Un'architettura multi-tenant ad alte prestazioni per la messaggistica server-to-server: Apache Pulsar

Ismam Abu
ismam.abu@studio.unibo.it

Aprile 2021

Il progetto ha lo scopo di studiare, presentare e testare le principali funzionalità e caratteristiche della tecnologia **Apache Pulsar** (https://pulsar.apache.org/). Apache Pulsar è un sistema open-source di messaggistica distribuita, basata su publisher / subscriber, nativa per il cloud e originariamente creata su Yahoo. È stato creato per alimentare le applicazioni critiche di Yahoo (e.g. Yahoo Mail, Yahoo Finance, Yahoo Sports).

Contents

1 Obiettivi del progetto

Il progetto sarà incentrato sullo studio e la comprensione delle tecnologie adottate da Apache Pulsar, con uno sguardo generale sul sistema distribuito in ambito reale e di utilizzo vero e proprio. Gli obiettivi principali sono:

- Pulsar Functions e relativi modelli di programmazione;
- Architettura dell'infrastruttura, Brokers, Clusters e Metadata storage (Apache ZooKeeper);
- Pulsar Geo-Replication, ovvero la replica dei dati dei messaggi archiviati in modo persistente su più cluster di un'istanza Pulsar;
- Interfaccia amministratore di Pulsar e relativo utilizzo;
- Pulsar IO, cenno allo spostamento dei dati al di fuori di Pulsar;
- Prove su un cluster Pulsar per valutare le modalità di errore e gli scenari di perdita dei messaggi.

Questi argomenti verranno in primo luogo studiati e successivamente spiegati nel dettaglio all'interno della relazione finale.
Pulsar supporta svariati linguaggi di programmazione lato client, questo progetto si concentrerà principalmente su Java ma non si esclude la possibilità di fare un confronto con le altre versioni (C#, Python).
Verranno fatti dei test da linea di comando (Pulsar CLI), ma anche attraverso delle interfacce.
In conclusione si farà un punto della situazione con tutti i vantaggi e svantaggi di tale tecnologia.

1.1 Piano di lavoro

Lato teorico; studio del funzionamento della tecnologia presentata, con particolare interesse sull'architettura dell'infrastruttura e le Pulsar Functions.
Lato pratico e di testing; test mirati a capire e valutare al meglio il funzionamento di questa tecnologia.
Nella relazione finale verranno spiegati nel dettaglio le parti teoriche principali insieme ai risultati che si otterranno dai test effettuati.

2 Introduzione a Pulsar

2.1 Che cos'è Apache Pulsar ?

Apache Pulsar è un sistema di messaggistica publisher-subscriber open source, sviluppato originariamente da Yahoo! ed assorbito successivamente da Apache Software Foundation nel 2016.

Pulsar ha come obiettivo quello di gestire grandi moli di messaggi scambiati su appositi canali di messaggistica fornendo un supporto a funzionalità come la replicazione geografica e il multi tenancy. E' altamente scalabile e può gestire i casi d'uso che prevedono un grande spostamento di dati.

Pulsar combina le caratteristiche migliori dei sistemi di messaggistica tradizionale come RabbitMQ e Apache Kafka, grazie a questo ha acquisito moltissima popolarità.

Nei seguenti sottocapitoli verranno date le definizioni principali dei concetti che sono alla base dell'architettura di Apache Pulsar.

2.2 Topic

I topic in Pulsar sono dei canali per la trasmissione di messaggi dai producer ai consumer, i topic sono la struttura in cui i messaggi vengono pubblicati.

2.3 Broker

Un broker di messaggi è un programma intermedio che traduce un messaggio dal protocollo di messaggistica del mittente al protocollo di messaggistica del ricevitore. I broker di messaggi sono elementi di telecomunicazione o reti di calcolatori in cui le applicazioni software comunicano scambiando messaggi.

Il broker dei messaggi di Pulsar è un componente **stateless** ed è responsabile dell'esecuzione di altri due componenti:

- **Un server HTTP**: si occupa esporre un'API REST sia per le attività amministrative che per la ricerca di Topics per producers e consumers. I producers si collegano ai broker per pubblicare i messaggi e i consumers si collegano ai broker per consumare i messaggi.
- **Un dispatcher**, è un server TCP asincrono su un protocollo binario personalizzato utilizzato per tutti i trasferimenti di dati.

2.4 Producers

Un producer è un processo che si collega ad un topic e si occupa pubblicare i messaggi su un broker Pulsar. Il broker Pulsar elabora i messaggi, che possono essere pubblicati sia in maniera sincrona che asincrona. È possibile avere diversi tipi di modalità di accesso per i producers:

- **Shared:** più producers pubblicano su un singolo topic, è il settaggio default.

- **Exclusive:** un solo producer scrive su un singolo topic, il vecchio producer viene rimosso e un nuovo producer viene selezionato per essere il prossimo producer esclusivo.
- **WaitForExclusive:** se c'è un producer connesso, allora la creazione del producer rimane in sospeso finché non si ottiene l'accesso esclusivo. Il producer esclusivo viene trattato come leader.

2.5 Consumers

Un consumer è un processo che si collega a un topic tramite una subscription e riceve i messaggi pubblicati su di esso.

Un consumer invia una richiesta di autorizzazione a un broker per ricevere messaggi. C'è una coda dal lato del consumer per ricevere i messaggi inviati dal broker. La ricezione del messaggio ha due tipi:

- **Ricezione Sincrona:** questa ricezione è bloccata finché non è disponibile un messaggio.
- **Ricezione Asincrona:** questa ricezione restituisce immediatamente una **future value** (ad esempio *CompletableFuture* in Java o le *Promise* in JavaScript) che viene completato quando è disponibile un nuovo messaggio.

2.6 Messaggi

Il messaggio è una delle unità alla base di Pulsar, essendo basato sul modello publish-subscribe, vuol dire che i *producer* pubblicano messaggi sui *topic* ed i *consumer* si iscrivono a tali topic, elaborano i messaggi in arrivo e inviano un *acknowledgement* al termine dell'elaborazione.

Quando viene creato un abbonamento, Pulsar conserva tutti i messaggi, anche se il consumer è disconnesso. I messaggi conservati vengono eliminati solo quando un consumer riconosce che tali messaggi sono stati elaborati correttamente.

2.7 Subscription

Una subscription è una regola di configurazione che determina il modo in cui i messaggi vengono recapitati ai consumers. In Pulsar sono disponibili quattro modalità di abbonamento: esclusiva, condivisa, failover e keyShared. Queste modalità sono illustrate in figura 1.

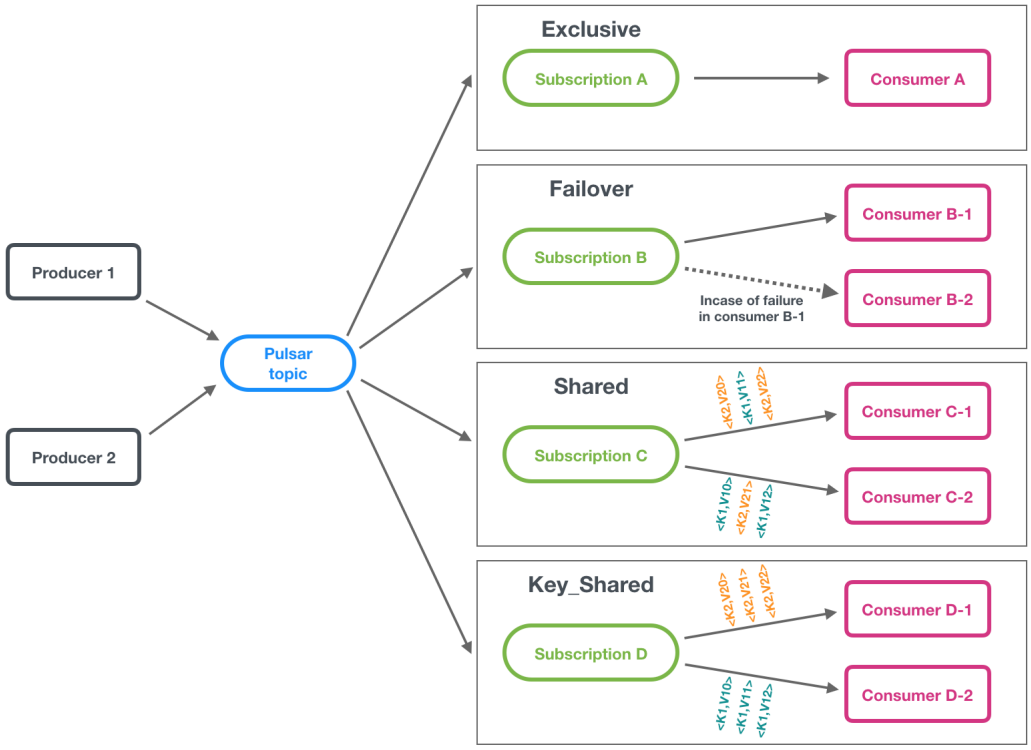

Figure 1: Diagramma che illustra le modalità di abbonamento delle subscriptions.

2.8 Cluster

Un'istanza Pulsar è costituita da uno o più cluster Pulsar. I cluster, a loro volta, sono costituiti da:

- Uno o più broker Pulsar.
- Un quorum ZooKeeper utilizzato per la configurazione e il coordinamento a livello di cluster.
- Un insieme di bookkeeper utilizzati per l'archiviazione persistente dei messaggi.

I cluster possono replicarsi tra loro utilizzando la replica geografica.

2.9 Apache BookKeeper

Apache BookKeeper è la tecnologia utilizzata da Pulsar che serve per rendere l'archiviazione dei messaggi persistente. BookKeeper è un sistema di log write-ahead distribuito (WAL) che fornisce una serie di vantaggi fondamentali:

- Consente a Pulsar di utilizzare molti registri indipendenti, chiamati **ledgers**. È possibile creare più ledgers per topics nel tempo.
- Offre uno storage molto efficiente per i dati e ne gestisce la replicazione.
- Garantisce la consistenza di lettura dei registri in presenza di vari errori di sistema.
- Offre una distribuzione uniforme dell'I/O tra i bookies.
- È scalabile orizzontalmente sia in termini di capacità che di throughput. La capacità può essere aumentata aggiungendo più bookies a un cluster.
- I bookies sono progettati per gestire migliaia di ledgers con letture e scritture simultanee. Utilizzando più dispositivi a disco, uno per il journal e l'altro per l'archiviazione generale, i bookies sono in grado di isolare gli effetti delle operazioni di lettura dalla latenza delle operazioni di scrittura in corso.

