestisce i processi in real-time.

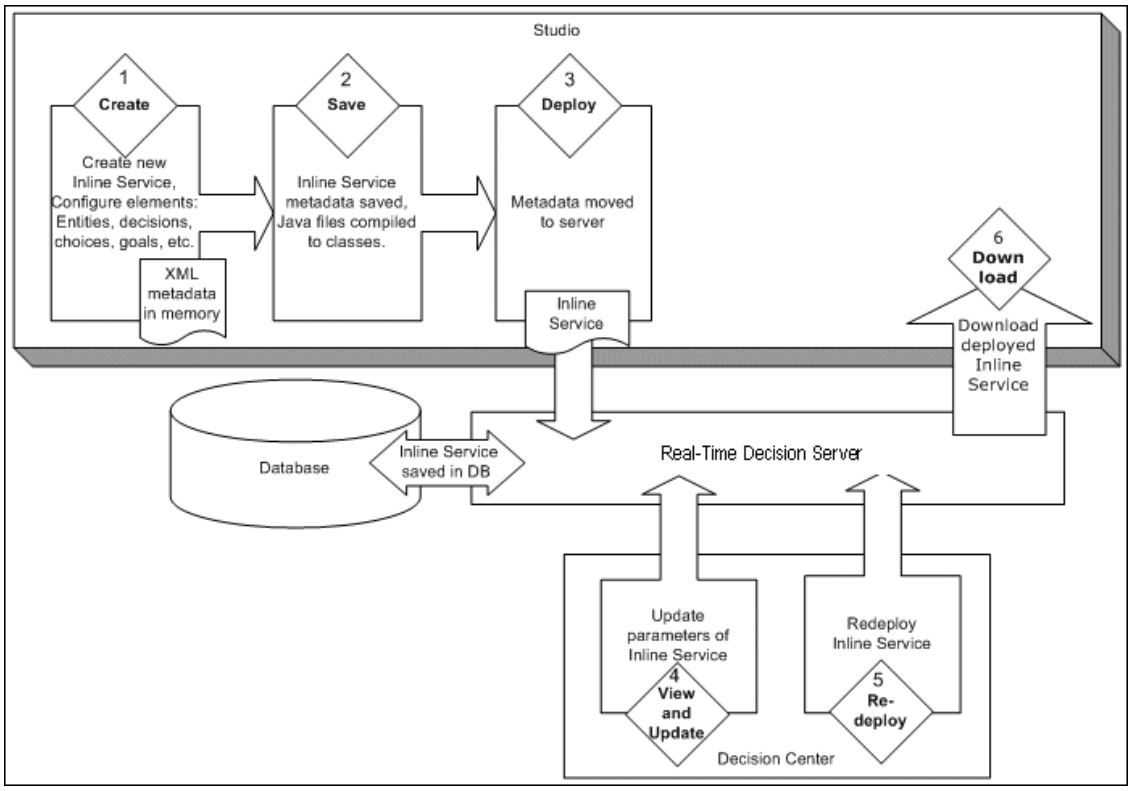


Figura 2.5: Ciclo di vita di una inline service [3]

Di conseguenza, il server su cui è implementato RTD può essere richiamato dal Front-end per richiedere le offerte commerciali. Inoltre, presenta una dashboard attraverso il quale l'amministratore può analizzare, monitorare e suggerire le raccomandazioni proposte tramite le inline services. Il compito principale dell'amministratore è quello di creare nuove inline services, definendo anche delle componenti necessarie per il loro corretto funzionamento, e caricarle sul server come descritto in figura 2.5. Ad ogni proposta viene assegnato uno score e tramite questo valore è possibile stabilire quale sia l'offerta più indicata da presentare al cliente. I punti di contatto con tutto ciò che è esterno a RTD vengono denominati ***integration points*** e vengono suddivisi in due categorie: *Informants* e *Advisors*. I primi ricevono dati dall'esterno, mentre i secondi una volta che ricevono i dati rispondono con una proposta. Ogni Advisor richiama degli oggetti decisionali parametrizzati che racchiudono al loro interno delle proposte commerciali. Tra queste proposte viene selezionata la migliore in base ai parametri specificati nella richiesta e allo scenario che è stato selezionato nella fase precedente. Ogni scenario ha le sue

regole ed è in base ad esse che vengono selezionati i banner (o i prodotti). Ad esempio, lo scenario 103 relativo all'evento LOGIN seleziona tutti quei banner per i quali sono verificate le seguenti condizioni:

1. Il banner non deve essere scaduto.
2. Il banner deve essere legato all'ndg.
3. L'utente non deve aver espresso un feedback "interessato", "non interessato" o N feedback "*visto*" (con $N \geqslant visualizationThreshold$) per quel banner.
4. L'utente non deve aver espresso per il prodotto legato al banner un feedback "*Negativo*", "*Positivo*" o "*Venduto*" negli ultimi N giorni.

Verificate queste condizioni viene costruita la risposta che conterrà la lista dei banner che possono essere presentati all'utente. Quest'ultima viene inviata al Front-End, attraverso l'advisor, completando il ciclo di processamento delle richieste.

Capitolo 3

Tecnologie usate

SNAP è in funzione da molti anni e numerose tecnologie innovative sono state presentate da quel momento. Nasce l'esigenza di aggiornare il sistema attuale introducendo un approccio più semplicistico e maggiormente orientato al *Real-time*. Con l'incremento dell'utilizzo del servizio di Home Banking segue una mole di richieste. Su questa notevole quantità di informazioni è possibile fare analisi statistiche in modo da avvantaggiare i business process incrementando i ricavi dell'azienda. Si potrebbe pensare di verificare quali banner sono stati valutati positivamente dai clienti e di conseguenza scartare quelli valutati in maniera negativa. Si cerca di puntare ad una soluzione modulare che possa consentire l'aggiunta di nuovi scenari in accordo con le necessità dell'azienda che ne ha commissionato lo sviluppo. È necessario creare un prodotto che possa essere utilizzato anche dagli amministratori meno esperti ricorrendo ad un'interfaccia *user-friendly* per ottenere delle informazioni eseguendo delle semplici operazioni di filtraggio e di aggregazione sulle richieste in arrivo. Per lo sviluppo di una soluzione conforme a quanto precedentemente detto, si è optato per *Oracle Stream Analytics*. Questo prodotto incorpora gli strumenti necessari per poter compiere operazioni su stream di dati in maniera efficiente. Un aspetto che è necessario tenere in considerazione è il valore dello storico dei dati che devono essere disponibili anche per i servizi aggiunti in un momento successivo all'inizio della raccolta dati. Le informazioni raccolte devono essere resilienti agli errori. Ciò si traduce nell'adozione di un'infrastruttura che possegga un meccanismo per garantire il *fault-tolerance* dove i dati vengono replicati per evitare perdita di informazioni e aumentare il grado di parallelismo. L'elaborazione dei dati in *real-time* deve essere un processo veloce che possa agire appena il dato è disponibile.

I dati vengono generati in continuazione e molto spesso le applicazioni che generano dati sono diverse da quelle che li prelevano. L'obiettivo è quello di avere un punto di contatto tra queste applicazioni senza stravolgere la loro implementazione [4]. Diversi studi affermano che Apache Kafka risulta essere un valido strumento che possa garantire tutte le proprietà precedentemente elencate [5]. Comparato agli altri sistemi più diffusi, Kafka risulta essere quello con le prestazioni più soddisfacenti [6]. Le tecnologie attualmente usate da SNAP sono destinate a cadere in disuso per fare spazio a nuovi prodotti molto più stabili ed efficienti. In quest'ottica si è scelto di adottare Kafka.

3.1 Apache Kafka

Kafka [7] è una piattaforma di messaggistica istantanea, sviluppata da Apache Software Foundation, che permette di creare applicazioni specializzate nello *stream processing*. Uno stream di dati costituisce una struttura senza vincoli di memoria in costante crescita. Lo *stream processing* consiste nel combinare i dati presenti nello stream in modo da generare degli aggregati che possano essere sfruttati per effettuare altre operazioni. Kafka fornisce supporto per questo tipo di applicazioni (es. Apache Spark).

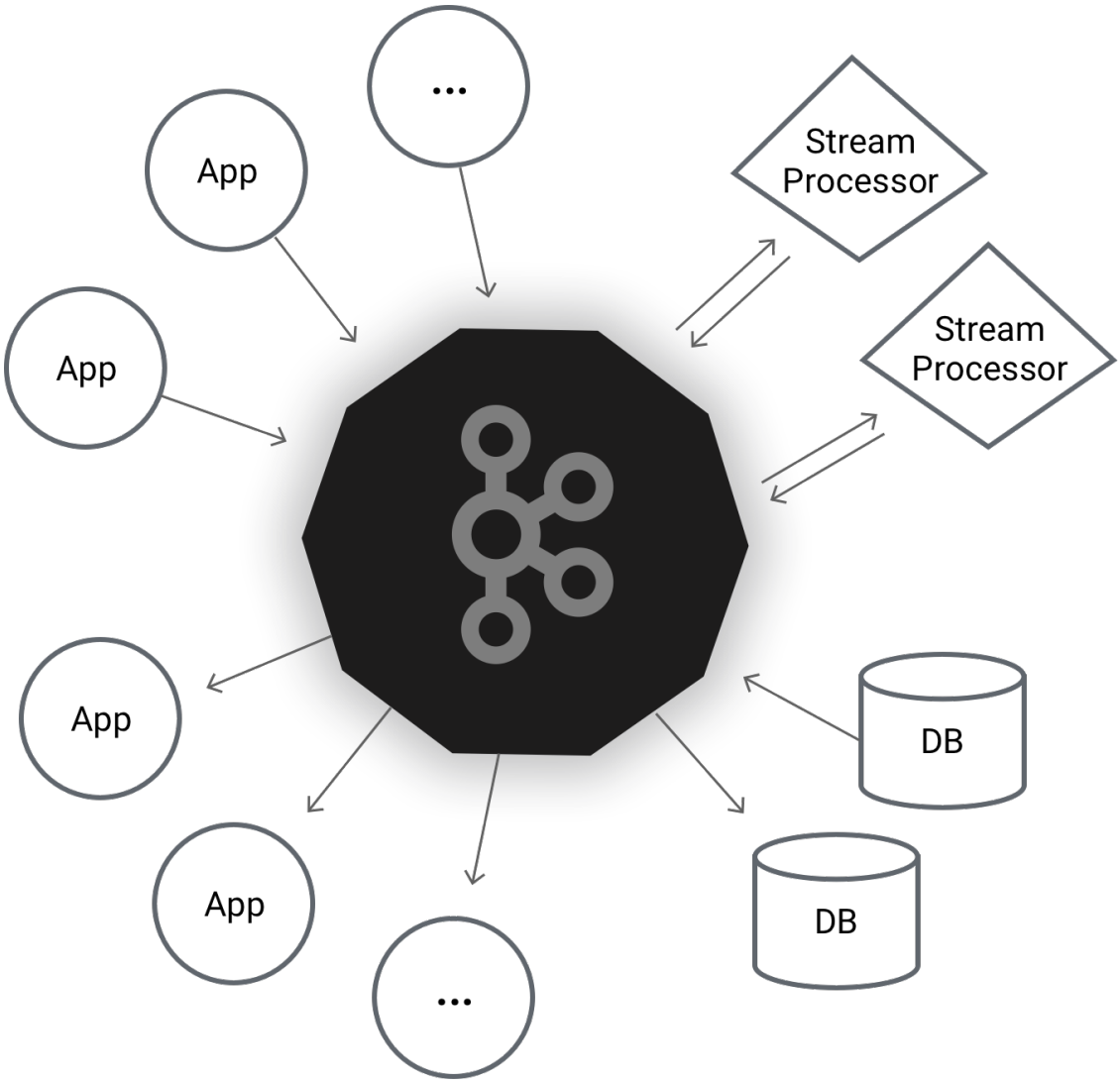


Figura 3.1: Architettura concettuale di Kafka [7]

Come tutti i sistemi di messaggistica, Kafka mette a disposizione un cluster dove i record vengono memorizzati e possono essere prelevati dalle applicazioni. Viene utilizzato il paradigma *producer/consume*r per questo tipo di architettura. In questo paradigma vi sono entità che assumo ruoli differenti:

- Il *producer* scrive i messaggi in delle strutture.
- Il *consumer* preleva questi messaggi dalle medesime.

