aws

Guida per gli sviluppatori

Flusso di dati Amazon Kinesis

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Flusso di dati Amazon Kinesis: Guida per gli sviluppatori

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

I marchi e l'immagine commerciale di Amazon non possono essere utilizzati in relazione a prodotti o servizi che non siano di Amazon, in una qualsiasi modalità che possa causare confusione tra i clienti o in una qualsiasi modalità che denigri o discrediti Amazon. Tutti gli altri marchi non di proprietà di Amazon sono di proprietà delle rispettive aziende, che possono o meno essere associate, collegate o sponsorizzate da Amazon.

Table of Contents

Cos'è Amazon Kinesis Data Streams? 1
Cosa posso fare con Kinesis Data Streams? 1
Vantaggi dell'utilizzo di Kinesis Data Streams 2
Servizi correlati 3
Concetti e terminologia 4
Esamina l'architettura di alto livello di Kinesis Data Streams 4
Acquisisci familiarità con la terminologia di Kinesis Data Streams 5
Kinesis Data Streams 5
Record di dati 5
Modalità di capacità 5
Periodo di retention 5
Producer 6
Consumer 6
Applicazione di flussi di dati Amazon Kinesis 6
Shard 6
Chiave di partizione 7
Numero sequenza 7
Kinesis Client Library 7
Nome applicazione 8
Crittografia lato server 8
Quote e limiti 9
Limiti delle API 12
Limiti per l'API del piano di controllo KDS 12
Limiti per l'API KDS Data Plane 17
Aumento delle quote 19
Prerequisiti completi per configurare Amazon Kinesis Data Streams 20
Registrati per AWS 20
Esecuzione del download di librerie e strumenti 20
Configura il tuo ambiente di sviluppo 21
Utilizza il AWS CLI per eseguire operazioni di Amazon Kinesis Data Streams 22
Tutorial: installazione e configurazione AWS CLI di Kinesis Data Streams 22
Installa il AWS CLI 22
Configura il AWS CLI 24
Tutorial: Esegui le operazioni di base di Kinesis Data Streams utilizzando AWS CLI 24

Passaggio 1: crea uno stream ... 24
Fase 2: Inserire un record ... 26
Passaggio 3: ottieni il record ... 27
Fase 4: pulizia ... 30
Tutorial introduttivi ... 32
Tutorial: elabora i dati di borsa in tempo reale utilizzando KPL e KCL 2.x ... 32
Completamento dei prerequisiti ... 33
Crea un flusso di dati ... 34
Crea una policy e un utente IAM ... 35
Scarica e crea il codice ... 40
Implementa il produttore ... 41
Implementa il consumatore ... 46
(Facoltativo) Estendi il consumatore ... 50
Pulizia delle risorse ... 52
Tutorial: elabora i dati di borsa in tempo reale utilizzando KPL e KCL 1.x ... 53
Completamento dei prerequisiti ... 54
Crea un flusso di dati ... 55
Crea una policy e un utente IAM ... 57
Scarica e crea il codice di implementazione ... 62
Implementa il produttore ... 63
Implementa il consumatore ... 67
(Facoltativo) Estendi il numero di utenti ... 71
Pulizia delle risorse ... 73
Tutorial: analizza i dati di borsa in tempo reale utilizzando Amazon Managed Service per Apache Flink ... 74
Prerequisiti ... 75
Fase 1: Impostazione di un account ... 76
Fase 2: configurare il AWS CLI ... 79
Fase 3: Creare un'applicazione ... 81
Tutorial: utilizzo AWS Lambda con Amazon Kinesis Data Streams ... 98
Usa la soluzione di AWS streaming di dati per Amazon Kinesis ... 98
Crea e gestisci flussi di dati Kinesis ... 100
Scegli la modalità di capacità del flusso di dati ... 100
Cos'è una modalità di capacità del flusso di dati? ... 101
Caratteristiche e casi d'uso della modalità on-demand ... 101
Caratteristiche e casi d'uso della modalità provisioned ... 103

Passa da una modalità di capacità all'altra 104
Crea uno stream utilizzando il AWS Management Console 105
Crea uno stream utilizzando il APIs 106
Crea il client Kinesis Data Streams 106
Crea lo stream 106
Aggiorna uno stream 108
Utilizzo della console 108
Utilizzare l'API 109
Usa il AWS CLI 110
Visualizzazione dell'elenco di flussi 110
Elenca i frammenti 111
Eliminare uno stream 114
Condividi nuovamente uno stream 115
Decidi una strategia per il resharding 116
Dividere un frammento 117
Unisci due frammenti 118
Completa l'azione di ricondivisione 120
Modifica il periodo di conservazione dei dati 122
Assegnazione di tag alle risorse 123
Rivedi le nozioni di base sui tag 124
Tieni traccia dei costi utilizzando i tag 125
Comprendi le restrizioni relative ai tag 125
Etichetta i flussi utilizzando la console Kinesis Data Streams 126
Contrassegna gli stream utilizzando AWS CLI 128
Etichetta i flussi utilizzando Kinesis Data Streams APIs 128
Tagga i consumatori utilizzando il AWS CLI 129
Tagga i consumatori utilizzando Kinesis Data Streams APIs 129
Scrivere dati su Kinesis Data Streams 131
Sviluppa produttori utilizzando Amazon Kinesis Producer Library (KPL) 132
Rivedi il ruolo del KPL 133
Scopri i vantaggi dell'utilizzo di KPL 133
Comprendi quando non usare KPL 134
Installa il KPL 135
Migrare a KPL 1.x 135
Transizione ai certificati Amazon Trust Services (ATS) per la KPL 140
Piattaforme supportate da KPL 140

Concetti chiave di KPL 141
Integra il KPL con il codice del produttore 143
Scrivi nel tuo flusso di dati Kinesis utilizzando KPL 145
Configura la KPL 147
Implementa la deaggregazione dei consumatori 148
Usa KPL con Amazon Data Firehose 152
Utilizzate il KPL con lo Schema Registry AWS Glue 152
Configura la configurazione del proxy KPL 153
Politica del ciclo di vita della versione KPL 153
Sviluppa i produttori utilizzando l'API Kinesis Data Streams con AWS SDK per Java 154
Aggiungi dati a uno stream 155
Interagisci con i dati utilizzando lo AWS Glue Schema Registry 162
Scrivi su Amazon Kinesis Data Streams utilizzando Kinesis Agent 162
Completa i prerequisiti per Kinesis Agent 163
Scarica e installa l'agente 164
Configura e avvia l'agente 165
Specificare le impostazioni di configurazione dell'agente 166
Monitora più directory di file e scrivi su più flussi 169
Usa l'agente per pre-elaborare i dati 170
Usa i comandi CLI dell'agente 174
Domande frequenti 175
Scrivi su Kinesis Data AWS Streams utilizzando altri servizi 176
Scrivi su Kinesis Data Streams utilizzando AWS Amplify 177
Scrivi su Kinesis Data Streams utilizzando Amazon Aurora 177
Scrivi su Kinesis Data Streams utilizzando Amazon CloudFront 177
Scrivi su Kinesis Data Streams CloudWatch utilizzando Amazon Logs 178
Scrivi su Kinesis Data Streams utilizzando Amazon Connect 178
Scrivi su Kinesis Data Streams utilizzando AWS Database Migration Service 178
Scrivi su Kinesis Data Streams utilizzando Amazon DynamoDB 179
Scrivi su Kinesis Data Streams utilizzando Amazon EventBridge 179
Scrivi su Kinesis Data Streams utilizzando AWS IoT Core 179
Scrivi su Kinesis Data Streams utilizzando Amazon Relational Database Service 179
Scrivi su Kinesis Data Streams utilizzando Amazon Pinpoint 180
Scrivi su Kinesis Data Streams utilizzando Amazon Quantum Ledger Database (Amazon QLDB) 180
Scrivi su Kinesis Data Streams utilizzando integrazioni di terze parti 180

Apache Flink 181
Fluentd 181
Debezium 181
Oracle GoldenGate 181
Connessione Kafka 181
Adobe Experience 182
Striim 182
Risolvi i problemi dei produttori di Kinesis Data Streams 182
La mia applicazione di produzione sta scrivendo a una velocità inferiore al previsto 182
Ricevo un errore di autorizzazione della chiave master KMS non autorizzata 184
Risolvi altri problemi comuni per i produttori 185
Ottimizza i produttori di Kinesis Data Streams 185
Personalizza i tentativi KPL e il comportamento dei limiti di frequenza 185
Applica le migliori pratiche all'aggregazione KPL 186
Leggi i dati da Kinesis Data Streams 188
Sviluppa consumatori con fan-out migliorati con un throughput dedicato 189
Differenze tra una soluzione condivisa tra tutti i consumatori e una versione estesa a tutti i consumatori 191
Gestisci i consumatori con un fan-out avanzato 192
Usa il Data Viewer nella console Kinesis 193
Interroga i tuoi flussi di dati nella console Kinesis 194
Usa la libreria client Kinesis 195
Cos'è Kinesis Client Library? 195
Caratteristiche e vantaggi principali di KCL 196
Concetti KCL 197
Tabelle di metadati DynamoDB e bilanciamento del carico in KCL 198
Sviluppa i consumatori con KCL 202
Elaborazione multi-stream con KCL 214
Usa il registro Schema con KCL AWS Glue 216
Autorizzazioni IAM richieste per le applicazioni consumer KCL 217
Configurazioni KCL 224
Politica del ciclo di vita della versione KCL 240
Esegui la migrazione dalle versioni precedenti di KCL 241
Documentazione della versione precedente di KCL 257
Sviluppa i consumatori con AWS SDK per Java 342
Sviluppa consumatori con produttività condivisa con AWS SDK per Java 342

Sviluppa una maggiore fidelizzazione dei consumatori con il AWS SDK per Java 348
Interagisci con i dati utilizzando lo Schema Registry AWS Glue 351
Sviluppa i consumatori utilizzando AWS Lambda 351
Sviluppa i consumatori utilizzando Managed Service per Apache Flink 352
Sviluppa i consumatori utilizzando Amazon Data Firehose 352
Usa altri AWS servizi per leggere i dati da Kinesis Data Streams 352
Leggi i dati da Kinesis Data Streams utilizzando Amazon EMR 353
Leggi i dati da Kinesis Data Streams EventBridge utilizzando Amazon Pipes 353
Leggi i dati da Kinesis Data Streams utilizzando AWS Glue 353
Leggi i dati da Kinesis Data Streams utilizzando Amazon Redshift 354
Leggi da Kinesis Data Streams utilizzando integrazioni di terze parti 354
Apache Flink 354
Piattaforma Adobe Experience 355
Apache Druid 355
Apache Spark 355
Databricks 355
Piattaforma Kafka Confluent 355
Kinesumer 356
Talend 356
Risolvi i problemi dei consumatori di Kinesis Data Streams 356
Errore di compilazione con il costruttore LeaseManagementConfig 356
Alcuni record Kinesis Data Streams vengono ignorati quando si utilizza la Kinesis Client Library. 358
I record che appartengono allo stesso frammento vengono elaborati da diversi processori di record contemporaneamente 358
L'applicazione consumer sta leggendo a una velocità inferiore al previsto 359
GetRecords restituisce un array di record vuoto anche quando ci sono dati nel flusso 360
Lo shard iterator scade in modo imprevisto 360
L'elaborazione dei record dei consumatori è in ritardo 361
Errore di autorizzazione della chiave KMS non autorizzata 362
DynamoDbException: il percorso del documento fornito nell'espressione di aggiornamento non è valido per l'aggiornamento 362
Risolvi altri problemi comuni per i consumatori 362
Ottimizzazione dei consumatori di Kinesis Data Streams 363
Migliora l'elaborazione a bassa latenza 363
Elabora dati serializzati utilizzando AWS Lambda Amazon Kinesis Producer Library 365

Usa il resharding, lo scaling e l'elaborazione parallela per modificare il numero di shard 365
Gestisci i record duplicati 367
Gestisci l'avvio, lo spegnimento e la limitazione 369
Monitoraggio dei flussi di dati Kinesis 372
Monitora il servizio Kinesis Data Streams con CloudWatch 372
Parametri e dimensioni del flusso di dati Amazon Kinesis 373
Accesso ai CloudWatch parametri di Amazon per il flusso di dati Kinesis 389
Monitoraggio dell'integrità dell'agente del flusso di dati Kinesis tramite CloudWatch 390
Monitora con CloudWatch 390
Registra le chiamate API di flusso di dati Amazon Kinesis tramite AWS CloudTrail 391
Informazioni sul flusso di dati Kinesis in CloudTrail 391
Esempio: voci dei file di log del flusso di dati Kinesis 393
Monitoraggio della KCL tramite CloudWatch 397
Parametri e spazio dei nomi 397
Parametri e dimensioni 397
Configurazione metrica 399
Elenco parametri 399
Monitoraggio della KCL con CloudWatch 417
Metriche, dimensioni e namespace 417
Livello metrico e granularità 417
Accesso locale e CloudWatch caricamento su Amazon 419
Elenco parametri 419
Sicurezza 424
Protezione dei dati in Kinesis Data Streams 425
Cos'è la crittografia lato server per Kinesis Streams? 425
Costi, regioni e considerazioni sulle prestazioni 427
Quali sono i primi passi per iniziare a utilizzare la crittografia lato server? 428
Crea e utilizza chiavi KMS generate dall'utente 429
Autorizzazioni per utilizzare le chiavi KMS generate dall'utente 429
Verifica e risolvi i problemi relativi alle autorizzazioni delle chiavi KMS 431
Utilizzo di flusso di dati Kinesis con endpoint VPC di interfaccia 432
Controllo dell'accesso alle risorse Kinesis Data Streams tramite IAM 435
Sintassi delle policy 437
Operazioni per il flusso di dati Kinesis 437
Nomi della risorsa Amazon (ARNs) per il flusso di dati Kinesis 438
Esempi di policy per il flusso di dati Kinesis 438

Condivisione del flusso di dati con un altro account 441
Configura una AWS Lambda funzione per la lettura da Kinesis Data Streams in un altro account 447
Condivisione dell'accesso utilizzando policy basate su risorse 447
Convalida della conformità per il flusso di dati Kinesis 449
Resilienza in Flusso di dati Kinesis 450
Ripristino di emergenza in Flusso di dati Kinesis 450
Sicurezza dell'infrastruttura nel flusso di dati Kinesis 451
Best practice di sicurezza per il flusso di dati Kinesis 452
Implementazione dell'accesso con privilegi minimi 452
Uso di ruoli IAM 452
Implementazione della crittografia lato server in risorse dipendenti 453
CloudTrail Da utilizzare per monitorare le chiamate API 453
Lavorare con AWS SDKs 454
Esempi di codice 456
Nozioni di base 457
Informazioni di base 457
Azioni 461
Esempi serverless 516
Richiamare una funzione Lambda da un trigger Kinesis 516
Segnalazione di errori di elementi batch per funzioni Lambda con un trigger Kinesis 527
Cronologia dei documenti 541
..... dxliv

Cos'è Amazon Kinesis Data Streams?

Puoi utilizzare Flusso di dati Amazon Kinesis per raccogliere ed elaborare flussi di grandi dimensioni di record di dati in tempo reale. Puoi creare applicazioni di elaborazione di dati, note come applicazioni del flusso di dati Kinesis. Una tipica applicazione del flusso di dati Kinesis legge da un flusso di dati come record di dati. Queste applicazioni possono utilizzare la Kinesis Client Library e possono essere eseguite su istanze Amazon EC2. Puoi inviare i record elaborati ai pannelli di controllo, utilizzarli per generare avvisi, modificare dinamicamente i prezzi e le strategie pubblicitarie, oppure inviare dati a una gamma di altri servizi AWS . Per ulteriori informazioni sulle funzionalità e sui prezzi del flusso di dati Kinesis, consulta Flusso di dati Amazon Kinesis.

Kinesis Data Streams fa parte della piattaforma di streaming dati Kinesis, insieme a Firehose, Kinesis Video Streams e Managed Service for Apache Flink.

Per ulteriori informazioni sulle soluzioni per i big data, consulta AWS Big Data on. AWS Per ulteriori informazioni sulle soluzioni AWS di dati in streaming, consulta Cosa sono i dati in streaming?.

Argomenti

- Cosa posso fare con Kinesis Data Streams?
- Vantaggi dell'utilizzo di Kinesis Data Streams
- Servizi correlati

Cosa posso fare con Kinesis Data Streams?

Puoi utilizzare il flusso di dati Kinesis per l'acquisizione e l'aggregazione di dati in maniera rapida e continua. Il tipo di dati utilizzato può includere dati di logo dell'infrastruttura IT, log di applicazioni, social media, feed di dati di mercato e dati clickstream Web. Poiché il tempo di risposta per il consumo e l'elaborazione dei dati è in tempo reale, l'elaborazione è in genere leggera.

Di seguito sono riportati alcuni scenari tipici per l'utilizzo del flusso di dati Kinesis:

Elaborazione accelerata di feed di dati e log

> Puoi fare in modo che i produttori inviino dati direttamente in un flusso. Ad esempio, i log di sistema e applicazioni push e sono disponibili per l'elaborazione in pochi secondi. In questo modo si impedisce la perdita di dati di log nel caso in cui il front-end o un server di applicazioni abbiano

esito negativo. Il flusso di dati Kinesis fornisce feed di dati accelerati perché non raggruppi i dati sul server prima di inviarli per il consumo.

Reportistica e parametri in tempo reale

Puoi utilizzare i dati raccolti nel flusso di dati Kinesis per la semplice analisi dei dati e la creazione di report in tempo reale. Ad esempio, l'applicazione di elaborazione dei dati può lavorare sui parametri e la creazione di report per log di sistema e applicazione durante l'arrivo dei dati, piuttosto che aspettare di ricevere batch di dati.

Analisi dei dati in tempo reale

Ciò combina la potenza di elaborazione parallela con il valore dei dati in tempo reale. Ad esempio, è possibile elaborare clickstream di siti Web in tempo reale e analizzare la fruibilità del sito utilizzando diverse applicazioni del flusso di dati Kinesis in esecuzione in parallelo.

Elaborazione complessa di flussi

È possibile creare grafici aciclici diretti (DAGs) di applicazioni e flussi di dati Kinesis Data Streams. Questo in genere implica l'inserimento di dati da più applicazioni del flusso di dati Kinesis in un altro flusso per l'elaborazione downstream da parte di un'altra applicazione del flusso di dati Kinesis.

Vantaggi dell'utilizzo di Kinesis Data Streams

Anche se è possibile utilizzare il flusso di dati Kinesis per risolvere una serie di problemi per lo streaming dei dati, un utilizzo comune è l'aggregazione di dati in tempo reale seguita dal caricamento di dati aggregati in un data warehouse o cluster di riduzione della mappa.

I dati vengono inviati in flussi di dati Kinesis e ciò garantisce durabilità ed elasticità. Il ritardo tra il momento in cui un record viene inserito nello stream e il momento in cui può essere recuperato (put-to-get ritardo) è in genere inferiore a 1 secondo. In altre parole, un'applicazione del flusso di dati Kinesis può iniziare a consumare i dati dal flusso quasi immediatamente dopo che i dati vengono aggiunti. L'aspetto di servizio gestito del flusso di dati Kinesis ti allevia dall'onere operativo di creare ed eseguire una pipeline di assunzione di dati. Puoi creare applicazioni di streaming map-reduce–type. L'elasticità del flusso di dati Kinesis consente di aumentare o diminuire lo streaming, in modo da non perdere mai record di dati prima della loro scadenza.

Diverse applicazioni del flusso di dati Kinesis possono utilizzare i dati da un flusso in modo che più azioni, come l'archiviazione e l'elaborazione, possano essere eseguite simultaneamente e in

modo indipendente. Ad esempio, due applicazioni possono leggere dati dallo stesso flusso. La prima applicazione calcola aggregati in esecuzione e aggiorna una tabella Amazon DynamoDB mentre la seconda applicazione comprime e archivia i dati in un datastore come Amazon Simple Storage Service (Amazon S3). La tabella DynamoDB con gli aggregati in esecuzione viene quindi letta da un dashboard per i report. up-to-the-minute

La Kinesis Client Library consente il consumo di dati da flussi tolleranti ai guasti e fornisce supporto di dimensionamento per le applicazioni del flusso di dati Kinesis.

Servizi correlati

Per informazioni su come usare i cluster Amazon EMR per leggere ed elaborare direttamente i flussi di dati Kinesis, consulta Connettore Kinesis.

Amazon Kinesis Data Streams Terminologia e concetti

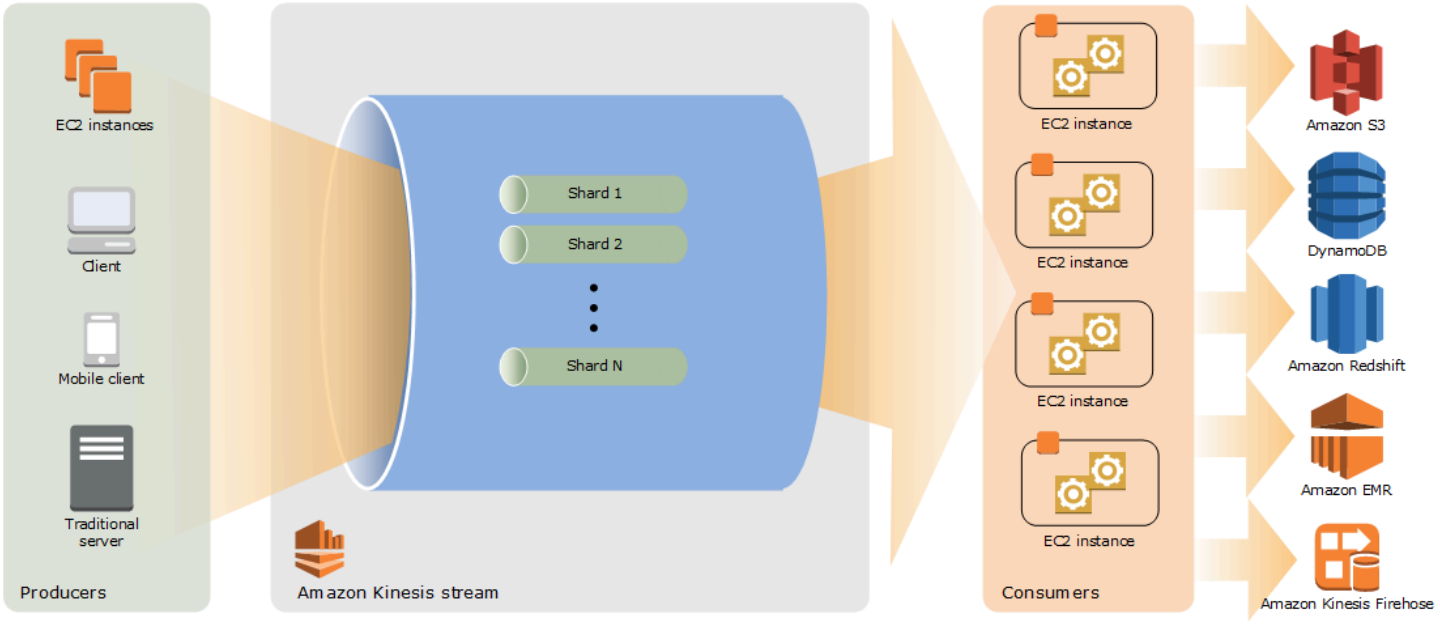
Prima di iniziare a usare Amazon Kinesis Data Streams, scopri la sua architettura e la sua terminologia.

Argomenti

- Esamina l'architettura di alto livello di Kinesis Data Streams
- Acquisisci familiarità con la terminologia di Kinesis Data Streams

Esamina l'architettura di alto livello di Kinesis Data Streams

Nel diagramma seguente viene illustrata l'architettura di alto livello di . I producer inviano continuamente dati al flusso di dati Kinesis e i consumer li elaborano in tempo reale. I consumatori (ad esempio un'applicazione personalizzata in esecuzione su Amazon EC2 o un flusso di distribuzione di Amazon Data Firehose) possono archiviare i risultati utilizzando un AWS servizio come Amazon DynamoDB, Amazon Redshift o Amazon S3.



Acquisisci familiarità con la terminologia di Kinesis Data Streams

Kinesis Data Streams

Un flusso di dati Kinesis è un insieme di partizioni. Ogni shard è una sequenza di record di dati. Ogni record di dati ha un numero di sequenza assegnato dal flusso di dati Kinesis.

Record di dati

Un record di dati è l'unità di dati archiviati in un flusso di dati Kinesis. I record di dati sono costituiti da un numero di sequenza, una chiave di partizione e di blob di dati, che è una sequenza di byte non modificabile. Il flusso di dati Kinesis non ispeziona, interpreta o modifica i dati nel blob in qualsiasi modo. Un data blob può essere fino a 1 MB.

Modalità di capacità

Una modalità di capacità del flusso di dati determina come viene gestita la capacità e come viene addebitato l'utilizzo del flusso di dati. Attualmente, in Kinesis Data Streams, puoi scegliere tra una modalità on demand e una modalità provisioning per i tuoi flussi di dati. Per ulteriori informazioni, consulta Scegli la modalità di capacità del flusso di dati.

Con la modalità on demand, il flusso di dati Kinesis gestisce automaticamente le partizioni per fornire la velocità di trasmissione effettiva necessaria. Ti viene addebitato solo la velocità di trasmissione effettiva che utilizzi e il flusso di dati Kinesis soddisfa automaticamente le esigenze di velocità di trasmissione effettiva dei tuoi carichi di lavoro man mano che aumentano o diminuiscono. Per ulteriori informazioni, consulta Caratteristiche e casi d'uso della modalità on-demand.

Con la modalità provisioning, devi specificare il numero di partizioni per il flusso di dati. La capacità totale di un flusso di dati è la somma di capacità delle relative partizioni. È possibile aumentare o diminuire il numero di partizioni in un flusso di dati in base alle esigenze e il numero di partizioni viene addebitato su base oraria. Per ulteriori informazioni, consulta Caratteristiche e casi d'uso della modalità provisioned.

Periodo di retention

Il periodo di ritenzione è il periodo di tempo in cui i record di dati sono accessibili dopo essere stati aggiunti al flusso. Un periodo di ritenzione del flusso è impostato su un periodo predefinito di 24 ore dopo la creazione. È possibile aumentare il periodo di conservazione fino a 8760 ore (365 giorni)

utilizzando l'IncreaseStreamRetentionPeriodoperazione e ridurlo a un minimo di 24 ore utilizzando l'operazione. DecreaseStreamRetentionPeriod Sono applicabili costi aggiuntivi per flussi con un periodo di ritenzione impostato su più di 24 ore. Per ulteriori informazioni, consulta Prezzi dei flussi di dati per Amazon Kinesis.

Producer

I produttori inseriscono record in Amazon Kinesis Data Streams. Ad esempio, un server Web che invia dati di log a un flusso è un producer.

Consumer

I consumer ottengono i record da Flusso di dati Amazon Kinesis e li elaborano. Questi consumatori sono conosciuti come Applicazione di flussi di dati Amazon Kinesis.

Applicazione di flussi di dati Amazon Kinesis

Un'applicazione Amazon Kinesis Data Streams è un consumatore di un flusso che viene generalmente eseguito su una EC2 flotta di istanze.