Oltre ai dati dei messaggi, in BookKeeper vengono memorizzati in modo permanente anche i **cursors**. I cursors sono posizioni di abbonamento per i consumers. BookKeeper consente a Pulsar di memorizzare la posizione del consumer in modo scalabile. Al momento, Pulsar supporta l'archiviazione permanente dei messaggi. In figura 2 è mostrato come i brokers e i bookies interagiscono.

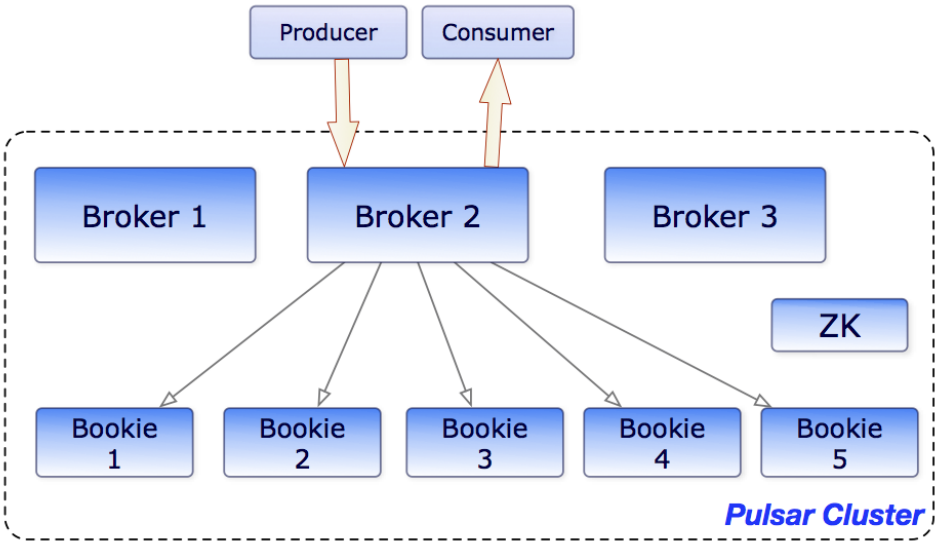

Figure 2: Diagramma che illustra le iterazioni tra i brokers e i bookies.

2.10 ZooKeeper

Ogni istanza Pulsar si basa su due quorum ZooKeeper separati (un quorum ZooKeeper è il numero minimo di nodi che devono essere attivi e funzionanti e disponibili per le richieste in arrivo).

1. **Local ZooKeeper**, opera a livello di cluster e fornisce la gestione e il coordinamento della configurazione specifica del cluster. Ogni cluster Pulsar deve avere un cluster ZooKeeper dedicato.

2. **Configuration Store**, opera a livello di istanza e fornisce la gestione della configurazione per l'intero sistema (e quindi tra i cluster). Un cluster indipendente di macchine o le stesse macchine utilizzate da ZooKeeper locale possono fornire il quorum dell'archivio di configurazione.

2.11 Ledgers

Un ledger è una struttura di dati immutabili con un singolo writer assegnato a più nodi di archiviazione di BookKeeper o bookmaker. Le entry vengono replicate su più bookmaker. I registri hanno una semantica molto semplice:

- Un broker Pulsar può creare, aggiungere voci e chiudere un ledger.
- Dopo che il ledger è stato chiuso (in modo esplicito o perché il processo di scrittura si è bloccato) può essere aperto solo in modalità di sola lettura.
- Infine, quando le voci nel ledger non sono più necessarie, l'intero ledger può essere eliminato dal sistema (in tutti i bookmaker).

2.11.1 Coerenza di lettura del ledger

Il principale punto di forza di BookKeeper è che garantisce la coerenza di lettura nei ledgers in presenza di errori. Poiché il ledger può essere scritto solo da un singolo processo, quel processo è libero di aggiungere voci in modo molto efficiente, senza bisogno di ottenere un permesso. Dopo un errore, il registro passerà attraverso un processo di ripristino che finalizzerà lo stato del registro e stabilirà quale voce è stata confermata per ultima nel registro. Dopo quel punto, tutti i lettori del ledger avranno la garanzia di vedere esattamente lo stesso contenuto.

2.12 Overview dell'architettura

Un'istanza Pulsar è composta da uno o più cluster Pulsar. I cluster all'interno di un'istanza possono replicare i dati tra di loro. In un cluster Pulsar abbiamo:

- In generale, uno o più broker:
  - gestiscono e bilanciano i messaggi in arrivo dai producers,
  - inviano messaggi ai consumers,
  - comunicano con l'archivio di configurazione Pulsar per gestire varie attività di coordinamento,
  - archiviano i messaggi in istanze di BookKeeper (ovvero bookies),

  - si basa su un cluster ZooKeeper specifico per determinate attività ed altro ancora.

- Un cluster BookKeeper è composto da uno o più bookies gestisce **l'archiviazione persistente** dei messaggi.
- Un cluster ZooKeeper specifico per quel cluster gestisce le attività di coordinamento tra i cluster Pulsar.

In figura 3 è illustrato un cluster Pulsar.

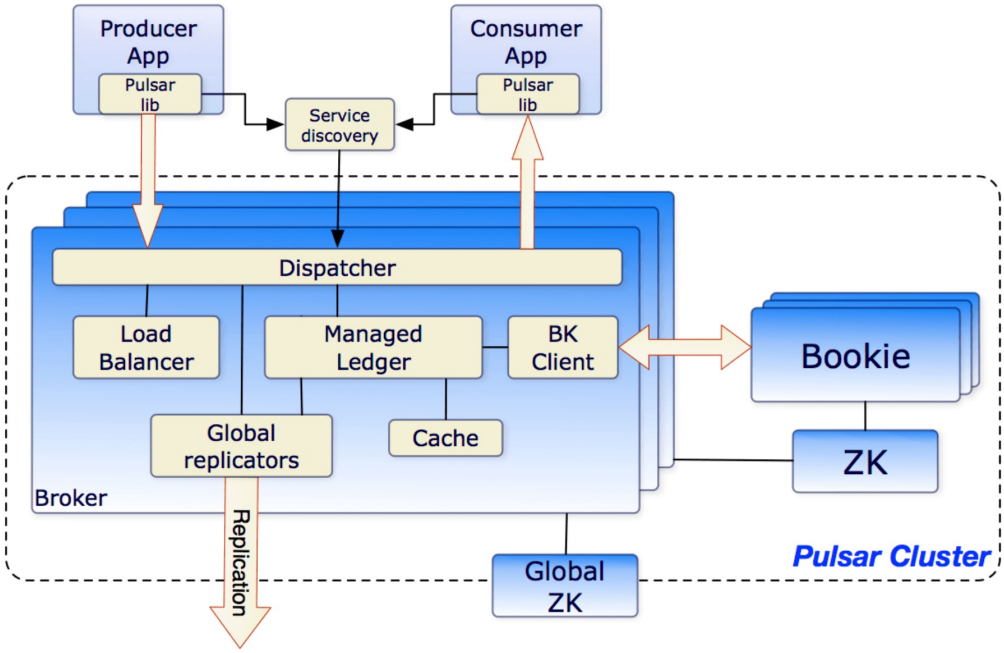

Figure 3: Diagramma che illustra come funziona un cluster Pulsar.

3 Caratteristiche alla base di Apache Pulsar

Di seguito verranno elencate le maggiori caratteristiche di Pulsar.

3.1 Modello Publish-Subscribe, streaming e code

Apache Pulsar è un sistema di messaggistica pub-sub, è nato con l'intento di gestire una alta mole di messaggi in tempo reale (uno streaming di messaggi). Si utilizza un modello complesso tipico delle code dei messaggi, questo modello però non è gestito dallo sviluppatore ma da Pulsar stesso.

3.2 Preservazione e ripetizione del messaggio

Generalmente, è necessario tenere traccia del fatto che un determinato messaggio sia stato effettivamente consumato, una volta che il client consuma il messaggio viene mandato un acknowledge al sistema comunicando che il messaggio sia stato effettivamente consumato. Generalmente, dopo questo, si provvede ad eliminare il messaggio, però dal momento che applicazioni possono bloccarsi e le zone di disponibilità possono diminuire, la possibilità di recuperare quel messaggio può essere fondamentale per ricostruire lo stato dell'applicazione, dunque la conservazione dei messaggi è importante. Se qualcosa va storto, Pulsar può riprodurre i messaggi che sono stati pubblicati su un topic, anche se sono già stati consumati. E' possibile che si abbia di nuovo bisogno di quel determinato messaggio.

La capacità di conservare i messaggi viene utilizzata anche per le architetture applicative basate sugli eventi, come l'origine degli eventi, in cui è importante registrare ogni cambiamento di stato come evento nell'ordine in cui si è verificato.

3.3 Progettazione basata su bassa latenza ed alta velocità di trasmissione

Fin dall'inizio, Pulsar è stato progettato per fornire una latenza bassa e costante insieme ad un throughput elevato. Lo fa separando due concetti:

- servire i messaggi tra producers e consumers;
- archiviare i messaggi per la persistenza.

Pulsar utilizza un'architettura multi-livello in cui i messaggi vengono serviti da broker e archiviati da Apache BookKeeper. Invece di costruire il proprio livello di archiviazione, Pulsar sfrutta BookKeeper per le sue prestazioni migliori.

BookKeeper è un registro distribuito progettato per archiviare in modo duraturo i messaggi con isolamento IO tra scrittura e lettura. Ciò significa che può fornire una latenza costante e bassa anche durante la scrittura o la lettura di grandi quantità di dati. A differenza dei sistemi di archiviazione tradizionali, le prestazioni non hanno problemi sotto un elevato livello di scrittura o sotto un elevato livello di lettura.

3.4 Architettura Cloud-Native

Poiché Apache Pulsar utilizza un approccio a più livelli, separando l'elaborazione (broker) dall'archiviazione (BookKeeper), si adatta molto bene alle infrastrutture cloud. I broker sono essenzialmente stateless e BookKeeper può essere facilmente gestito come StatefulSet in ambienti di gestione di container come Kubernetes e Docker, che è lo standard di fatto per il management cloud-native. Infatti Apache Pulsar funziona in maniera naturale all'interno Kubernetes, supportando aggiornamenti in sequenza, rollback e ridimensionamento orizzontale. Se abbinato a volumi di dati persistenti e supportati da cloud storage, Pulsar diventa un sistema di messaggistica altamente durevole e altamente flessibile che può scalare con facilità da piccole implementazioni di test a grandi implementazioni di produzione.

3.5 Conservazione infinita con archiviazione a più livelli

Un vantaggio dell'architettura multistrato di Pulsar è il fatto che sia possibile aggiungere nuovi livelli. Per prestazioni elevate, qualsiasi sistema di messaggistica persistente deve utilizzare dischi ad alte prestazioni, perché i messaggi devono essere scritti su disco e potrebbero dover essere necessario recuperarli in qualunque momento (se non vengono consumati immediatamente).