Queste operazioni di lettura/scrittura avvengo in totale sicurezza perché vi è un meccanismo interno per la gestione della concorrenza. Questo approccio

è molto usato nei sistemi distribuiti, in quanto garantisce la condivisione delle informazioni e la modularità delle applicazioni. Ciascuna applicazione che produce dati può collegarsi a questo sistema ed inviare attraverso una connessione TCP/IP i dati che acquisisce. I dati, definiti *eventi*, rimangono nelle strutture per tutto il tempo che serve, in modo tale che possano essere consumati da altre applicazioni. La scelta di Kafka non è casuale ma giustificata dai vantaggi che offre [8]:

- **High-throughput**: Kafka è capace di gestire enormi quantità di dati senza richiedere dell'hardware dedicato.
- **Low latency**: Kafka è in grado di gestire migliaia di richieste nel minor tempo possibile (nel range dei millisecondi).
- **Reliability**: Kafka è un sistema distribuito *fault-tolerant* che sfrutta il partizionamento e la replica dei dati.

3.1.1 Cluster

L'ecosistema Kafka è costituito da un cluster formato da differenti server chiamati *broker* in cui vengono memorizzati i dati. Questi ultimi vengono organizzati in unità logiche chiamate *topic* che verranno segmentate in partizioni [9]. La suddivisione in partizioni garantisce un aumento del throughput di rete e garantisce scalabilità.

Il *topic* è concettualmente assimilabile ad un insieme di code. Nello specifico, l'ordine di arrivo dei messaggi è garantito all'interno di ogni singola partizione. Non viene, per tanto, garantito l'ordine assoluto all'interno del *topic*.

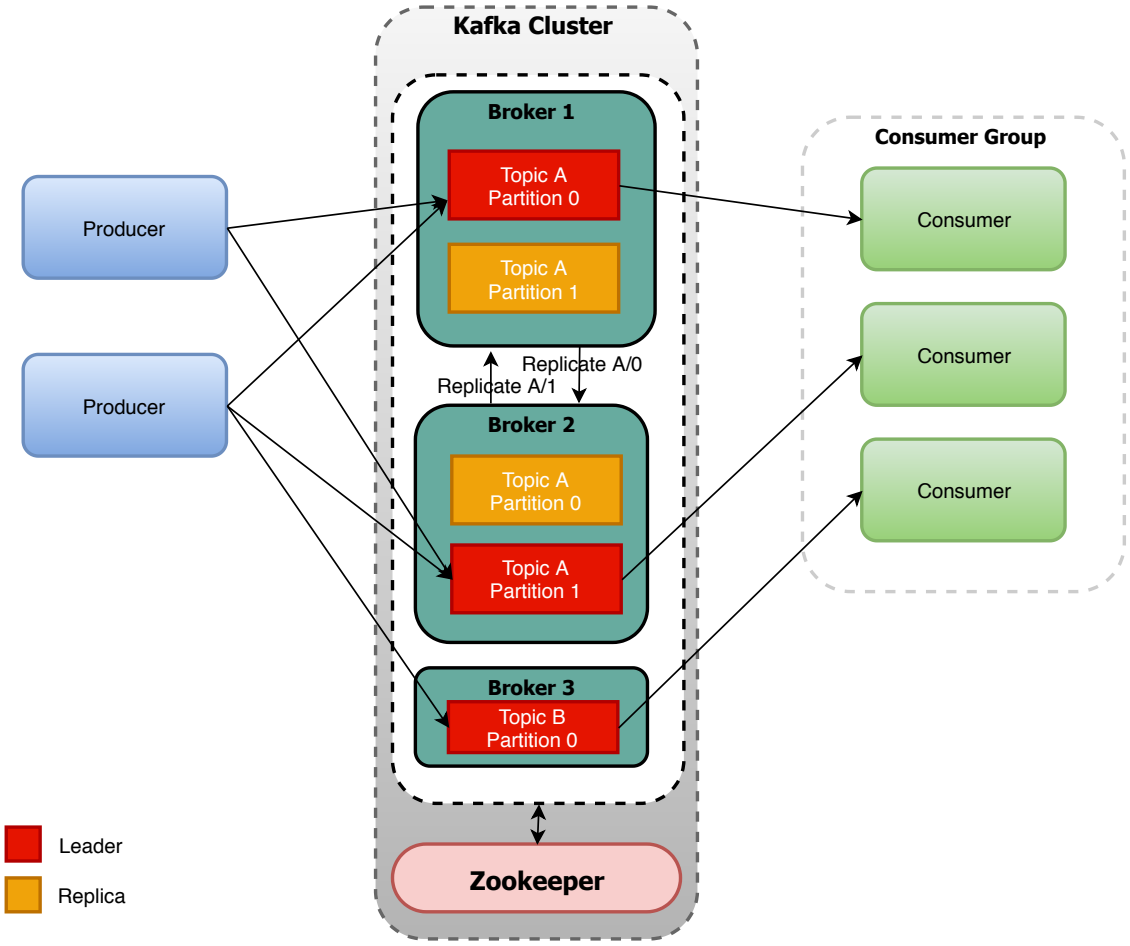


Figura 3.2: Architettura di un cluster Kafka

Zookeeper

Lo *Zookeeper* è un servizio centralizzato messo a disposizione da Kafka per registrare i metadati utili per il corretto funzionamento del sistema e il coordinamento delle attività di sincronizzazione. Si tratta di un'unità logica a cui le varie componenti del sistema possono riferirsi. Ad esempio, può registrare gli offset dei messaggi, in modo tale che i *consumer* possano sapere da dove iniziare a leggere i dati, ed informa gli eventuali *producer* e *consumer* dell'avvenuta aggiunzione o rimozione di un *broker*. Tramite lo *Zookeeper* è possibile conoscere lo stato dei vari broker e tenere traccia della posizione dei vari *topic* e delle partizioni.

Broker

Il suo compito è quello di registrare i messaggi mandati dai *producer* e servire le richieste dei *consumer*. Questi record vengono salvati nel disco come degli array di byte, opportunamente convertiti attraverso un processo di serializzazione. Ogniqualvolta un messaggio viene ricevuto, il broker provvede ad assegnargli un offset all'interno di una partizione. Un singolo broker può gestire migliaia di richieste al secondo [9].

Il funzionamento del cluster sfrutta un meccanismo basato sul concetto di Leadership. All'interno del cluster i broker notificano allo *Zookeeper* la loro volontà di voler acquisire la leadership, e solamenta il primo che ha inviato la notifica verrà eletto *controller* del cluster. Quest'ultimo ha la responsabilità di gestire gli stati delle partizioni e delle relative repliche, ed eseguire altre attività come la riassegnazione delle partizioni *leader*. Ciascuna partizione viene posseduta da un singolo broker, che verrà definito *Leader* per quella partizione. Il suo compito è quello di replicare le partizioni negli altri *broker* creando delle repliche per ottenere la ridondanza dei dati. Nel caso in cui il *Leader* smettesse di funzionare sarà compito del *Controller* eleggere un nuovo leader tra le repliche, andando a scegliere le partizioni *in-sync* con la partizione non più funzionante. In particolare, ciascuna partizione può essere replicata più volte e le repliche (o *followers*) hanno il compito di mantenersi sincronizzate con la partizione *leader*, in modo che possano sostituirla nel momento in cui questa non è più disponibile.

Producer

Affinchè un'applicazione possa scrivere su uno o più *topic*, deve implementare almeno un *producer*. Kafka fornisce delle API che permettono di crearli.

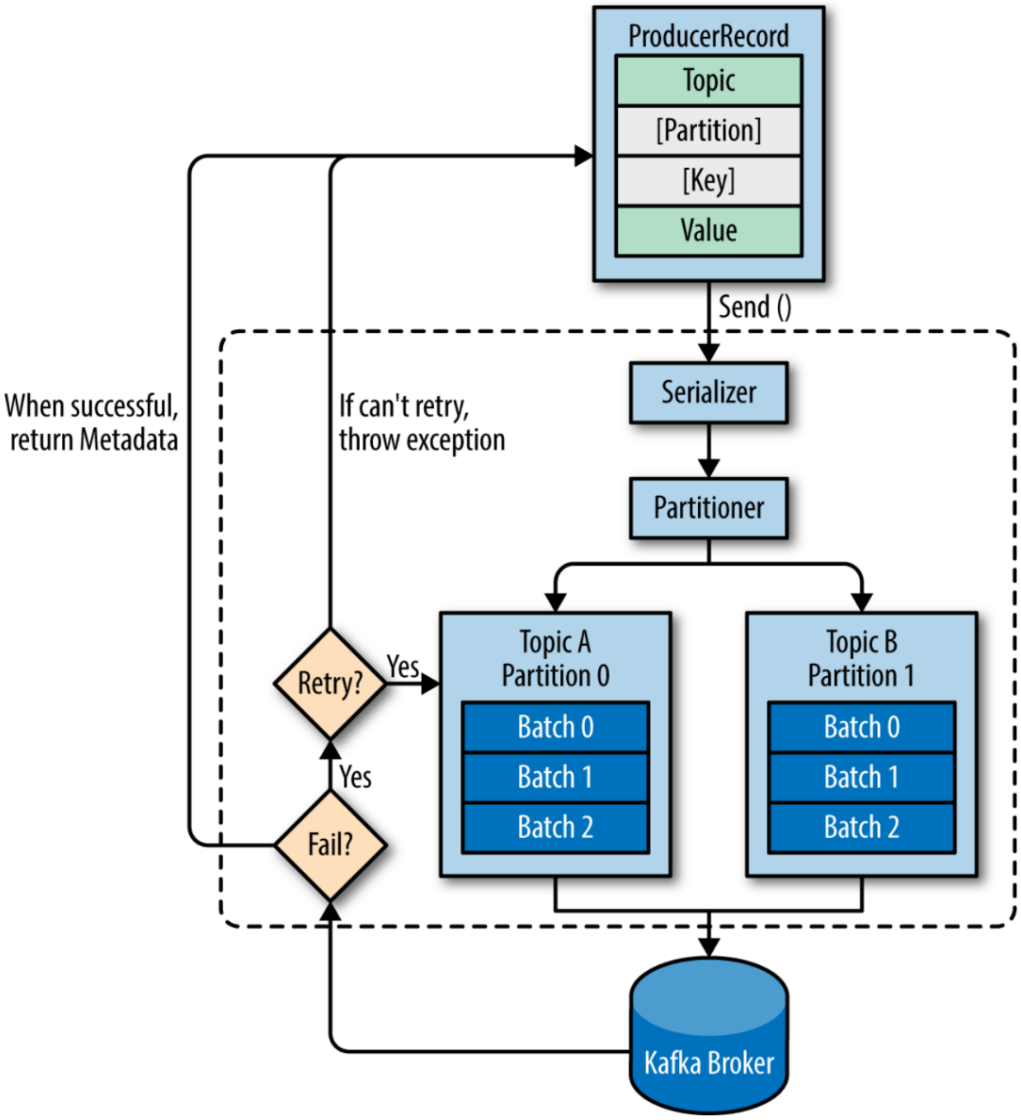


Figura 3.3: Scrittura da parte del Producer [9]