Sono disponibili due tipi di consumatori che è possibile sviluppare: i consumatori fan-out condivisi e i consumatori fan-out ottimizzati. Per ulteriori informazioni sulle differenze tra loro e per vedere in che modo è possibile creare ogni tipo di consumer, consulta Leggi i dati da Amazon Kinesis Data Streams.

L'output di un'applicazione del flusso di dati Kinesis può essere un input da un altro flusso, che consente di creare topologie complesse per l'elaborazione di dati in tempo reale. Un'applicazione può anche inviare dati a una varietà di altri servizi. AWS Possono esserci più applicazioni per un flusso e ogni applicazione può consumare i dati dal flusso in modo indipendente e simultaneo

Shard

Uno shard è una sequenza identificata in modo univoco di record di dati in un flusso. Un flusso è composto da uno o più shard, ciascuno dei quali fornisce un'unità fissa di capacità. Ogni partizione può supportare fino a 5 transazioni al secondo per le letture, fino a una velocità totale massima di lettura dei dati di 2 MB al secondo e fino a 1.000 record al secondo per le scritture, fino a una velocità di scrittura totale massima dei dati di 1 MB al secondo (incluse le chiavi di partizione). La capacità di dati del flusso è una funzione del numero di shard specificati per il flusso. La capacità totale del flusso è la somma delle capacità del suo shard.

Se il tasso di dati aumenta, è possibile aumentare o diminuire il numero di shard allocati al flusso. Per ulteriori informazioni, consulta Condividi nuovamente uno stream.

Chiave di partizione

Una chiave di partizione viene utilizzata per raggruppare i dati in base alla partizione in un flusso. Il flusso di dati Kinesis suddivide i record di dati appartenenti a un flusso in più partizioni. Utilizza la chiave di partizione che è associata a ogni record di dati per determinare a quale shard appartiene un determinato record di dati. Le chiavi di partizione sono stringhe Unicode, con un limite di lunghezza massima di 256 caratteri per ogni chiave. Una funzione MD5 hash viene utilizzata per mappare le chiavi di partizione su valori interi a 128 bit e per mappare i record di dati associati ai frammenti utilizzando gli intervalli di chiavi hash degli shard. Quando un'applicazione inserisce dati in un flusso, deve specificare una chiave di partizione.

Numero sequenza

Ogni record di dati ha un numero di sequenza univoco per chiave di partizione nella partizione. Il flusso di dati Kinesis assegna il numero di sequenza dopo aver scritto nel flusso con `client.putRecords` o `client.putRecord`. I numeri di sequenza per la stessa chiave di partizione generalmente aumentano nel corso del tempo. Maggiore è il periodo di tempo compreso tra le richieste di scrittura e più grande sarà il numero di sequenza.

> **Note**
>
> i numeri di sequenza non possono essere utilizzati come indici per set di dati all'interno dello stesso flusso. Per separare i set di dati logicamente, utilizza le chiavi di partizione o crea un flusso separato per ogni set di dati.

Kinesis Client Library

Kinesis Client Library è compilata nell'applicazione per consentire il consumo tollerante ai guasti dei dati dal flusso. Kinesis Client Library assicura che per ogni shard sia in esecuzione un processore di registrazione che elabora lo shard. La libreria, inoltre, semplifica la lettura di dati dal flusso. Kinesis Client Library utilizza le tabelle Amazon DynamoDB per archiviare i metadati relativi al consumo di dati. Crea tre tabelle per applicazione che elabora i dati. Per ulteriori informazioni, consulta Usa la libreria client Kinesis.

Nome applicazione

Il nome di un'applicazione del flusso di dati Amazon Kinesis identifica l'applicazione. Ciascuna delle applicazioni deve avere un nome univoco che corrisponda all' AWS account e alla regione utilizzati dall'applicazione. Questo nome viene utilizzato come nome per la tabella di controllo in Amazon DynamoDB e lo spazio dei nomi per i parametri Amazon. CloudWatch

Crittografia lato server

Flusso di dati Amazon Kinesis è in grado di criptare automaticamente i dati sensibili nel momento in cui un producer li inserisce in un flusso. Il flusso di dati Kinesis utilizza le chiavi master di AWS KMS per la crittografia. Per ulteriori informazioni, consulta Protezione dei dati in Flusso di dati Amazon Kinesis.

> **Note**
>
> Per eseguire operazioni di lettura o scrittura in un flusso crittografato, le applicazioni del producer e consumer devono avere l'autorizzazione di accedere alla chiave master. Per informazioni sulla concessione di autorizzazioni per le applicazioni di producer e consumer, consulta the section called “Autorizzazioni per utilizzare le chiavi KMS generate dall'utente”.

> **Note**
>
> L'utilizzo della crittografia lato server comporta dei costi (). AWS Key Management Service AWS KMS Per ulteriori informazioni, consulta Prezzi di AWS Key Management Service.

Quote e limiti

La tabella seguente descrive le quote e i limiti di stream e shard per Amazon Kinesis Data Streams.

Quota	Modalità on demand	Modalità provisioning
Numero di flussi di dati	Non esiste una quota massima per il numero di stream all'interno del tuo account. AWS Per impostazione predefinita, puoi creare fino a 50 flussi di dati con la modalità di capacità on demand. Se hai bisogno di aumentare questa quota, richiedi un ticket di assistenza.	Non è previsto un limite massimo per il numero di stream con la modalità fornita all'interno di un account.
Numero dle partizioni	Non esiste un limite massimo. Il numero di partizioni dipende dalla quantità di dati importati e dal livello di velocità di trasmissione effettiva richiesto . Il flusso di dati Kinesis dimensiona automaticamente il numero di partizioni in risposta alle variazioni del volume e del traffico dei dati.	Non c'è un limite massimo. La quota predefinita di frammenti è di 20.000 frammenti ciascuno Account AWS per quanto segue: Regioni AWS • Stati Uniti orientali (Virginia settentrionale) • Stati Uniti occidentali (Oregon) • Europa (Irlanda) Per tutte le altre regioni, la quota di frammenti predefinita è di 1.000 o 6.000 frammenti per area. Account AWS Puoi

Quota	Modalità on demand	Modalità provisioning
		visualizzare la quota e l'utilizzo degli shard del tuo account tramite la console Service Quotas all'indirizzo. https://console.aws.amazon.com/servicequotas/ Per richiedere un aumento della quota di shard, usa la console Service Quotas o. AWS CLI Per ulteriori informazioni, consulta la sezione Richiesta di un aumento di quota.

Quota	Modalità on demand	Modalità provisioning
Velocità di trasmissione effettiva del flusso di dati	Per impostazione predefinita, i nuovi flussi di dati creati con la modalità di capacità su richiesta hanno un throughput di lettura pari al 4%MB/s of write and 8 MB/s. Negli Stati Uniti orientali (Virginia settentrionale), negli Stati Uniti occidentali (Oregon) e in Europa (Irlanda)Regioni AWS, i flussi di dati con la modalità di capacità su richiesta sono scalabili fino a 10 capacità di GB/s of write and 20 GB/s read throughput. For other Regions, data streams with the on-demand capacity mode scale up to 200 MB/s of write and 400 MB/s read throughput. If you require an increase up to 10 GB/s write and 20 GB/s lettura per queste regioni, invia un ticket di assistenza.	Non vi è un limite superiore. La velocità di trasmissione effettiva massima dipende dal numero di partizioni fornite per il flusso. Ogni shard può supportare fino a 1 velocità di lettura. MB/sec or 1,000 records/sec write throughput or up to 2 MB/sec or 2,000 records/sec Se hai bisogno di una maggiore capacità di acquisizione, puoi facilmente aumentare il numero di shard nello stream utilizzando l' AWS Management Console API o. UpdateShardCount
Dimensioni del payload di dati	La dimensione massima di payload di dati di un record prima della `base64-encoding` è 1 MB.	
Dimensioni delle transazioni `GetRecords`	GetRecordspuò recuperare fino a 10 MB di dati per chiamata da un singolo shard e fino a 10.000 record per chiamata. Ogni chiamata `GetRecords` viene conteggiata come una transazione in lettura. Ogni shard può supportare fino a cinque operazioni di lettura al secondo. Ogni transazione di lettura può fornire un massimo di 10.000 record con una quota massima di 10 MiB per transazione.	

Quota	Modalità on demand	Modalità provisioning
Velocità di lettura dei dati per partizione	Ogni shard può supportare fino a una velocità di lettura totale massima dei dati di 2 MB al secondo tramite. GetRecords Se una chiamata a `GetRecords` restituisce 10 MiB, le successiv e chiamate effettuate nei prossimi 5 secondi generano un'eccezi one.	
Numero di consumer registrati per flusso di dati	Puoi creare fino a 20 consumer registrati (limite di fan-out avanzato) per ogni flusso di dati.	
Passaggio dalla modalità assegnata a quella on demand	Per ogni flusso di dati nel tuo AWS account, puoi passare dalla modalità di capacità on-demand a quella di capacità fornita due volte nell'arco di 24 ore.	

Limiti delle API

Come la maggior parte AWS APIs, le operazioni dell'API Kinesis Data Streams hanno una velocità limitata. I seguenti limiti si applicano per account per regione. AWS Per ulteriori informazioni su Kinesis APIs Data Streams, consulta l'Amazon Kinesis API Reference.

Limiti per l'API del piano di controllo KDS

La sezione seguente descrive i limiti per il piano di controllo KDS. APIs Il piano di controllo KDS ti APIs consente di creare e gestire i tuoi flussi di dati. Questi limiti si applicano per AWS account per regione.

Limiti per l'API del piano di controllo

API	Limite di chiamata API	Per account/flusso	Descrizione
AddTagsToStream	5 transazioni al secondo (TPS)	Per account	50 tag per flusso di dati
CreateStream	5 TPS	Per account	Non è prevista alcuna quota massima al numero di flussi che puoi avere

API	Limite di chiamata API	Per account/flusso	Descrizione
			in un account. Viene ricevuta una `LimitExceededException` quando si effettua una richiesta `CreateStream` e si prova a completare una delle seguenti operazioni: • Quando sono presenti più di cinque flussi nello stato `CREATING` in qualsiasi momento. • Quando crei più shard di quelli autorizzati per il tuo account.
DecreaseStreamRetentionPeriod	5 TPS	Per flusso	Il valore minimo del periodo di conservazione di un flusso di dati è di 24 ore.
DeleteResourcePolicy	5 TPS	Per account	Se hai bisogno di un aumento di questo limite, invia un ticket di supporto.
DeleteStream	5 TPS	Per account	
DeregisterStreamConsumer	5 TPS	Per flusso	

API	Limite di chiamata API	Per account/flusso	Descrizione
DescribeLimits	1 TPS	Per account	
DescribeStream	10 TPS	Per account	
DescribeStreamConsumer	20 TPS	Per flusso	
DescribeStreamSummary	20 TPS	Per account	
DisableEnhancedMonitoring	5 TPS	Per flusso	
EnableEnhancedMonitoring	5 TPS	Per flusso	
GetResourcePolicy	5 TPS	Per account	Se hai bisogno di un aumento di questo limite, invia un ticket di supporto.
IncreaseStreamRetentionPeriod	5 TPS	Per flusso	Il valore massimo del periodo di conservazione di un flusso è 8.760 ore (365 giorni).
ListShards	1000 TPS	Per flusso	
ListStreamConsumers	5 TPS	Per flusso	
ListStreams	5 TPS	Per account	
ListTagsForStream	5 TPS	Per flusso	
MergeShards	5 TPS	Per flusso	Applicabile solo per la capacità assegnata.

API	Limite di chiamata API	Per account/flusso	Descrizione
PutResourcePolicy	5 TPS	Per account	Se hai bisogno di un aumento di questo limite, invia un ticket di supporto.
RegisterStreamConsumer	5 TPS	Per flusso	Puoi registrare fino a 20 consumer per flusso di dati. Un determinato consumer può essere registrato a un solo flusso di dati alla volta. Possono essere creati contemporaneamente solo 5 consumer. In altre parole, non è possibile avere più di 5 consumer contemporaneamente in uno stato `CREATING`. Registrazione di un sesto consumer mentre ce ne sono 5 in un `CREATING`
RemoveTagsFromStream	5 TPS	Per flusso	
SplitShard	5 TPS	Per flusso	Applicabile solo per la capacità assegnata

API	Limite di chiamata API	Per account/flusso	Descrizione
StartStreamEncryption		Per flusso	Puoi applicare con successo una nuova chiave AWS KMS per la crittografia lato server 25 volte nell'arco di 24 ore consecutive.
StopStreamEncryption		Per flusso	Puoi disabilitare correttamente la crittografia lato server 25 volte in un periodo di 24 ore.
UpdateShardCount		Per flusso	Applicabile solo per la capacità assegnata. Il limite predefinito per il numero di shard è 10.000. Esistono limiti aggiuntivi per questa API. Per ulteriori informazioni, consulta UpdateShardCount.
UpdateStreamMode		Per flusso	Per ogni flusso di dati nel tuo AWS account, puoi passare dalla modalità di capacità on-demand a quella di capacità fornita due volte nell'arco di 24 ore.

Limiti per l'API KDS Data Plane

La sezione seguente descrive i limiti per il piano dati KDS. APIs Il piano dati KDS ti APIs consente di utilizzare i tuoi flussi di dati per raccogliere ed elaborare i record di dati in tempo reale. Questi limiti si applicano per ogni shard all'interno dei flussi di dati.

Limiti per l'API Data Plane

API	Limite di chiamata API	Limite del payload	Ulteriori dettagli
GetRecords	5 TPS	Il numero massimo di record che è possibile restituir e per chiamata è 10.000. La dimension e massima di dati che `GetRecords` può restituire è 10 MB.	Se una chiamata restituisce questa quantità di dati, le chiamate successiv e effettuate entro i 5 secondi successiv i genereranno `Provision edThrough putExceed edException` . Se il throughput assegnato nello stream è insuffici ente, le chiamate successive vengono effettuate entro il secondo successiv o. `Provision edThrough putExceed edException`
GetShardIterator	5 TPS		Un iteratore shard scade 5 minuti dopo che è stato restituit o al richiedente. Se una GetShardI

API	Limite di chiamata API	Limite del payload	Ulteriori dettagli
			terator richiesta viene effettuata troppo spesso, si riceve un. ProvisionedThroughputExceededException
PutRecord	1000 TPS	Ogni shard può supportare scritture fino a 1.000 record al secondo, fino a un massimo di 1 MB al secondo in scrittura dei dati.	
PutRecords		Ogni PutRecords richiesta può supportare fino a 500 record. Ciascun record nella richiesta può avere una dimensione massima pari a 1 MB, con un limite a 5 MB, per l'intera richiesta, incluse le chiavi di partizione. Ogni shard può supportare scritture fino a 1.000 record al secondo, fino a un massimo di 1 MB al secondo in scrittura dei dati.	

API	Limite di chiamata API	Limite del payload	Ulteriori dettagli
SubscribeToShard	È possibile effettuare una chiamata al secondo SubscribeToShard per consumatore registrato per shard.		Se chiami di SubscribeToShard nuovo con lo stesso ConsumerArn ShardId ed entro 5 secondi dall'esito positivo della chiamata, riceverai un. ResourceInUseException

Aumento delle quote

È possibile utilizzare Service Quotas per richiedere un aumento per una quota, se la quota è regolabile. Alcune richieste vengono risolte automaticamente, mentre altre vengono inviate a AWS Support. Puoi monitorare lo stato di una richiesta di aumento della quota inviata a AWS Support. Le richieste di aumento delle Service Quotas non ricevono assistenza prioritaria. Se hai una richiesta urgente, contatta l' AWS assistenza. Per ulteriori informazioni, consulta Cosa sono le Service Quotas?.

Per richiedere un aumento della quota di servizio, seguire la procedura descritta in Richiesta di un aumento di quota.

Prerequisiti completi per configurare Amazon Kinesis Data Streams

Prima di utilizzare Amazon Kinesis Data Streams per la prima volta, completa le seguenti attività per configurare il tuo ambiente.

Attività

- Registrati per AWS
- Esecuzione del download di librerie e strumenti
- Configura il tuo ambiente di sviluppo

Registrati per AWS

Quando ti registri ad Amazon Web Services (AWS), il tuo AWS account viene automaticamente registrato per tutti i servizi in AWS, incluso Kinesis Data Streams. Ti vengono addebitati solo i servizi che utilizzi.

Se hai già un AWS account, passa all'attività successiva. Se non disponi di un account AWS , utilizza la seguente procedura per crearne uno.

Per creare un account AWS

1. Apri la https://portal.aws.amazon.com/billing/registrazione.
2. Segui le istruzioni online.

   Nel corso della procedura di registrazione riceverai una telefonata, durante la quale sarà necessario inserire un codice di verifica attraverso la tastiera del telefono.

   Quando ti iscrivi a un Account AWS, Utente root dell'account AWSviene creato un. L'utente root dispone dell'accesso a tutte le risorse e tutti i Servizi AWS nell'account. Come best practice di sicurezza, assegna l'accesso amministrativo a un utente e utilizza solo l'utente root per eseguire attività che richiedono l'accesso di un utente root.

Esecuzione del download di librerie e strumenti

Le seguenti librerie e i seguenti strumenti consentono di iniziare a utilizzare il flusso di dati Kinesis:

- La Documentazione di riferimento delle API di Amazon Kinesis riporta il set di operazioni di base supportato dal flusso di dati Kinesis. Per ulteriori informazioni sull'esecuzione di operazioni di base utilizzando il codice Java, consulta:
  - Sviluppa i produttori utilizzando l'API Amazon Kinesis Data Streams con AWS SDK per Java
  - Sviluppa i consumatori con AWS SDK per Java
  - Crea e gestisci flussi di dati Kinesis
- Le versioni AWS SDKs per Go, Java JavaScript, .NET, PHP, Python e Ruby includono il supporto e gli esempi di Kinesis Data Streams. Se la tua versione di AWS SDK per Java non include esempi per Kinesis Data Streams, puoi anche scaricarli da. GitHub
- La Kinesis Client Library (KCL) fornisce un modello di easy-to-use programmazione per l'elaborazione dei dati. La KCL consente di iniziare rapidamente a utilizzare il flusso di dati Kinesis in Java, Node.js, .NET, Python e Ruby. Per ulteriori informazioni, consulta Lettura di dati da flussi.
- L'AWS Command Line Interface supporta il flusso di dati Kinesis. Ti AWS CLI consente di controllare più AWS servizi dalla riga di comando e di automatizzarli tramite script.

Configura il tuo ambiente di sviluppo

Per usare , accertati che il tuo ambiente di sviluppo Java soddisfi i seguenti requisiti:

- Java 1.7 (Java SE 7 JDK) o versioni successive. È possibile scaricare la versione più recente del software Java da Java SE Download sul sito Web di Oracle.
- Pacchetto Apache Commons (codice, client HTTP e Logging)
- Processore Jackson JSON

Tieni presente che AWS SDK per Java include Apache Commons e Jackson nella cartella di terze parti. Tuttavia, l'SDK per Java funziona con Java 1.6, mentre la Kinesis Client Library richiede Java 1.7.

Utilizza il AWS CLI per eseguire operazioni di Amazon Kinesis Data Streams

Questa sezione mostra come eseguire operazioni di base di Amazon Kinesis Data Streams utilizzando. AWS Command Line Interface Apprenderai i principi fondamentali sul flusso di dati Kinesis e le fasi necessarie per inviare e ottenere i dati provenienti da un flusso di dati Kinesis.

Se non hai utilizzato il flusso di dati Kinesis in precedenza, ti consigliamo di acquisire prima familiarità con i concetti chiave e la terminologia introdotti in Amazon Kinesis Data Streams Terminologia e concetti.

Argomenti

- Tutorial: installazione e configurazione AWS CLI di Kinesis Data Streams
- Tutorial: Esegui le operazioni di base di Kinesis Data Streams utilizzando AWS CLI

Per l'accesso alla CLI, sono necessari un ID chiave di accesso e una chiave di accesso segreta. Utilizza credenziali temporanee al posto delle chiavi di accesso a lungo termine quando possibile. Le credenziali temporanee includono un ID della chiave di accesso, una chiave di accesso segreta e un token di sicurezza che ne indica la scadenza. Per ulteriori informazioni, consulta Using temporary credenziali with AWS resources nella IAM User Guide.

Puoi trovare istruzioni dettagliate su step-by-step IAM e sulla configurazione delle chiavi di sicurezza in Create an IAM User.

In questa sezione, i comandi specifici descritti vengono forniti integralmente, salvo nel caso in cui valori specifici sono necessariamente diversi per ogni esecuzione. Inoltre, gli esempi utilizzano la regione Stati Uniti occidentali (Oregon), ma i passaggi in questa sezione funzionano in una qualsiasi delle regioni in cui è supportato il flusso di dati Kinesis.

Tutorial: installazione e configurazione AWS CLI di Kinesis Data Streams

Installa il AWS CLI

Per la procedura dettagliata su come installare i sistemi operativi AWS CLI per Windows e Linux, OS X e Unix, consulta Installazione della AWS CLI.

Utilizza il seguente comando per elencare le opzioni e i servizi disponibili:

```
aws help
```

Utilizzerai il servizio Kinesis Data Streams, quindi potrai AWS CLI rivedere i sottocomandi relativi a Kinesis Data Streams utilizzando il seguente comando:

```
aws kinesis help
```

Questo comando genera un output che include i comandi del flusso di dati Kinesis disponibili:

```
AVAILABLE COMMANDS

       o add-tags-to-stream

       o create-stream

       o delete-stream

       o describe-stream

       o get-records

       o get-shard-iterator

       o help

       o list-streams

       o list-tags-for-stream

       o merge-shards

       o put-record

       o put-records

       o remove-tags-from-stream

       o split-shard

       o wait
```

Questo elenco di comandi corrisponde all'API del flusso di dati Kinesis documentata nella Documentazione di riferimento delle API del servizio Amazon Kinesis. Ad esempio, il comando `create-stream` corrisponde all'operazione API `CreateStream`.

Ora AWS CLI è installato correttamente, ma non è configurato. Ciò viene mostrato nella sezione successiva.

Configura il AWS CLI

Per uso generale, il `aws configure` comando è il modo più veloce per configurare AWS CLI l'installazione. Per ulteriori informazioni, consulta Configurazione della AWS CLI.

Tutorial: Esegui le operazioni di base di Kinesis Data Streams utilizzando AWS CLI

Questa sezione descrive l'utilizzo di base di un flusso di dati Kinesis dalla riga di comando utilizzando la AWS CLI. Assicurati di avere familiarità con i concetti discussi in Amazon Kinesis Data Streams Terminologia e concetti.

> **Note**
>
> Dopo aver creato uno stream, sul tuo account vengono addebitati costi nominali per l'utilizzo di Kinesis Data Streams perché Kinesis Data Streams non è idoneo per il piano gratuito. AWS Al termine di questo tutorial, elimina le tue AWS risorse per evitare di incorrere in addebiti. Per ulteriori informazioni, consulta Fase 4: pulizia.

Argomenti

- Passaggio 1: crea uno stream
- Fase 2: Inserire un record
- Passaggio 3: ottieni il record
- Fase 4: pulizia

Passaggio 1: crea uno stream

Per prima cosa devi creare un flusso e verificare che sia stato creato correttamente. Utilizza il seguente comando per creare un flusso denominato "Foo":

```
aws kinesis create-stream --stream-name Foo
```

Quindi, invia il comando seguente per verificare l'avanzamento della creazione del flusso:

```
aws kinesis describe-stream-summary --stream-name Foo
```

Dovresti ottenere un output simile a quello dell'esempio seguente:

```
{
    "StreamDescriptionSummary": {
        "StreamName": "Foo",
        "StreamARN": "arn:aws:kinesis:us-west-2:123456789012:stream/Foo",
        "StreamStatus": "CREATING",
        "RetentionPeriodHours": 48,
        "StreamCreationTimestamp": 1572297168.0,
        "EnhancedMonitoring": [
            {
                "ShardLevelMetrics": []
            }
        ],
        "EncryptionType": "NONE",
        "OpenShardCount": 3,
        "ConsumerCount": 0
    }
}
```

In questo esempio, lo stream ha lo stato CREATING, il che significa che non è ancora pronto per l'uso. Ricontrolla dopo alcuni minuti e dovresti visualizzare un output simile a quello dell'esempio seguente:

```
{
    "StreamDescriptionSummary": {
        "StreamName": "Foo",
        "StreamARN": "arn:aws:kinesis:us-west-2:123456789012:stream/Foo",
        "StreamStatus": "ACTIVE",
        "RetentionPeriodHours": 48,
        "StreamCreationTimestamp": 1572297168.0,
        "EnhancedMonitoring": [
            {
```

```
                "ShardLevelMetrics": []
            }
        ],
        "EncryptionType": "NONE",
        "OpenShardCount": 3,
        "ConsumerCount": 0
    }
}
```

In questo output ci sono informazioni che non ti servono per questo tutorial. Le informazioni importanti per ora sono `"StreamStatus": "ACTIVE"` le informazioni che indicano che lo stream è pronto per essere utilizzato e le informazioni sul singolo shard che hai richiesto. Puoi inoltre verificare l'esistenza del tuo nuovo flusso utilizzando il comando `list-streams`, come mostrato qui:

```
aws kinesis list-streams
```

Output:

```
{
    "StreamNames": [
        "Foo"
    ]
}
```

Fase 2: Inserire un record

Ora che disponi di un flusso attivo, puoi iniziare ad aggiungere dati. Per questo tutorial, utilizzerai il comando più semplice, `put-record`, che aggiunge un singolo record di dati contenente il testo "testdata" nel flusso:

```
aws kinesis put-record --stream-name Foo --partition-key 123 --data testdata
```

Questo comando, se utilizzato con esito positivo, genera un output simile al seguente:

```
{
    "ShardId": "shardId-000000000000",
    "SequenceNumber": "49546986683135544286507457936321625675700192471156785154"
}
```

Congratulazioni, hai appena aggiunto dati a un flusso! Nella fase successiva scoprirai come estrarre dati dal flusso.

Passaggio 3: ottieni il record

GetShardIterator

Prima di poter ottenere dati dallo stream, devi procurarti lo shard iterator per lo shard che ti interessa. Un iteratore di shard definisce la posizione del flusso e lo shard da cui il consumer (il comando `get-record` in questo caso) effettuerà la lettura. Utilizzerai il `get-shard-iterator` comando come segue:

```
aws kinesis get-shard-iterator --shard-id shardId-000000000000 --shard-iterator-type
 TRIM_HORIZON --stream-name Foo
```

Ricorda che i comandi `aws kinesis` si basano su un'API del flusso di dati Kinesis, per cui se ti interessa ottenere maggiori informazioni sui parametri visualizzati, puoi consultare l'argomento della documentazione di riferimento delle API di `GetShardIterator`. L'esecuzione corretta produrrà un output simile al seguente esempio:

```
{
    "ShardIterator": "AAAAAAAAAAHSywljv0zEgPX4NyKdZ5wryMzP9yALs8NeKbUjp1IxtZs1Sp
+KEd9I6AJ9ZG4lNR1EMi+9Md/nHvtLyxpfhEzYvkTZ4D9DQVz/mBYWRO6OTZRKnW9gd
+efGN2aHFdkH1rJl4BL9Wyrk+ghYG22D2T1Da2EyNSH1+LAbK33gQweTJADBdyMwlo5r6PqcP2dzhg="
}
```

La lunga stringa di caratteri apparentemente casuali è l'iteratore di shard (il tuo sarà diverso). È necessario utilizzare copy/paste the shard iterator into the get command, shown next. Shard iterators have a valid lifetime of 300 seconds, which should be enough time for you to copy/paste lo shard iterator nel comando successivo. È necessario rimuovere tutte le nuove righe dall'iteratore dello shard prima di incollarle al comando successivo. Se ricevi un messaggio di errore che indica che lo shard iterator non è più valido, esegui nuovamente il comando. `get-shard-iterator`

GetRecords

Il comando `get-records` estrae i dati dal flusso e si risolve in una chiamata a `GetRecords` nell'API del flusso di dati Kinesis. L'iteratore di shard specifica la posizione nello shard da cui desideri iniziare a leggere i record di dati in sequenza. Se non sono disponibili record nella porzione dello shard a

cui l'iteratore punta, `GetRecords` restituisce un elenco vuoto. Potrebbero essere necessarie più chiamate per accedere a una parte dello shard che contiene i record.