Abbiamo il problema della conservazione dei messaggi se desideriamo riprodurli nuovamente o effettuare una ricerca di eventi, quindi è necessario conservare quei messaggi. Si potrebbe utilizzare l'archiviazione su dischi ad alte prestazioni ma è molto costosa.

Per risolvere questo problema, Pulsar supporta l'archiviazione a più livelli, consentendo di scaricare i messaggi meno recenti su opzioni di archiviazione più economiche, come i bucket S3. Quando un consumer ha bisogno di un messaggio precedente, Pulsar lo recupera automaticamente dal bucket S3 e lo consegna al consumer.

Le prestazioni sono inferiori, ma quando si tratta di messaggi vecchi di mesi o addirittura anni, le prestazioni contano poco, basta solo che quei messaggi siano disponibili quando se ne ha bisogno, senza spendere troppo.

3.6 Multi-tenancy e namespace

Dopo aver messo in piedi un sistema di messaggi, lo si deve condividere tra i vari team all'interno di un'organizzazione, quindi non ha senso dover replicare il sistema per assicurarsi che team diversi non si influenzino a vicenda o creare un sistema di sovrapposizione complesso per simulare il multi-tenancy.

Pulsar è stato progettato fin dall'inizio per essere un sistema multi-tenant, questo consente a più gruppi di utenti di condividere lo stesso cluster. Ogni tenant ha la propria autenticazione, autorizzazione e criteri. Inoltre, i tenant possono essere ulteriormente suddivisi in namespaces, il che semplifica il supporto di ambienti diversi, ad esempio all'interno di un singolo tenant, lo sviluppo, la gestione temporanea e la produzione.

Senza multi-tenancy, è necessario creare un livello di astrazione sopra il sistema di messaggistica o utilizzare un cluster completamente nuovo per un diverso gruppo di utenti.

3.7 Schema registry integrato

Una delle maggiori sfide di qualsiasi sistema di messaggistica è assicurarsi che producers e consumers non abbiano conflitti sul formato dei messaggi. Poiché producers e consumers sono disaccoppiati, è facile che uno o entrambi cambino il formato dei messaggi che stanno inviando o si aspettano di ricevere e le applicazioni finiscano per non funzionare.

La soluzione a questo è un registro dello schema che impone ai producers ed ai consumers di utilizzare messaggi con uno schema compatibile. Pulsar include un registro dello schema pronto all'uso. Serve semplicemente registrare lo schema con un topic e Pulsar si occupa di far rispettare le regole dello schema.

3.8 Geo-replicazione integrata

La replicazione dei messaggi in posizioni geografiche remote è importante per supportare il ripristino di emergenza o per consentire alle applicazioni di operare su scala globale. Se gli utenti della propria applicazione si spostano in tutto il mondo, si vuole che abbiano la stessa esperienza utente, indipendentemente da dove si trovino. Con la replica geografica, le applicazioni si connettono ad un cluster locale ma possono inviare e ricevere ai cluster in tutto il mondo.

Con Pulsar, la replica geografica dei messaggi è integrata. Se si pubblica un messaggio su un topic in un namespace replicato, quel messaggio viene replicato automaticamente nella posizione geografica o nelle posizioni remote configurate. Non sono necessarie configurazioni complesse o componenti aggiuntivi.

3.9 Flessibilità delle sottoscrizioni

Apache Pulsar supporta quattro diversi tipi di abbonamento: esclusivo, failover, condiviso e a chiave condivisa. Supporta anche più abbonamenti su un singolo topic. Utilizzando gli abbonamenti, si possono configurare facilmente i modelli di messaggistica, come code, pub-sub, fan-out e consumers concorrenti.

Pulsar implementa il modello dei consumers concorrenti utilizzando i suoi abbonamenti condivisi. Si può aumentare e diminuire il numero di consumers su un abbonamento condiviso senza problemi. Non ci sono partizioni coinvolte. Basta aggiungere un consumer ed inizia subito a ricevere messaggi.

3.10 Funzioni di streaming integrate

Di solito si vogliono ottenere informazioni dai dati che si stanno raccogliendo in tempo reale. Per ottenere insight in tempo reale, i dati devono essere elaborati in tempo reale. Con Pulsar, si possono integrare senza problemi funzioni per leggere nel flusso di messaggi, eseguendo la pulizia, l'arricchimento e l'analisi dei dati in tempo reale, quindi non è necessario scaricare tutto ed elaborarlo in un secondo momento. Con le funzioni Pulsar, si possono elaborare i dati mentre scorrono attraverso il sistema di messaggistica. Le funzioni Pulsar possono essere scritte in Java, Python o Go e possono essere configurate per essere eseguite come pod Kubernetes.

3.11 Connettori IO

Una delle funzioni principali di un sistema di messaggistica è quella di fondere insieme sistemi data-intensive come database, motori di elaborazione del flusso e altri sistemi di messaggistica. Poiché questo è comune, ha senso fornire un framework e connettori comuni per renderlo facile da usare. Questo è esattamente ciò che fa Pulsar con i suoi connettori IO.

Pulsar fornisce una serie di connettori già pronti che eseguono all'interno del cluster Pulsar e che facilitano la fusione dei propri sistemi. Pulsar viene fornito con un'ampia varietà di connettori, tra cui MySQL, MongoDb, Cassandra, RabbitMQ, Kafka, Flume, Redis e molti altri.

3.12 Topics partizionati e non-partizionati

Apache Pulsar supporta topics sia partizionati che non partizionati. Per casi d'uso con prestazioni inferiori, si può usare un topic non partizionato per semplificare le cose.

Tuttavia, se si ha a che fare con un caso d'uso ad alte prestazioni, in cui è necessario elaborare un volume elevato di dati su un singolo topic, è possibile utilizzare un topic partizionato per sfruttare il parallelismo nell'elaborazione. Si possono aggiungere facilmente partizioni man mano che i requisiti di prestazioni aumentano.

Pulsar è in grado di garantire l'ordine dei messaggi se si pubblica il proprio messaggio con le chiavi. Pulsar assegnerà messaggi con la stessa chiave alla stessa partizione, garantendo l'ordine per i messaggi inviati a quella chiave.

3.13 Messaggi persistenti e non persistenti

I messaggi persistenti vengono inviati ad Apache BookKeeper per l'archiviazione su disco. Questi messaggi sono garantiti per essere consegnati almeno una volta indipendentemente da un eventuale guasto della rete, dell'applicazione o persino dello stesso Pulsar.

Tuttavia, in alcuni casi non è richiesto questo livello di garanzie. Al massimo una consegna è sufficiente. In questi casi, Apache Pulsar supporta i messaggi non persistenti. I messaggi non persistenti non vengono archiviati su disco, riducendo i requisiti di risorse pur offrendo un throughput elevato e una bassa latenza.

3.14 Compressione del topic

A volte è interessante solo l'ultima istanza di un dato, non interessano tutti i valori storici ma solamente il valore più recente. In tal caso, si può utilizzare un topic compresso per memorizzare solo il valore più recente su una chiave particolare in un topic.

Tutti i dati vengono pubblicati in un topic compatto, Pulsar rimuoverà periodicamente i vecchi valori per una chiave, lasciando solo l'ultimo. I topics compatti impediscono al topic di crescere indefinitamente e consentono di accedere rapidamente ai valori più recenti su un topic.

3.15 Librerie Client

Pulsar dispone di un'ampia varietà di librerie client gestite dal progetto principale: Java, Python, C++, Golang, Node.js e C#. Se non si vuole utilizzare una libreria client Pulsar, Pulsar include un proxy WebSockets.

Ci sono molti altri client sviluppati dalla comunità, come Scala e Rust. E se si preferisce utilizzare HTTP per inviare e ricevere messaggi Pulsar, lo si può fare utilizzando Pulsar Beam.

4 Perché Pulsar ?

Al giorno d'oggi, Pulsar è una delle due soluzioni possibili poter utilizzare un sistema di messaggistica pub/sub scalabile, l'altro possibile candidato è Apache Kafka. Kafka è un sistema open source ed un riferimento per realizzare applicazioni di streaming distribuite. Pulsar d'altro canto offre il potenziale di un throughput più veloce e una latenza inferiore rispetto a Kafka in molte situazioni, insieme a un'API che consente agli sviluppatori di passare da Kafka a Pulsar con relativa facilità.

In questo capitolo verrà spiegato quale dei due sistemi scegliere e in che cosa differiscono.

4.1 Kafka

Sviluppato da LinkedIn e rilasciato come open source nel 2011, Apache Kafka si è diffuso in lungo e in largo, diventando praticamente la scelta predefinita per molti quando si vuole aggiungere un sistema pub/sub alla propria architettura.

Il caso d'uso standard per molte pipeline negli ultimi anni è stato quello di inserire i dati in Apache Kafka e quindi utilizzare tecnologie come *Apache Storm* o *Apache Spark* per estrarre dati, eseguire ed elaborare e successivamente pubblicare output su un altro topic per il consumo a valle.

Dal punto di vista dello sviluppatore, Apache Kafka è sempre stato molto user-friendly, mentre invece dal punto di vista operativo è stato un po' un miscuglio. Ottenere un piccolo cluster attivo e funzionante è relativamente facile, ma mantenere un cluster di grandi dimensioni è spesso problematico.

Kafka per supportare il multi-tenancy utilizza un'utilità chiamata MirrorMaker. Esso però è stato fonte di problemi tali per cui aziende come Uber hanno creato un proprio sistema di replicazione geografica nei datacenter.

Nel 2014, tre degli sviluppatori originali di Kafka (Jun Rao, Jay Kreps e Neha Narkhede) hanno formato Confluent, questa fornisce funzionalità aziendali aggiuntive nella sua piattaforma Confluent come Replicator, Control Center e plug-in di sicurezza aggiuntivi. Confluent ha anche un'offerta cloud, Confluent Cloud, che è un servizio Confluent Platform completamente gestito che viene eseguito su Amazon Web Services o Google Cloud Platform, se non si vuole affrontare da soli la parte del sovraccarico operativo dei cluster in esecuzione.

Se si è bloccati in AWS e si utilizzano i servizi Amazon, si tenga presente che Amazon ha introdotto un'anteprima pubblica di Amazon Managed Streaming for Kafka (MSK), che è un servizio Kafka completamente gestito all'interno dell'ecosistema AWS. (Si noti anche che Amazon MSK non è fornito in collaborazione con Confluent, quindi l'esecuzione di MSK non offre tutte le funzionalità di Confluent Platform, ma solo ciò che è fornito nell'open source Apache Kafka.)