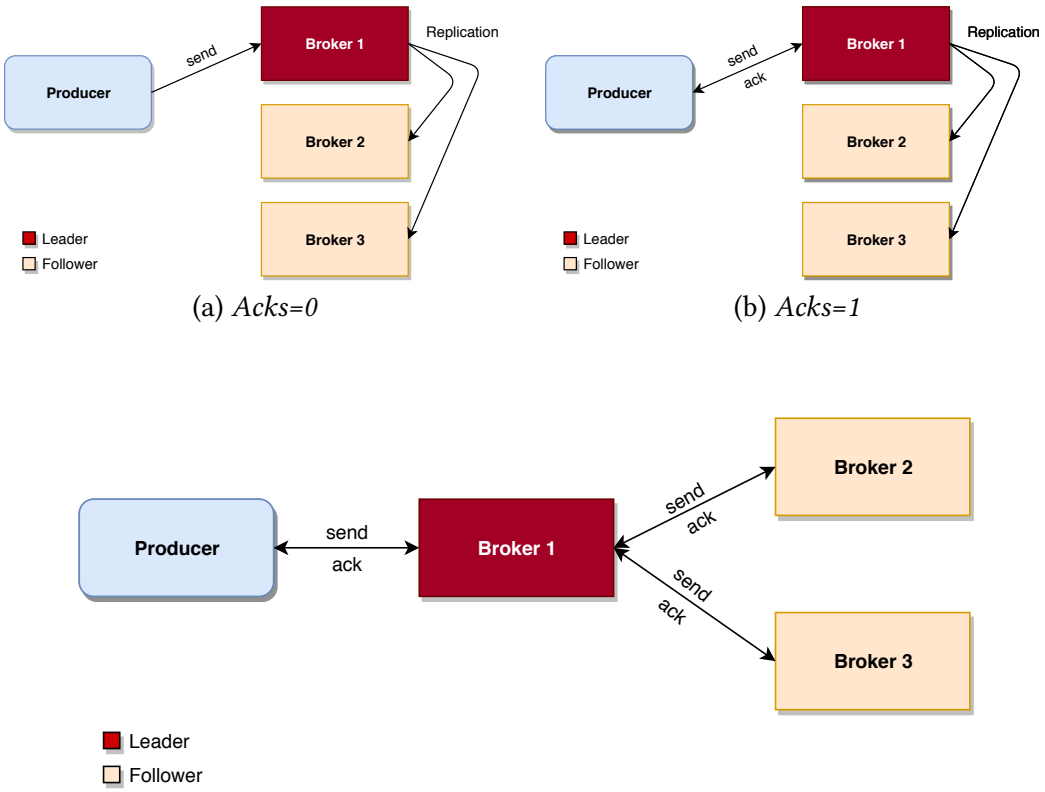
Ogni *producer* genera un record dove vengono specificati:

- topic di destinazione,
- partizione (opzionale),
- chiave (opzionale),
- dato.

I *Producer* possono scegliere in quale partizione inviare il record includendo la chiave, da cui verrà ricavato l'hash che servirà per scegliere la partizione, altrimenti sarà Kafka a suggerirne una. La prima opzione ha uno svantaggio: le partizioni non saranno equilibrate. Ciò è dovuto alle collisioni che occorrono quando si utilizzano delle tecniche di hashing. Nel caso in cui non venga specificata la chiave allora Kafka provvederà a scegliere una partizione, seguendo un approccio *Round-Robin* [11]. In questo modo i record saranno equamente distribuiti fra le partizioni. L'oggetto viene inviato ad un *serializer* che si occupa di effettuare la conversione ad array di byte. Successivamente, il *Partitioner* si occuperà di scegliere la partizione secondo le modalità descritte precedentemente. Una volta che la partizione è stata assegnata, il *producer* aggiunge l'oggetto ad un batch di messaggi (a meno che il parametro *max.in.flight.requests.per.connection* non sia pari ad uno) pronti per essere inviati al *broker* di destinazione. Quando il *broker* ha ricevuto i messaggi, ne notifica l'avvenuta ricezione e invia un metadato contenente informazioni addizionali sulla memorizzazione[9]. L'intero processo viene descritto in Figura 3.3. Con le API fornite da Kafka è possibile variare alcuni parametri di sistema per migliorare le prestazioni del producer a seconda del contesto di utilizzo. Alcuni di questi sono [10]:

- *acks*: specifica se il producer deve aspettare gli ack dal *broker leader* (vedi Figura 3.4).
- *compression.type*: specifica il tipo di compressione del record
- *batch.size*: specifica la dimensione (in Byte) del batch che conterrà i messaggi.

  L'overhead dovuto all'invio di ogni messaggio singolarmente sarebbe eccessivo, per cui si introduce il concetto di batch (collezione di messaggi) per migliorare il throughput. Lo svantaggio di questo approccio è che il singolo messaggio si propaga più lentamente
- *linger.ms*: specifica il tempo di attesa per l'invio dei messaggi (nel caso in cui il batch non sia pieno).
- *max.in.flight.requests.per.connection*: specifica quanti messaggi inviare al *broker* ad ogni connessione.



(a) *Acks=0*

(b) *Acks=1*

(c) *Acks=all*

Figura 3.4: Casi d'uso del parametro *Acks*

Consumer

I *Consumer*, dopo aver specificato una lista di *topic* ai quali iscriversi, prelevano i messaggi dai topic nell'ordine in cui vengono scritti dai *Producer*. Successivamente, il *consumer* eseguirà ciclicamente il *polling* dei messaggi. Durante la fase di polling le partizioni, dalle quali sarà possibile prelevare i dati, sono assegnate da Kafka. Nel caso in cui venisse innescata una fase di bilanciamento dei Consumer, dovuta da uno spegnimento improvviso di uno di essi, allora nuove partizioni verranno assegnate. È possibile specificare un tempo di polling, alle fine del quale il *consumer* preleverà tutti i messaggi che erano presenti nella coda, tenendo traccia dell'*offset* relativo all'ultimo messaggio letto. Questa informazione viene memorizzata anche dallo Zookeeper. In questo modo, se uno dei consumer non funzionasse, lo Zookeeper potrà condividere questa informazione con il *consumer* che prenderà il suo posto,

affinché possa riprendere dall'ultimo offset registrato. In una situazione reale, i *topic* vengono letti da più applicazioni. In quest'ottica viene introdotto il concetto di *Consumer Group* che costituisce uno dei principali vantaggi di Kafka, in quanto garantisce flessibilità e scalabilità. Nel momento in cui sono utilizzati più consumer, questi vengono associati ad un *consumer group*, identificato da un attributo *group.id*. Ciascun gruppo può iscriversi ad uno o più *topic*. Grazie a questo stratagemma, ciascun membro del gruppo leggerà un sottoinsieme di partizioni appertenti ad un topic. In particolare, ogni record del topic verrà letto da un solo membro di quel gruppo, aumentando il grado di parallelismo [16].

Per ogni gruppo, uno dei *broker* viene selezionato per coordinare le attività di lettura. Il *coordinatore* gestisce il gruppo, assegnando le partizioni a ciascun membro del gruppo, nei momenti in cui avviene un cambiamento all'interno del gruppo [17]. In genere, si utilizza un numero di *consumer* pari al numero di partizioni. Nel caso in cui fossero presenti in numero maggiore, allora resterebbero inattivi, causando un spreco di risorse. In Figura 3.5 viene descritto un *consumer group* formato da un solo membro che legge da un solo topic, formato da quattro partizioni. Quando un *consumer* si aggiunge a questo gruppo, inizia un processo di ridistribuzione delle partizioni da parte del *Coordinatore*. Nello specifico, al primo *consumer* vengono assegnate le prime due partizioni, mentre le ultime sono destinate all'altro *consumer*. Nell'ultimo riquadro viene descritta la situazione più conveniente, in quanto il numero di partizioni è eguagliato dal numero di *consumer*, e perciò ogni partizione sarà letta da uno solo dei membri del gruppo.

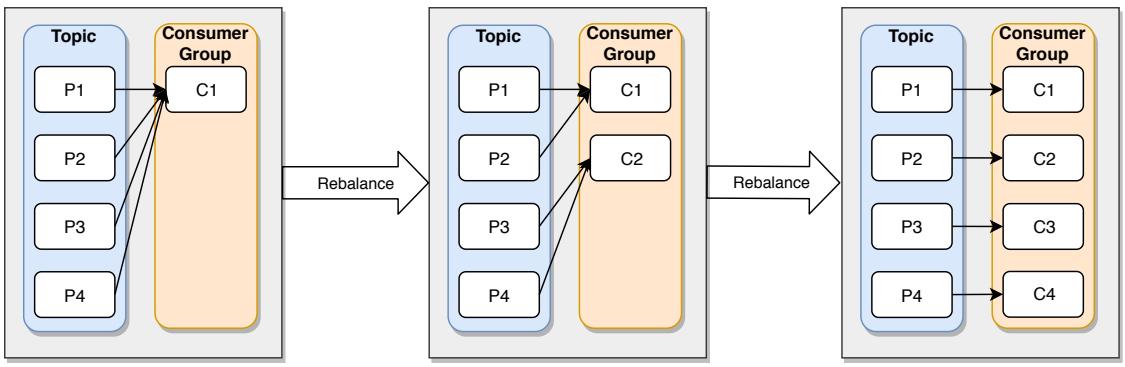


Figura 3.5: Scalabilità di Kafka grazie all'uso dei Consumer Group

3.2 Oracle Stream Analytics

Focalizzando l'attenzione sulla Figura 3.1 è possibile notare la presenza di alcuni elementi chiamati *Stream Processor*. In un contesto in cui i dati vengono visti come dei veri e propri stream nascono delle componenti chiamate *Stream Processor*. Questa componente viene introdotta al fine di analizzare in maniera ottimale un flusso di dati per poter applicare delle funzioni logiche elementari o complesse. Tramite questo meccanismo è possibile l'elaborazione dei dati in tempo reale, senza che i risultati siano salvati in memoria e con il vincolo che i tempi di reazione siano inferiori ai millisecondi [12].

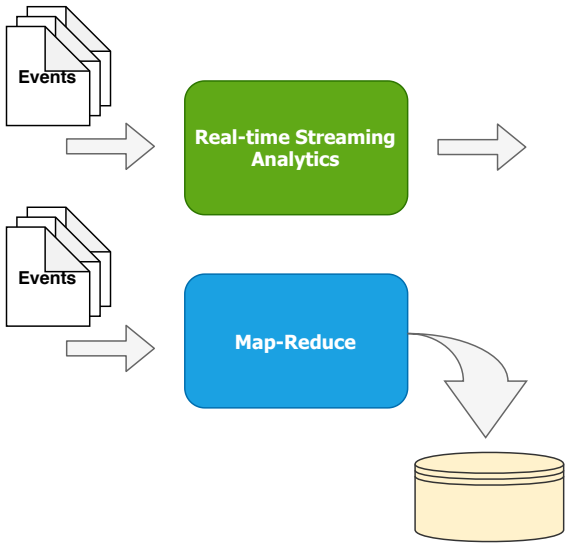


Figura 3.6: Comparazione tra Real-time Analytics e paradigma Map-Reduce

Nella nuova versione di SNAP, sviluppata nella presente tesi, si è scelto di utilizzare *Oracle Stream Analytics* (OSA) [13] come Stream Processor.

Questo prodotto di Oracle, implementato interamente su Spark, si presenta come la naturale evoluzione di Oracle Event processing. Con questo strumento è possibile processare e analizzare dati su larga scala in maniera automatica svolgendo determinate operazioni in tempo reale. Tutte queste operazioni sono gestite tramite una Dashboard:

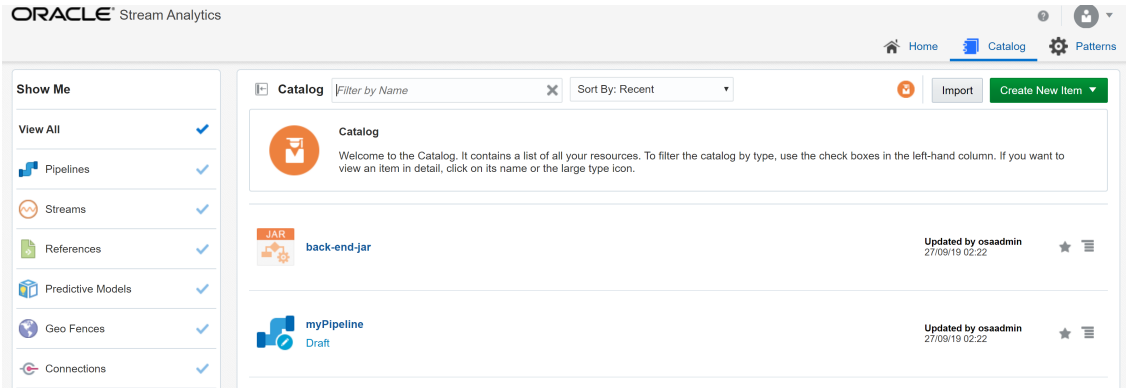


Figura 3.7: Dashboard di Oracle Stream Analytics

Grazie all'uso della dashboard, OSA è un prodotto rivolto sia agli sviluppatori che agli utenti meno esperti.