Nel seguente esempio del `get-records` comando:

```
aws kinesis get-records --shard-iterator
 AAAAAAAAAAHSywljv0zEgPX4NyKdZ5wryMzP9yALs8NeKbUjp1IxtZs1Sp+KEd9I6AJ9ZG4lNR1EMi
+9Md/nHvtLyxpfhEzYvkTZ4D9DQVz/mBYWRO6OTZRKnW9gd+efGN2aHFdkH1rJl4BL9Wyrk
+ghYG22D2T1Da2EyNSH1+LAbK33gQweTJADBdyMwlo5r6PqcP2dzhg=
```

Se state eseguendo questo tutorial da un processore di comandi di tipo Unix come bash, potete automatizzare l'acquisizione dell'iteratore shard usando un comando annidato, come questo:

```
SHARD_ITERATOR=$(aws kinesis get-shard-iterator --shard-id shardId-000000000000 --
shard-iterator-type TRIM_HORIZON --stream-name Foo --query 'ShardIterator')

aws kinesis get-records --shard-iterator $SHARD_ITERATOR
```

Se state eseguendo questo tutorial da un sistema che lo supporta PowerShell, potete automatizzare l'acquisizione dello shard iterator usando un comando come questo:

```
aws kinesis get-records --shard-iterator ((aws kinesis get-shard-iterator --shard-id
 shardId-000000000000 --shard-iterator-type TRIM_HORIZON --stream-name Foo).split('"')
[4])
```

Il risultato positivo del `get-records` comando richiederà i record dal vostro stream per lo shard che avete specificato quando avete ottenuto lo shard iterator, come nell'esempio seguente:

```
{
  "Records":[ {
    "Data":"dGVzdGRhdGE=",
    "PartitionKey":"123”,
    "ApproximateArrivalTimestamp": 1.441215410867E9,
    "SequenceNumber":"495449852569073700275708858640655777030226526385964318 74"
  } ],
  "MillisBehindLatest":24000,

 "NextShardIterator":"AAAAAAAAAAEDOW3ugseWPE4503kqN1yN1UaodY8unE0sYslMUmC6lX9hlig5+t4RtZM0/
tALfiI4QGjunVgJvQsjxjh2aLyxaAaPr
+LaoENQ7eVs4EdYXgKyThTZGPcca2fVXYJWL3yafv9dsDwsYVedI66dbMZFC8rPMWc797zxQkv4pSKvPOZvrUIudb8UkH3V
}
```

Nota che `get-records` è descritta sopra come una richiesta, il che significa che potresti ricevere zero o più record anche se ci sono record nel tuo stream. I record restituiti potrebbero non rappresentare tutti i record attualmente presenti nel tuo stream. Questo è normale e il codice di produzione cercherà i record nello stream a intervalli appropriati. Questa velocità di polling varierà in base ai requisiti di progettazione specifici dell'applicazione.

Nel tuo record in questa parte del tutorial, noterai che i dati sembrano spazzatura e non sono il testo `testdata` chiaro che abbiamo inviato. Ciò è dovuto al modo in cui `put-record` utilizza la codifica Base64 per consentirti di inviare dati binari. Tuttavia, il supporto AWS CLI di Kinesis Data Streams non fornisce la decodifica Base64 perché la decodifica Base64 su contenuto binario non elaborato stampato su stdout può portare a comportamenti indesiderati e potenziali problemi di sicurezza su determinate piattaforme e terminali. Se utilizzi un decoder Base64 (ad esempio, https://www.base64decode.org/) per eseguire la decodifica manuale di `dGVzdGRhdGE=`, osserverai che in realtà si tratta di `testdata`. Questo è sufficiente ai fini di questo tutorial perché, in pratica, viene usato raramente per consumare dati. AWS CLI Più spesso, viene utilizzato per monitorare lo stato dello stream e ottenere informazioni, come mostrato in precedenza (`describe-streamelist-streams`). Per ulteriori informazioni sulla KCL, consulta Sviluppo di consumer personalizzati con velocità di trasmissione effettiva condivisa tramite KCL.

`get-recordsn`on sempre restituisce tutti i record nello stream/shard specificato. Se ciò si verifica, utilizza `NextShardIterator` dall'ultimo risultato per ottenere il set di record successivo. Se nel flusso venissero inseriti più dati, che è la situazione normale nelle applicazioni di produzione, è possibile continuare a eseguire il polling dei dati utilizzati ogni volta. `get-records` Tuttavia, se non si chiama `get-records` using the next shard iterator entro la durata di 300 secondi dello shard iterator, verrà visualizzato un messaggio di errore e sarà necessario utilizzare il `get-shard-iterator` comando per ottenere un nuovo iteratore di shard.

L'output include anche `MillisBehindLatest`, che è il numero di millisecondi a cui la risposta dell'operazione GetRecords si trova rispetto all'estremità del flusso, a indicare il ritardo rispetto all'ora corrente del consumer. Un valore di zero indica che l'elaborazione dei record è aggiornata e che non sono presenti nuovi record da elaborare in questo momento. Nel nostro caso, potresti riscontrare un numero piuttosto elevato se hai dedicato diverso tempo ad acquisire familiarità con le fasi di questo tutorial. Per impostazione predefinita, i record di dati rimangono in un flusso per 24 ore in attesa che vengano recuperati. Questo intervallo di tempo viene chiamato periodo di conservazione ed è configurabile fino a 365 giorni.

Un `get-records` risultato positivo avrà sempre un valore `NextShardIterator` anche se non ci sono altri record attualmente nello stream. Si tratta di un modello di polling che presume che

un producer stia potenzialmente immettendo più record nel flusso in un dato momento. Sebbene possa scrivere le tue routine di polling, se utilizzi la KCL menzionata in precedenza per lo sviluppo di applicazioni consumer, il polling verrà gestito per tuo conto.

Se chiami `get-records` finché non ci sono più record nello stream e nello shard da cui stai estraendo, vedrai un output con record vuoti simile al seguente esempio:

```
{
    "Records": [],
    "NextShardIterator": "AAAAAAAAAAGCJ5jzQNjmdhO6B/YDIDE56jmZmrmMA/r1WjoHXC/
kPJXc1rckt3TFL55dENfe5meNgdkyCRpUPGzJpMgYHaJ53C3nCAjQ6s7ZupjXeJGoUFs5oCuFwhP+Wul/
EhyNeSs5DYXLSSC5XCapmCAYGFjYER69QSdQjxMmBPE/hiybFDi5qtkT6/PsZNz6kFoqtDk="
}
```

Fase 4: pulizia

Elimina lo stream per liberare risorse ed evitare addebiti involontari sul tuo account. Esegui questa operazione ogni volta che hai creato uno stream e non lo utilizzerai, perché i costi maturano per ogni stream, indipendentemente dal fatto che tu stia inserendo e ricevendo dati con esso o meno. Il comando di pulizia è il seguente:

```
aws kinesis delete-stream --stream-name Foo
```

Il successo non produce alcun risultato. Usa `describe-stream` per controllare lo stato di avanzamento dell'eliminazione:

```
aws kinesis describe-stream-summary --stream-name Foo
```

Se esegui questo comando immediatamente dopo il comando delete, vedrai un output simile al seguente esempio:

```
{
    "StreamDescriptionSummary": {
        "StreamName": "samplestream",
        "StreamARN": "arn:aws:kinesis:us-west-2:123456789012:stream/samplestream",
        "StreamStatus": "ACTIVE",
```

Quando il flusso è stato completamente eliminato, `describe-stream` genererà un errore di tipo "non trovato":

```
A client error (ResourceNotFoundException) occurred when calling the
 DescribeStreamSummary operation:
Stream Foo under account 123456789012 not found.
```

Tutorial introduttivi per Amazon Kinesis Data Streams

Amazon Kinesis Data Streams offre diverse soluzioni per l'acquisizione e il consumo di dati dai flussi di dati Kinesis. I tutorial in questa sezione sono progettati per aiutarti ulteriormente a comprendere i concetti e le funzionalità di Amazon Kinesis Data Streams e a identificare la soluzione che soddisfa le tue esigenze.

Argomenti

- Tutorial: elabora i dati di borsa in tempo reale utilizzando KPL e KCL 2.x
- Tutorial: elabora i dati di borsa in tempo reale utilizzando KPL e KCL 1.x
- Tutorial: analizza i dati di borsa in tempo reale utilizzando Amazon Managed Service per Apache Flink
- Tutorial: utilizzo AWS Lambda con Amazon Kinesis Data Streams
- Usa la soluzione di AWS streaming di dati per Amazon Kinesis

Tutorial: elabora i dati di borsa in tempo reale utilizzando KPL e KCL 2.x

Lo scenario di questo tutorial prevede l'inserimento delle negoziazioni azionarie in un flusso di dati e la scrittura di un'applicazione Amazon Kinesis Data Streams di base che esegue calcoli sul flusso. Imparerai come inviare un flusso di record a Kinesis Data Streams e implementare un'applicazione che consuma ed elabora i record quasi in tempo reale.

> **Important**
>
> Dopo aver creato uno stream, sul tuo account vengono addebitati costi nominali per l'utilizzo di Kinesis Data Streams perché Kinesis Data Streams non è idoneo per il piano gratuito. AWS Dopo l'avvio dell'applicazione consumer, inoltre, sul tuo account vengono addebitati costi nominali per l'utilizzo di Amazon DynamoDB. L'applicazione consumer utilizza DynamoDB per monitorare lo stato dell'elaborazione. Quando hai finito di utilizzare questa applicazione, elimina le risorse AWS per evitare di incorrere in costi aggiuntivi. Per ulteriori informazioni, consulta Pulizia delle risorse.

Il codice non accede ai dati del mercato azionario, ma simula il flusso delle negoziazioni. A tale scopo, utilizza un generatore di negoziazioni casuale che utilizza come punto di partenza i dati effettivi del mercato per i primi 25 titoli azionari in base al valore di mercato di febbraio 2015. Se disponi dell'accesso a un flusso di negoziazioni in tempo reale, potresti essere interessato a derivare statistiche utili e tempestive da quel flusso. Ad esempio, potresti eseguire un'analisi basata su finestra scorrevole per determinare i titoli più acquistati negli ultimi 5 minuti. Oppure potresti ricevere una notifica ogni volta che viene effettuato un ordine di vendita troppo grande (ossia, che include un numero eccessivo di titoli). Puoi estendere il codice in questa serie per fornire tale funzionalità.

Puoi seguire la procedura descritta in questo tutorial sul tuo computer desktop o portatile ed eseguire sia il codice producer che consumer sullo stesso computer o su qualsiasi piattaforma che supporta i requisiti predefiniti.

Gli esempi mostrati utilizzano la regione Stati Uniti occidentali (Oregon), ma si applicano a qualunque Regione AWS che supporta i flussi di dati.

Attività

- Completamento dei prerequisiti
- Crea un flusso di dati
- Crea una policy e un utente IAM
- Scarica e crea il codice
- Implementa il produttore
- Implementa il consumatore
- (Facoltativo) Estendi il consumatore
- Pulizia delle risorse

Completamento dei prerequisiti

È necessario soddisfare i seguenti requisiti per completare questo tutorial:

Creare e utilizzare un account Amazon Web Services

Prima di iniziare, assicurati di conoscere i concetti discussi inAmazon Kinesis Data Streams Terminologia e concetti, in particolare con gli stream, gli shard, i produttori e i consumatori. È anche utile aver completato le fasi riportate nella seguente guida: Tutorial: installazione e configurazione AWS CLI di Kinesis Data Streams.

È necessario disporre di un AWS account e di un browser Web per accedere a. AWS Management Console

Per l'accesso alla console, utilizza il nome utente IAM e la relativa password per accedere alla pagina AWS Management Console di accesso IAM. Per informazioni sulle credenziali AWS di sicurezza, incluso l'accesso programmatico e le alternative alle credenziali a lungo termine, consulta le credenziali AWS di sicurezza nella Guida per l'utente IAM. Per i dettagli sull'accesso al tuo Account AWS, consulta Come accedere a nella Guida per l' AWSutente.Accedi ad AWS

Per ulteriori informazioni su IAM e sulle istruzioni per la configurazione della chiave di sicurezza, consulta Creazione di un utente IAM.

Soddisfa i requisiti software di sistema

Il sistema che stati utilizzando per eseguire l'applicazione deve avere installato Java 7 o versioni successive. Per scaricare e installare la versione più recente di Java Development Kit (JDK), accedi al sito di installazione di Java SE di Oracle.

È necessaria la versione più recente di AWS SDK per Java.

L'applicazione consumer richiede la versione 2.2.9 o successiva della Kinesis Client Library (KCL), che è possibile ottenere da /tree/master. GitHub https://github.com/awslabs/ amazon-kinesis-client

Passaggi successivi

Crea un flusso di dati

Crea un flusso di dati

Innanzitutto, è necessario creare il flusso di dati che verrà utilizzato nelle fasi successive di questo tutorial.

Per creare un flusso

1. Accedi AWS Management Console e apri la console Kinesis all'indirizzo https://console.aws.amazon.com /kinesis.
2. Nel riquadro di navigazione, selezionare Data Streams (Flussi di dati).
3. Nella barra di navigazione, espandere il selettore delle regioni e selezionare una regione.
4. Selezionare Create Kinesis stream (Crea flusso Kinesis).
5. Inserire un nome per il flusso di dati (ad esempio, **`StockTradeStream`**).

6. Inserisci **1** il numero di frammenti, ma continua a Stimare il numero di frammenti che ti serviranno compressi.
7. Selezionare Create Kinesis stream (Crea flusso Kinesis).

Nella pagina dell'elenco Flussi Kinesis, durante la creazione del flusso, il relativo stato è `CREATING`. Quando il flusso è pronto per essere utilizzato, lo stato diventa `ACTIVE`.

Se si sceglie il nome del flusso, nella pagina che viene visualizzata, la scheda Details (Dettagli) mostra un riepilogo della configurazione del flusso di dati. La sezione Monitoring (Monitoraggio) mostra le informazioni di monitoraggio per il flusso.

Passaggi successivi

Crea una policy e un utente IAM

Crea una policy e un utente IAM

Le migliori pratiche di sicurezza per AWS imporre l'uso di autorizzazioni granulari per controllare l'accesso a diverse risorse. AWS Identity and Access Management (IAM) consente di gestire gli utenti e le autorizzazioni degli utenti in. AWS Una policy IAM elenca in modo esplicito le operazioni consentite e le risorse per le quali sono applicabili le operazioni.

Di seguito sono elencate le autorizzazioni minime generali richieste per i producer e i consumer del flusso di dati Kinesis.

Producer

Operazioni	Risorsa	Scopo
`DescribeStream` , `DescribeStreamSummary` , `DescribeStreamConsumer`	Flusso di dati Kinesis	Prima di leggere i record, il consumer verifica se il flusso di contiene il flusso di dati.
`SubscribeToShard` , `RegisterStreamConsumer`	Flusso di dati Kinesis	Sottoscrive e registra i consumer in un frammento.
`PutRecord` , `PutRecords`	Flusso di dati Kinesis	Scrivi i record in un flusso di dati Kinesis.

Consumer

Azioni	Risorsa	Scopo
`DescribeStream`	Flusso di dati Kinesis	Prima di leggere i record, il consumer verifica se il flusso di contiene il flusso di dati.
`GetRecords` , `GetShardIterator`	Flusso di dati Kinesis	Legge record da uno shard.
`CreateTable` , `DescribeTable` , `GetItem`, `PutItem`, `Scan`, `UpdateItem`	Tabella Amazon DynamoDB	Se il consumer viene sviluppato utilizzando la Kinesis Clien (versione 1.x o 2.x), deve disporre delle autorizzazioni a una per monitorare lo stato di elaborazione dell'applicazione.
`DeleteItem`	Tabella Amazon DynamoDB	Per quando il consumer esegue operazioni di divisione/unic flusso di dati Kinesis.
`PutMetricData`	CloudWatch Registro Amazon	KCL carica anche le metriche su CloudWatch, utili per mon

Per questo tutorial, creerai un'unica policy IAM che concede tutte le autorizzazioni precedenti. In produzione, si consiglia di creare due policy, una per i producer e una per i consumer.

Per creare una policy IAM

1. Individua l'Amazon Resource Name (ARN) per il nuovo flusso di dati creato nel passaggio precedente. Questo ARN è elencato come Stream ARN (ARN flusso) nella parte superiore della scheda Details (Dettagli). Di seguito è riportato il formato ARN:

   ```
   arn:aws:kinesis:region:account:stream/name
   ```

   Regione

   Il codice AWS regionale, `us-west-2` ad esempio. Per ulteriori informazioni, consulta Concetti di regione e zona di disponibilità.

account

L'ID AWS dell'account, come mostrato nelle Impostazioni dell'account.

nome

Il nome del flusso di dati creato nel passaggio precedente, ovveroStockTradeStream.

2. Determinare l'ARN per la tabella DynamoDB che verrà utilizzata dal consumer (e che verrà creata dalla prima istanza consumer). Deve essere nel seguente formato:

```
arn:aws:dynamodb:region:account:table/name
```

La regione e l'ID account sono identici ai valori nell'ARN del flusso di dati utilizzato per questo tutorial, ma il nome è il nome della tabella DynamoDB creata e utilizzata dall'applicazione consumer. KCL utilizza il nome dell'applicazione come nome della tabella. In questo passaggio, utilizzare `StockTradesProcessor` per il nome della tabella DynamoDB, poiché è il nome dell'applicazione utilizzato nelle fasi successive di questo tutorial.

3. Nella console IAM, in Policies (https://console.aws.amazon.com/iam/home #policies), scegli Crea policy. Se è la prima volta che si utilizzano le policy IAM, scegli Inizia, Crea policy.
4. Scegliere Select (Seleziona) accanto a Policy Generator (Generatore di policy).
5. Scegli Amazon Kinesis come servizio. AWS
6. Selezionare `DescribeStream`, `GetShardIterator`, `GetRecords`, `PutRecord` e `PutRecords` come operazioni consentite.
7. Inserire l'ARN del flusso di dati che si sta utilizzando in questo tutorial.
8. Utilizzare Add Statement (Aggiungi istruzione) per ciascuno dei seguenti elementi:

AWS Servizio	Operazioni	ARN
Amazon DynamoDB	`CreateTable` , `DeleteItem` , `DescribeTable` , `GetItem`, `PutItem`, `Scan`, `UpdateItem`	L'ARN della tabella DynamoDB creata nella fase 2 di questa procedura.
Amazon CloudWatch	`PutMetricData`	*

L'asterisco (*) che viene utilizzato quando si specifica un ARN non è richiesto. In questo caso, è perché non esiste una risorsa specifica CloudWatch su cui viene richiamata l'`PutMetricData`azione.

9. Selezionare Next Step (Fase successiva).
10. Modificare Policy Name (Nome policy) in `StockTradeStreamPolicy`, rivedere il codice e scegliere Create Policy (Crea policy).

Il documento della policy risultante dovrebbe essere simile a questo:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmt123",
      "Effect": "Allow",
      "Action": [
        "kinesis:DescribeStream",
        "kinesis:PutRecord",
        "kinesis:PutRecords",
        "kinesis:GetShardIterator",
        "kinesis:GetRecords",
        "kinesis:ListShards",
        "kinesis:DescribeStreamSummary",
        "kinesis:RegisterStreamConsumer"
      ],
      "Resource": [
        "arn:aws:kinesis:us-west-2:123:stream/StockTradeStream"
      ]
    },
    {
      "Sid": "Stmt234",
      "Effect": "Allow",
      "Action": [
        "kinesis:SubscribeToShard",
        "kinesis:DescribeStreamConsumer"
      ],
      "Resource": [
        "arn:aws:kinesis:us-west-2:123:stream/StockTradeStream/*"
      ]
    },
```

```
    {
      "Sid": "Stmt456",
      "Effect": "Allow",
      "Action": [
        "dynamodb:*"
      ],
      "Resource": [
        "arn:aws:dynamodb:us-west-2:123:table/StockTradesProcessor"
      ]
    },
    {
      "Sid": "Stmt789",
      "Effect": "Allow",
      "Action": [
        "cloudwatch:PutMetricData"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

Per creare un utente IAM

1. Aprire la console IAM all'indirizzo https://console.aws.amazon.com/iam/.
2. Nella pagina Users (Utenti), scegliere Add user (Aggiungi utente).
3. Per User Name (Nome utente), digitare `StockTradeStreamUser`.
4. Per Access type (Tipo di accesso), scegliere Programmatic access (Accesso programmatico), quindi selezionare Next: Permissions (Successivo: autorizzazioni).
5. Scegli Attach existing policies directly (Collega direttamente le policy esistenti).
6. Cerca per nome la politica che hai creato nella procedura precedente (. `StockTradeStreamPolicy` Selezionare la casella a sinistra del nome della policy, quindi scegliere Next: Review (Successivo: revisione).
7. Rivedere i dettagli e il riepilogo, quindi scegliere Create user (Crea utente).
8. Copiare l'ID chiave di accesso e salvarlo privatamente. In Secret access key (Chiave di accesso segreta), scegliere Show (Mostra) e salvare anche la chiave privatamente.

9. Incollare la chiave segreta e quella di accesso su un file locale in un luogo sicuro non accessibile ad altri. Per questa applicazione, è necessario creare un file denominato `~/.aws/credentials` (con autorizzazioni rigorose). Il file deve avere il formato seguente:

```
[default]
aws_access_key_id=access key
aws_secret_access_key=secret access key
```

Collegamento di una policy IAM a un utente

1. Nella console IAM, apri Policy e scegli Operazioni di policy.
2. Scegliere `StockTradeStreamPolicy` e Attach (Collega).
3. Scegliere `StockTradeStreamUser` e Attach Policy (Collega policy).

Passaggi successivi

Scarica e crea il codice

Scarica e crea il codice

In questo argomento viene fornito il codice di implementazione di esempio per l'inserimento di operazioni di trading di esempio nel flusso di dati (producer) e l'elaborazione di questi dati (consumer).

Per scaricare e creare il codice

1. Scarica il codice sorgente dal https://github.com/aws-samples/amazon-kinesis-learning GitHub repository sul tuo computer.
2. Creare un progetto nell'IDE con il codice sorgente, rispettando la struttura di directory fornita.
3. Aggiungere le seguenti librerie al progetto:
   - Amazon Kinesis Client Library (KCL)
   - AWS SDK
   - Apache HttpCore
   - Apache HttpClient
   - Apache Commons Lang

- Apache Commons Logging
- Guava (Google Core Libraries For Java)
- Jackson Annotations
- Jackson Core
- Jackson Databind
- Jackson Dataformat: CBOR
- Joda Time

4. A seconda dell'IDE, il progetto potrebbe essere creato automaticamente. In caso contrario, creare il progetto seguendo la procedura appropriata per l'IDE utilizzato.

Se la procedura viene completata correttamente, puoi passare alla sezione successiva, the section called “Implementa il produttore”.

Passaggi successivi

Implementa il produttore

Questo tutorial utilizza lo scenario reale del monitoraggio del commercio azionario. I seguenti principi illustrano brevemente in che modo questo scenario è mappato alla struttura del codice producer e di supporto.

Consulta il codice sorgente e rivedi le informazioni riportate di seguito.

StockTrade classe

Una singola negoziazione è rappresentata da un'istanza della classe StockTrade. Questa istanza include attributi come il simbolo dei titoli, il prezzo, il numero di azioni, il tipo di operazione (acquisto o vendita) e un ID univoco che identifica l'operazione. Questa classe è implementata per te.

Record di flusso

Un flusso è una sequenza di record. Un record è una serializzazione di un'istanza `StockTrade` in formato JSON. Per esempio:

```
{
  "tickerSymbol": "AMZN",
```

```
  "tradeType": "BUY",
  "price": 395.87,
  "quantity": 16,
  "id": 3567129045
}
```

StockTradeGenerator classe

StockTradeGenerator ha un metodo chiamato `getRandomTrade()` che restituisce una nuova compravendita di azioni generata casualmente ogni volta che viene richiamata. Questa classe è implementata per te.

StockTradesWriter classe

Il metodo `main` del producer, StockTradesWriter recupera continuamente uno scambio casuale e lo invia al flusso di dati Kinesis eseguendo queste operazioni:

1. Legge il nome del flusso di dati e il nome della regione come input.
2. Utilizza il `KinesisAsyncClientBuilder` per impostare la regione, le credenziali e la configurazione del client.
3. Verifica che il flusso esista e sia attivo (in caso contrario, si chiude con un errore).
4. In un ciclo continuo, chiama il metodo `StockTradeGenerator.getRandomTrade()` e quindi il metodo `sendStockTrade` per inviare lo scambio al flusso ogni 100 millisecondi.

Il metodo `sendStockTrade` della classe `StockTradesWriter` include il codice seguente:

```
private static void sendStockTrade(StockTrade trade, KinesisAsyncClient
 kinesisClient,
            String streamName) {
        byte[] bytes = trade.toJsonAsBytes();
        // The bytes could be null if there is an issue with the JSON serialization
 by the Jackson JSON library.
        if (bytes == null) {
            LOG.warn("Could not get JSON bytes for stock trade");
            return;
        }

        LOG.info("Putting trade: " + trade.toString());
        PutRecordRequest request = PutRecordRequest.builder()
                .partitionKey(trade.getTickerSymbol()) // We use the ticker symbol
 as the partition key, explained in the Supplemental Information section below.
```

```
                .streamName(streamName)
                .data(SdkBytes.fromByteArray(bytes))
                .build();
        try {
            kinesisClient.putRecord(request).get();
        } catch (InterruptedException e) {
            LOG.info("Interrupted, assuming shutdown.");
        } catch (ExecutionException e) {
            LOG.error("Exception while sending data to Kinesis. Will try again next
cycle.", e);
        }
    }
```

Fai riferimento alla seguente suddivisione del codice:

- L'`PutRecord`API prevede un array di byte ed è necessario convertire gli scambi in formato JSON. Questa singola riga di codice esegue tale operazione:

  ```
  byte[] bytes = trade.toJsonAsBytes();
  ```

- Prima di poter inviare lo scambio, devi creare una nuova istanza `PutRecordRequest` (denominata richiesta in questo caso): Ogni `request` richiede il nome del flusso, la chiave di partizione e un blob di dati.

  ```
  PutPutRecordRequest request = PutRecordRequest.builder()
      .partitionKey(trade.getTickerSymbol()) // We use the ticker symbol as the
   partition key, explained in the Supplemental Information section below.
      .streamName(streamName)
      .data(SdkBytes.fromByteArray(bytes))
      .build();
  ```

  L'esempio utilizza uno stock ticker come chiave di partizione, che mappa il record su uno shard specifico. In pratica, dovresti avere centinaia o migliaia di chiavi di partizione per shard, in modo che i record vengano distribuiti in modo uniforme in tutto il flusso. Per ulteriori informazioni su come aggiungere dati a un flusso, consulta Scrittura di dati su Amazon Kinesis Data Streams.