4.2 Apache Kafka o Apache Pulsar?

Apache Kafka è un prodotto datato, motivo per cui è stato molto approfondito e testato sul campo. Ha client scritti in quasi tutte le lingue più diffuse, oltre a una serie di connettori supportati per diverse origini dati in Kafka Connect. Con i servizi gestiti offerti da Amazon e Confluent, è facile ottenere un cluster Kafka di grandi dimensioni installato, funzionante e gestito, molto più semplice rispetto agli anni precedenti.

Tuttavia, se si ha intenzione di creare un sistema di messaggistica che deve essere multi-tenant o replicato geograficamente dall'inizio, o che ha grandi esigenze di archiviazione dei dati, oltre alla necessità di interrogare ed elaborare facilmente tutti quei dati, non importa come molto tempo fa in passato, quindi è consigliabile adottare Apache Pulsar. Si adatta sicuramente ad alcuni casi d'uso con cui Apache Kafka può avere difficoltà, pur funzionando bene in termini di funzionalità di base necessarie da una piattaforma di log distribuita. In più offre documentazione all'avanguardia ed una folta serie di risposte alle domande poste su Stack Overflow.

5 Pulsar Functions

5.1 Cosa sono le pulsar functions ?

Le Pulsar Functions sono l'infrastruttura che si occupa di elaborare i messaggi all'interno del sistema, consentono la creazione di complesse logiche di elaborazione in base al messaggio e permettono di sfruttare il concetto di serverless (consentendo agli sviluppatori di eseguire applicazioni senza gestire i server) nello streaming degli eventi, eliminando così l'utilizzo sistemi esterni come Apache Storm ed Apache Heron.

Queste funzioni si occupano di leggere e consumare i messaggi da uno o più topics Pulsar, applicano una logica di elaborazione che deve essere fornita dall'utente e successivamente i risultati del calcolo vengono pubblicati su altri topics.

Gli sviluppatori non devono apprendere nuove API, chiunque conosca Java, ad esempio, può utilizzare l'SDK di Java per scrivere una funzione, per esempio:

```
1 import java.util.function.Function;
2
3 public class ExclamationFunction implements Function<String, String>
4   @Override
5   public String apply(String input) {
6       return input + "!";
7   }
8 }
```

Listing 1: Esempio basilare di Pulsar Function in Java

L'obiettivo delle Pulsar Functions è poter permettere di utilizzare una API semplice e un framework di esecuzione per i casi dove è necessario effettuare il filtraggio, il routing e l'arricchimento dei messaggi.

Tutti possono scrivere una Pulsar Functions, inviarla al cluster Pulsar e utilizzarla immediatamente con le funzionalità di gestione integrate in Pulsar Functions. Ha un'API REST basata su CRUD, uno sviluppatore può anche inviare funzioni da qualsiasi workflow e renderle immediatamente operative.

Le funzioni Pulsar aumentano la produttività degli sviluppatori, forniscono una risoluzione dei problemi più semplice e semplicità operativa perché non è necessario un sistema di elaborazione esterno. Utilizzano un'API semplice e leggera senza SDK e un framework di esecuzione per il novanta percento dei casi d'uso di streaming che riguardano il filtraggio, il routing e l'arricchimento.

5.2 Modello di programmazione

Le Pulsar Functions forniscono un'ampia gamma di funzionalità e il modello di programmazione di base è semplice. Le funzioni ricevono messaggi da uno o più topics di input, ogni volta che viene ricevuto un messaggio, la funzione completerà le seguenti attività:

- Applicare una logica di elaborazione all'input e scrivere l'output su:
  - Un topic di output in Pulsar.
  - Apache Bookkeeper.

- Scrivere i log nel log topic.
- Incrementare un contatore.

E' possibile utilizzare le funzioni Pulsar per impostare la seguente catena di elaborazione:

- Una funzione Python ascolta il topic delle frasi non elaborate e "disinfetta" le stringhe in ingresso (rimuovendo gli spazi bianchi estranei e convertendo tutti i caratteri in minuscolo) e quindi pubblica i risultati in un topic delle frasi sanificate.
- Una funzione Java ascolta il topic delle frasi elaborate, conta il numero di volte in cui ogni parola appare entro una finestra di tempo specificata e pubblica i risultati in un topic dei risultati
- Infine, una funzione Python ascolta il topic dei risultati e scrive i risultati in una tabella MySQL.

5.3 Workflow dell'invio

L'elaborazione del workflow di invio di una funzione viene chiamato rappresentazione della funzione. La sua struttura, chiamata *FunctionConfig*, ha un tenant, un namespace ed un nome.

Dopo aver inviato la funzione, vengono eseguiti o un controllo di invio o le convalide per verificare che l'utente disponga dei privilegi per inviare la funzione al namespace e al tenant specifici. Per Java, le classi vengono caricate al momento dell'invio per assicurarsi che siano effettivamente nel file JAR. Tutto questo viene fatto in modo che l'utente riceva un messaggio di errore il più rapidamente possibile, evitando di dover guardare i log degli errori stessi.

Il passaggio successivo consiste nel copiare il codice in BookKeeper. Tutti i parametri del codice sono rappresentati come *FunctionMetaData* in una struttura di buffer di protocollo come di seguito:

```
message FunctionMetaData {
    FunctionDetails functionDetails;
    PackageLocationMetaData packageLocation;
    uint64 version ;
    uint64 createTime;
    map<int32 , FunctionState> instanceStates ;
    FunctionAuthenticationSpec functionAuthSpec ;
}
```

Listing 2: Struttura del buffer di protocollo

La struttura FunctionMetaData è interamente gestita dal Function MetaData Manager. Dal punto di vista del worker, la funzione MetaData Manager mantiene il sistema di registrazione. Mappa dal *Fully Qualified Function Name* (FQFN) ai metadati della funzione. Inoltre, aggiorna e gestisce gli stati della macchina, nonché eventuali conflitti quando vengono inviati più Worker, in base a ciò che viene inviato, scrive i metadati nel topic.

5.4 Scheduling dei workflows

Una volta che il sistema ha accettato una funzione, questa viene pianificata utilizzando *Pluggable Scheduler*. Viene invocato ogni volta che una nuova funzione viene inviata ed eseguita da un Leader. Un leader viene eletto avendo una subscription di failover su un topic di coordinamento. Il consumer del Topic viene quindi eletto come Leader.

Il leader scrive i compiti sui topics, noti come topics di assegnazione. Esistono all'interno di un particolare spazio dei nomi all'interno di Pulsar e sono assegnati a singoli Worker. Tutti gli worker conoscono tutte le assegnazioni che sono compresse e includono tutta la logica di sistema come il FQFN e l'ID dell'istanza all'interno delle tabelle delle assegnazioni.

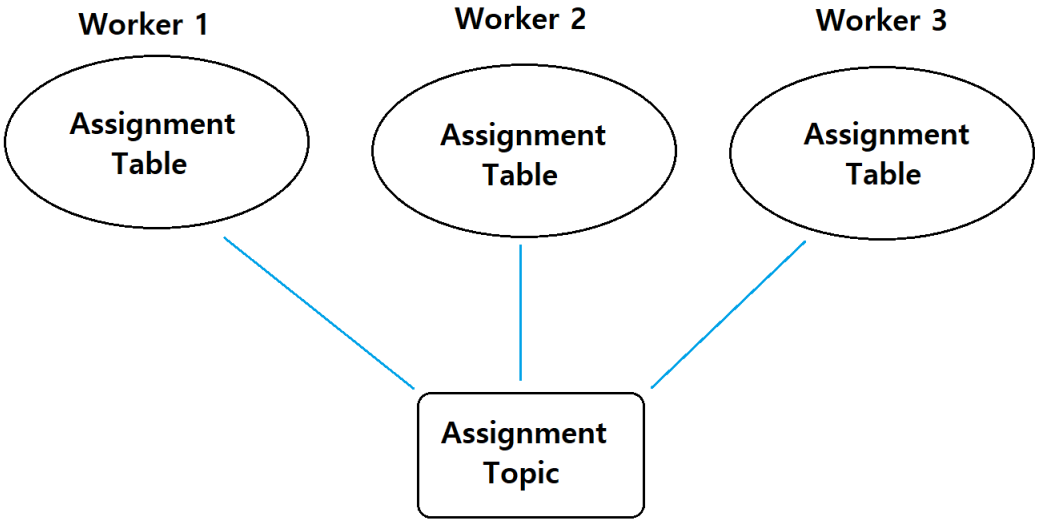

Figure 4: Diagramma che illustra lo Scheduling degli workflows.

5.5 Execution Workflows

L'execution workflow viene attivato dalle modifiche alla tabella di assegnazione. I componenti all'interno del worker, chiamati *Function RunTime Manager*, gestiscono l'assegnazione del ciclo di vita della funzione come l'avvio o l'arresto di un messaggio utilizzando uno Spawner.

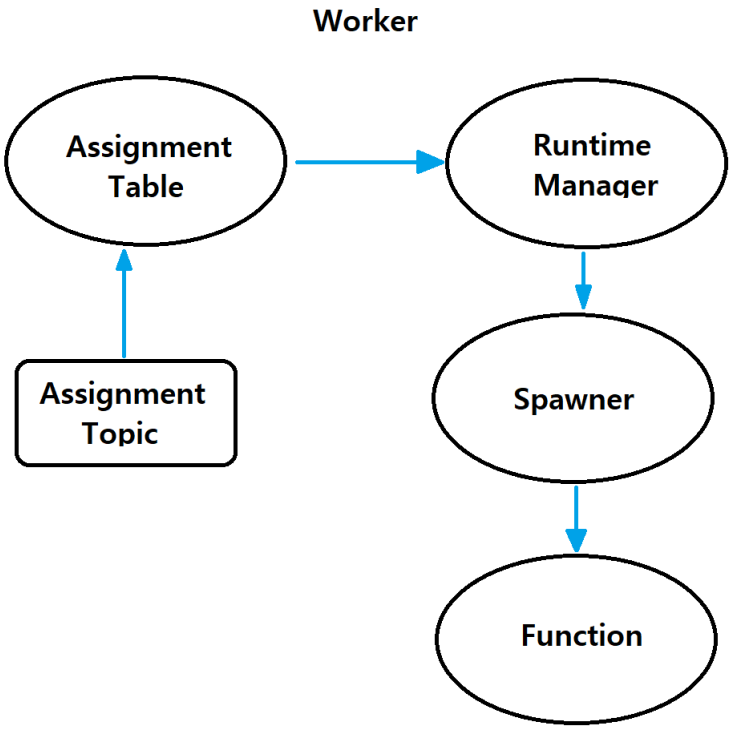

Figure 5: Diagramma che illustra l'execution workflow.