OSA consta di diverse componenti, chiamati *item*:

- Connection,
- Stream,
- Pipeline,
- Custom Jar,
- Target.

Nei sotto paragrafi successivi sono illustrate le componenti rilevanti ai fini della presente ricerca.

3.2.1 Connection

Una *connessione* è un oggetto che fa riferimento ad una sorgente esterna, ed è costituita da una collezione di metadati richiesti per effettuare la connessione. I tipi di connessioni supportati nella versione utilizzata (18.0.0.0) sono:

- Database,
- Coherence,
- Druid,

- JNDI,
- Kafka.

Inoltre, la stessa connessione può essere riutilizzata più volte per accedere a differenti risorse (e.g. è possibile sfruttare una connessione al cluster Kafka per fare riferimento a differenti topic).

3.2.2 Stream

L'oggetto *stream* è un flusso dinamico di dati e viene definito a partire da una connessione esistente. Tramite questo oggetto è possibile referenziare un *topic*.

3.2.3 Pipeline

Una pipeline definisce una coda formata da *stage*. Ciascuno di essi assume il ruolo di *event processor* e si occupa di trasformare e arricchire i dati provenienti da risorse esterne visualizzandoli su di una interfaccia grafica. Si costruisce una rete atta a elaborare i dati, dove ogni *stage* ha il proprio scopo ed effettua operazioni differenti. Specificatamente, per ciascun nodo si possono applicare le seguenti operazioni:

- Pattern,
- Query,
- Rule,
- Scoring,
- Target.

Inoltre, è possibile avere dei branch nei vari nodi. Ogni cambiamento su un nodo viene automaticamente propagato su tutta la pipeline.

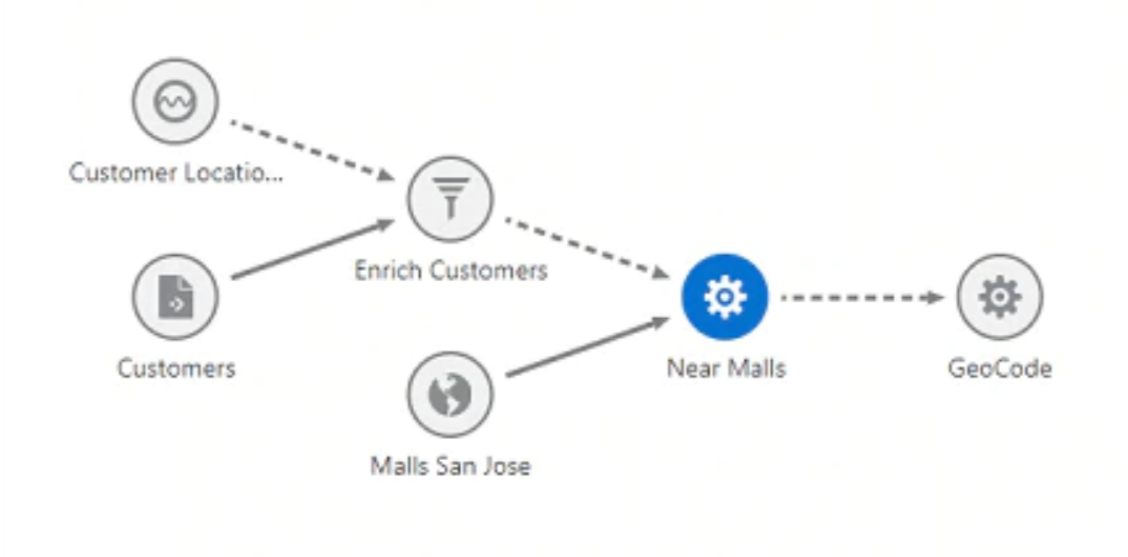


Figura 3.8: Schema grafico di una Pipeline [14]

3.2.4 Custom Jar

Gli strumenti messi a disposizione da OSA permettono di eseguire le operazioni più basilari. Nel momento in cui si ha la necessità di eseguire delle operazioni più complesse, è possibile sfruttare dei file jar. Questi file contengono gli algoritmi, scritti in linguaggio JAVA, che consentono di applicare la logica voluta dal programmatore. Nello specifico, è possibile definire due tipi di *custom jar*:

- Custom Functions

  Tramite un *Custom Stage* si definisce un proprio stage personalizzato da applicare ai dati della pipeline.

- Custom Stage.

  le *Custom Functions* vengono utilizzate per definire le proprie funzioni che saranno accessibili attraverso l'*Expression Builder*. Per maggiori dettagli circa l'utilizzo dell'Expression Builder consultare la documentazione [18].

3.2.5 Target

I risultati ottenuti attraverso l'elaborazione dei dati che fluiscono all'interno della pipeline possono essere salvati all'interno di una risorsa esterna. Oracle Stream Analytics mette a disposizione un artefatto chiamato *Target* che referenzia una risorsa esterna. Questo stage rappresenta lo stadio finale della pipeline ed è opzionale.

Capitolo 4

Architettura proposta: SNAP 2.0

In questo capitolo viene descritta la metodologia adottata per la progettazione e l'implementazione del sistema SNAP 2.0. Per semplicità, sono stati implementati gli scenari relativi agli eventi *Login* e *Vetrina Prodotti*. In particolare, gli scenari presi in considerazione sono:

- Scenario 103, relativo all'evento *Login*.
- Scenario 400, relativo all'evento *Vetrina Prodotti*

4.1 Progettazione

Durante questa fase sono state prese in considerazione diverse tecnologie che potessero adattarsi al caso in questione. Nel dettaglio, l'azienda richiede le seguenti specifiche di progetto:

- Semplicità nell'aggiunta di un nuovo scenario.
- Adozione di uno strumento per l'elaborazione dei dati in real-time.
- Condivisione dei dati tra i vari servizi offerti.

In base alle specifiche espresse è stato possibile individuare le tecnologie ampiamente descritte nel Capitolo 3. Le macrocomponenti adottate dall'architettura, di conseguenza, sono tre:

- *Oracle Web Logic Server*

Questa è l'unica componente che si è deciso di mantenere per non stravolgere la logica del precedente sistema. Sono state comunque apportate delle modifiche al fine di renderlo compatibile con Kafka.

- *Apache Kafka*

  La versione di Kafka utilizzata è la 2.1.1 mentre nel Front-end e nei custom jar si utilizzano le API di Kafka versione 2.3.0. Tramite questa tecnologia vengono sfruttati i Topic per memorizzare le richieste e le risposte, in modo tale che siano condivisibili tra i vari servizi del front-end. Grazie alla gestione automatica dei topic e alla loro suddivisione in partizioni, Kafka permette di fornire le risposte in tempi molto brevi.

Figura 4.1: Logo di Kafka [7]

- *Oracle Stream Analytics*

  Si è scelto di adottare OSA versione 18.0.0.0 per via della sua compatibilità con l'ecosistema Kafka, in quanto permette di leggere flussi di dati prelevati dai topic e di poter applicare le operazioni necessarie per fornire un'offerta personalizzata, come nel vecchio sistema.

Figura 4.2: Logo di Oracle Stream Analytics [14]

4.2 Implementazione

In questa sezione viene spiegato nel dettaglio il funzionamento delle diverse componenti presenti nel sistema SNAP 2.0, la cui architettura è rappresentata in Figura 4.3.

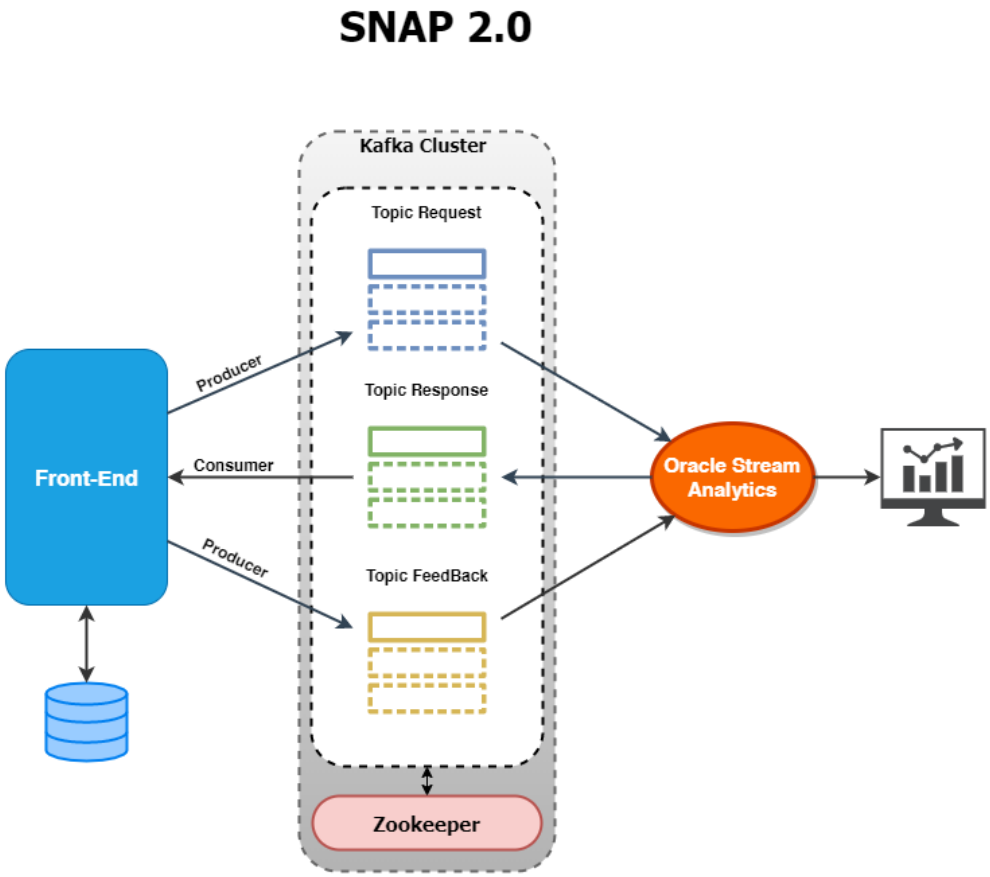


Figura 4.3: Architettura di SNAP 2.0

4.2.1 Front-End 2.0

La nuova versione mantiene la maggior parte dei servizi che erano presenti in SNAP, ad esclusione dei servizi relativi alle sessioni. Nel precedente sistema

il cliente doveva richiere una sessione prima di poter richiamare gli altri servizi. Si è scelto di rinunciare al concetto di sessione, in quanto risulta ereditato da un altro progetto e non aveva nessuna utilità nel contesto dell'applicazione. Di conseguenza, non sono stati implementati i servizi di *StartSession* e *EndSession*.

Si è deciso dunque di implementare esclusivamente i servizi *OfferService*, *PostOutcomeService* e *PostEventService*. Si rimanda al Capitolo 2 per una descrizione dettagliata. In aggiunta, sono state apportate delle migliorie per rendere il sistema più performante e compatibile con Kafka. Sono state aggiunte le classi per la gestione dei *Consumer* e dei *Producer*.