  Ora `request` è pronto per l'invio al client (operazione put):

```
   kinesisClient.putRecord(request).get();
```

- La verifica e la registrazione degli errori sono sempre aggiunte utili. Questo codice registra le condizioni di errore:

```
if (bytes == null) {
    LOG.warn("Could not get JSON bytes for stock trade");
    return;
}
```

  Aggiungi il blocco try/catch per l'operazione `put`:

```
try {
  kinesisClient.putRecord(request).get();
} catch (InterruptedException e) {
            LOG.info("Interrupted, assuming shutdown.");
} catch (ExecutionException e) {
            LOG.error("Exception while sending data to Kinesis. Will try again
 next cycle.", e);
}
```

  Questo è perché un'operazione put del flusso di dati Kinesis può non riuscire a causa di un errore di rete o perché il flusso di dati raggiunge il limite di velocità di trasmissione effettiva e viene sottoposto a limitazione. Si consiglia di valutare attentamente la politica di nuovi tentativi per le `put` operazioni volte a evitare la perdita di dati, come l'utilizzo di un nuovo tentativo.

- La registrazione dello stato è utile, ma opzionale:

```
LOG.info("Putting trade: " + trade.toString());
```

Il producer mostrato qui utilizza la funzionalità record singolo dell'API del flusso di dati Kinesis, `PutRecord`. In pratica, se un producer genera numerosi record, spesso è più efficiente utilizzare

la funzionalità record multipli di `PutRecords` e inviare batch di record ogni volta. Per ulteriori informazioni, consulta Scrittura di dati su Amazon Kinesis Data Streams.

Per eseguire il producer

1. Verificare che la chiave di accesso e la coppia di chiavi segrete recuperate in Crea una policy e un utente IAM siano salvate nel file `~/.aws/credentials`.
2. Eseguire la classe `StockTradeWriter` con i seguenti argomenti:

```
StockTradeStream us-west-2
```

   Se è stato creato un flusso in una regione diversa da `us-west-2`, è necessario specificare quella regione qui.

Verrà visualizzato un output simile al seguente:

```
Feb 16, 2015 3:53:00 PM
com.amazonaws.services.kinesis.samples.stocktrades.writer.StockTradesWriter
 sendStockTrade
INFO: Putting trade: ID 8: SELL 996 shares of BUD for $124.18
Feb 16, 2015 3:53:00 PM
com.amazonaws.services.kinesis.samples.stocktrades.writer.StockTradesWriter
 sendStockTrade
INFO: Putting trade: ID 9: BUY 159 shares of GE for $20.85
Feb 16, 2015 3:53:01 PM
com.amazonaws.services.kinesis.samples.stocktrades.writer.StockTradesWriter
 sendStockTrade
INFO: Putting trade: ID 10: BUY 322 shares of WMT for $90.08
```

Il flusso di negoziazioni viene ora importato dal flusso di dati Kinesis.

Passaggi successivi

Implementa il consumatore

Implementa il consumatore

L'applicazione consumer in questo tutorial elabora continuamente le operazioni azionarie nel flusso di dati. Quindi, genera i titoli più acquistati e venduti ogni minuto. L'applicazione si basa sulla Kinesis Client Library (KCL), che esegue numerose delle attività impegnative comuni alle applicazioni consumer. Per ulteriori informazioni, consulta Informazioni su KCL 1.x e 2.x.

Consulta il codice sorgente e rivedi le informazioni riportate di seguito.

StockTradesProcessor classe

La classe principale del consumatore, fornita per te, che svolge le seguenti attività:

- Legge l'applicazione, il flusso di dati e i nomi delle regioni, passati come argomenti.
- Crea un'`KinesisAsyncClient`istanza con il nome della regione.
- Crea un'istanza `StockTradeRecordProcessorFactory` che serve istanze di `ShardRecordProcessor`, implementate da un'istanza `StockTradeRecordProcessor`.
- Crea un'`ConfigsBuilder`istanza con l'`StockTradeRecordProcessorFactory`istanza `KinesisAsyncClient` `StreamNameApplicationName`,, e. Questo è utile per creare tutte le configurazioni con valori predefiniti.
- Crea uno scheduler KCL (in precedenza, nelle versioni di KCL 1.x era noto come worker KCL) con l'istanza `ConfigsBuilder`.
- Lo scheduler crea un nuovo thread per ciascun shard (assegnato a questa istanza consumer), che in un ciclo continuo legge i record dai flussi di dati. Quindi invoca l'istanza `StockTradeRecordProcessor` per elaborare ogni batch di record ricevuto.

StockTradeRecordProcessor classe

Implementazione dell'istanza `StockTradeRecordProcessor`, che a sua volta implementa cinque metodi richiesti: `initialize`, `processRecords`, `leaseLost`, `shardEnded` e `shutdownRequested`.

I metodi `initialize` e `shutdownRequested` vengono utilizzati da KCL per consentire all'elaboratore di record di sapere quando dovrebbe essere pronto a iniziare a ricevere record e quando dovrebbe aspettarsi di non ricevere più record, in modo da poter effettuare qualsiasi attività di configurazione e cessazione specifica per l'app. `leaseLost` e `shardEnded` sono utilizzati per implementare qualsiasi logica su cosa fare quando un lease viene perso o quando una elaborazione ha raggiunto la fine del frammento. In questo esempio, registriamo semplicemente i messaggi che indicano questi eventi.

Ti forniamo il codice per questi metodi. L'elaborazione principale si verifica nel metodo `processRecords`, che a sua volta utilizza `processRecord` per ogni record. Quest'ultimo metodo viene fornito come codice di base per lo più vuoto da implementare nella fase successiva, dove è spiegato in modo dettagliato.

Da segnalare è anche l'implementazione dei metodi di supporto per `processRecord`, ovvero `reportStats` e `resetStats`, che sono vuoti nel codice sorgente originale.

Il metodo `processRecords` viene implementato per te ed esegue questa procedura:

- Per ogni record passato, chiama `processRecord` su di esso.
- Se è trascorso almeno 1 minuto dall'ultimo report, chiama `reportStats()`, che consente di stampare le statistiche più recenti, seguito da `resetStats()`, che cancella le statistiche in modo che l'intervallo successivo includa solo i nuovi record.
- Imposta l'orario della creazione di report successiva.
- Se è trascorso almeno 1 minuto dall'ultimo checkpoint, chiama `checkpoint()`.
- Imposta l'orario della creazione di checkpoint successiva.

Questo metodo utilizza intervalli di 60 secondi per la frequenza di creazione di report e checkpoint. Per ulteriori informazioni sul checkpointing, consulta Utilizzo della Kinesis Client Library.

StockStats classe

Questa classe fornisce la conservazione dei dati e il monitoraggio delle statistiche per i titoli più comuni nel tempo. Questo codice viene fornito per te e include i seguenti metodi:

- `addStockTrade(StockTrade)`: inserisce un dato `StockTrade` nelle statistiche in esecuzione.
- `toString()`: restituisce le statistiche in una stringa formattata.

Questa classe tiene traccia delle azioni più popolari tenendo un conteggio progressivo del numero totale di scambi per ogni azione e del conteggio massimo. Aggiorna questi conteggi ogni volta che si verifica uno scambio.

Aggiungi codice ai metodi della classe `StockTradeRecordProcessor`, come mostrato nella procedura seguente.

Per implementare il consumer

1. Implementare il metodo `processRecord` creando un'istanza di un oggetto `StockTrade` delle dimensioni corrette e aggiungendo a essa i dati del record, registrando un avviso se si verifica un problema.

```
byte[] arr = new byte[record.data().remaining()];
record.data().get(arr);
StockTrade trade = StockTrade.fromJsonAsBytes(arr);
    if (trade == null) {
        log.warn("Skipping record. Unable to parse record into StockTrade.
 Partition Key: " + record.partitionKey());
        return;
        }
stockStats.addStockTrade(trade);
```

2. Implementa un `reportStats` metodo. Modifica il formato di output in base alle tue preferenze.

```
System.out.println("****** Shard " + kinesisShardId + " stats for last 1 minute
 ******\n" +
stockStats + "\n" +
"*****************************************************************\n");
```

3. Implementare il metodo `resetStats`, che crea una nuova istanza `stockStats`.

```
stockStats = new StockStats();
```

4. Implementa i seguenti metodi richiesti dall'`ShardRecordProcessor`interfaccia:

```
@Override
public void leaseLost(LeaseLostInput leaseLostInput) {
    log.info("Lost lease, so terminating.");
}

@Override
```

```
public void shardEnded(ShardEndedInput shardEndedInput) {
    try {
        log.info("Reached shard end checkpointing.");
        shardEndedInput.checkpointer().checkpoint();
    } catch (ShutdownException | InvalidStateException e) {
        log.error("Exception while checkpointing at shard end. Giving up.", e);
    }
}

@Override
public void shutdownRequested(ShutdownRequestedInput shutdownRequestedInput) {
    log.info("Scheduler is shutting down, checkpointing.");
    checkpoint(shutdownRequestedInput.checkpointer());
}

private void checkpoint(RecordProcessorCheckpointer checkpointer) {
    log.info("Checkpointing shard " + kinesisShardId);
    try {
        checkpointer.checkpoint();
    } catch (ShutdownException se) {
        // Ignore checkpoint if the processor instance has been shutdown (fail
 over).
        log.info("Caught shutdown exception, skipping checkpoint.", se);
    } catch (ThrottlingException e) {
        // Skip checkpoint when throttled. In practice, consider a backoff and
 retry policy.
        log.error("Caught throttling exception, skipping checkpoint.", e);
    } catch (InvalidStateException e) {
        // This indicates an issue with the DynamoDB table (check for table,
 provisioned IOPS).
        log.error("Cannot save checkpoint to the DynamoDB table used by the Amazon
 Kinesis Client Library.", e);
    }
}
```

Per eseguire il consumer

1. Eseguire il producer scritto in per inserire record di scambi simulati nel flusso.
2. Verifica che la coppia chiave di accesso e chiave segreta recuperata durante la creazione dell'utente IAM sia stata salvata nel file `~/.aws/credentials`.

3. Eseguire la classe `StockTradesProcessor` con i seguenti argomenti:

```
StockTradesProcessor StockTradeStream us-west-2
```

Nota: se è stato creato un flusso in una regione diversa da `us-west-2`, è necessario specificare quella regione qui.

Dopo un minuto, si dovrebbe visualizzare un output come il seguente, aggiornato ogni minuto:

```
****** Shard shardId-000000000001 stats for last 1 minute ******
Most popular stock being bought: WMT, 27 buys.
Most popular stock being sold: PTR, 14 sells.
************************************************************
```

Passaggi successivi

[(Facoltativo) Estendi il consumatore]

(Facoltativo) Estendi il consumatore

Questa sezione opzionale mostra in che modo puoi ampliare la funzionalità del codice consumer per uno scenario leggermente più elaborato.

Se desideri conoscere i maggiori ordini di vendita ogni minuto, puoi modificare la classe `StockStats` in tre punti per supportare questa nuova priorità.

Per ampliare la funzionalità del consumer

1. Aggiungere nuove variabili di istanza:

```
// Ticker symbol of the stock that had the largest quantity of shares sold
private String largestSellOrderStock;
// Quantity of shares for the largest sell order trade
private long largestSellOrderQuantity;
```

2. Aggiungere il seguente codice a `addStockTrade`:

```
if (type == TradeType.SELL) {
     if (largestSellOrderStock == null || trade.getQuantity() >
 largestSellOrderQuantity) {
         largestSellOrderStock = trade.getTickerSymbol();
         largestSellOrderQuantity = trade.getQuantity();
     }
 }
```

3. Modificare il metodo `toString` per stampare le informazioni aggiuntive:

```
public String toString() {
    return String.format(
        "Most popular stock being bought: %s, %d buys.%n" +
        "Most popular stock being sold: %s, %d sells.%n" +
        "Largest sell order: %d shares of %s.",
        getMostPopularStock(TradeType.BUY),
 getMostPopularStockCount(TradeType.BUY),
        getMostPopularStock(TradeType.SELL),
 getMostPopularStockCount(TradeType.SELL),
        largestSellOrderQuantity, largestSellOrderStock);
}
```

Se si esegue il consumer ora (ricordare di eseguire anche il producer), dovrebbe essere visualizzato un output simile a questo:

```
****** Shard shardId-000000000001 stats for last 1 minute ******
Most popular stock being bought: WMT, 27 buys.
Most popular stock being sold: PTR, 14 sells.
Largest sell order: 996 shares of BUD.
************************************************************
```

Passaggi successivi

[Pulizia delle risorse]

Pulizia delle risorse

Poiché l'utilizzo del flusso di dati Kinesis è a pagamento, una volta terminato di usarlo assicurati di eliminarlo insieme alle tabelle Amazon DynamoDB corrispondenti. Vengono infatti applicati costi nominali a un flusso attivo, anche quando non invii o ricevi record. Ciò si verifica perché un flusso attivo utilizza risorse monitorando in modo continuo i record in entrata e le richieste per ottenere record.

Per eliminare il flusso e la tabella

1. Chiudete tutti i produttori e i consumatori che potreste avere ancora in gestione.
2. Apri la console Kinesis in /kinesis. https://console.aws.amazon.com
3. Scegliere il flusso creato per questa applicazione (`StockTradeStream`).
4. Scegliere Delete Stream (Elimina flusso).
5. Apri la console DynamoDB all'indirizzo. https://console.aws.amazon.com/dynamodb/
6. Eliminare la tabella `StockTradesProcessor`.

Riepilogo

L'elaborazione di una grande quantità di dati quasi in tempo reale non richiede la scrittura di codice complicato o lo sviluppo di un'infrastruttura enorme. Elaborare una piccola quantità di dati (come la scrittura`processRecord(Record)`) è tanto semplice quanto scrivere la logica, ma usare Kinesis Data Streams per scalare in modo da funzionare per una grande quantità di dati in streaming. Non devi preoccuparti del dimensionamento dell'elaborazione, perché il flusso di dati Kinesis lo gestisce per conto tuo. Devi soltanto inviare i tuoi record di streaming al flusso di dati Kinesis e scrivere la logica per elaborare ogni nuovo record ricevuto.

Di seguito sono elencati alcuni miglioramenti potenziali per questa applicazione.

Aggregazione tra tutti gli shard

> Al momento, puoi ottenere statistiche derivate dall'aggregazione dei record di dati ricevuti da un singolo ruolo di lavoro e provenienti da un solo shard (uno shard non può essere elaborato da più di un ruolo di lavoro in un'unica applicazione allo stesso tempo). Quando esegui il dimensionamento e disponi di più di uno shard, potresti voler effettuare l'aggregazione tra tutti gli shard. Per farlo, ti occorre un'architettura pipeline in cui l'output di ogni ruolo di lavoro viene utilizzato in un altro flusso con un singolo shard, che viene elaborato da un ruolo di lavoro che

aggrega gli output della prima fase. Poiché i dati della prima fase sono limitati (un campione al minuto per shard), possono essere gestiti facilmente da uno shard.

Dimensionamento dell'elaborazione

Quando il flusso viene incrementato per disporre di numerosi shard (dal momento che molti producer inviano dati), per dimensionare l'elaborazione è possibile aggiungere ulteriori ruoli di lavoro. Puoi eseguire i worker in EC2 istanze Amazon e utilizzare i gruppi di Auto Scaling.

Usa i connettori per Amazon S3/DynamoDB/Amazon Redshift/Storm

Poiché uno stream viene elaborato continuamente, il relativo output può essere inviato ad altre destinazioni. AWS fornisce connettori per integrare Kinesis Data Streams AWS con altri servizi e strumenti di terze parti.

Tutorial: elabora i dati di borsa in tempo reale utilizzando KPL e KCL 1.x

Lo scenario per questo tutorial richiede l'importazione del flusso di negoziazioni in un flusso di dati e la scrittura di una semplice applicazione Flusso di dati Amazon Kinesis che esegua calcoli sul flusso. Imparerai come inviare un flusso di record a Kinesis Data Streams e implementare un'applicazione che consuma ed elabora i record quasi in tempo reale.

> **Important**
>
> Dopo aver creato uno stream, sul tuo account vengono addebitati costi nominali per l'utilizzo di Kinesis Data Streams perché Kinesis Data Streams non è idoneo per il piano gratuito. AWS Dopo l'avvio dell'applicazione consumer, inoltre, sul tuo account vengono addebitati costi nominali per l'utilizzo di Amazon DynamoDB. L'applicazione consumer utilizza DynamoDB per monitorare lo stato dell'elaborazione. Quando hai finito di utilizzare questa applicazione, elimina le risorse AWS per evitare di incorrere in costi aggiuntivi. Per ulteriori informazioni, consulta Pulizia delle risorse.

Il codice non accede ai dati del mercato azionario, ma simula il flusso delle negoziazioni. A tale scopo, utilizza un generatore di negoziazioni casuale che utilizza come punto di partenza i dati effettivi del mercato per i primi 25 titoli azionari in base al valore di mercato di febbraio 2015. Se disponi dell'accesso a un flusso di negoziazioni in tempo reale, potresti essere interessato a derivare

statistiche utili e tempestive da quel flusso. Ad esempio, potresti eseguire un'analisi basata su finestra scorrevole per determinare i titoli più acquistati negli ultimi 5 minuti. Oppure potresti ricevere una notifica ogni volta che viene effettuato un ordine di vendita troppo grande (ossia, che include un numero eccessivo di titoli). Puoi estendere il codice in questa serie per fornire tale funzionalità.

Puoi completare i passaggi di questo tutorial sul tuo computer desktop o laptop ed eseguire sia il codice produttore che quello consumer sulla stessa macchina o su qualsiasi piattaforma che supporti i requisiti definiti, come Amazon Elastic Compute Cloud (Amazon EC2).

Gli esempi mostrati utilizzano la regione Stati Uniti occidentali (Oregon), ma si applicano a qualunque Regione AWS che supporta i flussi di dati.

Attività

- Completamento dei prerequisiti
- Crea un flusso di dati
- Crea una policy e un utente IAM
- Scarica e crea il codice di implementazione
- Implementa il produttore
- Implementa il consumatore
- (Facoltativo) Estendi il numero di utenti
- Pulizia delle risorse

Completamento dei prerequisiti

Di seguito sono riportati i requisiti per il completamento del Tutorial: elabora i dati di borsa in tempo reale utilizzando KPL e KCL 1.x.

Creare e utilizzare un account Amazon Web Services

Prima di iniziare, accertati di avere familiarità con i concetti discussi in Amazon Kinesis Data Streams Terminologia e concetti, con particolare riguardo a flussi, shard, producer e consumer. È inoltre utile aver completato Tutorial: installazione e configurazione AWS CLI di Kinesis Data Streams.

Sono necessari un AWS account e un browser Web per accedere a AWS Management Console.

Per l'accesso alla console, utilizza il nome utente IAM e la relativa password per accedere alla pagina AWS Management Console di accesso IAM. Per informazioni sulle credenziali AWS di

sicurezza, incluso l'accesso programmatico e le alternative alle credenziali a lungo termine, consulta le credenziali AWS di sicurezza nella IAM User Guide. Per i dettagli sull'accesso al tuo Account AWS, consulta Come accedere a nella Guida per l' AWSutente.Accedi ad AWS

Per ulteriori informazioni su IAM e sulle istruzioni per la configurazione della chiave di sicurezza, consulta Creazione di un utente IAM.

Soddisfa i requisiti software di sistema

Sul sistema utilizzato per eseguire l'applicazione deve essere installato Java 7 o versioni successive. Per scaricare e installare la versione più recente di Java Development Kit (JDK), accedi al sito di installazione di Java SE di Oracle.

Se disponi di un IDE Java, ad esempio Eclipse, puoi aprire, modificare, creare ed eseguire il codice sorgente.

È necessaria la versione più recente di AWS SDK per Java. Se utilizzi Eclipse come IDE, puoi installare invece AWS Toolkit for Eclipse.

L'applicazione consumer richiede la versione 1.2.1 o successiva di Kinesis Client Library (KCL), disponibile presso GitHub Kinesis Client Library (Java).

Fasi successive

Crea un flusso di dati

Crea un flusso di dati

Nella prima fase del Tutorial: elabora i dati di borsa in tempo reale utilizzando KPL e KCL 1.x, è necessario creare il flusso da utilizzare nelle fasi successive.

Per creare un flusso

1. Accedi AWS Management Console e apri la console Kinesis all'indirizzo https://console.aws.amazon.com /kinesis.
2. Nel riquadro di navigazione, selezionare Data Streams (Flussi di dati).
3. Nella barra di navigazione, espandere il selettore delle regioni e selezionare una regione.
4. Selezionare Create Kinesis stream (Crea flusso Kinesis).
5. Inserire un nome per il flusso (ad esempio, **`StockTradeStream`**).

6. Inserisci **1** il numero di frammenti, ma continua a Stimare il numero di frammenti che ti serviranno compressi.
7. Selezionare Create Kinesis stream (Crea flusso Kinesis).

Nella pagina dell'elenco Flussi Kinesis, lo stato del flusso mentre viene creato è `CREATING`. Quando il flusso è pronto per essere utilizzato, lo stato diventa `ACTIVE`. Seleziona il nome del flusso. Nella pagina che viene visualizzata, la scheda Details (Dettagli) mostra un riepilogo della configurazione del flusso. La sezione Monitoring (Monitoraggio) mostra le informazioni di monitoraggio per il flusso.

Informazioni aggiuntive sui frammenti

Quando inizi a utilizzare il flusso di dati Kinesis al di fuori di questo tutorial, potresti dover pianificare il processo di creazione di un flusso più attentamente. Dovresti effettuare una stima della domanda massima quando esegui il provisioning degli shard. Utilizzando questo scenario come esempio, il numero di scambi nel mercato azionario statunitense raggiunge il picco durante il giorno (fuso orario EST) e le stime della domanda dovrebbero essere effettuate per quell'ora del giorno. A questo punto, puoi scegliere di effettuare il provisioning per la domanda massima attesa o incrementare o ridurre il flusso in risposta alle fluttuazioni della domanda.

Uno shard è un'unità di capacità di throughput. Nella pagina Crea flusso Kinesis, espandi Calcola il numero di partizioni necessarie. Inserisci le dimensioni medie dei record, il numero massimo di record scritti al secondo e il numero di applicazioni in uso attenendoti alle seguenti indicazioni:

Dimensione media record

Una stima delle dimensioni medie calcolate dei record. Se non conosci questo valore, utilizza le dimensioni massime stimate dei record.

Numero massimo di record scritti

Considerate il numero di entità che forniscono dati e il numero approssimativo di record al secondo prodotti da ciascuna di esse. Ad esempio, se ottieni dati sulle negoziazioni da 20 server di trading e ognuno di essi genera 250 scambi al secondo, il numero totale di scambi (record) è di 5.000 al secondo.

Numero di applicazioni in uso

Il numero di applicazioni che in modo indipendente effettuano la lettura dal flusso per elaborarlo in modo diverso e produrre un output diverso. Ciascuna applicazione può avere più istanze in

esecuzione su macchine diverse (esecuzione in cluster) in modo da poter tenere il passo con un flusso di volume elevato.

Se il numero stimato di shard mostrato supera il limite attuale di shard, potresti dover inviare una richiesta per aumentare tale limite prima di poter creare un flusso con quel numero di shard. Per richiedere un aumento del limite di partizioni, utilizza il modulo dei limiti del flusso di dati Kinesis. Per ulteriori informazioni sui flussi e sulle partizioni, consulta Crea e gestisci flussi di dati Kinesis.

Passaggi successivi

Crea una policy e un utente IAM

Crea una policy e un utente IAM

Le migliori pratiche di sicurezza per AWS imporre l'uso di autorizzazioni granulari per controllare l'accesso a diverse risorse. AWS Identity and Access Management (IAM) consente di gestire gli utenti e le autorizzazioni degli utenti in. AWS Una policy IAM elenca in modo esplicito le operazioni consentite e le risorse per le quali sono applicabili le operazioni.

Di seguito sono elencate le autorizzazioni minime generali richieste per un producer e un consumer del flusso di dati Kinesis.

Producer

Operazioni	Risorsa	Scopo
`DescribeStream` , `DescribeStreamSummary` , `DescribeStreamConsumer`	Flusso di dati Kinesis	Prima di tentare di scrivere record, il producer controlla se i attivo, se i frammenti sono contenuti nel flusso e se il flusso
`SubscribeToShard` , `RegisterStreamConsumer`	Flusso di dati Kinesis	Sottoscrive e registra un consumer a uno shard flusso di da
`PutRecord` , `PutRecords`	Flusso di dati Kinesis	Scrivi i dati in un flusso di dati Kinesis.

Consumer

Azioni	Risorsa	Scopo
`DescribeStream`	Flusso di dati Kinesis	Prima di leggere i record, il consumer verifica se il flusso es shard.
`GetRecords` , `GetShardIterator`	Flusso di dati Kinesis	Leggi i record da una partizione del flusso di dati Kinesis.
`CreateTable` , `DescribeTable` , `GetItem`, `PutItem`, `Scan`, `UpdateItem`	Tabella Amazon DynamoDB	Se il consumer viene sviluppato utilizzando la Kinesis Clien bisogno delle autorizzazioni a una tabella DynamoDB per m elaborazione dell'applicazione. Il primo consumer avviato cr
`DeleteItem`	Tabella Amazon DynamoDB	Per quando il consumer esegue operazioni di divisione/unic flusso di dati Kinesis.
`PutMetricData`	CloudWatch Registro Amazon	KCL carica anche le metriche su CloudWatch, utili per mon

Per questa applicazione, devi creare una singola policy IAM che conceda tutte le autorizzazioni precedenti. In pratica, potresti voler creare due policy, una per i producer e una per i consumer.

Per creare una policy IAM

1. Individuare l'Amazon Resource Name (ARN) per il nuovo flusso. Questo ARN è elencato come Stream ARN (ARN flusso) nella parte superiore della scheda Details (Dettagli). Di seguito è riportato il formato ARN:

   ```
   arn:aws:kinesis:region:account:stream/name
   ```

   Regione

   Il codice regione, ad esempio, `us-west-2`. Per ulteriori informazioni, consulta Concetti di regione e zona di disponibilità.

account

L'ID AWS dell'account, come mostrato in Impostazioni account.

nome

Il nome del flusso dal Crea un flusso di dati, ovvero `StockTradeStream`.

2. Determinare l'ARN per la tabella DynamoDB che verrà utilizzata dal consumer (e creata dalla prima istanza consumer). Deve essere nel seguente formato:

```
arn:aws:dynamodb:region:account:table/name
```

La regione e l'account sono gli stessi della fase precedente, ma questa volta il nome è quello della tabella creata e utilizzata dall'applicazione consumer. La KCL utilizzata dal consumer usa il nome dell'applicazione come nome della tabella. Utilizzare `StockTradesProcessor`, che è il nome dell'applicazione utilizzato successivamente.