6 Pulsar IO

I connettori Pulsar IO permettono di creare, distribuire e gestire facilmente i connettori che i sistemi esterni (come Cassandra, Aerospike e tanti altri) utilizzano per interagire con Pulsar.

Una delle domande più comuni che sorgono quando si utilizzano piattaforme come Pulsar è: qual è il modo migliore per spostare i dati dentro e fuori dalla piattaforma ? Gli utenti possono scrivere codice personalizzato utilizzando le API consumer e producer di Pulsar per realizzare lo spostamento dei dati desiderato, ma esistono alternative più semplici. La domanda che ci si pone è dove e come l'applicazione deve essere eseguita per pubblicare o consumare dati da Pulsar? Altri sistemi di messaggistica non hanno modi organizzati per essere gestiti o gestire applicazioni che inviano o ricevono dati da e verso sistemi esterni, ciò costringe a creare soluzioni personalizzate ed eseguirle manualmente.

Per risolvere questi problemi è stato creato Pulsar IO, un framework che consente di inviare o ricevere dati da e verso Pulsar utilizzando il framework Pulsar Functions esistente. In questo modo si hanno tutti i vantaggi Pulsar Functions (parallelismo, elasticità ecc) e possono essere utilizzati dalle applicazioni che inviano e ricevono dati da Pulsar.

6.1 Architettura di Pulsar IO

Un'applicazione che immette dati all'interno di Pulsar viene chiamata *Source*, mentre un'applicazione che estrapola dati da Pulsar viene detta *Sink*. Quindi abbiamo che una source acquisisce i dati da un sistema esterno e li scrive su un topic Pulsar, mentre un Sink legge i dati da uno o più topic Pulsar e li scrive su un sistema esterno.

Il framework Pulsar IO viene eseguito al di sopra del framework delle funzioni Pulsar esistente. Singole source e sink possono essere eseguiti come qualsiasi funzione insieme ai broker Pulsar. Pertanto, tutti i vantaggi del framework Pulsar Functions sono disponibili anche per il framework Pulsar IO, ad esempio applicazioni sink e source.

Come accennato in precedenza, l'obiettivo è che gli utenti non debbano scrivere alcuna applicazione personalizzata o alcun codice per l'ingresso e l'uscita dei dati da e verso Pulsar. Pertanto, sono presenti un'ampia varietà di source e sink integrati (Kafka, Twitter Firehose, Cassandra, Aerospike, ecc.) che gli utenti possono eseguire con un solo comando. Ciò consente all'utente di non doversi preoccupare dei dettagli di livello implementativo.

6.2 Vantaggi dell'utilizzo di Pulsar IO Framework

Come accennato in precedenza, il framework Pulsar IO viene eseguito al di sopra del framework delle funzioni Pulsar esistente. Sfruttando il framework Pulsar Functions, le source e i sink che fanno parte di Pulsar IO ottengono tutti i vantaggi esistenti delle funzioni Pulsar:

Beneficio	**Dettaglio**
Flessibilità di esecuzione	Source e sink possono essere eseguiti come parte di un cluster esistente o come processo autonomo
Parallelismo	Per aumentare la velocità effettiva di un sink o di una source, è possibile eseguire più istanze di source e sink semplicemente aggiungendo una configurazione.
Bilanciamento del carico	Se source e sink vengono eseguiti in modalità "cluster"
Tolleranza agli errori, monitoraggio e metriche	Se le source e i sink vengono eseguiti in modalità "cluster", il servizio monitorerà automaticamente le source e i sink distribuiti. Quando i nodi falliscono, le source e i sink vengono ridistribuiti ai nodi operativi. Anche le metriche vengono raccolte automaticamente
Aggiornamenti dinamici	Il parallelismo di ciascun connettore, il codice source, i topics di ingresso e uscita e molte altre configurazioni possono essere modificati al volo
Località dei dati	Poiché i broker servono le richieste di lettura e scrittura sui topics, l'esecuzione di source e sink adiacenti ai broker può ridurre la latenza di rete e l'utilizzo della larghezza di banda della rete

7 Geo-replicazione

La Geo-replicazione è la replicazione dei messaggi archiviati in modo permanente su più cluster di un'istanza Pulsar. I modelli di replicazione adottati da Apache Pulsar sono:

- Replica asincrona
- Replica sincrona

7.1 Funzionamento della geo-replicazione

Nella figura 6, quando i producers P1, P2 e P3 pubblicano dei messaggi sull'argomento T1, i messaggi vengono replicati istantaneamente tra i loro cluster, rispettivamente Cluster-A, Cluster-B e Cluster-C. Una volta che i messaggi sono stati replicati, i consumatori C1 e C2 possono consumare quei messaggi dai rispettivi cluster. Senza la replica geografica, i consumatori C1 e C2 non sarebbero in grado di utilizzare i messaggi pubblicati dal producer P3.

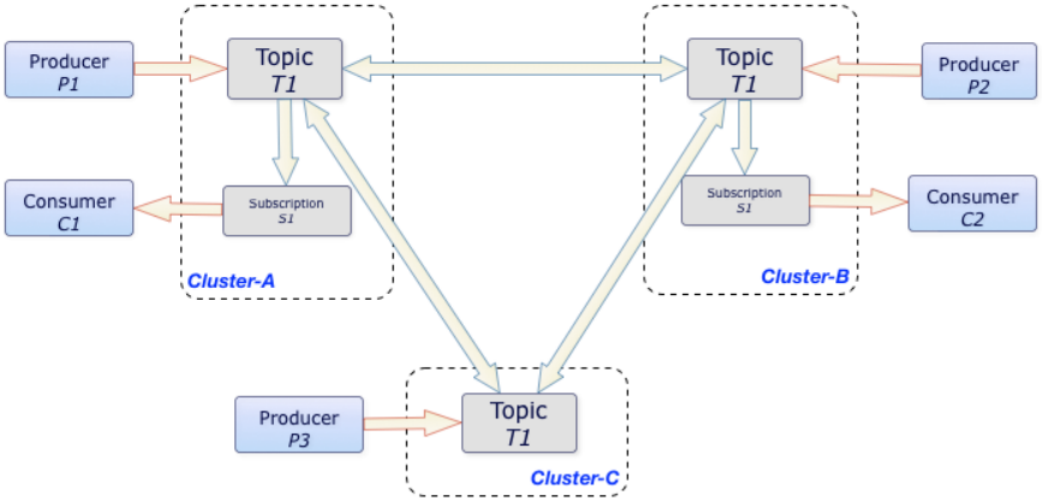

Figure 6: Diagramma che illustra il processo di replica geografica tra i cluster A, B e C.

La replica geografica è abilitata in base al tenant in Pulsar, può essere abilitata tra due cluster specifici solo quando un tenant ha accesso ad entrambi i cluster. Questo è gestito a livello di namespaces, consentendo di creare e configurare un namespace da replicare tra due o più cluster di cui è stato eseguito il provisioning a cui può accedere un tenant. Qualsiasi messaggio pubblicato su qualsiasi topic di quel namespace verrà quindi replicato in tutti i cluster nel set specificato.

7.2 Replicazione asincrona

Nella replicazione asincrona, i dati pubblicati sul topic vengono replicati in cluster situati in aree diverse. La replica asincrona può essere configurata a livello di namespace per il quale il tenant ha accesso ad essa. I producers vengono riconosciuti non appena il messaggio viene pubblicato/persistito localmente nel cluster. Il broker replica quindi i dati nel cluster di replica. Quando i dati vengono replicati, si conserva l'ordine ma non la posizione del cursore (la posizione del cursore viene preservata e mantenuta all'interno del cluster locale). La replicazione asincrona ha tre modelli differenti:

1. Attivo - Attivo
2. Failover
3. Replica aggregata

7.2.1 Replicazione Attiva - Attiva

Il diagramma in figura 7 mostra i cluster attivi - attivi tra le regioni, due in questo esempio. I dati vengono replicati tra US-East e US-West, date le autorizzazioni a livello di namespace. Il producer/consumer può produrre/consumare messaggi localmente nel cluster perché la subscription è locale nel cluster. Oppure possiamo avere un solo consumer per consumare i messaggi quando il producer produce messaggi in entrambi i cluster. Questo modello produce lo stesso set di dati tra le regioni. Dal punto di vista dell'utente, i dati elaborati a est vengono replicati anche a ovest per essere riprodotti. Man mano che lo stesso set di dati viene pubblicato tra i cluster, verrà elaborato nei sistemi di backend. Se il sistema di backend non viene replicato, questo è il modello da utilizzare. Se il sistema prevede un insieme univoco di messaggi, è responsabilità dell'applicazione che lo utilizza filtrare prima dell'inoltro a sistemi di backend o implementazioni diverse.

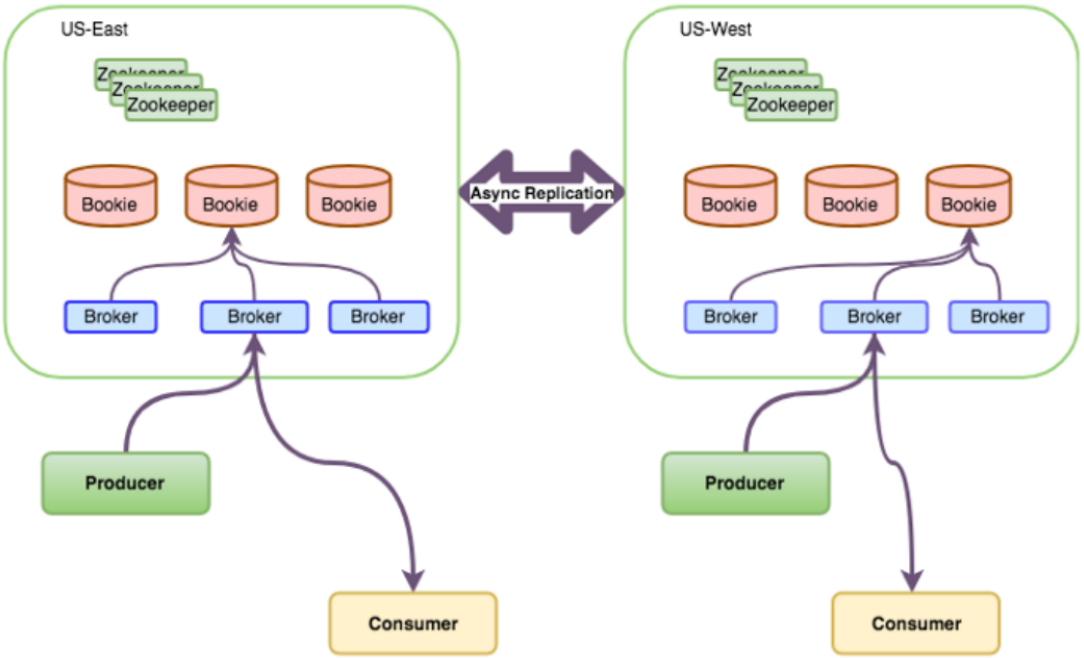

Figure 7: Diagramma che illustra la replicazione attiva - attiva.