Quando il sistema riceve una richiesta, si instanzia un Producer che provvede a memorizzare le richieste in un relativo Topic denominato *Topic-Offer-Request*. Di contro, vengono inizializzati altri consumer su più thread, in modo da migliorare le prestazioni, che si occupano di fare il *polling* su un opportuno topic, chiamato *Topic-Offer-Response*, dove vengono salvate le risposte al termine della loro eleborazione. L'architettura di Kafka non supporta il paradigma Request-Response. Perciò è stato reso necessaria l'introduzione di un meccanismo di associazione richiesta/risposta. Conseguentemente si assegna ad una richiesta una chiave:

$$Key = Abi@Ndg@Evento@Timestamp$$

Viene utilizzata una HashMap di oggetti del tipo *EventOffer*, che tiene traccia dello storico delle richieste in attesa di completamento. Un oggetto *EventOffer* contiene al suo interno altri due oggetti: *OfferRequest* e *OfferResponse*, che includono le informazioni contenute rispettivamente nella richiesta e nella risposta. Una volta che la richiesta viene incapsulta in un oggetto *EventOffer* viene processata dal sistema per un tempo limite di 5 secondi, al termine del quale verrà restituito un messaggio di errore. Durante questo intervallo viene verificato il valore di un campo booleano chiamato *isComplete*, la quale verrà settato a *True* nel momento in cui la risposta viene elaborata.

Dal momento in cui viene eseguita l'applicazione, i consumer operano constantemente, leggendo dai topic ogni 10 millisecondi. Non appena la risposta è disponibile, il consumer si occupa di recuperare il riferimento al relativo oggetto *EventOffer* tramite la chiave ed incapsularla, cambiando il valore del campo *isComplete*.

Per quanto riguarda i servizi relativi ai FeedBack, sono stati implementati dei Producer che, banalmente, inviano le richieste ai relativi Topic.

4.2.2 Kafka Cluster

Il Cluster ha un proprio ciclo di vita ed è indipendente dall'applicazione del Front-end. In esso girano i broker contenenti i topic definiti in fase di progettazione. In particolare, nel presente caso studio sono instanziati due broker, per sfruttare al meglio il fattore di replicazione nel momento in cui un nodo smette di funzionare.

Di seguito vengono mostrati alcuni dei parametri di configurazione del broker 0:

Tabella 4.1: File di configurazione del broker

SERVER.PROPERTIES	
Parametro	**Valore**
broker.id	0
num.network.threads	3
socket.send.buffer.bytes	102400
socket.receive.buffer.bytes	102400
socket.request.max.bytes	104857600
num.partitions	2
log.flush.interval.ms	20000
log.retention.check.interval.ms	300000
zookeeper.connect	localhost:2181

Al fine di garantire una certa separazione logica tra richieste, risposte e feedback, sono stati definiti all'interno del cluster quattro Topic:

- Topic-Offer-Request, contenente le richieste.
- Topic-Offer-Response, contenente le risposte.
- Topic-Feedback-Banner, contenente i feedback relativi ai banner.
- Topic-Feedback-Prodotti, contenente i feedback relativi ai prodotti.

Ciascun Topic possiede due partizioni e un fattore di replica pari a due. La replica introduce una ridondanza nei dati, e ciò significa che il sistema può tollerare al massimo un solo guasto ad uno dei due nodi.

4.2.3 Oracle Stream Analytics

Nel capitolo 3 vengono descritte le caratteristiche di Oracle Stream Analytics, tra cui la possibilità di leggere da uno stream. Si configura OSA in modo tale che si colleghi al cluster di Kafka attraverso la creazione di un oggetto di tipo *connection*, in cui vengono specificati i parametri visibile in Figura 4.4.

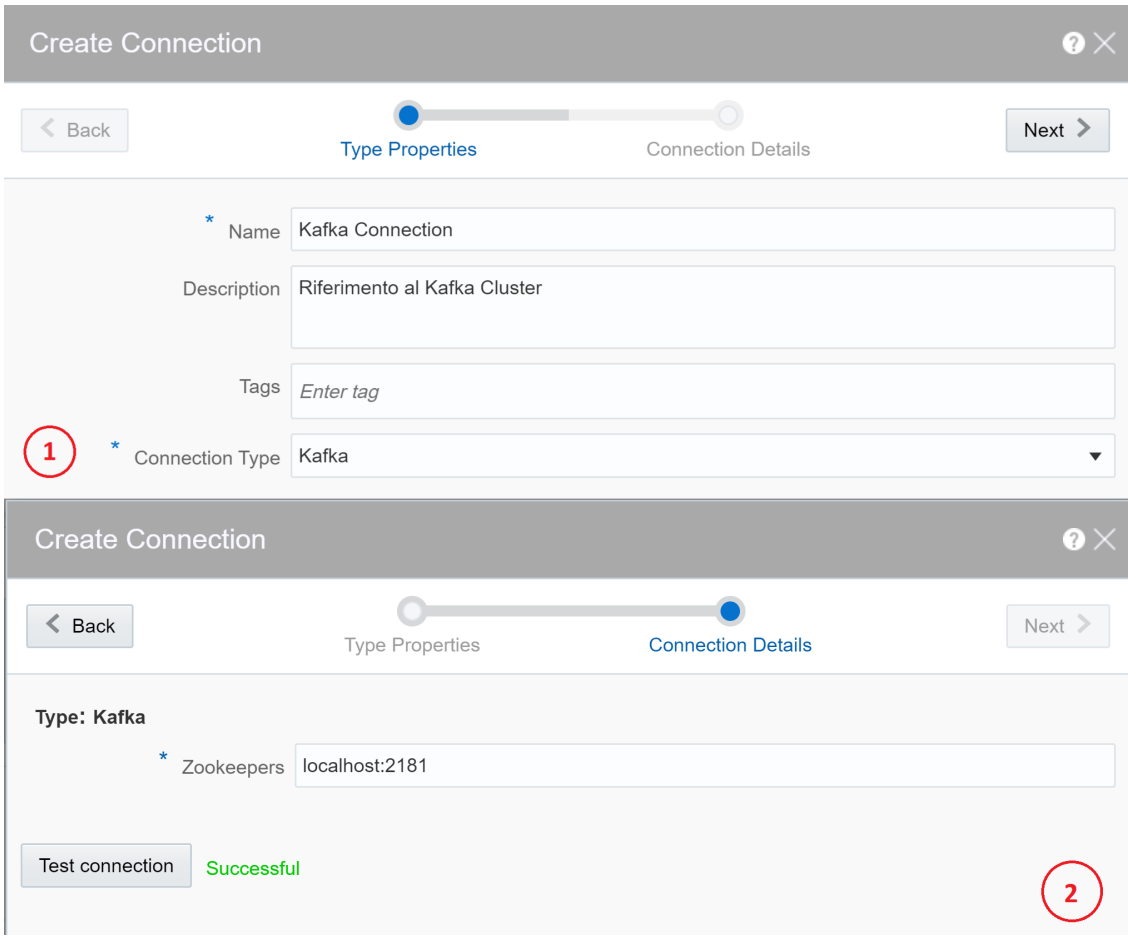


Figura 4.4: Creazione guidata di una Connection

Successivamente, viene creato un oggetto *Stream*, in cui viene specificata la connessione creata precedentemente e il topic che si vuole analizzare.

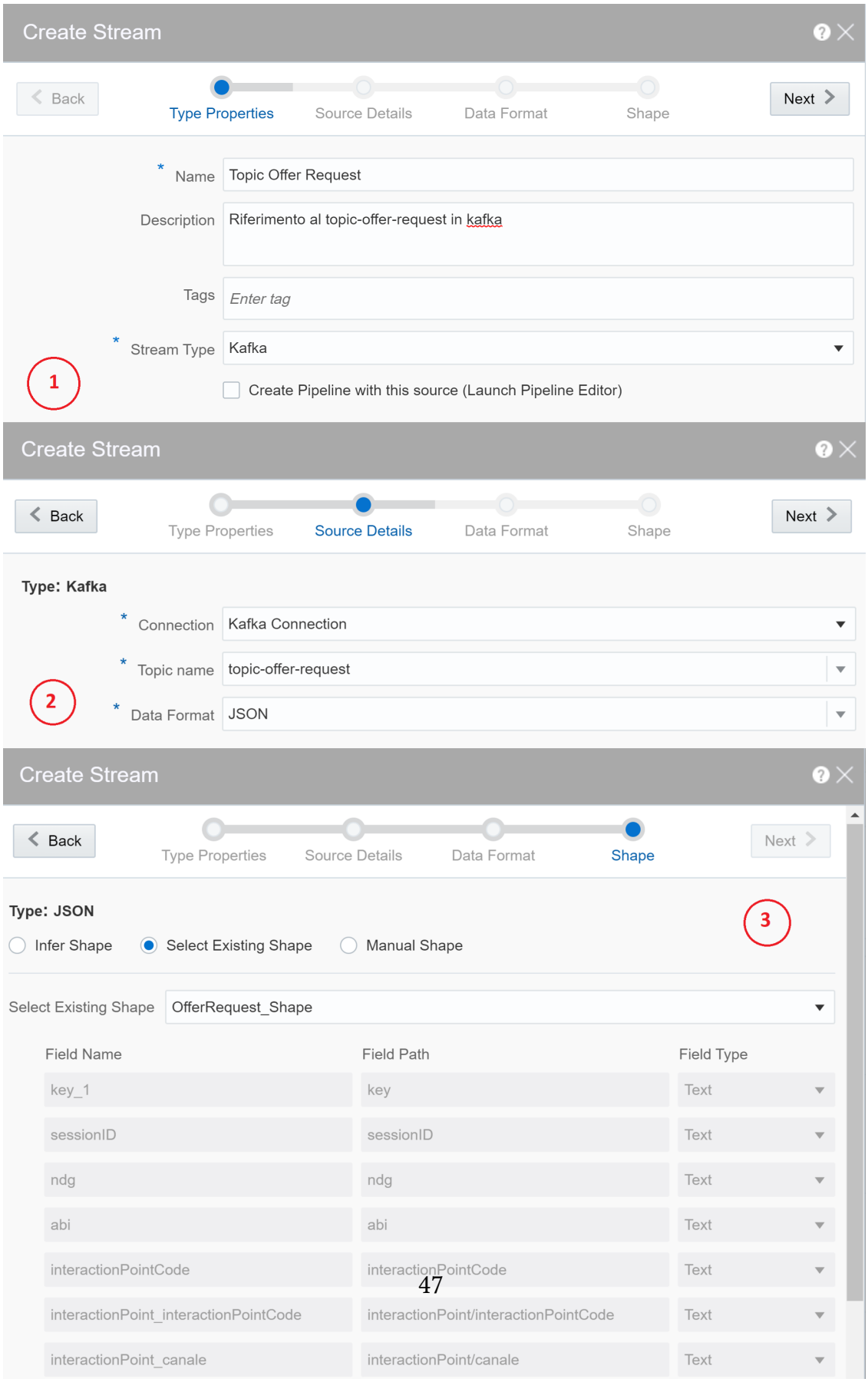


Figura 4.5: Creazione guidata di uno Stream

Una volta completati tutti gli step, si può procedere con la creazione di una pipeline il cui stage referenzia lo stream precedentemente creato. A questo punto viene eseguito automaticamente il deploy della pipeline, e successivamente si potranno vedere i dati in real-time come in Figura 4.6.