3. Nella console IAM, in Policies (https://console.aws.amazon.com/iam/home #policies), scegli Crea policy. Se è la prima volta che si utilizzano le policy IAM, scegli Inizia, Crea policy.
4. Scegliere Select (Seleziona) accanto a Policy Generator (Generatore di policy).
5. Scegli Amazon Kinesis come servizio. AWS
6. Selezionare `DescribeStream`, `GetShardIterator`, `GetRecords`, `PutRecord` e `PutRecords` come operazioni consentite.
7. Inserire l'ARN creato alla fase 1.
8. Utilizzare Add Statement (Aggiungi istruzione) per ciascuno dei seguenti elementi:

AWS Servizio	Operazioni	ARN
Amazon DynamoDB	`CreateTable` , `DeleteItem` , `DescribeTable` , `GetItem`, `PutItem`, `Scan`, `UpdateItem`	L'ARN creato alla fase 2
Amazon CloudWatch	`PutMetricData`	*

L'asterisco (*) che viene utilizzato quando si specifica un ARN non è richiesto. In questo caso, è perché non esiste una risorsa specifica CloudWatch su cui viene richiamata l'`PutMetricData`azione.

9. Selezionare Next Step (Fase successiva).
10. Modificare Policy Name (Nome policy) in `StockTradeStreamPolicy`, rivedere il codice e scegliere Create Policy (Crea policy).

Il documento della policy risultante dovrebbe essere simile a quello seguente:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmt123",
      "Effect": "Allow",
      "Action": [
        "kinesis:DescribeStream",
        "kinesis:PutRecord",
        "kinesis:PutRecords",
        "kinesis:GetShardIterator",
        "kinesis:GetRecords",
        "kinesis:ListShards",
        "kinesis:DescribeStreamSummary",
        "kinesis:RegisterStreamConsumer"
      ],
      "Resource": [
        "arn:aws:kinesis:us-west-2:123:stream/StockTradeStream"
      ]
    },
    {
      "Sid": "Stmt234",
      "Effect": "Allow",
      "Action": [
        "kinesis:SubscribeToShard",
        "kinesis:DescribeStreamConsumer"
      ],
      "Resource": [
        "arn:aws:kinesis:us-west-2:123:stream/StockTradeStream/*"
      ]
    },
```

```
    {
      "Sid": "Stmt456",
      "Effect": "Allow",
      "Action": [
        "dynamodb:*"
      ],
      "Resource": [
        "arn:aws:dynamodb:us-west-2:123:table/StockTradesProcessor"
      ]
    },
    {
      "Sid": "Stmt789",
      "Effect": "Allow",
      "Action": [
        "cloudwatch:PutMetricData"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

Per creare un utente IAM

1. Aprire la console IAM all'indirizzo https://console.aws.amazon.com/iam/.
2. Nella pagina Users (Utenti), scegliere Add user (Aggiungi utente).
3. Per User Name (Nome utente), digitare `StockTradeStreamUser`.
4. Per Access type (Tipo di accesso), scegliere Programmatic access (Accesso programmatico), quindi selezionare Next: Permissions (Successivo: autorizzazioni).
5. Scegli Attach existing policies directly (Collega direttamente le policy esistenti).
6. Cercare la policy creata per nome. Selezionare la casella a sinistra del nome della policy, quindi scegliere Next: Review (Successivo: revisione).
7. Rivedere i dettagli e il riepilogo, quindi scegliere Create user (Crea utente).
8. Copiare l'ID chiave di accesso e salvarlo privatamente. In Secret access key (Chiave di accesso segreta), scegliere Show (Mostra) e salvare anche la chiave privatamente.
9. Incollare la chiave segreta e quella di accesso su un file locale in un luogo sicuro non accessibile ad altri. Per questa applicazione, è necessario creare un file denominato `~/.aws/credentials` (con autorizzazioni rigorose). Il file deve avere il formato seguente:

```
[default]
aws_access_key_id=access key
aws_secret_access_key=secret access key
```

Collegamento di una policy IAM a un utente

1. Nella console IAM, apri Policy e scegli Operazioni di policy.
2. Scegliere `StockTradeStreamPolicy` e Attach (Collega).
3. Scegliere `StockTradeStreamUser` e Attach Policy (Collega policy).

Fasi successive

Scarica e crea il codice di implementazione

Scarica e crea il codice di implementazione

Viene fornito codice di base per il the section called “Tutorial: elabora i dati di borsa in tempo reale utilizzando KPL e KCL 1.x”. Include l'implementazione di uno stub sia per l'acquisizione dei flussi delle negoziazioni (producer) che per l'elaborazione dei dati (consumer). La procedura seguente mostra come completare l'implementazione.

Per scaricare e compilare il codice di implementazione

1. Scaricare il codice sorgente sul computer.
2. Creare un progetto nell'IDE di preferenza con il codice sorgente, rispettando la struttura di directory fornita.
3. Aggiungere le seguenti librerie al progetto:

   - Amazon Kinesis Client Library (KCL)
   - AWS SDK
   - Apache HttpCore
   - Apache HttpClient
   - Apache Commons Lang
   - Apache Commons Logging
   - Guava (Google Core Libraries For Java)

- Jackson Annotations
- Jackson Core
- Jackson Databind
- Jackson Dataformat: CBOR
- Joda Time

4. A seconda dell'IDE, il progetto potrebbe essere creato automaticamente. In caso contrario, creare il progetto seguendo la procedura appropriata per l'IDE utilizzato.

Se la procedura viene completata correttamente, puoi passare alla sezione successiva, the section called “Implementa il produttore”. Se la build genera errori in una fase, analizzali e correggili prima di continuare.

Passaggi successivi

Implementa il produttore

L'applicazione nel Tutorial: elabora i dati di borsa in tempo reale utilizzando KPL e KCL 1.x utilizza come scenario quello del monitoraggio del mercato azionario reale. I seguenti principi illustrano brevemente in che modo questo scenario è mappato alla struttura del codice producer e di supporto.

Consulta il codice sorgente e rivedi le informazioni riportate di seguito.

StockTrade classe

Una singola negoziazione è rappresentata da un'istanza della classe `StockTrade`. Questa istanza include attributi come il simbolo dei titoli, il prezzo, il numero di azioni, il tipo di operazione (acquisto o vendita) e un ID univoco che identifica l'operazione. Questa classe è implementata per te.

Record di flusso

Un flusso è una sequenza di record. Un record è una serializzazione di un'istanza `StockTrade` in formato JSON. Per esempio:

```
{
  "tickerSymbol": "AMZN",
```

```
  "tradeType": "BUY",
  "price": 395.87,
  "quantity": 16,
  "id": 3567129045
}
```

StockTradeGenerator classe

`StockTradeGenerator` include un metodo denominato `getRandomTrade()` che restituisce una negoziazione generata casualmente ogni volta che viene invocata. Questa classe è implementata per te.

StockTradesWriter classe

Il metodo `main` del producer, `StockTradesWriter` recupera continuamente uno scambio casuale e lo invia al flusso di dati Kinesis eseguendo queste operazioni:

1. Legge il nome del flusso e il nome della regione come input.
2. Crea un `AmazonKinesisClientBuilder`.
3. Utilizza il generatore client per impostare la regione, le credenziali e la configurazione del client.
4. Crea un client `AmazonKinesis` utilizzando il generatore di client.
5. Verifica che il flusso esista e sia attivo (in caso contrario, si chiude con un errore).
6. In un ciclo continuo, chiama il metodo `StockTradeGenerator.getRandomTrade()` e quindi il metodo `sendStockTrade` per inviare lo scambio al flusso ogni 100 millisecondi.

Il metodo `sendStockTrade` della classe `StockTradesWriter` include il codice seguente:

```
private static void sendStockTrade(StockTrade trade, AmazonKinesis kinesisClient,
 String streamName) {
    byte[] bytes = trade.toJsonAsBytes();
    // The bytes could be null if there is an issue with the JSON serialization by
 the Jackson JSON library.
    if (bytes == null) {
        LOG.warn("Could not get JSON bytes for stock trade");
        return;
    }

    LOG.info("Putting trade: " + trade.toString());
    PutRecordRequest putRecord = new PutRecordRequest();
    putRecord.setStreamName(streamName);
```

```
    // We use the ticker symbol as the partition key, explained in the Supplemental
 Information section below.
    putRecord.setPartitionKey(trade.getTickerSymbol());
    putRecord.setData(ByteBuffer.wrap(bytes));

    try {
        kinesisClient.putRecord(putRecord);
    } catch (AmazonClientException ex) {
        LOG.warn("Error sending record to Amazon Kinesis.", ex);
    }
}
```

Fai riferimento alla seguente suddivisione del codice:

- L'`PutRecord`API prevede un array di byte e devi convertirlo in formato `trade` JSON. Questa singola riga di codice esegue tale operazione:

  ```
  byte[] bytes = trade.toJsonAsBytes();
  ```

- Prima di poter inviare lo scambio, devi creare una nuova istanza `PutRecordRequest` (denominata `putRecord` in questo caso):

  ```
  PutRecordRequest putRecord = new PutRecordRequest();
  ```

  Ogni chiamata `PutRecord` richiede il nome del flusso, la chiave di partizione e il blob di dati. Il codice seguente compila questi campi nell'oggetto `putRecord` utilizzando i relativi metodi `setXxxx()`:

  ```
  putRecord.setStreamName(streamName);
  putRecord.setPartitionKey(trade.getTickerSymbol());
  putRecord.setData(ByteBuffer.wrap(bytes));
  ```

  Questo esempio utilizza un ticket per titoli come chiave di partizione, che mappa il record a un determinato shard. In pratica, dovresti avere centinaia o migliaia di chiavi di partizione per shard, in modo che i record vengano distribuiti in modo uniforme in tutto il flusso. Per ulteriori informazioni su come aggiungere dati a un flusso, consulta Aggiungi dati a uno stream.

  Ora `putRecord` è pronto per l'invio al client (operazione `put`):

  ```
  kinesisClient.putRecord(putRecord);
  ```

- La verifica e la registrazione degli errori sono sempre aggiunte utili. Questo codice registra le condizioni di errore:

```
if (bytes == null) {
    LOG.warn("Could not get JSON bytes for stock trade");
    return;
}
```

Aggiungi il blocco try/catch per l'operazione `put`:

```
try {
       kinesisClient.putRecord(putRecord);
} catch (AmazonClientException ex) {
       LOG.warn("Error sending record to Amazon Kinesis.", ex);
}
```

Questo è perché un'operazione `put` del flusso di dati Kinesis potrebbe non riuscire a causa di un errore di rete o perché il flusso di dati raggiunge il limite di velocità di trasmissione effettiva e viene sottoposto a limitazione. Ti consigliamo di valutare attentamente la politica di riprova per quanto riguarda `put` le operazioni volte a evitare la perdita di dati, come l'utilizzo di un nuovo tentativo.

- La registrazione dello stato è utile, ma opzionale:

```
LOG.info("Putting trade: " + trade.toString());
```

Il producer mostrato qui utilizza la funzionalità record singolo dell'API del flusso di dati Kinesis, `PutRecord`. In pratica, se un producer genera numerosi record, spesso è più efficiente utilizzare la funzionalità record multipli di `PutRecords` e inviare batch di record ogni volta. Per ulteriori informazioni, consulta Aggiungi dati a uno stream.

Per eseguire il producer

1. Verifica che la coppia chiave di accesso e chiave segreta recuperata durante la creazione dell'utente IAM sia stata salvata nel file `~/.aws/credentials`.
2. Eseguire la classe `StockTradeWriter` con i seguenti argomenti:

```
StockTradeStream us-west-2
```

Se è stato creato un flusso in una regione diversa da `us-west-2`, è necessario specificare quella regione qui.

Verrà visualizzato un output simile al seguente:

```
Feb 16, 2015 3:53:00 PM
com.amazonaws.services.kinesis.samples.stocktrades.writer.StockTradesWriter
 sendStockTrade
INFO: Putting trade: ID 8: SELL 996 shares of BUD for $124.18
Feb 16, 2015 3:53:00 PM
com.amazonaws.services.kinesis.samples.stocktrades.writer.StockTradesWriter
 sendStockTrade
INFO: Putting trade: ID 9: BUY 159 shares of GE for $20.85
Feb 16, 2015 3:53:01 PM
com.amazonaws.services.kinesis.samples.stocktrades.writer.StockTradesWriter
 sendStockTrade
INFO: Putting trade: ID 10: BUY 322 shares of WMT for $90.08
```

Il flusso di negoziazioni viene ora importato dal flusso di dati Kinesis.

Passaggi successivi

Implementa il consumatore

Implementa il consumatore

L'applicazione consumer nel Tutorial: elabora i dati di borsa in tempo reale utilizzando KPL e KCL 1.x elabora in modo continuo il flusso di negoziazioni creato in . Quindi, genera i titoli più acquistati e venduti ogni minuto. L'applicazione si basa sulla Kinesis Client Library (KCL), che esegue numerose delle attività impegnative comuni alle applicazioni consumer. Per ulteriori informazioni, consulta Sviluppa i consumatori di KCL 1.x.

Consulta il codice sorgente e rivedi le informazioni riportate di seguito.

StockTradesProcessor classe

Classe principale del consumer, fornita per te, che esegue queste attività:

- Legge il nome dell'applicazione, del flusso e della regione, passati come argomenti.
- Legge le credenziali da `~/.aws/credentials`.

- Crea un'istanza `RecordProcessorFactory` che serve istanze di `RecordProcessor`, implementate da un'istanza `StockTradeRecordProcessor`.
- Crea un worker KCL con l'istanza `RecordProcessorFactory` e una configurazione standard, inclusi il nome del flusso, le credenziali e il nome dell'applicazione.
- Il worker crea un nuovo thread per ciascuna partizione (assegnata a questa istanza consumer), che in un ciclo continuo legge i record dal flusso di dati Kinesis. Quindi invoca l'istanza `RecordProcessor` per elaborare ogni batch di record ricevuto.

StockTradeRecordProcessor classe

Implementazione dell'istanza `RecordProcessor`, che a sua volta implementa tre metodi richiesti: `initialize`, `processRecords` e `shutdown`.

Come suggeriscono i nomi, `initialize` e `shutdown` vengono utilizzati dalla Kinesis Client Library per consentire all'elaboratore di record di sapere quando dovrebbe essere pronto a iniziare a ricevere record e quando dovrebbe aspettarsi di non ricevere più record, in modo da poter effettuare qualsiasi attività di configurazione e cessazione specifica per l'applicazione. Il codice relativo viene fornito per te. L'elaborazione principale si verifica nel metodo `processRecords`, che a sua volta utilizza `processRecord` per ogni record. Quest'ultimo metodo viene fornito come codice di base per lo più vuoto da implementare nella fase successiva, dove è spiegato ulteriormente.

Da segnalare è anche l'implementazione dei metodi di supporto per `processRecord`, ovvero `reportStats` e `resetStats`, che sono vuoti nel codice sorgente originale.

Il metodo `processRecords` viene implementato per te ed esegue questa procedura:

- Per ogni record passato, chiama `processRecord` su di esso.
- Se è trascorso almeno 1 minuto dall'ultimo report, chiama `reportStats()`, che consente di stampare le statistiche più recenti, seguito da `resetStats()`, che cancella le statistiche in modo che l'intervallo successivo includa solo i nuovi record.
- Imposta l'orario della creazione di report successiva.
- Se è trascorso almeno 1 minuto dall'ultimo checkpoint, chiama `checkpoint()`.
- Imposta l'orario della creazione di checkpoint successiva.

Questo metodo utilizza intervalli di 60 secondi per la frequenza di creazione di report e checkpoint. Per ulteriori informazioni sulla creazione di checkpoint, consulta Informazioni aggiuntive sul consumatore.

StockStats classe

Questa classe fornisce la conservazione dei dati e il monitoraggio delle statistiche per i titoli più comuni nel tempo. Questo codice viene fornito per te e include i seguenti metodi:

- `addStockTrade(StockTrade)`: inserisce un dato `StockTrade` nelle statistiche in esecuzione.
- `toString()`: restituisce le statistiche in una stringa formattata.

Questa classe tiene traccia delle azioni più popolari tenendo un conteggio progressivo del numero totale di negoziazioni per ogni azione e del conteggio massimo. Aggiorna questi conteggi ogni volta che si verifica uno scambio.

Aggiungi codice ai metodi della classe `StockTradeRecordProcessor`, come mostrato nella procedura seguente.

Per implementare il consumer

1. Implementare il metodo `processRecord` creando un'istanza di un oggetto `StockTrade` delle dimensioni corrette e aggiungendo a essa i dati del record, registrando un avviso se si verifica un problema.

```
StockTrade trade = StockTrade.fromJsonAsBytes(record.getData().array());
if (trade == null) {
    LOG.warn("Skipping record. Unable to parse record into StockTrade. Partition
 Key: " + record.getPartitionKey());
    return;
}
stockStats.addStockTrade(trade);
```

2. Implementare un metodo `reportStats` semplice. Modificare liberamente il formato di output in base alle proprie preferenze.

```
System.out.println("****** Shard " + kinesisShardId + " stats for last 1 minute
 ******\n" +
                  stockStats + "\n" +

 "****************************************************************\n");
```

3. Infine, implementare il metodo `resetStats`, che crea una nuova istanza `stockStats`.

```
stockStats = new StockStats();
```

Per eseguire il consumer

1. Eseguire il producer scritto in per inserire record di scambi simulati nel flusso.
2. Verifica che la coppia chiave di accesso e chiave segreta recuperata durante la creazione dell'utente IAM sia stata salvata nel file `~/.aws/credentials`.
3. Eseguire la classe `StockTradesProcessor` con i seguenti argomenti:

   ```
   StockTradesProcessor StockTradeStream us-west-2
   ```

   Nota: se è stato creato un flusso in una regione diversa da `us-west-2`, è necessario specificare quella regione qui.

Dopo un minuto, si dovrebbe visualizzare un output come il seguente, aggiornato ogni minuto:

```
****** Shard shardId-000000000001 stats for last 1 minute ******
Most popular stock being bought: WMT, 27 buys.
Most popular stock being sold: PTR, 14 sells.
************************************************************
```

Informazioni aggiuntive sul consumatore

Se hai familiarità con i vantaggi della Kinesis Client Library discussi in Sviluppa i consumatori di KCL 1.x e altrove, potresti chiederti perché utilizzarla qui. Sebbene sia possibile utilizzare solo un singolo flusso di partizioni e una singola istanza consumer per elaborarlo, è comunque più semplice implementare il consumer utilizzando la KCL. Confronta la procedura di implementazione di codice nella sezione del producer con quella del consumer e potrai avere un'idea della maggiore semplicità di implementazione di un consumer. Ciò è principalmente dovuto ai servizi forniti dalla KCL.

In questa applicazione, devi concentrarti sull'implementazione di una classe di processore di record in grado di elaborare singoli record. Non devi preoccuparti di come i record vengono recuperati dal flusso di dati Kinesis; la KCL recupera i record e invoca il processore di record ogni volta che non sono presenti nuovi record disponibili. Inoltre, non devi preoccuparti nemmeno di quante istanze shard e consumer sono presenti. Se il flusso è stato incrementato, non è necessario riscrivere l'applicazione per gestire più di un'istanza shard o consumer.

Il termine checkpoint significa registrare il punto del flusso fino ai record di dati che sono stati consumati ed elaborati finora. Se l'applicazione si blocca, lo stream viene letto da quel punto e non dall'inizio dello stream. Il soggetto della creazione di checkpoint e i relativi modelli di progettazione e best practice non verranno trattati in questo capitolo. Tuttavia, è qualcosa che potresti incontrare negli ambienti di produzione.

Come illustrato in , le operazioni `put` nell'API del flusso di dati Kinesis richiedono una chiave di partizione come input. Il flusso di dati Kinesis utilizza una chiave di partizione come meccanismo per dividere i record tra più partizioni (quando è presente più di una partizione nel flusso). La stessa chiave di partizione esegue l'instradamento sempre allo stesso shard. In questo modo, il consumer che elabora un determinato shard può essere progettato presupponendo che i record con la stessa chiave di partizione vengano inviati solo a quel consumer e che nessun record con la stessa chiave di partizione venga inviato a un altro consumer. Pertanto, il ruolo di lavoro di un consumer può aggregare tutti i record con la stessa chiave di partizione senza trascurare dati necessari.

In questa applicazione, l'elaborazione di record da parte del consumer non è un processo intensivo, per cui puoi utilizzare una partizione ed eseguire l'elaborazione nello stesso thread della KCL. Tuttavia, prima valuta la possibilità di incrementare il numero di shard. In alcuni casi, potresti voler trasferire l'elaborazione a un altro thread oppure utilizzare un pool di thread se prevedi che il processo di elaborazione dei record sia intensivo. In questo modo, la KCL è in grado di recuperare i nuovi record più rapidamente, mentre gli altri thread possono elaborare i record in parallelo. La progettazione multithread non è banale e deve essere affrontata con tecniche avanzate, quindi aumentare il numero di shard è di solito il modo più efficace per scalare.

Passaggi successivi

(Facoltativo) Estendi il numero di utenti

(Facoltativo) Estendi il numero di utenti

L'applicazione in Tutorial: elabora i dati di borsa in tempo reale utilizzando KPL e KCL 1.x potrebbe già essere sufficiente per i tuoi scopi. Questa sezione opzionale mostra in che modo puoi ampliare la funzionalità del codice consumer per uno scenario leggermente più elaborato.

Se desideri conoscere i maggiori ordini di vendita ogni minuto, puoi modificare la classe `StockStats` in tre punti per supportare questa nuova priorità.

Per ampliare la funzionalità del consumer

1. Aggiungere nuove variabili di istanza:

```
// Ticker symbol of the stock that had the largest quantity of shares sold
private String largestSellOrderStock;
// Quantity of shares for the largest sell order trade
private long largestSellOrderQuantity;
```

2. Aggiungere il seguente codice a `addStockTrade`:

```
if (type == TradeType.SELL) {
    if (largestSellOrderStock == null || trade.getQuantity() >
largestSellOrderQuantity) {
        largestSellOrderStock = trade.getTickerSymbol();
        largestSellOrderQuantity = trade.getQuantity();
    }
}
```

3. Modificare il metodo `toString` per stampare le informazioni aggiuntive:

```
public String toString() {
    return String.format(
            "Most popular stock being bought: %s, %d buys.%n" +
            "Most popular stock being sold: %s, %d sells.%n" +
            "Largest sell order: %d shares of %s.",
            getMostPopularStock(TradeType.BUY),
getMostPopularStockCount(TradeType.BUY),
            getMostPopularStock(TradeType.SELL),
getMostPopularStockCount(TradeType.SELL),
            largestSellOrderQuantity, largestSellOrderStock);
}
```

Se si esegue il consumer ora (ricordare di eseguire anche il producer), dovrebbe essere visualizzato un output simile a questo:

```
****** Shard shardId-000000000001 stats for last 1 minute ******
 Most popular stock being bought: WMT, 27 buys.
 Most popular stock being sold: PTR, 14 sells.
 Largest sell order: 996 shares of BUD.
 ************************************************************
```

Passaggi successivi

Pulizia delle risorse

Pulizia delle risorse

Poiché l'utilizzo del flusso di dati Kinesis è a pagamento, una volta terminato di usarlo assicurati di eliminarlo insieme alle tabelle Amazon DynamoDB corrispondenti. Vengono infatti applicati costi nominali a un flusso attivo, anche quando non invii o ricevi record. Ciò si verifica perché un flusso attivo utilizza risorse monitorando in modo continuo i record in entrata e le richieste per ottenere record.

Per eliminare il flusso e la tabella

1. Chiudere i producer e i consumer ancora in esecuzione.
2. Apri la console Kinesis in /kinesis. https://console.aws.amazon.com
3. Scegliere il flusso creato per questa applicazione (`StockTradeStream`).
4. Scegliere Delete Stream (Elimina flusso).
5. Apri la console DynamoDB all'indirizzo. https://console.aws.amazon.com/dynamodb/
6. Eliminare la tabella `StockTradesProcessor`.

Riepilogo

L'elaborazione di una grande quantità di dati quasi in tempo reale non richiede la scrittura di codice complicato o lo sviluppo di un'infrastruttura enorme. Elaborare una piccola quantità di dati (come la scrittura`processRecord(Record)`) è tanto semplice quanto scrivere la logica, ma usare Kinesis Data Streams per scalare in modo da funzionare per una grande quantità di dati in streaming. Non devi preoccuparti del dimensionamento dell'elaborazione, perché il flusso di dati Kinesis lo gestisce per conto tuo. Devi soltanto inviare i tuoi record di streaming al flusso di dati Kinesis e scrivere la logica per elaborare ogni nuovo record ricevuto.

Di seguito sono elencati alcuni miglioramenti potenziali per questa applicazione.

Aggregazione tra tutti gli shard

> Al momento, puoi ottenere statistiche derivate dall'aggregazione dei record di dati ricevuti da un singolo ruolo di lavoro e provenienti da un solo shard (uno shard non può essere elaborato da più di un ruolo di lavoro in un'unica applicazione allo stesso tempo). Quando esegui il

dimensionamento e disponi di più di uno shard, potresti voler effettuare l'aggregazione tra tutti gli shard. Per farlo, ti occorre un'architettura pipeline in cui l'output di ogni ruolo di lavoro viene utilizzato in un altro flusso con un singolo shard, che viene elaborato da un ruolo di lavoro che aggrega gli output della prima fase. Poiché i dati della prima fase sono limitati (un campione al minuto per shard), possono essere gestiti facilmente da uno shard.

Dimensionamento dell'elaborazione

Quando il flusso viene incrementato per disporre di numerosi shard (dal momento che molti producer inviano dati), per dimensionare l'elaborazione è possibile aggiungere ulteriori ruoli di lavoro. Puoi eseguire i worker in EC2 istanze Amazon e utilizzare i gruppi di Auto Scaling.

Usa i connettori per Amazon S3/DynamoDB/Amazon Redshift/Storm

Poiché uno stream viene elaborato continuamente, il relativo output può essere inviato ad altre destinazioni. AWS fornisce connettori per integrare Kinesis Data Streams AWS con altri servizi e strumenti di terze parti.

Passaggi successivi

- Per ulteriori informazioni sull'utilizzo delle operazioni API del flusso di dati Kinesis, consulta Sviluppa i produttori utilizzando l'API Amazon Kinesis Data Streams con AWS SDK per Java, Sviluppa consumatori con produttività condivisa con AWS SDK per Java e Crea e gestisci flussi di dati Kinesis.
- Per ulteriori informazioni sulla Kinesis Client Library, consulta Sviluppa i consumatori di KCL 1.x.
- Per ulteriori informazioni su come ottimizzare l'applicazione, consulta Ottimizza i consumatori di Amazon Kinesis Data Streams.

Tutorial: analizza i dati di borsa in tempo reale utilizzando Amazon Managed Service per Apache Flink

Lo scenario per questo tutorial richiede l'importazione del flusso di negoziazioni in un flusso di dati e la scrittura di una semplice applicazione del Servizio gestito da Amazon per Apache Flink che esegua calcoli sul flusso. Imparerai come inviare un flusso di record a Kinesis Data Streams e implementare un'applicazione che consuma ed elabora i record quasi in tempo reale.