7.2.2 Replicazione Failover

L'impostazione dei cluster è come illustrato in figura 8, tranne per il fatto che abbiamo un consumer per il cluster. Poiché i consumatori sono locali nel cluster con subscription localizzata. Quando il cluster US-East si interrompe, il producer e il consumer possono essere spostati nel cluster US-West per conservare i messaggi anche in assenza di abbonamento attivo. I dati non vengono persi in questo evento poiché la replica avviene da US-East a US-West. Quando avviamo il consumer su US-West, dobbiamo reimpostare il cursore per riprodurre il messaggio dal momento in cui non è stato possibile avviare l'elaborazione. Il ripristino del cursore si riferisce all'ora in cui si è verificato l'evento. Man mano che l'ordine viene mantenuto, l'applicazione può iniziare a elaborare i messaggi dal punto in cui è stata interrotta. Lo svantaggio principale di questo modello è migrare manualmente il producer/consumer, reimpostare il cursore per riprodurre il messaggio che può comportare una minore percentuale di duplicati che si basa sul livello di precisione del ripristino del cursore.

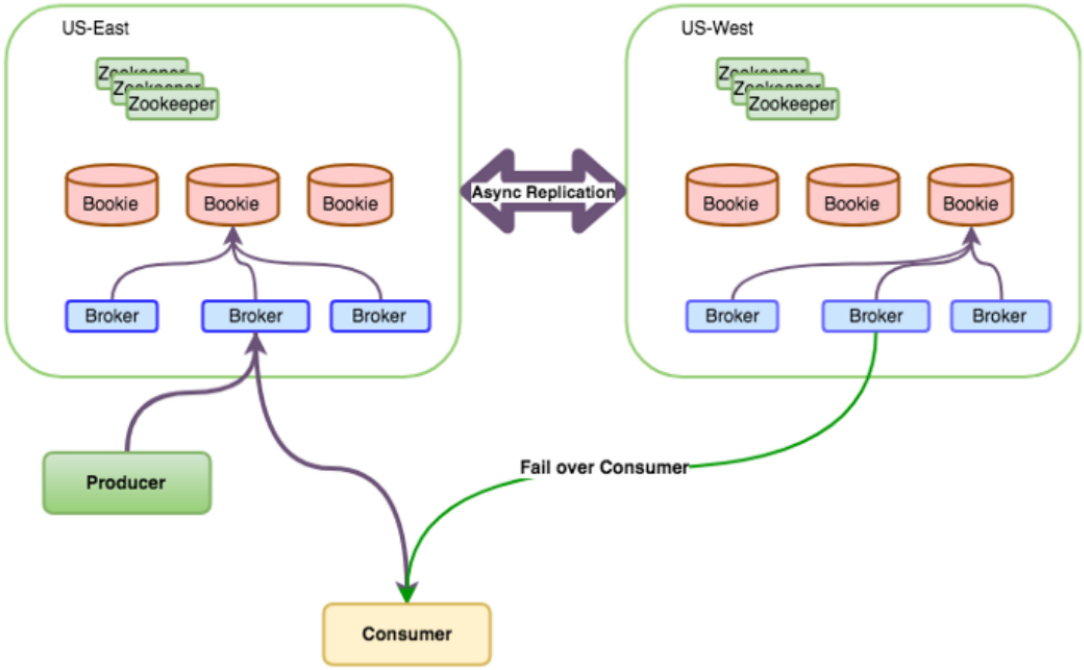

Figure 8: Diagramma che illustra la replicazione failover.

7.3 Replicazione sincrona

La replica sincrona offre all'utilizzatore una maggiore flessibilità per disporre di un cluster. La configurazione principale consiste nell'abilitare *rackAwarePlacementPolicy* sul broker per mantenere i dati nei bookkeeper all'interno delle regioni. Questa funzione è fuori dagli schemi di pulsar. I vantaggi si vedono quando una regione improvvisamente va offline, in questo caso un'altra regione riprende attivamente da dove è stata interrotta la precedente senza duplicati e interventi manuali. L'aspetto negativo di questo schema è la complessità nell'installarlo e farlo funzionare in *bare metal*. I modelli precedenti possono essere avviati come contenitori in kubernetes. Un altro difetto è lo zookeeper, dobbiamo mantenere un numero dispari di cluster come nell'immagine in figura 9.

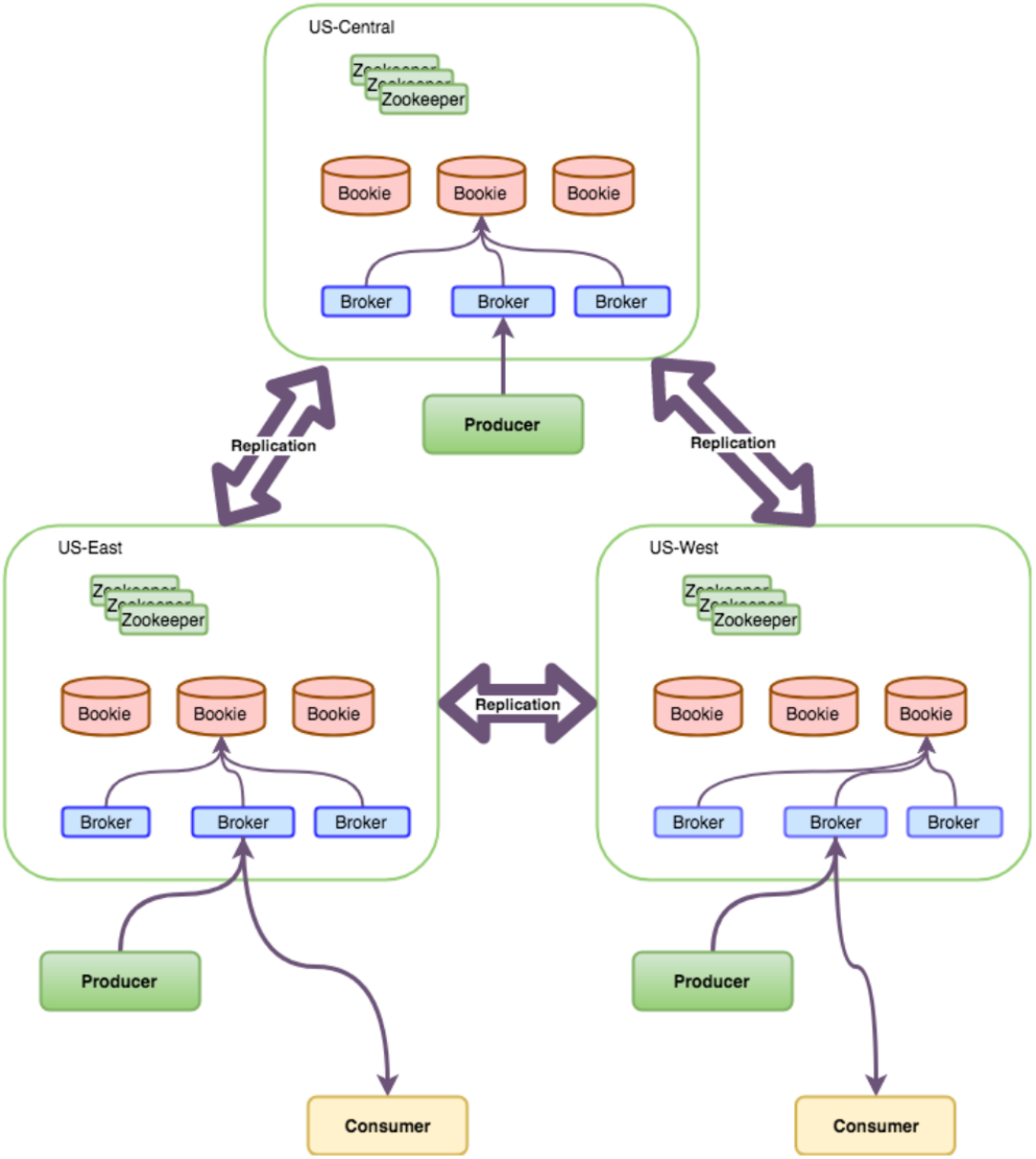

Figure 9: Diagramma che illustra la replicazione sincrona, abbiamo 3 cluster dentro 3 regioni.

8 Installazioni

Nella prima parte di questa sezione verrà illustrato la maniera corretta per installare ed eseguire Pulsar, nella seconda c'è una piccola guida per utilizzare l'applicazione usando la libreria fornita da Pulsar.

8.1 Set up di Pulsar in versione standalone

La modalità standalone include un broker Pulsar, i componenti di ZooKeeper e BookKeeper che devono essere in esecuzione all'interno di un singolo processo JVM. Ecco i passi per installare usando il binario rilasciato da Apache:

- Scaricare il binario attraverso il comando:

```
$ wget https://archive.apache.org/dist/pulsar/pulsar-2.8.0/apache-pulsar-2.8.0-bin.tar.gz
```

- Una volta scaricato, decomprimere il file ed entrare nella directory risultante:

```
$ tar xvfz apache-pulsar-2.8.0-bin.tar.gz
$ cd apache-pulsar-2.8.0
```

8.1.1 Contenuto del pacchetto scaricato

Il pacchetto binario scaricato contiene inizialmente le seguenti directory:

Directory	Contenuto
bin	Strumenti da riga di comando, come pulsar e pulsar-admin.
conf	File di configurazione per Pulsar, come file di configurazione del broker, dello ZooKeeper e altro.
examples	Un file JAR contenente Pulsar Functions di esempio.
lib	Files JAR utilizzato da Pulsar.
licenses	File di licenza, in formato .txt, per vari componenti del codebase Pulsar.

Una volta eseguito Pulsar vengono create ulteriormente queste directory:

Directory	Contenuto
data	La directory di archiviazione dei dati utilizzata da ZooKeeper e BookKeeper.
istances	Artefatti creati per le Pulsar Functions.
logs	Log creati dall'installazione.