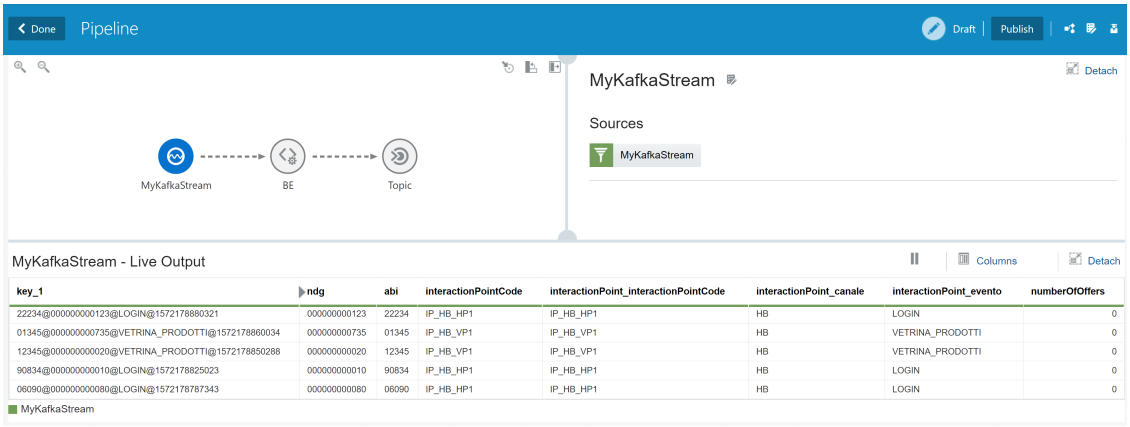


Figura 4.6: Pipeline Live Output

Questi sono gli step necessari per tenere sotto osservazione un topic.

Processo di selezione Scenario ed estrazione offerta personalizzata

Nella sezione 3.2.4 vengono citati i custom jar, ovvero file contenenti codice JAVA, che possono essere inseriti nella pipeline. Si è deciso di sfruttare questa caratteristica di Oracle Stream Analytics per poter includer la logica della profilazione del cliente. In sostanza, la selezione dei banner viene eseguita direttamente da OSA, in unico step e per ogni richiesta figurante nella pipeline in real-time. La scelta dei banner avviene sempre mediante un'opportuna selezione dello scenario più adatto. Come gia discusso, gli scenari sono legati a degli eventi ed per ognuno di essi viene definita un ordine di priorità. Gli eventi presi in considerazione nella presente tesi sono *Login* e *Vetrina Prodotti*.

- *Login*

  Al momento l'unico scenario legato a questo evento è il 103. Questo scenario si attiva sempre, tranne nel caso in cui sia stato forzatamente disattivato da un operatore esterno. Ciò può essere verificato andando a controllare una tabella chiamata *ScenariDisattivati*. Una volta che si è attivato lo scenario, si procede con il prelievo dei banner che rispettano le

regole imposte dallo Scenario 103 (si veda 2.1.3). Tra tutti i banner che rispettano le condizioni, viene selezionato solamente il primo. Quest'ultimo verrà visualizzato nella pipeline e verrà inviato al successivo stage di tipo *Target* collegato al Topic-Offer-Response.

- *Vetrina Prodotti*

  Analogamente al *Login*, questo evento possiede un solo scenario, ovvero il 400. Quando l'evento è di tipo Vetrina Prodotti ciò che vengono restituiti non sono i banner, ma una lista di prodotti. Il processo di selezione avviene esattamente come descritto in precedenza. L'unica cosa che cambia sono le condizioni a cui devono sottostare i prodotti che saranno selezionati. In particolare, un prodotto risulta idoneo se e solo se è associato all'ndg e, opzionalmente, ad un banner. Tra tutti i prodotti selezionati se ne sceglie solamente uno.

Per quanto riguarda i perfezionamenti apportati rispetto al vecchio sistema, si è deciso di cambiare la logica precedente in modo tale che vengano fornite risposte non nulle. In passato, veniva attivato il primo scenario disponibile, se possibile, e si elaborava la risposta anche nel caso in cui fosse nulla. In SNAP 2.0 si esegue un'ulteriore verifica, ovvero si controlla che lo scenario presenti almeno un banner. Nel caso in cui, per lo scenario preso in esame, non vi siano banner idonei, quest'ultimo verrà disattivato, e si procederà con l'analisi dello scenario successivo.

Capitolo 5

Analisi delle Performance

In questo capitolo verranno fornite le specifiche tecniche riguardanti le macchine utilizzate per l'esecuzione di entrambi i sistemi informativi. Inoltre, si eseguirà una valutazione comparativa dei tempi di risposta, con relativa analisi delle singole fasi del processo di elaborazione dell'offerta personalizzata per il cliente.

5.1 Specifiche Tecniche

Attualmente il sistema SNAP usa un cluster di quattro macchine in cui sono ripartite le varie componenti logiche del sistema. Le loro posizioni possono essere visualizzate nella Figura 2.2. Per lo sviluppo di SNAP 2.0 non è stato possibile sfruttare gli stessi nodi del sistema originale per motivi aziendali e, quindi, si è optato per l'utilizzo di un Notebook. In Tabella 5.1 sono riassunte le caratteristiche tecniche di ogni macchina.

Tabella 5.1: Specifiche Hardware dei dispositivi utilizzati da entrambi i sistemi

	Architecture	**CPU(s)**	**Thread/core**	**Model Name**	**RAM**
Macchina 1-2	x86_64	4	1	Intel(R) Xeon(R) CPU E5-2697 v2 @ 2.70GHz	16
Macchina 3-4	x86_64	4	1	Intel(R) Xeon(R) CPU E5-2699 v4 @ 2.20GHz	16
Notebook (Dell XPS 15)	x86_64	4	1	Intel(R) Core(TM) CPU i5-7300HQ @ 2.50GHz	8

5.1.1 Comparazione Risultati ottenuti

In questa sezione vengono analizzati i risultati in tempi di risposta del sistema sviluppato, tenendo in considerazione i tempi di elaborazione di ogni singola fase del processo di selezione dei banner. I risultati sono puramente indicativi e il sistema risulta chiaramente sottodimensionato per via del suo sviluppo su un unico nodo.

Come si evince dalla Figura 5.1, SNAP ha un tempo di risposta medio che si aggira sui 450ms con un picco massimo che supera il secondo.

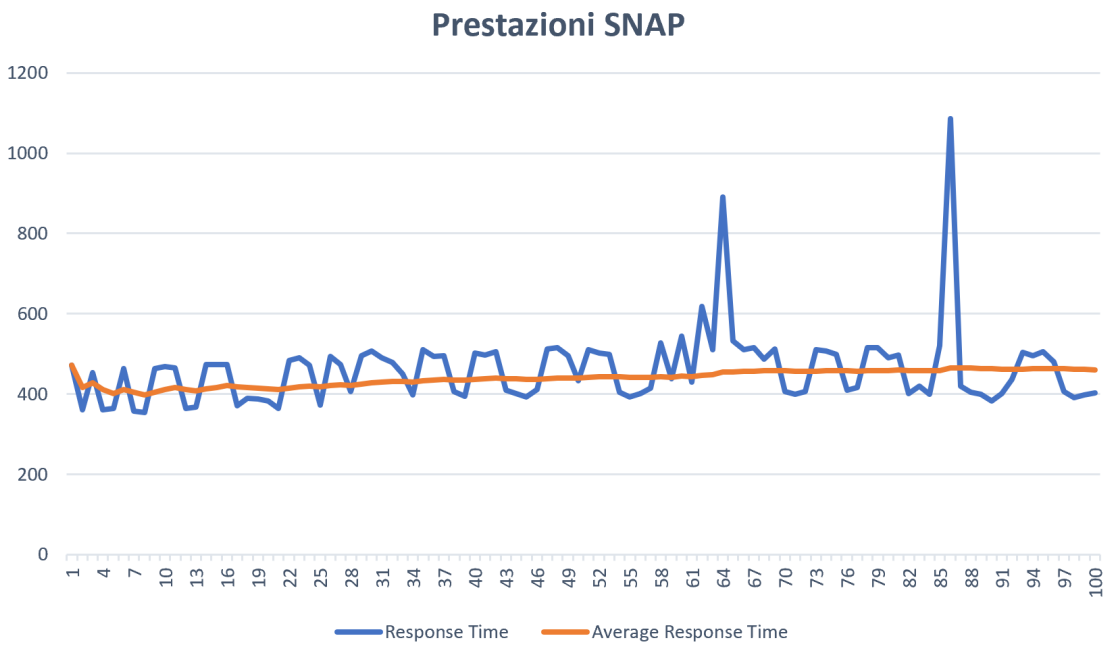


Figura 5.1: Tempo di risposta medio di SNAP

Di seguito, vengono mostrate delle immagini che mostrano i tempi medi di risposta di SNAP 2.0. Le analisi sono state svolte utilizzando gli strumenti forniti da Oracle Stream Analytics e si è deciso di analizzare solamente il servizio di *OfferService*, che risulta essere il più corposo, nonché il principale servizio. Nella Figura 5.2, si può notare il tempo medio complessivo, che risulta essere pari a 300 ms. Abbiamo un aumento di prestazioni rispetto al sistema originale ma bisogna considerare che le valutazioni non sono eque, in quanto non sono state fatte nelle stesse condizioni. Ciò nonostante il tempo medio risulta essere più che accettabile ed, inoltre, con buona probabilità, in una situazione reale in cui le varie componenti sono distribuite in un cluster di computer, si potrebbero ottenere prestazioni migliori.

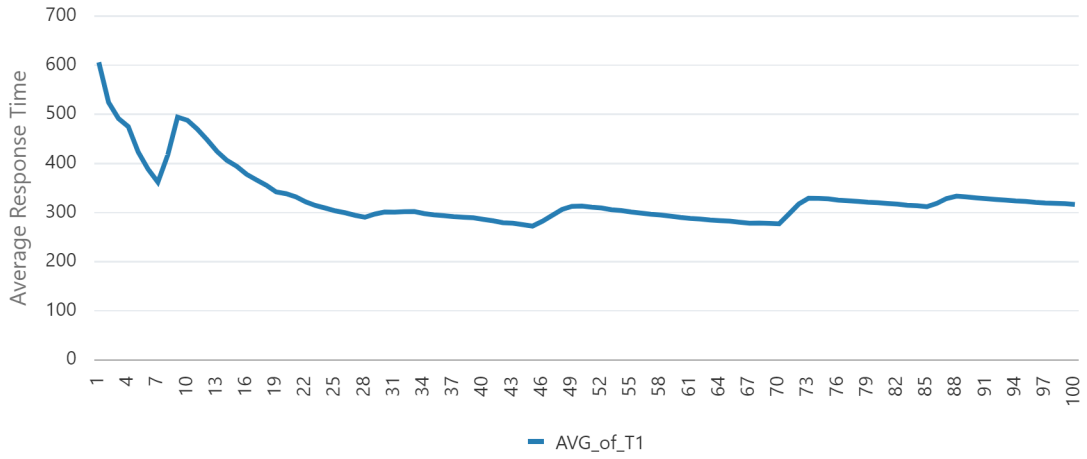


Figura 5.2: Tempo di risposta medio di SNAP 2.0

Adesso, si analizzano le singole fasi di processamento della richiesta, al fine di determinare quale delle operazioni richieda una maggiore attenzione in termini di ottimizzazione.

Per l'analisi dei singoli tempi si è scelto di considerare tre momenti:

- τ_{req}: instante di tempo in cui è stata fatta la richiesta.
- τ_{read}: instante di tempo in cui il dato è prelevato dallo stream.
- τ_{write}: instante di tempo in cui la risposta è stata elaborata.

Da questi instanti possiamo estrarre due intervalli di tempo definiti come segue:

$$\Delta\tau_{pull} = \tau_{read} - \tau_{req} \quad \wedge \quad \Delta\tau_{process} = \tau_{write} - \tau_{read}$$

Da cui si può ricavare:

$$\Delta\tau_{tot} \simeq \Delta\tau_{pull} + \Delta\tau_{process}$$

Il tempo totale di risposta è uguale, approssimativamente, alla somma dei due intervalli. Sono da considerarsi trascurabili i tempi di lettura dei *consumer*.

5.1.2 Kafka Monitoring

L'analisi delle metriche relative al cluster Kafka sono state condotte utilizzando il framework *Kafka-Monitor* [19]. Questo framework permette di effettuare dei test sul cluster in modo da collezionare le metriche necessarie alla

valutazione delle performance. *Kafka-monitor* fa uso di *app* e *servizi*. Questi ultimi vengono istanziati in thread separati e svolgono determinate azioni riportando le relative metriche.