Con Amazon Managed Service for Apache Flink, puoi usare Java o Scala per elaborare e analizzare i dati di streaming. Il servizio consente di creare ed eseguire codice Java o Scala su sorgenti di

streaming per eseguire analisi di serie temporali, alimentare dashboard in tempo reale e creare metriche in tempo reale.

È possibile creare applicazioni Flink nel servizio gestito per Apache Flink utilizzando le librerie open source basate su Apache Flink. Apache Flink è un celebre framework e motore per l'elaborazione di flussi di dati.

> **Important**
>
> Dopo aver creato due flussi di dati e un'applicazione, sul tuo account vengono addebitati costi nominali per l'utilizzo di Kinesis Data Streams e Managed Service for Apache Flink perché non sono idonei per il piano gratuito. AWS Quando hai finito con questa applicazione, elimina le tue risorse per evitare di incorrere in addebiti. AWS

Il codice non accede ai dati del mercato azionario, ma simula il flusso delle negoziazioni. A tale scopo, utilizza un generatore di negoziazioni casuale. Se disponi dell'accesso a un flusso di negoziazioni in tempo reale, potresti essere interessato a derivare statistiche utili e tempestive da quel flusso. Ad esempio, potresti eseguire un'analisi basata su finestra scorrevole per determinare i titoli più acquistati negli ultimi 5 minuti. Oppure potresti ricevere una notifica ogni volta che viene effettuato un ordine di vendita troppo grande (ossia, che include un numero eccessivo di titoli). Puoi estendere il codice in questa serie per fornire tale funzionalità.

Gli esempi mostrati utilizzano la regione Stati Uniti occidentali (Oregon), ma si applicano a qualunque Regione AWS che supporta il servizio gestito per Apache Flink.

Attività

- Prerequisiti per il completamento degli esercizi
- Configura un AWS account e crea un utente amministratore
- Configura il AWS Command Line Interface (AWS CLI)
- Crea ed esegui un servizio gestito per l'applicazione Apache Flink

Prerequisiti per il completamento degli esercizi

Per completare le fasi in questa guida, è richiesto quanto segue:

- Java Development Kit (JDK) versione 8. Imposta la variabile di ambiente `JAVA_HOME` in modo che punti alla posizione di installazione di JDK.

- Ti consigliamo di utilizzare un ambiente di sviluppo (ad esempio Eclipse Java Neon o IntelliJ IDEA) per sviluppare e compilare l'applicazione.
- Client Git. Installa il client Git se non lo hai già fatto.
- Apache Maven Compiler Plugin. Maven deve trovarsi nel percorso di lavoro. Per testare l'installazione Apache Maven, immetti quanto segue:

```
$ mvn -version
```

Per iniziare, vai alla pagina Configura un AWS account e crea un utente amministratore.

Configura un AWS account e crea un utente amministratore

Prima di utilizzare Amazon Managed Service for Apache Flink per la prima volta, completa le seguenti attività:

1. Registrati per AWS
2. Crea un utente IAM

Registrati per AWS

Quando ti registri ad Amazon Web Services (AWS), il tuo AWS account viene automaticamente registrato per tutti i servizi in AWS, incluso Amazon Managed Service per Apache Flink. Ti vengono addebitati solo i servizi che utilizzi.

Con il servizio gestito per Apache Flink vengono addebitati esclusivamente i costi per le risorse utilizzate. Se sei un nuovo cliente AWS , puoi iniziare a utilizzare il servizio gestito per Apache Flink gratuitamente. Per ulteriori informazioni, consulta Piano gratuito AWS.

Se hai già un AWS account, passa all'attività successiva. Se non disponi di un account AWS , procedi nel modo seguente per crearne uno.

Per creare un account AWS

1. Apri la https://portal.aws.amazon.com/billing/registrazione.
2. Segui le istruzioni online.

   Nel corso della procedura di registrazione riceverai una telefonata, durante la quale sarà necessario inserire un codice di verifica attraverso la tastiera del telefono.

Quando ti iscrivi a un Account AWS, Utente root dell'account AWSviene creato un. L'utente root dispone dell'accesso a tutte le risorse e tutti i Servizi AWS nell'account. Come best practice di sicurezza, assegna l'accesso amministrativo a un utente e utilizza solo l'utente root per eseguire attività che richiedono l'accesso di un utente root.

Annota l'ID del tuo AWS account perché ti servirà per l'attività successiva.

Crea un utente IAM

I servizi in AWS, come Amazon Managed Service for Apache Flink, richiedono l'immissione di credenziali al momento dell'accesso. In questo modo, il servizio può stabilire se si dispone delle autorizzazioni per accedere alle risorse di proprietà del servizio stesso. AWS Management Console Richiede l'immissione della password.

Puoi creare chiavi di accesso per il tuo AWS account per accedere a AWS Command Line Interface (AWS CLI) o all'API. Tuttavia, non ti consigliamo di accedere AWS utilizzando le credenziali del tuo AWS account. Ti consigliamo invece di utilizzare AWS Identity and Access Management (IAM). Crea un utente IAM, aggiungilo a un gruppo IAM con autorizzazioni amministrative, quindi concedi le autorizzazioni amministrative all'utente IAM creato. Potrai quindi accedere ad AWS utilizzando un URL speciale e le credenziali dell'utente IAM.

Se ti sei registrato AWS ma non hai ancora creato un utente IAM, puoi crearne uno utilizzando la console IAM.

Per eseguire gli esercizi delle Nozioni di base di questa guida è necessario disporre di un utente (`adminuser`) con autorizzazioni di amministratore. Per creare un `adminuser` nel tuo account, utilizza la procedura indicata di seguito.

Per creare un gruppo di amministratori

1. Accedi AWS Management Console e apri la console IAM all'indirizzo https://console.aws.amazon.com/iam/.
2. Nel riquadro di navigazione, scegliere Groups (Gruppi), quindi Create New Group (Crea nuovo gruppo).
3. In Group Name (Nome gruppo), digitare un nome per il gruppo, come **`Administrators`** , quindi scegliere Next Step (Fase successiva).

4. Nell'elenco delle politiche, seleziona la casella di controllo accanto alla AdministratorAccesspolitica. È possibile utilizzare il menu Filter (Filtro) e la casella Search (Cerca) per filtrare l'elenco di policy.
5. Scegliere Next Step (Fase successiva), quindi Create Group (Crea gruppo).

Il nuovo gruppo viene elencato nell'area Group Name (Nome gruppo).

Creazione di un utente IAM per se stessi, aggiunta al gruppo Amministratori e creazione di una password

1. Nel riquadro di navigazione, scegli Users (Utenti), quindi scegli Add user (Aggiungi utente).
2. Nella casella User name (Nome utente) , immettere un nome utente.
3. Scegli Accesso programmatico e Accesso alla Console di gestione AWS .
4. Scegli Successivo: autorizzazioni.
5. Selezionare la casella di controllo accanto al gruppo Administrators (Amministratori). Scegli quindi Next: Review (Avanti: Verifica).
6. Selezionare Create user (Crea utente).

Per accedere come nuovo utente IAM

1. Esci da AWS Management Console.
2. Per accedere alla console, utilizzare il seguente formato URL:

   `https://aws_account_number.signin.aws.amazon.com/console/`

   *aws_account_number*È l'ID AWS del tuo account senza trattini. Ad esempio, se l'ID AWS dell'account è 1234-5678-9012, sostituiscilo con. *aws_account_number* **123456789012** Per informazioni su come trovare il numero del tuo account, consulta l'ID del tuo account e il suo alias nella Guida per l' AWS utente IAM.
3. Immettere il nome utente e la password di IAM appena creati. Una volta effettuato l'accesso, la barra di navigazione mostra *your_user_name* @*your_aws_account_id*.

> **Note**
>
> Se non desideri che l'URL della pagina di accesso contenga l'ID AWS dell'account, puoi creare un alias per l'account.

Per creare o rimuovere un alias dell'account

1. Aprire la console IAM all'indirizzo https://console.aws.amazon.com/iam/.
2. Nel riquadro di navigazione, selezionare Dashboard (Pannello di controllo).
3. Individuare il link di accesso agli utenti IAM.
4. Per creare l'alias, scegliere Customize (Personalizza). Immettere il nome che si desidera usare per l'alias, quindi selezionare Yes, Create (Sì, crea).
5. Per rimuovere l'alias, scegliere Customize (Personalizza) e quindi Yes, Delete (Sì, elimina). L'URL di accesso torna a utilizzare l'ID dell'account. AWS

Per effettuare l'accesso dopo aver creato un alias dell'account, utilizzare il seguente URL:

`https://your_account_alias.signin.aws.amazon.com/console/`

Per verificare il link di accesso degli utenti IAM al tuo account, apri la console IAM e controlla in IAM users sign-in link (Link di accesso utenti IAM) nel pannello di controllo.

Per ulteriori informazioni su IAM, consulta:

- AWS Identity and Access Management (IAM)
- Nozioni di base
- Guida per l'utente di IAM

Approfondimenti

Configura il AWS Command Line Interface (AWS CLI)

Configura il AWS Command Line Interface (AWS CLI)

In questa fase, scarichi e configuri il file AWS CLI per utilizzarlo con Amazon Managed Service for Apache Flink.

> **Note**
>
> Gli esercizi sulle Nozioni di base di questa guida presuppongono che tu disponga di credenziali di amministratore (`adminuser`) nell'account per eseguire le operazioni.

> **Note**
>
> Se l'hai già AWS CLI installato, potrebbe essere necessario eseguire l'aggiornamento per ottenere le funzionalità più recenti. Per ulteriori informazioni, vedere Installazione dell'interfaccia a riga di AWS comando nella Guida AWS Command Line Interface per l'utente. Per verificare la versione di AWS CLI, esegui il seguente comando:
>
> ```
> aws --version
> ```
>
> Gli esercizi di questo tutorial richiedono la seguente AWS CLI versione o successiva:
>
> ```
> aws-cli/1.16.63
> ```

Per configurare il AWS CLI

1. Scarica e configura la AWS CLI. Per le istruzioni, consulta i seguenti argomenti nella Guida per l'utente dell'AWS Command Line Interface :
   - Installazione dell' AWS Command Line Interface
   - Configurazione della AWS CLI
2. Aggiungi un profilo denominato per l'utente amministratore nel file di AWS CLI configurazione. Questo profilo viene utilizzato durante l'esecuzione dei AWS CLI comandi. Per ulteriori informazioni sui profili denominati, consulta Profili denominati in Guida per l'utente dell'AWS Command Line Interface .

   ```
   [profile adminuser]
   aws_access_key_id = adminuser access key ID
   aws_secret_access_key = adminuser secret access key
   region = aws-region
   ```

Per un elenco delle AWS regioni disponibili, consulta AWS Regioni ed endpoint in. Riferimenti generali di Amazon Web Services

3. Verifica la configurazione digitando il comando help riportato di seguito al prompt dei comandi:

```
aws help
```

Dopo aver configurato un AWS account AWS CLI, puoi provare l'esercizio successivo, in cui configurerai un'applicazione di esempio e verificherai la end-to-end configurazione.

Approfondimenti

Crea ed esegui un servizio gestito per l'applicazione Apache Flink

Crea ed esegui un servizio gestito per l'applicazione Apache Flink

In questo esercizio, viene creata un'applicazione del servizio gestito per Apache Flink con flussi di dati come origine e come sink.

Questa sezione contiene le fasi seguenti:

- Crea due flussi di dati Amazon Kinesis
- Scrittura di record di esempio nel flusso di input
- Scarica ed esamina il codice Java per lo streaming di Apache Flink
- Compilate il codice dell'applicazione
- Caricate il codice Java di streaming Apache Flink
- Crea ed esegui l'applicazione Managed Service for Apache Flink

Crea due flussi di dati Amazon Kinesis

Prima di creare un Amazon Managed Service per Apache Flink per questo esercizio, crea due flussi di dati Kinesis (e). `ExampleInputStream` `ExampleOutputStream` L'applicazione utilizza questi flussi per i flussi di origine e di destinazione dell'applicazione.

Puoi creare questi flussi utilizzando la console Amazon Kinesis o il comando AWS CLI seguente. Per istruzioni sulla console, consulta Creazione e aggiornamento dei flussi di dati.

Per creare i flussi di dati (AWS CLI)

1. Per creare il primo stream (`ExampleInputStream`), usa il seguente comando Amazon Kinesis `create-stream` AWS CLI .

```
$ aws kinesis create-stream \
--stream-name ExampleInputStream \
--shard-count 1 \
--region us-west-2 \
--profile adminuser
```

2. Per creare il secondo flusso utilizzato dall'applicazione per scrivere l'output, esegui lo stesso comando, modificando il nome del flusso in `ExampleOutputStream`.

```
$ aws kinesis create-stream \
--stream-name ExampleOutputStream \
--shard-count 1 \
--region us-west-2 \
--profile adminuser
```

Scrittura di record di esempio nel flusso di input

In questa sezione, viene utilizzato uno script Python per scrivere record di esempio nel flusso per l'applicazione da elaborare.

> **Note**
>
> Questa sezione richiede AWS SDK for Python (Boto).

1. Crea un file denominato `stock.py` con i seguenti contenuti:

```
import datetime
import json
import random
import boto3

STREAM_NAME = "ExampleInputStream"
```

```
def get_data():
    return {
        "EVENT_TIME": datetime.datetime.now().isoformat(),
        "TICKER": random.choice(["AAPL", "AMZN", "MSFT", "INTC", "TBV"]),
        "PRICE": round(random.random() * 100, 2),
    }


def generate(stream_name, kinesis_client):
    while True:
        data = get_data()
        print(data)
        kinesis_client.put_record(
            StreamName=stream_name, Data=json.dumps(data),
 PartitionKey="partitionkey"
        )


if __name__ == "__main__":
    generate(STREAM_NAME, boto3.client("kinesis"))
```

2. Successivamente nel tutorial, esegui lo script `stock.py` per inviare dati all'applicazione.

```
$ python stock.py
```

Scarica ed esamina il codice Java per lo streaming di Apache Flink

Il codice dell'applicazione Java per questi esempi è disponibile da. GitHub Per scaricare il codice dell'applicazione, esegui le operazioni descritte di seguito:

1. Clona il repository remoto con il comando seguente:

```
git clone https://github.com/aws-samples/amazon-kinesis-data-analytics-java-
examples.git
```

2. Passa alla directory `GettingStarted`.

Il codice dell'applicazione si trova nei file `CustomSinkStreamingJob.java` e `CloudWatchLogSink.java`. Tieni presente quanto segue riguardo al codice dell'applicazione:

- L'applicazione utilizza un'origine Kinesis per leggere dal flusso di origine. Il seguente snippet crea il sink Kinesis:

```
return env.addSource(new FlinkKinesisConsumer<>(inputStreamName,
                new SimpleStringSchema(), inputProperties));
```

Compilate il codice dell'applicazione

In questa sezione, viene utilizzato il compilatore Apache Maven per creare il codice Java per l'applicazione. Per ulteriori informazioni sull'installazione di Apache Maven e Java Development Kit (JDK), consulta Prerequisiti per il completamento degli esercizi.

L'applicazione Java richiede i componenti seguenti:

- Un file Project Object Model (pom.xml). Questo file contiene informazioni sulla configurazione e le dipendenze dell'applicazione, tra cui le librerie Amazon Managed Service per Apache Flink.
- Un metodo `main` contenente la logica dell'applicazione.

> **Note**
>
> Per utilizzare il connettore Kinesis per la seguente applicazione, è necessario scaricare il codice sorgente del connettore e crearlo come descritto nella documentazione di Apache Flink.

Per creare e compilare il codice dell'applicazione

1. Creare un'applicazione Java/Maven nell'ambiente di sviluppo. Per ulteriori informazioni su come creare un'applicazione, consulta la documentazione per l'ambiente di sviluppo:
   - Creazione del primo progetto Java (Eclipse Java Neon)
   - Creazione, esecuzione e compressione della prima applicazione Java (IntelliJ Idea)
2. Utilizzare il codice seguente per un file denominato `StreamingJob.java`.

```
package com.amazonaws.services.kinesisanalytics;

import com.amazonaws.services.kinesisanalytics.runtime.KinesisAnalyticsRuntime;
```

```
import org.apache.flink.api.common.serialization.SimpleStringSchema;
import org.apache.flink.streaming.api.datastream.DataStream;
import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
import org.apache.flink.streaming.connectors.kinesis.FlinkKinesisConsumer;
import org.apache.flink.streaming.connectors.kinesis.FlinkKinesisProducer;
import
 org.apache.flink.streaming.connectors.kinesis.config.ConsumerConfigConstants;

import java.io.IOException;
import java.util.Map;
import java.util.Properties;

public class StreamingJob {

    private static final String region = "us-east-1";
    private static final String inputStreamName = "ExampleInputStream";
    private static final String outputStreamName = "ExampleOutputStream";

    private static DataStream<String>
 createSourceFromStaticConfig(StreamExecutionEnvironment env) {
        Properties inputProperties = new Properties();
        inputProperties.setProperty(ConsumerConfigConstants.AWS_REGION, region);

 inputProperties.setProperty(ConsumerConfigConstants.STREAM_INITIAL_POSITION,
 "LATEST");

        return env.addSource(new FlinkKinesisConsumer<>(inputStreamName, new
 SimpleStringSchema(), inputProperties));
    }

    private static DataStream<String>
 createSourceFromApplicationProperties(StreamExecutionEnvironment env)
            throws IOException {
        Map<String, Properties> applicationProperties =
 KinesisAnalyticsRuntime.getApplicationProperties();
        return env.addSource(new FlinkKinesisConsumer<>(inputStreamName, new
 SimpleStringSchema(),
                applicationProperties.get("ConsumerConfigProperties")));
    }

    private static FlinkKinesisProducer<String> createSinkFromStaticConfig() {
        Properties outputProperties = new Properties();
        outputProperties.setProperty(ConsumerConfigConstants.AWS_REGION, region);
        outputProperties.setProperty("AggregationEnabled", "false");
```

```
        FlinkKinesisProducer<String> sink = new FlinkKinesisProducer<>(new
SimpleStringSchema(), outputProperties);
        sink.setDefaultStream(outputStreamName);
        sink.setDefaultPartition("0");
        return sink;
    }

    private static FlinkKinesisProducer<String>
createSinkFromApplicationProperties() throws IOException {
        Map<String, Properties> applicationProperties =
KinesisAnalyticsRuntime.getApplicationProperties();
        FlinkKinesisProducer<String> sink = new FlinkKinesisProducer<>(new
SimpleStringSchema(),
                applicationProperties.get("ProducerConfigProperties"));

        sink.setDefaultStream(outputStreamName);
        sink.setDefaultPartition("0");
        return sink;
    }

    public static void main(String[] args) throws Exception {
        // set up the streaming execution environment
        final StreamExecutionEnvironment env =
StreamExecutionEnvironment.getExecutionEnvironment();

        /*
         * if you would like to use runtime configuration properties, uncomment the
         * lines below
         * DataStream<String> input = createSourceFromApplicationProperties(env);
         */

        DataStream<String> input = createSourceFromStaticConfig(env);

        /*
         * if you would like to use runtime configuration properties, uncomment the
         * lines below
         * input.addSink(createSinkFromApplicationProperties())
         */

        input.addSink(createSinkFromStaticConfig());

        env.execute("Flink Streaming Java API Skeleton");
    }
```

```
}
```

Notare quanto segue riguardo l'esempio di codice precedente:

- Questo file contiene il metodo `main` che definisce la funzionalità dell'applicazione.
- L'applicazione crea connettori di origine e sink per accedere alle risorse esterne utilizzando un oggetto `StreamExecutionEnvironment`.
- L'applicazione crea connettori di origine e sink utilizzando proprietà statiche. Per usare proprietà dell'applicazione dinamiche, utilizza i metodi `createSourceFromApplicationProperties` e `createSinkFromApplicationProperties` per creare i connettori. Questi metodi leggono le proprietà dell'applicazione per configurare il connettori.

3. Per usare il codice dell'applicazione, compila il codice e comprimilo in un file JAR. È possibile compilare e creare un pacchetto del codice in uno di due modi:

   - Utilizzare lo strumento Maven della riga di comando. Crea il file JAR eseguendo il comando seguente nella directory che contiene il file `pom.xml`:

     ```
     mvn package
     ```

   - Utilizza il tuo ambiente di sviluppo. Per informazioni dettagliate, consulta la documentazione relativa all'ambiente di sviluppo.

   È possibile caricare il pacchetto come un file JAR, oppure comprimere il pacchetto e caricarlo come un file ZIP. Se create l'applicazione utilizzando AWS CLI, specificate il tipo di contenuto del codice (JAR o ZIP).
4. Se si verificano errori durante la compilazione, verifica che la variabile di ambiente `JAVA_HOME` sia impostata correttamente.

Se l'applicazione viene compilata correttamente, viene creato il seguente file:

`target/java-getting-started-1.0.jar`

Caricate il codice Java di streaming Apache Flink

In questa sezione, viene creato un bucket Amazon Simple Storage Service (Amazon S3) e caricato il codice dell'applicazione.

Per caricare il codice dell'applicazione

1. Apri la console Amazon S3 all'indirizzo. https://console.aws.amazon.com/s3/
2. Seleziona Crea bucket.
3. Inserisci **ka-app-code-*<username>*** nel campo Nome bucket. Aggiungi un suffisso al nome del bucket, ad esempio il nome utente, per renderlo globalmente univoco. Scegli Next (Successivo).
4. Nella fase Configura opzioni, non modificare le impostazioni e scegli Successivo.
5. Nella fase Imposta autorizzazioni, non modificare le impostazioni e scegli Successivo.
6. Seleziona Crea bucket.
7. Nella console Amazon S3, scegli il ***<username>*** bucket ka-app-code- e scegli Carica.
8. Nella fase Seleziona file, scegli Aggiungi file. Individua il file `java-getting-started-1.0.jar` creato nella fase precedente. Scegli Next (Successivo).
9. Nella fase Imposta autorizzazioni, non modificare le impostazioni. Scegli Next (Successivo).
10. Nella fase Imposta proprietà, non modificare le impostazioni. Scegli Carica.

Il codice dell'applicazione è ora archiviato in un bucket Amazon S3 accessibile dall'applicazione.

Crea ed esegui l'applicazione Managed Service for Apache Flink

È possibile creare ed eseguire un'applicazione del servizio gestito per Apache Flink utilizzando la console o la AWS CLI.

> **Note**
>
> Quando crei l'applicazione utilizzando la console, le tue risorse AWS Identity and Access Management (IAM) e Amazon CloudWatch Logs vengono create automaticamente. Quando crei l'applicazione utilizzando AWS CLI, crei queste risorse separatamente.

Argomenti

- Crea ed esegui l'applicazione (Console)
- Crea ed esegui l'applicazione (AWS CLI)

Crea ed esegui l'applicazione (Console)

Segui questi passaggi per creare, configurare, aggiornare ed eseguire l'applicazione utilizzando la console.

Creazione dell'applicazione

1. Apri la console Kinesis in /kinesis. https://console.aws.amazon.com
2. Sul pannello di controllo di Amazon Kinesis, scegli Crea applicazione di analisi.
3. Nella pagina Kinesis Analytics - Create application (Kinesis Analytics - Crea applicazione), fornire i dettagli dell'applicazione come segue:
   - Per Nome applicazione, immetti **`MyApplication`**.
   - Per Descrizione, inserisci **`My java test app`**.
   - Per Runtime, scegliere Apache Flink 1.6.
4. Per Autorizzazioni di accesso, scegli Crea/aggiorna **`kinesis-analytics-MyApplication-us-west-2`** per il ruolo IAM.
5. Scegli Crea applicazione.

> **Note**
>
> Quando crei un'applicazione Amazon Managed Service for Apache Flink utilizzando la console, hai la possibilità di creare un ruolo e una policy IAM per la tua applicazione. L'applicazione utilizza questo ruolo e questa policy per accedere alle sue risorse dipendenti. Queste risorse IAM sono denominate utilizzando il nome dell'applicazione e la Regione come segue:
>
> - Policy: `kinesis-analytics-service-MyApplication-us-west-2`
> - Ruolo: `kinesis-analytics-MyApplication-us-west-2`

Modifica la policy IAM

Modifica la policy IAM per aggiungere le autorizzazioni per accedere ai flussi di dati Kinesis.

1. Aprire la console IAM all'indirizzo https://console.aws.amazon.com/iam/.

2. Seleziona Policy. Scegli la policy **`kinesis-analytics-service-MyApplication-us-west-2`** creata dalla console nella sezione precedente.
3. Nella pagina Riepilogo, scegli Modifica policy. Scegli la scheda JSON.
4. Aggiungi alla policy la sezione evidenziata del seguente esempio di policy. Sostituisci l'account di esempio IDs (*012345678901*) con l'ID del tuo account.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ReadCode",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": [
                "arn:aws:s3:::ka-app-code-username/java-getting-started-1.0.jar"
            ]
        },
        {
            "Sid": "ListCloudwatchLogGroups",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:*"
            ]
        },
        {
            "Sid": "ListCloudwatchLogStreams",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogStreams"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:*"
            ]
        },
        {
```

```
            "Sid": "PutCloudwatchLogs",
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:kinesis-analytics-log-stream"
            ]
        },
        {
            "Sid": "ReadInputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
        },
        {
            "Sid": "WriteOutputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleOutputStream"
        }
    ]
}
```

Configura l'applicazione

1. Nella MyApplicationpagina, scegli Configura.
2. Nella pagina Configura applicazione, fornisci la Posizione del codice:
   - Per Bucket Amazon S3, inserisci **ka-app-code-*<username>***.
   - Per Percorso dell'oggetto Amazon S3, inserisci **java-getting-started-1.0.jar**
3. In Accesso alle risorse dell'applicazione, per Autorizzazioni di accesso, scegli Crea/aggiorna **kinesis-analytics-MyApplication-us-west-2** per il ruolo IAM.
4. In Proprietà, per ID gruppo, inserisci **ProducerConfigProperties**.
5. Immetti i valori e le proprietà dell'applicazione seguenti:

Chiave	Valore
flink.inputstream.initpos	**LATEST**
aws:region	**us-west-2**
AggregationEnabled	**false**

6. In Monitoraggio, accertati che il Monitoraggio del livello dei parametri sia impostato su Applicazione.
7. Per la CloudWatch registrazione, seleziona la casella di controllo Abilita.
8. Scegli Aggiorna.

> **Note**
>
> Quando scegli di abilitare la CloudWatch registrazione, Managed Service for Apache Flink crea un gruppo di log e un flusso di log per te. I nomi di tali risorse sono i seguenti:
>
> - Gruppo di log: `/aws/kinesis-analytics/MyApplication`
> - Flusso di log: `kinesis-analytics-log-stream`

Esecuzione dell'applicazione.

1. Nella MyApplicationpagina, scegli Esegui. Conferma l'operazione.
2. Quando l'applicazione è in esecuzione, aggiorna la pagina. La console mostra il Grafico dell'applicazione.

Arresta l'applicazione

Nella MyApplicationpagina, scegli Stop. Conferma l'operazione.

Aggiornamento dell'applicazione

Tramite la console, puoi aggiornare le impostazioni dell'applicazione, ad esempio le proprietà dell'applicazione, le impostazioni di monitoraggio e la posizione o il nome di file del JAR

dell'applicazione. Puoi anche ricaricare il JAR dell'applicazione dal bucket Amazon S3 se è necessario aggiornare il codice dell'applicazione.

Nella MyApplicationpagina, scegli Configura. Aggiorna le impostazioni dell'applicazione e scegli Aggiorna.

Crea ed esegui l'applicazione (AWS CLI)

In questa sezione, si utilizza AWS CLI per creare ed eseguire l'applicazione Managed Service for Apache Flink. Managed Service for Apache Flink utilizza il `kinesisanalyticsv2` AWS CLI comando per creare e interagire con le applicazioni Managed Service for Apache Flink.

Creazione di una policy di autorizzazione

Innanzitutto, crea una policy di autorizzazione con due istruzioni: una che concede le autorizzazioni per l'operazione `read` sul flusso di origine e un'altra che concede le autorizzazioni per operazioni `write` sul flusso di sink. Collega quindi la policy a un ruolo IAM (che verrà creato nella sezione successiva). Pertanto, quando il servizio gestito per Apache Flink assume il ruolo, il servizio disporrà delle autorizzazioni necessarie per leggere dal flusso di origine e scrivere nel flusso di sink.

Utilizza il codice seguente per creare la policy di autorizzazione `KAReadSourceStreamWriteSinkStream`. Sostituisci *`username`* con il nome utente utilizzato per creare il bucket Amazon S3 per archiviare il codice dell'applicazione. Sostituisci l'ID dell'account in Amazon Resource Names (ARNs) (*`012345678901`*) con l'ID del tuo account.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "S3",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": ["arn:aws:s3:::ka-app-code-username",
                "arn:aws:s3:::ka-app-code-username/*"
            ]
        },
        {
            "Sid": "ReadInputStream",
```

```
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
        },
        {
            "Sid": "WriteOutputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleOutputStream"
        }
    ]
}
```

Per step-by-step istruzioni su come creare una politica di autorizzazioni, consulta il Tutorial: Create and Attach Your First Customer Managed Policy nella IAM User Guide.