8.1.2 Eseguire Pulsar Stand Alone

Una volta scaricata una copia locale della release, è possibile avviare un cluster locale usando il comando pulsar, che è memorizzato nella directory bin, specificando che si vuole avviare Pulsar in modalità standalone.

```
$ cd bin
$ ./pulsar standalone
```

Sempre da dentro la cartella bin è possibile attraverso il comando *./pulsar* visualizzare tutti i comandi che si possono utilizzare.

```
Usage: pulsar <command>
where command is one of:

    broker              Run a broker server
    bookie              Run a bookie server
    zookeeper           Run a zookeeper server
    configuration-store Run a configuration-store server
    discovery           Run a discovery server
    proxy               Run a pulsar proxy
    websocket           Run a web socket proxy server
    functions-worker    Run a functions worker server
    sql-worker          Run a sql worker server
    sql                 Run sql CLI
    standalone          Run a broker server with local bookies and local
    zookeeper

    initialize-cluster-metadata     One-time metadata initialization
    delete-cluster-metadata         Delete a clusters metadata
    initialize-transaction-coordinator-metadata     One-time transaction
    coordinator metadata initialization
    initialize-namespace     namespace initialization
    compact-topic       Run compaction against a topic
    zookeeper-shell     Open a ZK shell client
    broker-tool         CLI to operate a specific broker
    tokens              Utility to create authentication tokens
    version             Get the current version of pulsar

    help                This help message

or command is the full name of a class with a defined main() method.
```

Pulsar in versione standalone aiuta a capire il funzionamento della tecnologia, per poter provare in maniera più avanzata le possibilità che offre è necessario utilizzare alcuni dei comandi mostrati in precedenza.

Come abbiamo detto in precedenza pulsar è composto da due sottosistemi esterni essenziali, ZooKeeper e BookKeeper, nelle seguenti sezioni verrà illustrato come si è creato il cluster di pulsar che poi è servito nei vari test che sono stati creati.

8.2 Creazione del cluster Pulsar

In questa sottosezione verrà illustrato come creare un cluster Pulsar. Per alcune prove sono necessari più cluster, quindi se li si vogliono provare è necessario mettere in piedi più clusters seguendo le seguenti procedure.

8.2.1 Deploy locale di ZooKeeper

Come abbiamo detto, ZooKeeper serve per gestire le attività essenziali relative al coordinamento e alla configurazione di Pulsar. Per distribuire un'istanza di Pulsar è necessario creare un cluster ZooKeeper locale per cluster Pulsar.

All'interno del file zookeeper.conf è necessario apportare le seguenti modifiche: Per iniziare, occorre aggiungere tutti i server ZooKeeper alla configurazione specificata in conf/zookeeper.conf (contenuta nella cartella che si è scaricato).

```
server.1=127.0.0.1:2888:3888
server.2=127.0.0.1:4888:5888
server.3=127.0.0.1:6888:7888
```

Ora è necessario creare la directory *nodeX/zookeeperData/myid* e specificare all'interno del file l'ID del nodo, nel nostro caso, per semplicità è uguale a X di "nodeX".

Per gestire un cluster Pulsar è necessario gestire un archivio di configurazione per la gestione di alcune attività di coordinamento. Se si vuole gestire un cluster Pulsar singolo allora non è necessario gestire l'archivio, se invece si vuole distribuire un'istanza di Pulsar multi-cluster allora va gestito anche questo caso. Qui verrà illustrato solo Pulsar single-cluster.

La configurazione di ZooKeeper è gestita da due file di configurazione, una è *conf/zookeeper.conf* e serve per l'installazione in locale e *conf/global_zookeeper.conf* per l'archivio di configurazione.

Dopo aver aggiunto ogni server alla configurazione di zooKeeper.conf e le relative voci myid, possiamo avviare ZooKeeper su tutti gli host utilizzando il seguente comando:

```
$ bin/pulsar-daemon start zookeeper
```

8.2.2 BookKeeper

Per configurare i bookies di BookKeeper si utilizza il file *conf/bookkeeper.conf*. Quando si deve configurare ogni bookmaker serve assicurarsi che il parametro *zkServers* sia impostato sulla stringa di connessione per lo ZooKeeper locale. Queste sono le modifiche minime necessarie per la configurazione.

```
# Change to point to journal disk mount point
journalDirectory=.../cluster/node1/journalData

# Point to ledger storage disk mount point
ledgerDirectories=data/bookkeeper/ledgers

# Point to local ZK quorum
zkServers=localhost:2181
```

Ogni broker Pulsar possiede il proprio gruppo di bookies, il cluster BookKeeper condivide un quorum ZooKeeper con il cluster pulsar.

Per avviare un bookie in foreground si usa il seguente comando:

```
$ bin/bookkeeper bookie
```

Per verificare che il bookie sia stato avviato correttamente e che funzioni si usa il seguente comando:

```
$ bin/bookkeeper shell bookiesanity
```

Attraverso questo comando si crea un nuovo ledger nel bookmaker locale, si scrivono alcune voci, le si rileggono e alla fine si elimina il ledger.

8.2.3 Inizializzazione Pulsar Metadata

E' necessario inizializzare i metadati del cluster appena creato, per inizializzare tali metadati è necessario lanciare il seguente comando:

```
$ bin/pulsar initialize-cluster-metadata \
  --cluster node1 \
  --zookeeper 127.0.0.1:2181,127.0.0.1:2182,127.0.0.1:2183 \
  --configuration-store 127.0.0.1:2181,127.0.0.1:2182,127.0.0.1:2183 \
  --web-service-url http://127.0.0.1:8080 \
  --broker-service-url pulsar://127.0.0.1:6650
```

8.2.4 Conclusioni

Una volta avviato correttamente il cluster Pulsar, con il suo ZooKeeper, BookKeeper e inizializzati i metadati possiamo provare a produrre un messaggio. Tramite il seguente comando si produce un messaggio su un certo topic:

```
$ bin/pulsar-client produce \
  persistent://public/default/test \
  -n 1 \
  -m "Hello Pulsar"
```

Se tutto è andato a buon fine dovrebbe vedersi la produzione del messaggio sul topic.

8.3 Pulsar Functions

Ci sono due modi per provare le Pulsar Functions, la prima consiste nell'utilizzarle da fuori il cluster (attraverso la modalità standalone) e la seconda eseguirle all'interno del cluster.

8.3.1 Modalità esterna

Per provare le pulsar functions è necessario un cluster, può essere creato utilizzando la modalità standalone illustrata in precedenza.
Un esempio di funzione pulsar potrebbe essere la seguente:

```
import java.util.function.Function;

public class ExclamationFunction implements Function<String, String> {
    @Override
    public String apply(String input) {
        return String.format("%s!", input);
    }
}
```

Come input viene presa una stringa da un topic, alla stringa in input viene aggiunto un punto esclamativo alla fine e quindi viene pubblicata la nuova stringa manipolata su un altro topic.

Il contenuto delle funzioni (in Java) deve essere contenuta in un file JAR, per eseguire le funzioni in locale è necessario eseguire il seguente comando:

```
$ bin/pulsar-admin functions localrun \
  --jar examples/api-examples.jar \
  --classname org.apache.pulsar.functions.api.examples.
    ExclamationFunction \
  --inputs persistent://public/default/exclamation-input \
  --output persistent://public/default/exclamation-output \
  --name exclamation
```

E' possibile anche inserire più topic di input, nell'esempio è settato a uno, ma modificando il parametro –inputs si possono specificare più topics.

Ora apriamo un'altra shell e utilizziamo pulsar-client per ascoltare i messaggi sull'argomento di output:

```
$ bin/pulsar-client consume persistent://public/default/exclamation-
    output \
  --subscription-name my-subscription \
  --num-messages 0
```

Il parametro –num-messages 0 vuole dire che il consumer ascolterà il topic all'infinito (senza accettare un numero specifico di messaggi).

Con un listener attivo, possiamo aprire un'altra shell e produrre un messaggio sul topic in input:

```
$ bin/pulsar-client produce persistent://public/default/exclamation-input
    \
  --num-produce 1 \
  --messages "Hello world"
```

Se tutto è andato correttamente dovremmo ottenere il messaggio "Hello World!" sul consumatore.

8.3.2 Modalità interna

Questa modalità è più adatta allo sviluppo, nelle distribuzioni reali di Pulsar si procede in questa maniera. In questo modo le funzioni Pulsar vengono eseguite all'interno del cluster e vengono gestite utilizzando al stessa interfaccia delle funzioni pulsar-admin. Si può utilizzare un cluster creato come spiegato primo sotto-capitolo creando prima localmente ZooKeeper e poi un BookKeeper.

Con il seguente comando si effettua il deploy della funzione di esclamazione illustrata prima:

```
$ bin/pulsar-admin functions create \
  --jar examples/api-examples.jar \
  --classname org.apache.pulsar.functions.api.examples.
    ExclamationFunction \
  --inputs persistent://public/default/exclamation-input \
  --output persistent://public/default/exclamation-output \
  --name exclamation
```

Una volta lanciato il comando, nel caso in qui tutto sia andato correttamente verrà stampato *"Created successfully"*, ora possiamo guardare la lista delle funzioni all'interno del nostro cluster:

```
$ bin/pulsar-admin functions list \
  --tenant public \
  --namespace default
```

Per ottenere tutte le informazioni riguardo la funzione che abbiamo messo in esecuzione possiamo lanciare il seguente comando:

```
$ bin/pulsar-admin functions get \
  --tenant public \
  --namespace default \
  --name exclamation
```

Per eliminare una funzione dal nostro cluster possiamo eseguire il seguente comando:

```
$ bin/pulsar-admin functions delete \
  --tenant public \
  --namespace default \
  --name exclamation
```

8.3.3 Conclusioni

Nelle due sotto-sezioni precedenti è stato spiegato come eseguire il setup delle Pulsar Functions, per provarle, come mostrato, si possono mandare dei semplici messaggi da linea di comando. Si può anche utilizzare l'applicazione con la gui nel repository e utilizzare quella per testare le varie funzioni.

8.4 Pulsar Georeplication

In questa sotto-sezione verrà illustrato come si è testata la replicazione geografica su più cluster di pulsar. Verranno creati 3 cluster pulsar, Cluster A, Cluster B e Cluster C.

Come prima operazione è necessario connettere i cluster creati, per farlo si utilizza il comando pulsar-admin, nel codice sottostante è mostrato come connettere il cluster A con il cluster C, all'interno del cluster A lanciare il seguente comando:

```
$ bin/pulsar-admin clusters create \
  --broker-url pulsar://<DNS di clusterA>:<Porta> \
  --url http://<DNS del clusterC>:<Porta> \
  ClusterC
```

Ora è necessario configurare la connessione tra il cluster A e il cluster B:

```
$ bin/pulsar-admin clusters create \
  --broker-url pulsar://<DNS di clusterA>:<Porta> \
  --url http://<DNS del clusterC>:<Porta> \
  ClusterB
```

Si fa lo stesso tra il cluster B e il cluster C.