Alcuni dei servizi che possono essere utilizzati sono:

- *ProduceService*

  Tramite questo servizio viene istanziato un producer, con i parametri specificati nel file di configurazione, che scriverà i messaggi nei topic. Nel mentre vengono svolte le scritture, il servizio misura l'affidabilità del cluster definita come:

$$produce\_availability = \frac{produce\_rate}{produced\_rate + error\_rate} \tag{5.1}$$

- *ConsumeService*

  Legge i record dai topic ed analizza le informazioni contenute nel payload dei record per ricavare i tempi di latenza end-to-end e verificare che alcuni messaggi non siano andati perduti.

Per entrambi i servizi è possibile specificare una classe Producer/Consumer custom purchè implementi l'interfaccia *com.linkedin.kmf.producer.KMBaseProducer*, altrimenti verrà usato il consumer/producer di default. Tramite l'uso di *kafka-monitor* è possibile istanziare più servizi contemporaneamente. L'interazione tra il framework e il cluster viene descritta in Figura 5.3.

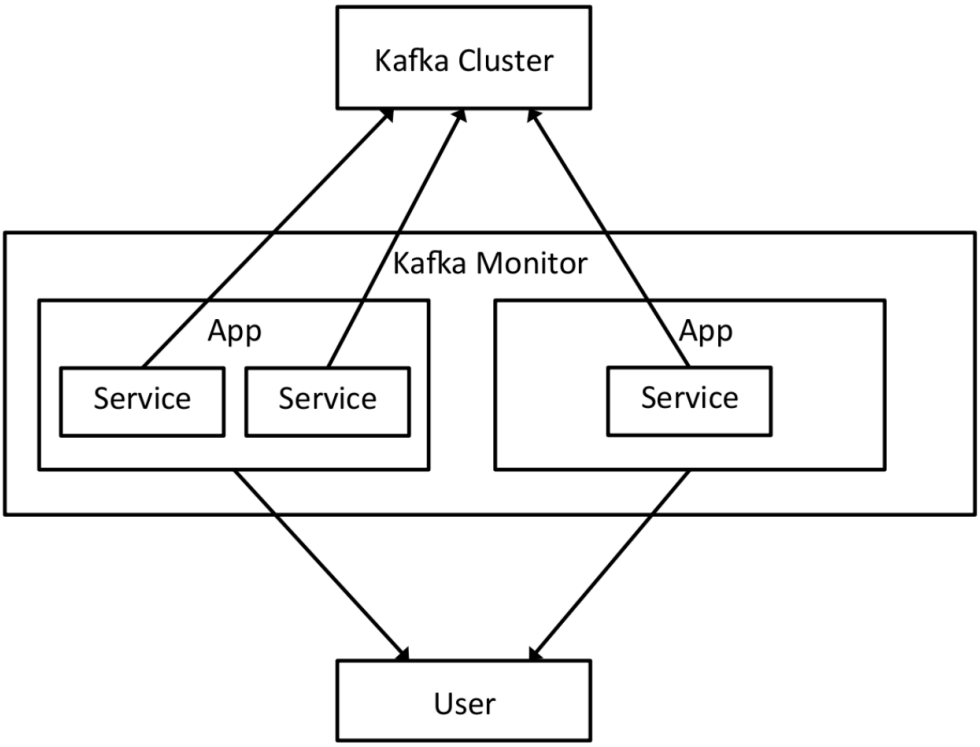


Figura 5.3: Architettura Kafka Monitor

Per l'elaborazione delle metriche viene avviata l'app denominata *SingleClusterMonitor* che include i servizi descritti in precedenza. Le analisi sono state condotte in una finestra temporale di circa 20 minuti e sono stati analizzati in totale 25000 record con un tasso di produzione che si aggira intorno a 19 record al secondo (si veda Figura 5.6).

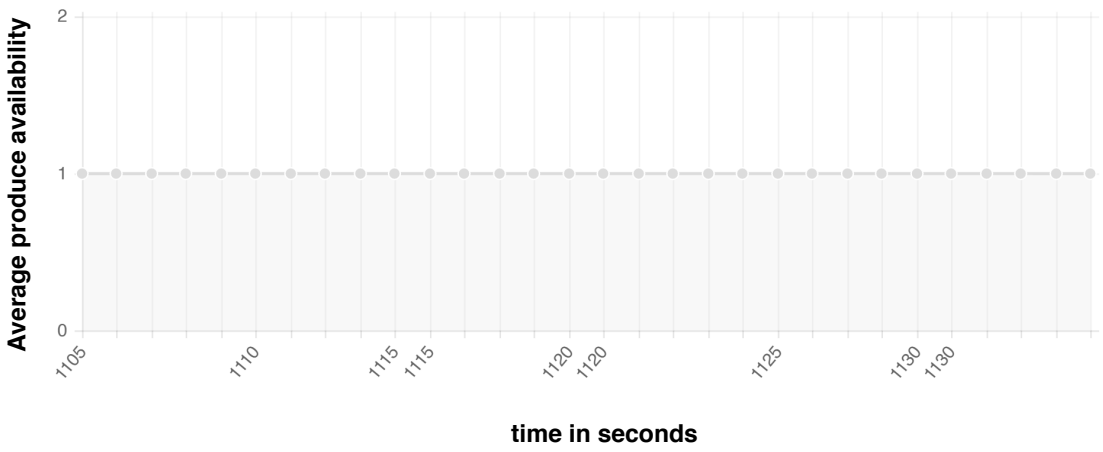


Figura 5.4: produce-availability-avg

In Figura 5.4 viene calcolata la media della frazione specifica in 5.1. La media è unitaria e di conseguenza l'*error_rate* è nullo. Ciò può essere confermato dalla Figura 5.5.

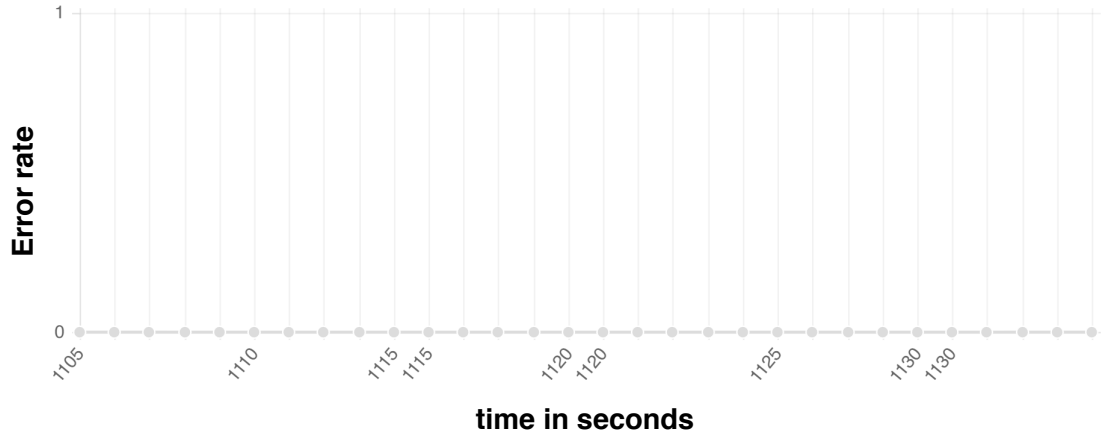


Figura 5.5: records-lost-rate

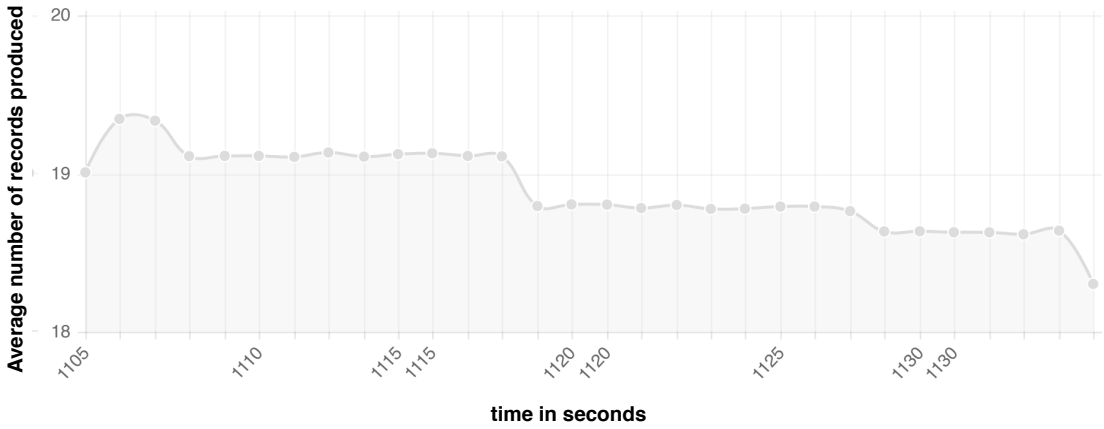


Figura 5.6: records-produced-rate

La Figura 5.7 riporta le informazioni relative alla metrica *produce-delay-ms-avg*, ovvero la media dei tempi di ingestion dei messaggi in millisecondi, che risulta essere di poco inferiore ai 5ms. Il tempo di ingestion rappresenta il tempo che impiega Kafka a salvare il record nella partizione del topic.

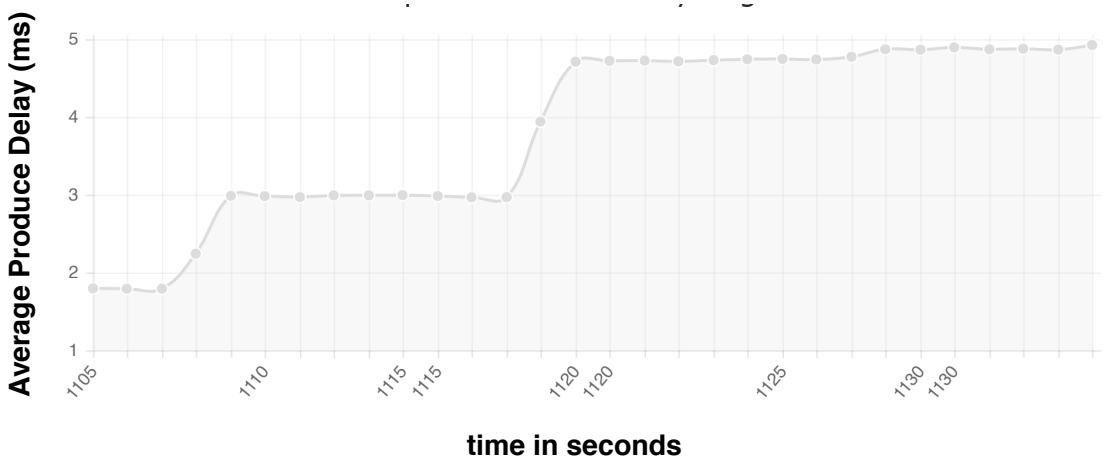


Figura 5.7: produce-delay-ms-avg

Infine, si riscontrano risultati soddisfacenti relativi alle operazioni effettuate dal consumer come si pùo notare dalla Figura 5.8. Il service relativo al consumer riesce a prelevare i record in un intervallo di tempo pari a circa 7 millisecondi.