> **Note**
>
> Per accedere ad altri AWS servizi, puoi utilizzare il AWS SDK per Java. Il servizio gestito per Apache Flink imposta automaticamente le credenziali richieste dall'SDK su quelle del ruolo IAM di esecuzione del servizio associato all'applicazione. Non sono richieste fasi aggiuntive.

Creazione di un ruolo IAM

In questa sezione, crei un ruolo IAM che Managed Service for Apache Flink può assumere per leggere un flusso di origine e scrivere nel flusso sink.

Il servizio gestito per Apache Flink non può accedere al tuo flusso senza autorizzazioni. Queste autorizzazioni possono essere assegnate con un ruolo IAM. Ad ogni ruolo IAM sono collegate due policy. La policy di attendibilità concede al servizio gestito per Apache Flink l'autorizzazione per assumere il ruolo e la policy di autorizzazione determina cosa può fare il servizio assumendo questo ruolo.

Collega la policy di autorizzazione creata nella sezione precedente a questo ruolo.

Per creare un ruolo IAM

1. Aprire la console IAM all'indirizzo https://console.aws.amazon.com/iam/.

2. Nel riquadro di navigazione, seleziona Ruoli, quindi Crea nuovo ruolo.
3. In Seleziona tipo di identità attendibile, scegli Servizio AWS . In Scegli il servizio che utilizzerà questo ruolo, scegli Kinesis. In Seleziona il tuo caso d'uso, scegli Analisi dei dati Kinesis.

   Scegli Successivo: Autorizzazioni.
4. Nella pagina Allega policy di autorizzazione, seleziona Successivo: esamina. Collega le policy di autorizzazione dopo aver creato il ruolo.
5. Nella pagina Crea ruolo, immetti **`KA-stream-rw-role`** per Nome ruolo. Scegli Crea ruolo.

   È stato creato un nuovo ruolo IAM denominato `KA-stream-rw-role`. Successivamente, aggiorna le policy di attendibilità e di autorizzazione per il ruolo.
6. Collega la policy di autorizzazione al ruolo.

   > **Note**
   >
   > Per questo esercizio, il servizio gestito per Apache Flink assume questo ruolo per la lettura di dati da un flusso di dati Kinesis (origine) e la scrittura dell'output in un altro flusso di dati Kinesis. Pertanto, devi collegare la policy creata nella fase precedente, the section called “Creazione di una policy di autorizzazione”.

   a. Nella pagina Riepilogo, scegli la scheda Autorizzazioni.
   b. Scegliere Collega policy.
   c. Nella casella di ricerca, immetti **`KAReadSourceStreamWriteSinkStream`** (la policy creata nella sezione precedente).
   d. Scegli la policy KAReadInputStreamWriteOutputStream e seleziona Collega policy.

È stato creato il ruolo di esecuzione del servizio che l'applicazione utilizzerà per accedere alle risorse. Prendi nota dell'ARN del nuovo ruolo.

Per step-by-step istruzioni sulla creazione di un ruolo, consulta Creating an IAM Role (Console) nella IAM User Guide.

Creazione dell'applicazione del servizio gestito per Apache Flink

1. Salvare il seguente codice JSON in un file denominato `create_request.json`. Sostituisci l'ARN del ruolo di esempio con l'ARN per il ruolo creato in precedenza. Sostituisci il suffisso

dell'ARN del bucket (*username*) con il suffisso scelto nella sezione precedente. Sostituisci l'ID account di esempio (*012345678901*) nel ruolo di esecuzione del servizio con il tuo ID account.

```
{
    "ApplicationName": "test",
    "ApplicationDescription": "my java test app",
    "RuntimeEnvironment": "FLINK-1_6",
    "ServiceExecutionRole": "arn:aws:iam::012345678901:role/KA-stream-rw-role",
    "ApplicationConfiguration": {
        "ApplicationCodeConfiguration": {
            "CodeContent": {
                "S3ContentLocation": {
                    "BucketARN": "arn:aws:s3:::ka-app-code-username",
                    "FileKey": "java-getting-started-1.0.jar"
                }
            },
            "CodeContentType": "ZIPFILE"
        },
        "EnvironmentProperties":  {
         "PropertyGroups": [
            {
               "PropertyGroupId": "ProducerConfigProperties",
               "PropertyMap" : {
                    "flink.stream.initpos" : "LATEST",
                    "aws.region" : "us-west-2",
                    "AggregationEnabled" : "false"
               }
            },
            {
               "PropertyGroupId": "ConsumerConfigProperties",
               "PropertyMap" : {
                    "aws.region" : "us-west-2"
               }
            }
         ]
      }
    }
}
```

2. Esegui l'operazione `CreateApplication` con la richiesta precedente per creare l'applicazione:

```
aws kinesisanalyticsv2 create-application --cli-input-json file://
create_request.json
```

L'applicazione è ora creata. Avvia l'applicazione nella fase successiva.

Avvio dell'applicazione

In questa sezione, viene utilizzata l'operazione `StartApplication` per avviare l'applicazione.

Per avviare l'applicazione

1. Salvare il seguente codice JSON in un file denominato `start_request.json`.

```
{
    "ApplicationName": "test",
    "RunConfiguration": {
        "ApplicationRestoreConfiguration": {
         "ApplicationRestoreType": "RESTORE_FROM_LATEST_SNAPSHOT"
         }
    }
}
```

2. Esegui l'operazione `StartApplication` con la richiesta precedente per avviare l'applicazione:

```
aws kinesisanalyticsv2 start-application --cli-input-json file://start_request.json
```

L'applicazione è ora in esecuzione. Puoi controllare i parametri del servizio gestito per Apache Flink sulla CloudWatch console Amazon per verificare che l'applicazione funzioni.

Interruzione dell'applicazione

In questa sezione, viene utilizzata l'operazione `StopApplication` per interrompere l'applicazione.

Per interrompere l'applicazione

1. Salvare il seguente codice JSON in un file denominato `stop_request.json`.

```
{"ApplicationName": "test"
}
```

2. Esegui l'operazione `StopApplication` con la seguente richiesta di interrompere l'applicazione:

```
aws kinesisanalyticsv2 stop-application --cli-input-json file://stop_request.json
```

L'applicazione è ora interrotta.

Tutorial: utilizzo AWS Lambda con Amazon Kinesis Data Streams

In questa esercitazione, è possibile creare una funzione Lambda per utilizzare gli eventi da un flusso di dati Kinesis. In questo scenario di esempio, un'applicazione personalizzata scrive i record in un flusso di dati Kinesis. AWS Lambda quindi esegue il polling di questo flusso di dati e, quando rileva nuovi record di dati, richiama la funzione Lambda. AWS Lambda esegue quindi la funzione Lambda assumendo il ruolo di esecuzione specificato al momento della creazione della funzione Lambda.

Per istruzioni dettagliate, consulta Tutorial: Using AWS Lambda with Amazon Kinesis.

> **Note**
>
> Questo tutorial presuppone che tu abbia una certa conoscenza delle operazioni di base di Lambda e AWS Lambda della console. Se non l'hai già fatto, segui le istruzioni in Getting Started with AWS Lambda per creare la tua prima funzione Lambda.

Usa la soluzione di AWS streaming di dati per Amazon Kinesis

La soluzione AWS Streaming Data per Amazon Kinesis configura automaticamente i AWS servizi necessari per acquisire, archiviare, elaborare e distribuire dati in streaming con facilità. La soluzione offre diverse opzioni per risolvere i casi d'uso dei dati di streaming che utilizzano più AWS servizi, tra cui Kinesis Data AWS Lambda Streams, Amazon API Gateway e Amazon Managed Service for Apache Flink.

Ogni soluzione include i seguenti componenti:

- Un AWS CloudFormation pacchetto per distribuire l'esempio completo.
- Una CloudWatch dashboard per la visualizzazione delle metriche delle applicazioni.
- CloudWatch allarmi sulle metriche applicative più rilevanti.
- tutti i ruoli e le politiche IAM necessari.

La soluzione è disponibile qui: Soluzione di streaming di dati per Amazon Kinesis

Crea e gestisci flussi di dati Kinesis

Flusso di dati Amazon Kinesis ottiene grandi quantità di dati in tempo reale, li memorizza in modo durevole e li rende disponibili per il consumo. L'unità di dati archiviati dal flusso di dati Kinesis è un record di dati. Un flusso di dati rappresenta un gruppo di record di dati. I record di dati in un flusso di dati sono distribuiti in shard.

Uno shard è una sequenza di record di dati in un flusso. Funge da unità di velocità di trasmissione effettiva di base di un flusso di dati Kinesis. Una partizione supporta 1 MB/s e 1.000 record al secondo per le scritture e 2 MB/s per le letture sia in modalità di capacità on demand che in modalità di capacità assegnata. I limiti delle partizioni garantiscono prestazioni prevedibili, semplificando la progettazione e la gestione di un flusso di lavoro di streaming di dati altamente affidabile.

In questa sezione, imparerai come impostare la modalità di capacità per lo stream e come creare uno stream utilizzando o. AWS Management Console APIs Potrai quindi intraprendere azioni aggiuntive sullo stream.

Argomenti

- Scegli la modalità di capacità del flusso di dati
- Crea uno stream utilizzando il AWS Management Console
- Crea uno stream utilizzando il APIs
- Aggiorna uno stream
- Visualizzazione dell'elenco di flussi
- Elenca i frammenti
- Eliminare uno stream
- Condividi nuovamente uno stream
- Modifica il periodo di conservazione dei dati
- Etichetta le tue risorse Amazon Kinesis Data Streams

Scegli la modalità di capacità del flusso di dati

I seguenti argomenti spiegano come scegliere la modalità di capacità appropriata per l'applicazione e come passare da una modalità di capacità all'altra, se necessario.

Argomenti

- Cos'è una modalità di capacità del flusso di dati?
- Caratteristiche e casi d'uso della modalità on-demand
- Caratteristiche e casi d'uso della modalità provisioned
- Passa da una modalità di capacità all'altra

Cos'è una modalità di capacità del flusso di dati?

Una modalità di capacità del flusso di dati determina come viene gestita la capacità e come viene addebitato l'utilizzo del flusso di dati. In Flussoi di dati Amazon Kinesis è possibile scegliere tra la modalità di capacità on demand e la modalità assegnata per i flussi di dati.

- On demand: i flussi di dati con modalità on demand non richiedono alcuna pianificazione della capacità e si dimensionano automaticamente per gestire i gigabyte di velocità di trasmissione effettiva di scrittura e lettura al minuto. Con la modalità on demand, il flusso di dati Kinesis gestisce automaticamente le partizioni per fornire la velocità di trasmissione effettiva necessaria.
- Assegnata: per i flussi di dati con la modalità assegnata, è necessario specificare il numero di partizioni per il flusso di dati. La capacità totale di un flusso di dati è la somma di capacità delle relative partizioni. Puoi aumentare o diminuire il numero di partizioni in un flusso di dati in base alle esigenze.

Puoi utilizzare Kinesis Data `PutRecords` APIs Streams e scrivere `PutRecord` dati nei tuoi flussi di dati sia in modalità on-demand che in modalità di capacità fornita. Per recuperare i dati, entrambe le modalità di capacità supportano i consumer predefiniti che utilizzano l'API `GetRecords` e i consumer Enhanced Fan-Out (EFO) che utilizzano l'API `SubscribeToShard`.

Tutte le funzionalità del flusso di dati Kinesis, tra cui la modalità di conservazione, la crittografia, i parametri di monitoraggio e altre, sono supportate sia per la modalità on demand che per quella assegnata. Il flusso di dati Kinesis offre durabilità e disponibilità elevate sia in modalità di capacità on demand che in modalità di capacità assegnata.

Caratteristiche e casi d'uso della modalità on-demand

I flussi di dati in modalità on demand non richiedono alcuna pianificazione della capacità e si dimensionano automaticamente per gestire i gigabyte di velocità di trasmissione effettiva di scrittura e lettura al minuto. La modalità on demand semplifica l'acquisizione e l'archiviazione di grandi volumi di dati a bassa latenza perché elimina il provisioning e la gestione di server, spazio di archiviazione

o velocità di trasmissione effettiva. È possibile importare miliardi di record al giorno senza alcun sovraccarico operativo.

La modalità on demand è ideale per soddisfare le esigenze di traffico applicativo altamente variabile e imprevedibile. Non è più necessario effettuare il provisioning di questi carichi di lavoro per la massima capacità, il che può comportare costi più elevati a causa del basso utilizzo. La modalità on demand è adatta per carichi di lavoro con modelli di traffico imprevedibili e altamente variabili.

Con la modalità di capacità on demand, si paga per i GB di dati scritti e letti dai flussi di dati. Non è necessario specificare la velocità di trasmissione effettiva di lettura e scrittura che si prevede l'applicazione esegua. Il flusso di dati Kinesis si adatta istantaneamente ai carichi di lavoro siano essi aumentati o diminuiti. Per ulteriori informazioni, consulta Prezzi di Amazon Kinesis Data Streams.

Un flusso di dati in modalità on demand supporta fino al doppio della velocità di trasmissione effettiva di scrittura di picco osservato nei 30 giorni precedenti. Man mano che la velocità trasmissione effettiva di scrittura del flusso di dati raggiunge un nuovo picco, il flusso di dati Kinesis dimensiona automaticamente la capacità del flusso di dati. Ad esempio, se il flusso di dati ha un throughput di scrittura che varia tra il 10% MB/s and 40 MB/s, then Kinesis Data Streams ensures that you can easily burst to double your previous peak throughput, or 80 MB/s. If the same data stream sustains a new peak throughput of 50 MB/s, Kinesis Data Streams ensures that there is enough capacity to ingest 100 MB/s del throughput di scrittura. Tuttavia, si può verificare una limitazione se il traffico aumenta fino a più del doppio del picco precedente nell'arco di 15 minuti. Dovrai riprovare a eseguire queste richieste limitate.

La capacità di lettura aggregata di un flusso di dati con la modalità on demand aumenta proporzionalmente alla velocità di scrittura. Ciò contribuisce a garantire che le applicazioni consumer abbiano sempre una velocità di lettura adeguata per elaborare i dati in arrivo in tempo reale. Ottieni almeno il doppio della velocità di trasmissione effettiva di scrittura rispetto ai dati di lettura utilizzando l'API `GetRecords`. Ti consigliamo di utilizzare un'applicazione consumer con l'API `GetRecord` in modo che abbia abbastanza spazio per recuperare il ritardo dell'applicazione quando deve essere ripristinata dopo un periodo di inattività. Si consiglia di utilizzare la funzionalità di fan-out avanzato del flusso di dati Kinesis per scenari che richiedono l'aggiunta di più di un'applicazione consumer. Il fan-out avanzato supporta l'aggiunta di un massimo di 20 applicazioni consumer a un flusso di dati tramite l'API `SubscribeToShard`, con ogni applicazione consumer con una velocità di trasmissione effettiva dedicata.

Gestisci le eccezioni della velocità effettiva di lettura e scrittura

Con la modalità di capacità on demand (uguale alla modalità di capacità assegnata), è necessario specificare una chiave di partizione per ogni record per scrivere i dati nel flusso di dati. Il flusso di dati Kinesis utilizza le chiavi di partizione per distribuire i dati tra le partizioni. Il flusso di dati Kinesis monitora il traffico per ogni partizione. Quando il traffico in entrata supera i 500 KB/s per partizione, divide la partizione entro 15 minuti. I valori della chiave hash della partizione principale vengono ridistribuiti in modo uniforme tra le partizioni secondarie.

Se il traffico in entrata supera il doppio del picco precedente, è possibile che si verifichino eccezioni di lettura o scrittura per circa 15 minuti, anche quando i dati sono distribuiti uniformemente tra le partizioni. Ti consigliamo di riprovare tutte queste richieste in modo che tutti i record vengano archiviati correttamente nel flusso di dati Kinesis.

Potrebbero verificarsi eccezioni di lettura e scrittura se si utilizza una chiave di partizione che causa una distribuzione non uniforme dei dati e se i record assegnati a una determinata partizione superano i limiti. Con la modalità on demand, il flusso di dati si adatta automaticamente alla gestione di modelli di distribuzione dei dati non uniformi, a meno che una singola chiave di partizione non superi la velocità di trasmissione effettiva di 1 MB/s e i limiti di 1.000 record al secondo.

Nella modalità on demand, il flusso di dati Kinesis divide le partizioni in modo uniforme quando rileva un aumento del traffico. Tuttavia, non rileva né isola le chiavi hash che indirizzano una parte maggiore del traffico in entrata verso una partizione particolare. Se si utilizzano chiavi di partizione molto irregolari, è possibile che continuino a ricevere eccezioni di scrittura. Per questi casi d'uso, si consiglia di utilizzare la modalità di capacità assegnata che supporta le suddivisioni di partizioni granulari.

Caratteristiche e casi d'uso della modalità provisioned

Con la modalità provisioning, dopo aver creato il flusso di dati, puoi aumentare o ridurre dinamicamente la capacità dello shard utilizzando o l' AWS Management Console API. UpdateShardCount È possibile effettuare aggiornamenti mentre è presente un'applicazione producer o consumer del flusso di dati Kinesis che scrive o legge dati dal flusso.

La modalità assegnata è adatta per un traffico prevedibile con requisiti di capacità facili da stimare. È possibile utilizzare la modalità assegnata se si desidera un controllo preciso su come i dati sono distribuiti tra le partizioni.

Con la modalità assegnata, devi specificare il numero di partizioni per il flusso di dati. Per determinare le dimensioni iniziali di un flusso di dati, è necessario soddisfare i seguenti valori di input:

- Le dimensioni medie del record di dati scritti nel flusso in kilobyte (KB), arrotondate al KB più vicino (`average_data_size_in_KB`).
- Il numero di record di dati scritti e letti dal flusso al secondo (`records_per_second`).
- Il numero di applicazioni consumer, che sono le applicazioni del flusso di dati Kinesis che elaborano i dati simultaneamente e indipendentemente dal flusso (`number_of_consumers`).
- La larghezza di banda in scrittura in entrata in KB (`incoming_write_bandwidth_in_KB`), che è uguale a `average_data_size_in_KB` moltiplicato per `records_per_second`.
- La larghezza di banda in lettura in uscita in KB (`outgoing_read_bandwidth_in_KB`), che è uguale a `incoming_write_bandwidth_in_KB` moltiplicato per `number_of_consumers`.

Puoi calcolare il numero di partizioni (`number_of_shards`) di cui il flusso ha bisogno utilizzando i valori di input nella formula seguente:

```
number_of_shards = max(incoming_write_bandwidth_in_KiB/1024,
 outgoing_read_bandwidth_in_KiB/2048)
```

Nella modalità assegnata è possibile che si verifichino comunque eccezioni di velocità di trasmissione effettiva di lettura e scrittura se non si configura il flusso di dati per gestire i picchi di velocità di trasmissione effettiva. In questo caso, è necessario dimensionare manualmente il flusso di dati in modo da adattarlo al traffico di dati.

Potrebbero verificarsi eccezioni di lettura e scrittura se si utilizza una chiave di partizione che causa una distribuzione non uniforme dei dati e se i record assegnati a una determinata partizione superano i limiti. Per risolvere questo problema nella modalità assegnata, identifica tali frammenti e dividili manualmente per adattarli meglio al traffico. Per ulteriori informazioni, consulta Ripartizionamento di un flusso.

Passa da una modalità di capacità all'altra

È possibile cambiare la modalità di capacità del flusso di dati da on demand ad assegnata e viceversa. Per ogni flusso di dati del tuo account AWS , puoi passare dalla modalità di capacità on-demand a quella di capacità assegnata e viceversa due volte nell'arco di 24 ore.

Il passaggio da una modalità di capacità all'altra di un flusso di dati non causa interruzioni alle applicazioni che utilizzano il flusso di dati. È possibile continuare a scrivere e leggere da questo flusso di dati. Quando si passa da una modalità di capacità all'altra, da on demand ad assegnata o

viceversa, lo stato del flusso viene impostato su Aggiornamento in corso. È necessario attendere che lo stato del flusso di dati diventi Attivo prima di poterne modificare nuovamente le proprietà.

Quando si passa dalla modalità di capacità assegnata a quella on demand, il flusso di dati inizialmente conserva il numero di partizioni che aveva prima della transizione e, da questo momento in poi, il flusso di dati Kinesis monitorerà il traffico di dati e dimensionerà il numero di partizioni di questo flusso di dati on demand in base alla velocità di trasmissione effettiva di scrittura.

Quando passi dalla modalità on demand alla modalità assegnata, il flusso di dati mantiene inizialmente anche il numero di partizioni che aveva prima della transizione, ma da questo momento in poi, sarai responsabile del monitoraggio e della regolazione del numero di partizioni di questo flusso di dati per adattarlo adeguatamente alla tua velocità di trasmissione effettiva di scrittura.

Crea uno stream utilizzando il AWS Management Console

È possibile creare un flusso utilizzando la console del flusso di dati Kinesis, l'API del flusso di dati Kinesis o la AWS Command Line Interface (AWS CLI).

Creazione di un flusso di dati tramite la console

1. Accedi AWS Management Console e apri la console Kinesis all'indirizzo https://console.aws.amazon.com /kinesis.
2. Nella barra di navigazione, espandere il selettore delle regioni e selezionare una regione.
3. Selezionare Create data stream (Crea flusso di dati).
4. Nella pagina Crea flusso Kinesis, inserisci un nome per il tuo flusso di dati, quindi scegli la modalità di capacità On demand o Assegnata. La modalità On demand è selezionata per impostazione predefinita. Per ulteriori informazioni, consulta Scegli la modalità di capacità del flusso di dati.

   Con la modalità On demand, puoi quindi scegliere Crea flusso Kinesis per creare il tuo flusso di dati. Con la modalità Assegnata, devi specificare il numero di partizioni necessarie e quindi scegliere Crea flusso Kinesis.

   Nella pagina Kinesis streams (Flussi Kinesis), lo Status (Stato) del flusso è Creating (Creazione in corso) durante la creazione del flusso. Quando il flusso è pronto per essere usato, lo Status (Stato) passa su Active (Attivo).
5. Seleziona il nome del flusso. La pagina Stream Details (Dettagli del flusso) visualizza un riepilogo della configurazione del flusso, oltre alle informazioni sul monitoraggio.

Creazione di un flusso tramite l'API del flusso di dati Kinesis

- Per informazioni sulla creazione di un flusso tramite l'API del flusso di dati Kinesis, consulta Crea uno stream utilizzando il APIs.

Per creare uno stream utilizzando il AWS CLI

- Per informazioni sulla creazione di uno stream utilizzando AWS CLI, vedete il comando create-stream.

Crea uno stream utilizzando il APIs

Utilizzare la procedura seguente per creare il tuo flusso di dati Kinesis.

Crea il client Kinesis Data Streams

Prima di utilizzare i flusso di dati Kinesis, è necessario creare un oggetto client. Il seguente codice Java avvia un'istanza da un generatore client e la utilizza per impostare la regione, le credenziali e la configurazione del client. Quindi crea un oggetto client.

```
AmazonKinesisClientBuilder clientBuilder = AmazonKinesisClientBuilder.standard();

clientBuilder.setRegion(regionName);
clientBuilder.setCredentials(credentialsProvider);
clientBuilder.setClientConfiguration(config);

AmazonKinesis client = clientBuilder.build();
```

Per ulteriori informazioni, consulta Regioni ed endpoint del flusso di dati Kinesis nella Riferimenti generali di AWS.

Crea lo stream

Ora che hai creato il tuo client Kinesis Data Streams, puoi creare uno stream utilizzando la console o a livello di codice. Per creare uno stream a livello di codice, crea un'istanza di un `CreateStreamRequest` oggetto e specifica un nome per lo stream. Se desideri utilizzare la modalità provisioned, specifica il numero di shard da utilizzare per lo stream.

- On demand:

```
CreateStreamRequest createStreamRequest = new CreateStreamRequest();
createStreamRequest.setStreamName( myStreamName );
```

- Assegnata:

```
CreateStreamRequest createStreamRequest = new CreateStreamRequest();
createStreamRequest.setStreamName( myStreamName );
createStreamRequest.setShardCount( myStreamSize );
```

Il nome del flusso identifica il flusso. Il nome si riferisce all' AWS account utilizzato dall'applicazione. È inoltre definito in base alla regione. Cioè, due stream in due AWS account diversi possono avere lo stesso nome e due stream nello stesso AWS account ma in due regioni diverse possono avere lo stesso nome, ma non due stream sullo stesso account e nella stessa regione.

La velocità effettiva dello stream è una funzione del numero di shard. Per una maggiore velocità effettiva assegnata, sono necessari più shard. Un numero maggiore di shard aumenta anche il costo AWS addebitato per lo streaming. Per ulteriori informazioni su come calcolare il numero appropriato di shard per la tua applicazione, vedere Scegli la modalità di capacità del flusso di dati.