Una volta collegati i cluster dai l'autorizzazione per abilitare la replicazione nei cluster. Quando si crea un tenant si concede l'autorizzazione alla replicazione:

```
$ bin/pulsar-admin tenants create my-tenant \
  --admin-roles my-admin-role \
  --allowed-clusters clusterA,clusterB,clusterC
```

Ora è necessario abilitare la replica geografica, lo si fa a livello di namespace, per creare un namespace:

```
$ bin/pulsar-admin namespaces create my-tenant/my-namespace
```

Inizialmente il namespace non è assegnato a nessun cluster, per assegnarlo usare il seguente comando:

```
$ bin/pulsar-admin namespaces set-clusters my-tenant/my-namespace \
  --clusters clusterA,clusterB,clusterC
```

Successivamente si abilità la replica geografica a livello di topic, è possibile impostarlo tramite:

```
$ bin/pulsar-admin topics set-replication-clusters --clusters clusterA,
    clusterB,clusterC my-tenant/my-namespace/my-topic
```

Di default, i messaggi vengono replicati in tutti i cluster configurati per il namespace. E' possibile limitare la replica specificando un elenco per un messaggio.

E' possibile controllare le statistiche specifiche del topic per i topic di replica, utilizzando il seguente comando:

```
$ bin/pulsar-admin topics stats persistent://my-tenant/my-namespace/my-
    topic
```

Ogni cluster riporta le proprie statistiche locali, inclusi i tassi di replica in entrata, in uscita e quelli arretrati.

8.5 Pulsar IO

In questa sotto sezione verrà illustrato come si sono testati i connettori Pulsar IO.

8.5.1 Configurazione di Pulsar IO per File Source Connector

File Source Connector è un connettore built in di Pulsar che estrae i messaggi dai file nelle directory (quindi un *source connector*) e mantiene i messaggi nei Pulsar topic. La configurazione del connettore ha le proprietà mostrate nella seguente tabella.

Nome	Descrizione
inputDirectory	La directory di input per estrarre i file.
recurse	Se estrarre o meno i file dalla sottodirectory.
keepFile	Se impostato su true, il file non viene eliminato dopo l'elaborazione, il che significa che il file può essere prelevato continuamente.
fileFilter	Viene prelevato il file il cui nome corrisponde all'espressione regolare data.
pathFilter	Se *recurse* è impostato su true, viene scansionata la sottodirectory il cui percorso corrisponde all'espressione regolare specificata.
minimumFileAge	L'età minima per l'elaborazione di un file. Qualsiasi file più giovane di *MinimumFileAge* (secondo la data dell'ultima modifica) viene ignorato.
maximumFileAge	L'età massima per l'elaborazione di un file. Qualsiasi file precedente a *maximumFileAge* (in base alla data dell'ultima modifica) viene ignorato.
minimumSize	La dimensione minima (in byte) che un file può essere elaborato.
maximumSize	La dimensione massima (in byte) che un file può essere elaborato.
ignoreHiddenFiles	Se i file nascosti debbano essere ignorati o meno.
pollingInterval	Indica quanto tempo attendere prima di eseguire un elenco di directory.
numWorkers	Il numero di thread di lavoro che elaborano i file.

Prima di utilizzare il connettore, è necessario creare un file di configurazione, per esempio un file simile a questo (è un file YAML):

```
    configs:
        inputDirectory: "/Users/ismam"
        recurse: true
        keepFile: true
        fileFilter: "[^\\.].*"
        pathFilter: "*"
        minimumFileAge: 0
        maximumFileAge: 9999999999
        minimumSize: 1
        maximumSize: 5000000
        ignoreHiddenFiles: true
        pollingInterval: 5000
        numWorkers: 1
```

8.5.2 Esempio di File Source Connecter

Un esempio fornito da Pulsar su come utilizzare File Source Connecter:

1. Pullare l'immagine di docker:

```
$ docker pull apachepulsar/pulsar:{version}

```

2. Fare partire Pulsar in modalità standalone:

```
$ docker run -d -it -p 6650:6650 -p 8080:8080 -v $PWD/data:/
pulsar/data --name pulsar-standalone apachepulsar/pulsar:{version
} bin/pulsar standalone

```

3. Creare un file di configurazione denominato *file-connector-yaml*, al suo interno scrivere:

```
configs:
    inputDirectory: "/opt"

```

4. Copiare il file di configurazione nel container:

```
$ docker cp connectors/file-connector.yaml pulsar-standalone:/
pulsar/

```

5. Scaricare il source file connector

```
$ curl -O https://mirrors.tuna.tsinghua.edu.cn/apache/pulsar/
pulsar-{version}/connectors/pulsar-io-file-{version}.nar

```

6. Eseguire il source file connector

```
$ docker exec -it pulsar-standalone /bin/bash

$ ./bin/pulsar-admin sources localrun \
--archive /pulsar/pulsar-io-file-{version}.nar \
--name file-test \
--destination-topic-name  pulsar-file-test \
--source-config-file /pulsar/file-connector.yaml

```

7. Fare partire un consumer:

```
./bin/pulsar-client consume -s file-test -n 0 pulsar-file-test

```

8. Scrivere un messaggio nel file test.txt

```
echo "hello world!" > /opt/test.txt

```

8.6 Applicazione Java

Sono stati implementati 2 piccoli progetti per testare le librerie di Pulsar in Java (*org.apache.pulsar.client.api._*):

1. Pulsar Advanced: contiene i test che si sono scritti per capire bene il funzionamento di consumers, producers, topic e messaggi.

2. Pulsar GUI Client: è un client con un'interfaccia grafica da cui si possono eseguire una serie di azioni, nelle sotto sezioni successive ne verrà spiegato l'utilizzo.

8.6.1 Architettura di Pulsar GUI Client

L'applicazione è divisa molto semplicemente in 3 package:

1. View: contiene solamente la parte grafica (molto banale).

2. Model: contiene i meccanismi di base dell'applicazione, al suo interno troviamo i metodi che effettivamente vanno a creare il client, i consumer, i producer ecc. NB: quando viene creato il client, per motivi logistici è stata usata una stringa di connessione al proprio localhost, nella porta 6650 (se si fanno le azioni standard usando il comando standalone non si creano problemi, però quando si va a provare con cluster multipli bisogna ricordarsi di modificare questo dato).

3. Controller: questa classe si occupa di far interagire view e model, prima creando un'istanza di view, poi di model e poi facendo partire il tutto tramite il metodo *initView()*, sono poi presenti tutti i vari action listeners.

8.6.2 Test di Pulsar GUI Client

Prima di utilizzare l'applicazione è necessario utilizzare il comando standalone per creare un cluster locale Pulsar. Una volta lanciata l'applicazione si ha la schermata mostrata in figura 10:

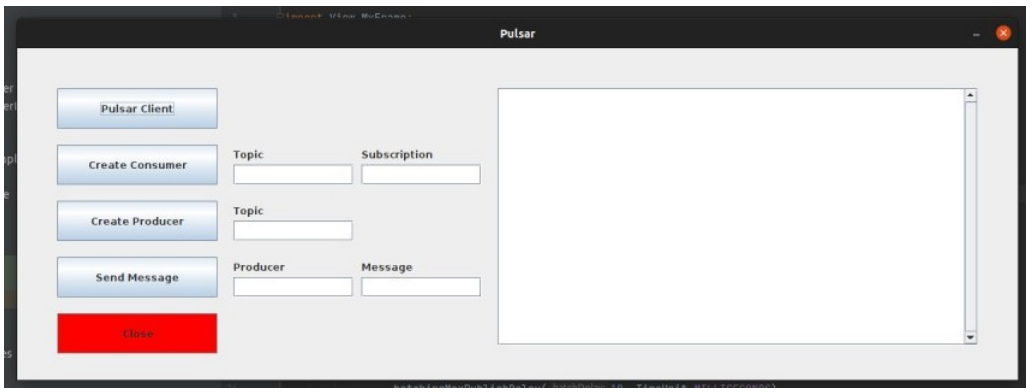

Figure 10: Applicazione per provare la libreria Java di Pulsar.

Una volta lanciata l'applicazione è necessario cliccare su Pulsar Client per creare un nuovo client per poter effettuare le richieste (come creazione di consumer, producer, mandare messaggi). Una volta fatto ciò possiamo:

- Creare un consumer specificando un topic e una subscription (se non esistono vengono creati)
- Creare un producer specificando il topic creato in precedenza (o eventualmente un topic esistente)
- Mandare un messaggio specificando l'id del producer (che viene stampato sul terminale alla creazione del producer) e il messaggio da mandare
- Chiudere il client (vengono chiusi in ordine tutti i producer, consumer ed in fine il client)

Un esempio di questa sequenza di comandi in figura 11:

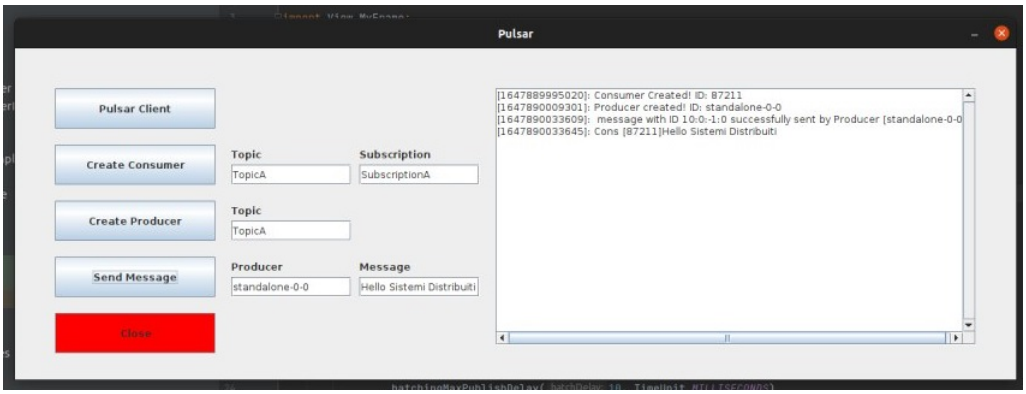

Figure 11: Esempio sull'applicazione.

9 Conclusioni

Il progetto ha soddisfatto gli obiettivi con cui si è iniziato, sono stati toccati tutti i punti di interesse, dall'architettura generale di Pulsar fino alle sue funzionalità specifiche. E' un sistema che stiamo cercando di implementare dove lavoro per poter gestire una serie di sensori che devono inviare dati e questi dati devono subire una serie di manipolazioni real time. L'aspetto più interessante è stata la Geo Replicazione.