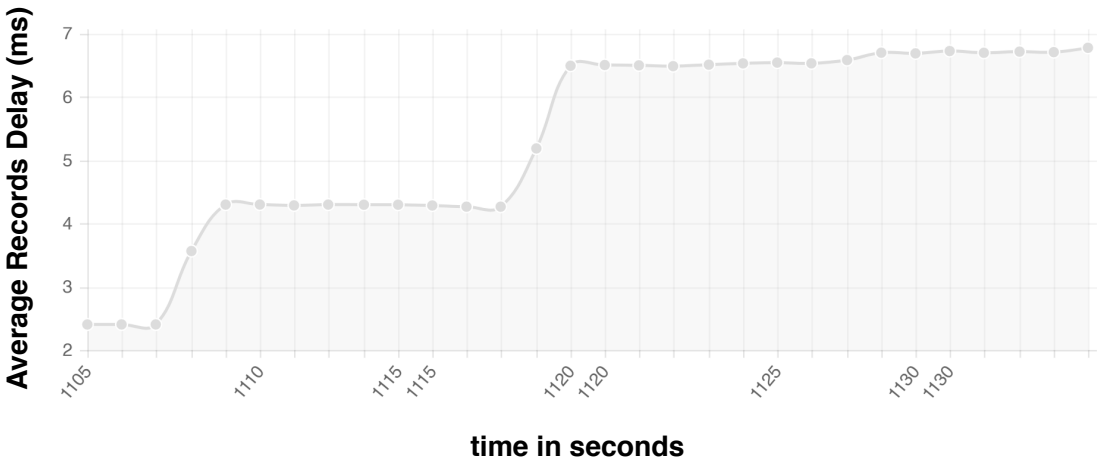


Figura 5.8: records-delay-ms-avg

5.1.3 Lettura Stream

Prima che la richiesta venga processata da OSA, è necessario che venga ricevuta dal Front-end, il quale instanzierà un Producer che invierà il dato al Kafka Topic. Qualunque dato presente nel Topic verrà elaborato secondo le specifiche della pipeline definita in OSA. In questa sezione, si fornisce una descrizione dei tempi necessari per la lettura del dato.

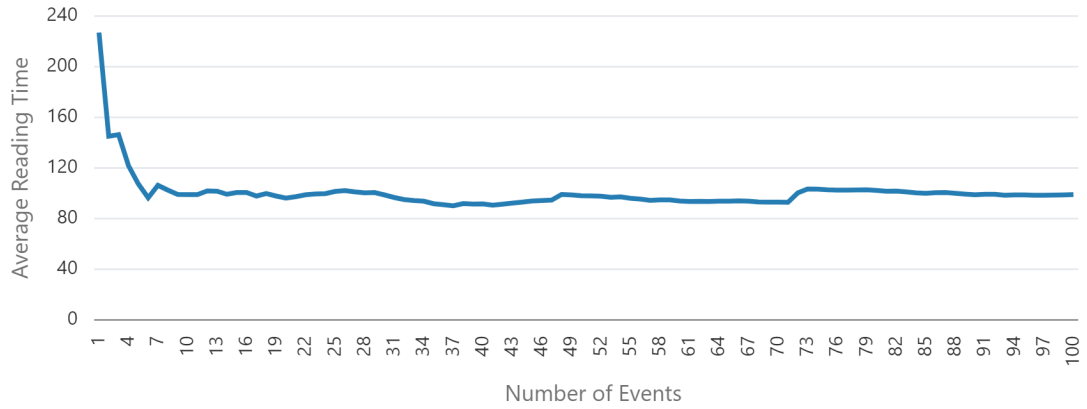


Figura 5.9: Tempo di risposta medio di lettura dallo Stream

Dalla Figura 5.9 si evince che il sistema ha un tempo di risposta medio pari a 100 ms. Il valore minimo registrato risulta essere pari a 93 m, mentre il massimo è equivalente a 128 ms. Si nota un picco nei primi pacchetti, ciò è dovuto ai tempi di *deploy* dell'applicazione relativa al particolare stage.

5.1.4 Elaborazione dell'offerta

In questa sezione ci si concentra sull'analisi dei tempi medi di elaborazione della richiesta al fine di selezionare l'offerta migliore per il cliente. Quest'ultima viene inviata attraverso uno stage di tipo *Target* al *Kafka Topic*, denominato *Topic-Offer-Response*. L'elaborazione comprende l'attivazione di un dato scenario e la selezione dei relativi banner. Questa operazione viene applicata per ciascun data e richiede alcune chiamate a dei database esterni, per ottenere i parametri necessari alla determinazione dell'offerta migliore.

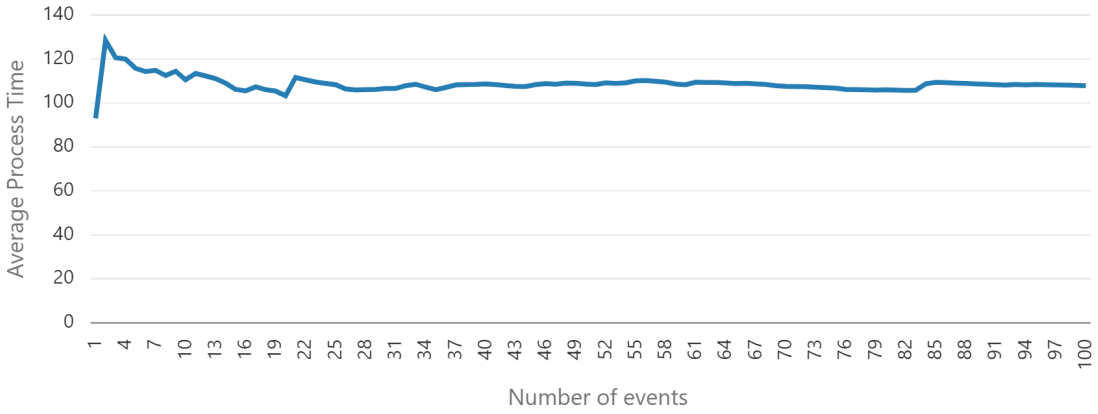


Figura 5.10: Tempo di risposta medio di processamento

In Figura 5.10 è possibile notare che non ci sono particolari variazioni di tempo per tutti i record. Il tempo medio di elaborazione si aggira sui 110 ms.

Ai tempi precedentemente citati vanno sommati i tempi di lettura dei Consumer. Nella Sezione 3.1.1 si è detto che un Consumer preleva i dati in intervalli di tempo decisi dal programmatore. Nel presente lavoro si è scelto di settare i tempi di *polling* a 10 ms.

Capitolo 6

Conclusioni

Durante il corso della presente tesi è stato sviluppato un nuovo sistema, che andrà a sostituire un prodotto attualmente in funzione, che permette di fornire un'offerta mirata ai clienti bancari sulla base di alcuni indicatori che tengano traccia dell'esperienza dell'utente all'interno del servizio di Home Banking. Le valutazioni finali hanno messo in luce come il sistema posso essere utilizzato per altri progetti, in quanto presenta dei tempi di risposta inferiori al sistema originario. È opportuno sottolineare che le comparazioni tra i due sistemi non possono essere eseguite in maniera equa. SNAP utilizza delle macchine special-purpose sulle quali sono eseguiti solamente i servizi necessari al suo funzionamento, mentre SNAP 2.0 viene eseguito su un PC dove sono attivi processi secondari che consumano risorse che potrebbero essere utilizzate dal sistema stesso. La valutazione risulta, quindi, compromessa per la presenza di questi fattori esterni. Nonostante questo, è stato possibile sviluppare questo prototipo che svolge le operazioni richieste in tempi più rapidi. Inoltre, ha permesso di darci un'idea del suo potenziale e di porre le basi per l'avvio di altri progetti che sfruttino le medesime tecnologie. La presente tesi ha cercato di fornire una spiegazione dettagliata dei due sistemi nei precedenti capitoli, rispettando le specifiche espresse dall'azienda presso cui si è svolto il tirocinio.

6.1 Sviluppi Futuri

L'azienda Reply è rimasta soddisfatta dalle tecnologie usate in questo progetto di tesi, che hanno rispettato a pieno le specifiche riportate nel Capitolo 4. D'altronde la soluzione Kafka è stata adottata da aziende famose come Netflix, Microsoft o Airbnb [15]. Il motivo è dovuto alla sua semplicità d'uso e

alle sue potenzialità. Con la diffusione di servizi basati sullo streaming di dati diventa necessaria l'adozione di questa tecnologia, in quanto fornisce una gestione automatica degli stream di dati, categorizzati in topic, che possono essere condivisi da differenti servizi, per una elaborazione dei flussi di dati più rapida ed efficiente possibile.

Oltre a Kafka, la soluzione Oracle Stream Analytics porta con sè altrettanti vantaggi. Si è deciso di sfruttare questa tecnologia per via della sua semplicità d'uso. Attualmente, qualunque modifica al sistema SNAP richiede un'interruzione dei servizi e la creazione di un progetto ad-hoc che implementi le specifiche richieste. Con Oracle Stream Analytics basta semplicemente sviluppare un jar che rispetti i requisiti di progetto e lo si inserisca nella pipeline. Inoltre, Oracle Stream Analytics permette la possibilità di sfruttare dei modelli per il Machine Learning. I progetti futuri su cui Reply punterà, sfrutteranno questa funzionalità.

Infine, con l'ausilio delle macchine di Reply, sarà possibile distribuire le unità logiche nei vari server della rete Reply, al fine di fornire un servizio efficiente e funzionale.

Bibliografia

[1] Alves,Alexandre, Robin J., and Williams, Lloyd. Getting Started with Oracle Complex Event Processing 11g. Packt, 2013. Web.

[2] Docs.oracle.com. (2019). Fusion Middleware Platform Developer's Guide for Oracle Real-Time Decisions - Contents. [online] Available at: https://-docs.oracle.com/cd/E23943_01/bi.1111/e16630/toc.htm [Accessed 9 Oct. 2019].

[3] Docs.oracle.com. (2019). About Integrating with Oracle RTD. [online] Available at: https://docs.oracle.com/cd/E23943_01/bi.1111/e16630/about_int.htm#BIRDG250 [Accessed 9 Oct. 2019].

[4] Aung, Thandar, Hla Yin Min, and Aung Htein Maw. "Performance Evaluation for Real-Time Messaging System in Big Data Pipeline Architecture." 2018 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC) (2018): 198-986. Web.

[5] Nishant Garg, "Apache Kafka", PACKT Publishing UK, October 2013

[6] Jay Kreps, Neha Narkhede, Jun Rao, Kafka: a Distributed Messaging System for Log Processing, May 2015

[7] Apache Kafka. (2019). Apache Kafka. [online] Available at: https://kafka.apache.org/ [Accessed 9 Oct. 2019].

[8] Hiraman, Bhole Rahul, Chapte Viresh M, and C. Karve Abhijeet. "A Study of Apache Kafka in Big Data Stream Processing." 2018 International Conference on Information , Communication, Engineering and Technology (ICICET) (2018): 1-3. Web.

[9] Narkhede, N., Shapira, G. and Palino, T. (2017). Kafka: the definitive guide. Beijing: O'Reilly.

[10] Docs.confluent.io. (2019). Producer Configurations — Confluent Platform. [online] Available at: https://docs.confluent.io/current/installation/configuration/producer-configs.html [Accessed 10 Oct. 2019].

[11] Apache Kafka. (2019). Apache Kafka. [online] Available at: https://kafka.apache.org/documentation/#producerapi [Accessed 10 Oct. 2019].
[12] Perera, Srinath & Suhothayan, Sriskandarajah. (2015). Solution patterns for Realtime Streaming Analytics. 10.1145/2675743.2774214.
[13] Docs.oracle.com. (2019). Introduction to Oracle Stream Analytics. [online] Available at: https://docs.oracle.com/middleware/1221/osa/using-streamanalytics/GUID-81454309-164B-4933-B972-A9FEF06159D0.htm#OEPSX107 [Accessed 15 Oct. 2019].
[14] Oracle.com. (2019). Oracle Stream Analytics. [online] Available at: https://www.oracle.com/middleware/technologies/stream-processing.html [Accessed 15 Oct. 2019].
[15] How, K., & breve, A. (2019). Apache Kafka in breve. Retrieved 27 October 2019, from https://www.ionos.it/digitalguide/server/know-how/apache-kafka-in-breve/
[16] http://cloudurable.com/blog/kafka-architecture-consumers/index.html
[17]
[18] Using Oracle Stream Analytics. (2019). Retrieved 28 October 2019, from https://docs.oracle.com/en/middleware/fusion-middleware/osa/18.1/using-stream-analytics/understanding-expression-builder-functions.html#GUID-6FD6D232-E056-4073-B942-4CD356ECA0E6
[19] GitHub. (2019). linkedin/kafka-monitor. [online] Available at: https://github.com/linkedin/kafka-monitor [Accessed 15 Nov. 2019].