Dopo aver configurato l'`createStreamRequest`oggetto, create uno stream chiamando il `createStream` metodo sul client. Dopo la chiamata a `createStream`, attendere che il flusso raggiunga lo stato `ACTIVE` prima di eseguire qualunque operazione sul flusso. Per controllare lo stato del flusso, chiamare il metodo `describeStream`. Tuttavia, `describeStream` genera un'eccezione se il flusso non esiste. Pertanto, racchiudere la chiamata `describeStream` in un blocco `try/catch`.

```
client.createStream( createStreamRequest );
DescribeStreamRequest describeStreamRequest = new DescribeStreamRequest();
describeStreamRequest.setStreamName( myStreamName );

long startTime = System.currentTimeMillis();
long endTime = startTime + ( 10 * 60 * 1000 );
while ( System.currentTimeMillis() < endTime ) {
  try {
    Thread.sleep(20 * 1000);
  }
  catch ( Exception e ) {}

  try {
```

```
    DescribeStreamResult describeStreamResponse =
 client.describeStream( describeStreamRequest );
    String streamStatus =
 describeStreamResponse.getStreamDescription().getStreamStatus();
    if ( streamStatus.equals( "ACTIVE" ) ) {
      break;
    }
    //
    // sleep for one second
    //
    try {
      Thread.sleep( 1000 );
    }
    catch ( Exception e ) {}
  }
  catch ( ResourceNotFoundException e ) {}
}
if ( System.currentTimeMillis() >= endTime ) {
  throw new RuntimeException( "Stream " + myStreamName + " never went active" );
}
```

Aggiorna uno stream

È possibile aggiornare i dettagli di un flusso utilizzando la console del flusso di dati Kinesis, l'API del flusso di dati Kinesis o la AWS CLI.

> **Note**
>
> Puoi abilitare la crittografia lato server per flussi esistenti o per flussi che hai creato di recente.

Utilizzo della console

Aggiornamento di un flusso di dati tramite la console

1. Apri la console Amazon Kinesis all'indirizzo. https://console.aws.amazon.com/kinesis/
2. Nella barra di navigazione, espandere il selettore delle regioni e selezionare una regione.
3. Selezionare il nome del flusso nell'elenco. La pagina Stream Details (Informazioni flusso) visualizza un riepilogo della configurazione del flusso e informazioni sul monitoraggio.

4. Per passare dalla modalità di capacità on demand a quella assegnata per un flusso di dati, scegli Modifica modalità di capacità nella scheda Configurazione. Per ulteriori informazioni, consulta Scegli la modalità di capacità del flusso di dati.

   > **Important**
   >
   > Per ogni flusso di dati nel tuo AWS account, puoi passare dalla modalità on-demand a quella fornita due volte nell'arco di 24 ore.

5. Per un flusso di dati con la modalità assegnata, per modificare il numero di partizioni, scegli Modifica partizioni assegnate nella scheda Configurazione, quindi inserisci un nuovo numero di partizioni.
6. Per abilitare la crittografia lato server di record di dati, selezionare Edit (Modifica) nella sezione Server-side encryption (Crittografia lato server). Seleziona una chiave KMS da utilizzare come chiave master per la crittografia oppure utilizza la chiave master predefinita aws/kinesis gestita da Kinesis. Se abiliti la crittografia per uno stream e utilizzi la tua chiave AWS KMS master, assicurati che le applicazioni di produzione e consumo abbiano accesso alla chiave AWS KMS master che hai utilizzato. Per assegnare le autorizzazioni a un'applicazione e accedere a una chiave AWS KMS generata dall'utente, consultare the section called “Autorizzazioni per utilizzare le chiavi KMS generate dall'utente”.
7. Per modificare il periodo di ritenzione dei dati, selezionare Edit (Modifica) nella sezione Data retention period (Periodo di ritenzione dei dati) e immettere un nuovo periodo di ritenzione dei dati.
8. Se sono stati attivati i parametri personalizzati nell'account, selezionare Edit (Modifica) nella sezione Shard level metrics (Parametri a livello di shard) e specificare i parametri per il flusso. Per ulteriori informazioni, consulta the section called “Monitora il servizio Kinesis Data Streams con CloudWatch”.

Utilizzare l'API

Per aggiornare le informazioni sul flusso utilizzando l'API, consulta i seguenti metodi:

- AddTagsToStream
- DecreaseStreamRetentionPeriod
- DisableEnhancedMonitoring
- EnableEnhancedMonitoring

- IncreaseStreamRetentionPeriod
- RemoveTagsFromStream
- StartStreamEncryption
- StopStreamEncryption
- UpdateShardCount

Usa il AWS CLI

Per informazioni sull'aggiornamento di uno stream utilizzando il AWS CLI, consulta il riferimento alla CLI di Kinesis.

Visualizzazione dell'elenco di flussi

Gli stream sono limitati all' AWS account associato alle AWS credenziali utilizzate per creare un'istanza del client Kinesis Data Streams e anche alla regione specificata per il client. Un account AWS può avere molti flussi attivi simultaneamente. È possibile creare un elenco dei flussi nella console del flusso di dati Kinesis, oppure in modo programmatico. Il codice in questa sezione mostra come elencare tutti gli stream per il tuo account. AWS

```
ListStreamsRequest listStreamsRequest = new ListStreamsRequest();
listStreamsRequest.setLimit(20);
ListStreamsResult listStreamsResult = client.listStreams(listStreamsRequest);
List<String> streamNames = listStreamsResult.getStreamNames();
```

Questo esempio di codice crea prima una nuova istanza di `ListStreamsRequest` e chiama il suo metodo `setLimit` per specificare che un massimo di 20 flussi deve essere restituito per ogni chiamata a `listStreams`. Se non si specifica un valore per `setLimit`, il flusso di dati Kinesis restituisce un numero di flussi minore o uguale al numero contenuto nell'account. Il codice quindi passa `listStreamsRequest` al metodo `listStreams` del client. Il valore restituito `listStreams` viene memorizzato in un oggetto `ListStreamsResult`. Il codice chiama il metodo `getStreamNames` su questo oggetto e memorizza i nomi di flusso restituiti nell'elenco `streamNames`. Si noti che il flusso di dati Kinesis può restituire un numero di flussi minore di quanto indicato dal limite specificato anche se non ci sono più flussi oltre quelli nell'account e nella regione. Per verificare di avere recuperato tutti i flussi, utilizzare il metodo `getHasMoreStreams` come descritto nel prossimo esempio di codice.

```
while (listStreamsResult.getHasMoreStreams())
```

```
{
    if (streamNames.size() > 0) {
      listStreamsRequest.setExclusiveStartStreamName(streamNames.get(streamNames.size()
 - 1));
    }
    listStreamsResult = client.listStreams(listStreamsRequest);
    streamNames.addAll(listStreamsResult.getStreamNames());
}
```

Questo codice chiama il metodo `getHasMoreStreams` su `listStreamsRequest` per controllare se sono disponibili flussi aggiuntivi oltre a quelli restituiti nella chiamata iniziale a `listStreams`. In tal caso, il codice chiama il metodo `setExclusiveStartStreamName` con il nome dell'ultimo flusso restituito nella precedente chiamata a `listStreams`. Il metodo `setExclusiveStartStreamName` fa si che la chiamata successiva a `listStreams` inizi dopo il flusso. Il gruppo di nomi di flusso restituiti dalla chiamata viene quindi aggiunto all'elenco `streamNames`. Questo processo continua finché tutti i nomi di flusso sono stati raccolti nell'elenco.

I flussi restituiti da `listStreams` possono essere in uno dei seguenti stati:

- `CREATING`
- `ACTIVE`
- `UPDATING`
- `DELETING`

È possibile controllare lo stato di un flusso utilizzando il metodo `describeStream`, come illustrato nella sezione precedente Crea uno stream utilizzando il APIs.

Elenca i frammenti

Un flusso di dati può avere una o più partizioni. Il metodo consigliato per elencare o recuperare gli shard da un flusso di dati consiste nell'utilizzare l'API. ListShards L'esempio seguente mostra in che modo è possibile ottenere un elenco di partizioni di un flusso di dati. Per una descrizione completa dell'operazione principale utilizzata in questo esempio e di tutti i parametri che è possibile impostare per l'operazione, vedere. ListShards

```
import software.amazon.awssdk.services.kinesis.KinesisAsyncClient;
import software.amazon.awssdk.services.kinesis.model.ListShardsRequest;
```

```
import software.amazon.awssdk.services.kinesis.model.ListShardsResponse;

import java.util.concurrent.TimeUnit;

public class ShardSample {

    public static void main(String[] args) {

        KinesisAsyncClient client = KinesisAsyncClient.builder().build();

        ListShardsRequest request = ListShardsRequest
                .builder().streamName("myFirstStream")
                .build();

        try {
            ListShardsResponse response = client.listShards(request).get(5000,
 TimeUnit.MILLISECONDS);
            System.out.println(response.toString());
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Per eseguire il codice di esempio precedente è possibile utilizzare un file POM come il seguente.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/
xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>kinesis.data.streams.samples</groupId>
    <artifactId>shards</artifactId>
    <version>1.0-SNAPSHOT</version>
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <configuration>
                    <source>8</source>
```

```
                    <target>8</target>
                </configuration>
            </plugin>
        </plugins>
    </build>
    <dependencies>
        <dependency>
            <groupId>software.amazon.awssdk</groupId>
            <artifactId>kinesis</artifactId>
            <version>2.0.0</version>
        </dependency>
    </dependencies>
</project>
```

Con l'`ListShards`API, puoi utilizzare il ShardFilterparametro per filtrare la risposta dell'API. È possibile specificare un solo filtro alla volta.

Se si utilizza il `ShardFilter` parametro quando si richiama l' ListShardsAPI, `Type` è la proprietà richiesta e deve essere specificata. Se si specificano i tipi `AT_TRIM_HORIZON`, `FROM_TRIM_HORIZON` o `AT_LATEST`, non è necessario specificare né la proprietà `ShardId` né la proprietà `Timestamp` facoltative.

Se si specifica il tipo `AFTER_SHARD_ID`, sarà necessario fornire anche il valore per la proprietà `ShardId` facoltativa. La `ShardId` proprietà ha funzionalità identiche al `ExclusiveStartShardId` parametro dell' ListShards API. Quando viene specificata la proprietà `ShardId`, la risposta includerà le partizioni che iniziano con la partizione il cui ID segue immediatamente la `ShardId` assegnata.

Se si specifica il tipo `AT_TIMESTAMP` o `FROM_TIMESTAMP_ID`, sarà necessario fornire anche il valore per la proprietà `Timestamp` facoltativa. Se si specifica il tipo `AT_TIMESTAMP`, verranno restituite tutte le partizioni aperte nel timestamp fornito. Se si specifica il tipo `FROM_TIMESTAMP`, verranno restituite tutte le partizioni dal timestamp fornito al TIP.

> **Important**
>
> `DescribeStreamSummarye` `ListShard` APIs forniscono un modo più scalabile per recuperare informazioni sui flussi di dati. Più specificamente, le quote per l' DescribeStream API possono causare limitazioni. Per ulteriori informazioni, consulta Quote e limiti. Tieni inoltre presente che le `DescribeStream` quote sono condivise tra tutte le applicazioni che interagiscono con tutti i flussi di dati del tuo account. AWS Le quote per l' ListShards API, invece, sono specifiche per un singolo flusso di dati. Quindi non solo ottieni un TPS più

elevato con l' ListShards API, ma l'azione si adatta meglio man mano che crei più flussi di dati.
Ti consigliamo di migrare tutti i produttori e i consumatori che chiamano l' DescribeStream API per invocare invece the and the. DescribeStreamSummary ListShard APIs Per identificare questi produttori e consumatori, consigliamo di utilizzare Athena per analizzare i CloudTrail log man mano che gli user agent per KPL e KCL vengono acquisiti nelle chiamate API.

```
SELECT useridentity.sessioncontext.sessionissuer.username,
useridentity.arn,eventname,useragent, count(*) FROM
cloudtrail_logs WHERE Eventname IN ('DescribeStream')  AND
eventtime
    BETWEEN ''
        AND ''
GROUP BY
 useridentity.sessioncontext.sessionissuer.username,useridentity.arn,eventname,useragent
ORDER BY  count(*) DESC LIMIT 100
```

Consigliamo inoltre di riconfigurare le integrazioni AWS Lambda e Amazon Firehose con Kinesis Data Streams che richiamano l'API in modo che le integrazioni richiamino `DescribeStream` invece e. `DescribeStreamSummary` `ListShards` In particolare, per AWS Lambda, è necessario aggiornare la mappatura delle sorgenti degli eventi. Per Amazon Firehose, le autorizzazioni IAM corrispondenti devono essere aggiornate in modo da includere l'autorizzazione `ListShards` IAM.

Eliminare uno stream

È possibile eliminare un flusso con la console del flusso di dati Kinesis oppure in modo programmatico. Per eliminare un flusso in modo programmatico, utilizzare `DeleteStreamRequest`, come illustrato nel codice seguente.

```
DeleteStreamRequest deleteStreamRequest = new DeleteStreamRequest();
deleteStreamRequest.setStreamName(myStreamName);
client.deleteStream(deleteStreamRequest);
```

Arrestare tutte le applicazioni che operano nel flusso prima di eliminarlo. Se un'applicazione tenta di intervenire su un flusso eliminato, riceve le eccezioni `ResourceNotFound`. Inoltre, se successivamente si crea un nuovo flusso con lo stesso nome del flusso precedente, e le applicazioni che operavano sul flusso precedente sono ancora in esecuzione, queste applicazioni potrebbero tentare di interagire con il nuovo flusso come se si trovasse nel flusso precedente, con risultati imprevedibili.

Condividi nuovamente uno stream

> **Important**
>
> Puoi condividere nuovamente il tuo stream utilizzando l'API. UpdateShardCount Altrimenti, è possibile continuare a eseguire suddivisioni e unioni come illustrato qui.

Flusso di dati Amazon Kinesis supporta il ripartizionamento, che consente di regolare il numero di partizioni nel flusso in modo da adattarsi alle modifiche alla velocità del flusso di dati tramite lo streaming. Il resharding è considerata un'operazione avanzata. Se non hai mai utilizzato il flusso di dati Kinesis, torna a questo argomento dopo aver acquisito familiarità con tutti gli altri aspetti del flusso di dati Kinesis.

Esistono due tipi di operazioni di resharding: suddivisione di shard e unione di shard. In una suddivisione di shard, si divide un singolo shard in due shard. In una unione di shard, si uniscono due hard in un singolo shard. Il resharding è sempre a livello di coppia nel senso che non è possibile suddividere più di due shard in un'unica operazione e non è possibile unire più di due shard in un'unica operazione. Lo shard o la coppia di shard su cui agisce l'operazione di resharding viene definito shard padre. Lo shard o la coppia di shard, risultato dell'operazione di resharding, viene definito shard figlio.

Il frazionamento aumenta il numero di shard nel tuo flusso e pertanto aumenta la capacità di dati del flusso. Poiché si riceverà un addebito per ogni shard, la suddivisione aumenta il costo del flusso. Analogamente, l'unione riduce il numero di shard nel tuo flusso e pertanto diminuisce la capacità, quindi il costo, dei dati del flusso.

Il resharding viene in genere eseguito da un'applicazione di amministrazione che è distinta dall'applicazione producer (put) e dall'applicazione consumer (get). Tale applicazione amministrativa monitora le prestazioni complessive dello stream in base ai parametri forniti da Amazon CloudWatch o in base ai parametri raccolti dai produttori e dai consumatori. L'applicazione amministrativa

necessita inoltre di un set più ampio di autorizzazioni IAM rispetto ai consumatori o ai produttori, poiché i consumatori e i produttori di solito non dovrebbero aver bisogno di accedere a ciò che viene utilizzato per il resharding. APIs Per ulteriori informazioni sulle autorizzazioni IAM per il flusso di dati Kinesis, consulta Controllo dell'accesso alle risorse del flusso di dati Amazon Kinesis tramite IAM.

Per ulteriori informazioni sul ripartizionamento, consulta Come posso modificare il numero di partizioni aperte nel flusso di dati Kinesis?

Argomenti

- Decidi una strategia per il resharding
- Dividere un frammento
- Unisci due frammenti
- Completa l'azione di ricondivisione

Decidi una strategia per il resharding

Lo scopo del ripartizionamento in Flusso di dati Amazon Kinesis è quello di consentire al flusso di adattarsi ai cambiamenti alla velocità del flusso di dati. Si esegue la suddivisione in shard per aumentare la capacità (e i costi) del flusso. Si esegue l'unione dei shard per ridurre i costi (e la capacità) del flusso.

Un approccio al ripartizionamento potrebbe essere la suddivisione di ogni partizione nel flusso; tale operazione raddoppierebbe la capacità del flusso. Tuttavia, questo potrebbe fornire più capacità aggiuntiva a quella effettivamente necessaria e, di conseguenza, causare spese superflue.

È inoltre possibile utilizzare i parametri per determinare quali sono le partizioni calde o fredde, ovvero le partizioni che ricevono più dati o meno dati del previsto. È possibile scegliere di suddividere gli shard hot per aumentare la capacità per le chiavi hash indirizzate a questi shard. Analogamente, è possibile unire shard cold per sfruttare al meglio la loro capacità inutilizzata.

Puoi ottenere alcuni dati sulle prestazioni del tuo stream dai CloudWatch parametri Amazon pubblicati da Kinesis Data Streams. Tuttavia, è anche possibile raccogliere alcuni parametri per i tuoi flussi. Un approccio consiste nel registrare i valori chiave hash generati dalle chiavi di partizione per i tuoi record di dati. Tenere presente che una chiave di partizione è stata specificata nel momento in cui il record è stato aggiunto al flusso.

```
putRecordRequest.setPartitionKey( String.format( "myPartitionKey" ) );
```

Kinesis Data MD5Streams utilizza per calcolare la chiave hash dalla chiave di partizione. Poiché si specifica la chiave di partizione per il record, è possibile utilizzarla per calcolare il valore della chiave hash MD5 per quel record e registrarlo.

Puoi anche registrare gli IDs shard a cui sono assegnati i tuoi record di dati. L'ID shard è disponibile utilizzando il metodo `getShardId` dell'oggetto `putRecordResults` restituito dal metodo `putRecords` e l'oggetto `putRecordResult` restituito dal metodo `putRecord`.

```
String shardId = putRecordResult.getShardId();
```

Con i valori dello shard IDs e delle chiavi hash, puoi determinare quali shard e hash key ricevono più o meno traffico. Quindi è possibile utilizzare il resharding per fornire più o meno capacità, a seconda della chiave.

Dividere un frammento

Per suddividere una partizione in Flusso di dati Amazon Kinesis, è necessario specificare il modo in cui i valori della chiave hash devono essere ridistribuiti dalla partizione principale alle partizioni secondarie. Quando si aggiunge un record di dati al flusso, viene assegnato a uno shard in base al valore chiave hash. Il valore della chiave hash è l'MD5hash della chiave di partizione specificata per il record di dati nel momento in cui aggiungi il record di dati allo stream. I record di dati che hanno la stessa chiave di partizione, hanno anche lo stesso valore chiave hash.

I possibili valori chiave hash per un determinato shard costituiscono un set ordinato e contiguo di numeri interi non negativi. Questo intervallo di possibili valori chiave hash è determinato da quanto segue:

```
shard.getHashKeyRange().getStartingHashKey();
shard.getHashKeyRange().getEndingHashKey();
```

Quando si suddivide lo shard, è necessario specificare un valore in questo intervallo. Questo valore chiave hash e tutti i valori chiave hash più alti sono distribuiti a uno shard figlio. Tutti i valori chiave hash inferiori vengono distribuiti ad un altro shard figlio.

Il seguente codice illustra un'operazione di suddivisione shard che ridistribuisce le chiavi hash in modo uniforme tra ciascuno degli shard figlio, essenzialmente suddividendo lo shard padre a metà. Questo è solo uno dei modi possibili di ripartire lo shard padre. È possibile, ad esempio, suddividere lo shard in modo che il terzo inferiore delle chiavi dal padre vada ad uno shard figlio e i due terzi

superiori delle chiavi vadano all'altro shard figlio. Tuttavia, per molte applicazioni, la suddivisione degli shard a metà è un approccio efficace.

Il codice presuppone che `myStreamName` contiene il nome del flusso e la variabile oggetto `shard` contiene lo shard da suddividere. Per cominciare, creare un nuovo oggetto `splitShardRequest` e impostare il nome del flusso e l'ID shard.

```
SplitShardRequest splitShardRequest = new SplitShardRequest();
splitShardRequest.setStreamName(myStreamName);
splitShardRequest.setShardToSplit(shard.getShardId());
```

Determinare il valore di chiave hash che si trova a metà tra i valori minimo e massimo contenuti nello shard. Questo è il valore chiave hash di partenza per lo shard figlio che conterrà la metà superiore delle chiavi hash dallo shard padre. Specificare questo valore nel metodo `setNewStartingHashKey`. È necessario specificare solo questo valore. Il flusso di dati Kinesis distribuisce automaticamente le chiavi hash sotto questo valore all'altra partizione secondaria creata dalla suddivisione. L'ultima fase è la chiamata del metodo `splitShard` sul client del flusso di dati Kinesis.

```
BigInteger startingHashKey = new
 BigInteger(shard.getHashKeyRange().getStartingHashKey());
BigInteger endingHashKey   = new
 BigInteger(shard.getHashKeyRange().getEndingHashKey());
String newStartingHashKey  = startingHashKey.add(endingHashKey).divide(new
 BigInteger("2")).toString();

splitShardRequest.setNewStartingHashKey(newStartingHashKey);
client.splitShard(splitShardRequest);
```

La prima fase dopo questa procedura è illustrata in Attendi che uno stream torni ad essere attivo.

Unisci due frammenti

Un'operazione di unione shard prende due shard specificati e li combina in un singolo shard. Dopo l'unione, il singolo shard figlio riceve i dati per tutti i valori chiave hash coperti dai due shard padre.

Adiacenza di shard

Per unire due shard, gli shard devono essere adiacenti. Due shard sono considerati adiacenti se l'unione degli intervalli chiave hash per i due shard formano un set contiguo senza spazi. Ad esempio, supponiamo di avere due shard, uno con una gamma di chiavi hash di 276...381 e l'altro con una

gamma di chiavi hash di 382...454. È possibile unire questi due shard in un singolo shard che avrebbe così una gamma di chiavi hash 276... 454.

Per fare un altro esempio, supponiamo di avere due shard, uno con una gamma di chiavi hash di 276...381 e l'altro con una gamma di chiavi hash di 455...560. In questo caso non è possibile unire i due shard, poiché ci sarebbero uno o più shard tra questi due a copertura della gamma 382...454.

L'insieme di tutti gli `OPEN` shard in un flusso, come gruppo, copre sempre l'intera gamma di valori delle chiavi hash. MD5 Per ulteriori informazioni sugli stati degli shard, come ad esempio `CLOSED`, vedere Prendi in considerazione il routing dei dati, la persistenza dei dati e lo stato dello shard dopo una reshard.

Per identificare gli shard candidati per l'unione, è necessario filtrare tutti gli shard che sono in uno stato `CLOSED`. Le partizioni con stato `OPEN`, ossia non `CLOSED`, hanno un numero di sequenza finale pari a `null`. È possibile testare il numero di sequenza finale per un shard utilizzando:

```
if( null == shard.getSequenceNumberRange().getEndingSequenceNumber() )
{
  // Shard is OPEN, so it is a possible candidate to be merged.
}
```

Dopo aver filtrato gli shard chiusi, ordinare i restanti shard dal valore chiave hash più alto supportato per ogni shard. È possibile recuperare questo valore utilizzando:

```
shard.getHashKeyRange().getEndingHashKey();
```

Se due shard sono adiacenti in questo elenco filtrato e ordinato, allora possono essere uniti.

Codice dell'operazione di unione

Il seguente codice unisce due shard. Il codice presuppone che `myStreamName` contiene il nome del flusso e che le variabili oggetto `shard1` e `shard2` contengono due shard adiacenti da unire.

Per l'operazione di unione, iniziare creando istanze di un nuovo oggetto `mergeShardsRequest`. Specificare il nome del flusso con il metodo `setStreamName`. Quindi specificare i due shard da unire utilizzando i metodi `setShardToMerge` e `setAdjacentShardToMerge`. Infine, chiama il metodo `mergeShards` sul client del flusso di dati Kinesis per eseguire l'operazione.

```
MergeShardsRequest mergeShardsRequest = new MergeShardsRequest();
mergeShardsRequest.setStreamName(myStreamName);
mergeShardsRequest.setShardToMerge(shard1.getShardId());
```

```
mergeShardsRequest.setAdjacentShardToMerge(shard2.getShardId());
client.mergeShards(mergeShardsRequest);
```

La prima fase dopo questa procedura è illustrata in Attendi che uno stream torni ad essere attivo.

Completa l'azione di ricondivisione

Dopo qualsiasi tipo di procedura di ripartizionamento in Flusso di dati Amazon Kinesis e prima della ripresa dell'elaborazione record normale, sono necessarie altre procedure e considerazioni. Le sezioni seguenti descrivono queste procedure.

Argomenti

- Attendi che uno stream torni ad essere attivo
- Prendi in considerazione il routing dei dati, la persistenza dei dati e lo stato dello shard dopo una reshard

Attendi che uno stream torni ad essere attivo

Dopo aver chiamato un'operazione di resharding, `splitShard` o `mergeShards`, devi attendere che lo stream diventi nuovamente attivo. Il codice da usare è lo stesso di quando si attende che il flusso diventi attivo dopo la creazione di un flusso. Il codice è il seguente:

```
DescribeStreamRequest describeStreamRequest = new DescribeStreamRequest();
describeStreamRequest.setStreamName( myStreamName );

long startTime = System.currentTimeMillis();
long endTime = startTime + ( 10 * 60 * 1000 );
while ( System.currentTimeMillis() < endTime )
{
  try {
    Thread.sleep(20 * 1000);
  }
  catch ( Exception e ) {}

  try {
    DescribeStreamResult describeStreamResponse =
 client.describeStream( describeStreamRequest );
    String streamStatus =
 describeStreamResponse.getStreamDescription().getStreamStatus();
    if ( streamStatus.equals( "ACTIVE" ) ) {
```

```
      break;
    }
   //
    // sleep for one second
    //
    try {
      Thread.sleep( 1000 );
    }
    catch ( Exception e ) {}
  }
  catch ( ResourceNotFoundException e ) {}
}
if ( System.currentTimeMillis() >= endTime )
{
  throw new RuntimeException( "Stream " + myStreamName + " never went active" );
}
```

Prendi in considerazione il routing dei dati, la persistenza dei dati e lo stato dello shard dopo una reshard

Kinesis Data Streams è un servizio di streaming di dati in tempo reale. Le tue applicazioni devono presupporre che i dati fluiscano continuamente attraverso gli shard del tuo stream. Quando si esegue il resharding, i record di dati che scorrevano verso gli shard padre vengono instradati agli shard figlio in base ai valori chiave hash che sono mappati dalle chiavi di partizione del record di dati. Tuttavia, qualsiasi record di dati presente negli shard padre prima del resharding rimane in tali shard. Gli shard principali non scompaiono quando si verifica un nuovo shard. Rimangono con i dati contenuti prima del resharding. I record di dati nelle partizioni principali sono accessibili tramite le operazioni `getShardIterator` e `getRecords` nelle API del flusso di dati Kinesis oppure tramite la Kinesis Client Library.

> **Note**
>
> I record di dati sono accessibili dal momento in cui sono stati aggiunti al flusso fino al periodo di conservazione attuale. Questo vale indipendentemente da eventuali modifiche apportate agli shard nel flusso durante tale periodo di tempo. Per altre informazioni sul periodo di conservazione di un flusso, vedere Modifica il periodo di conservazione dei dati.

Nel processo di resharding, uno shard padre passa dallo stato `OPEN` allo stato `CLOSED` allo stato `EXPIRED`.

- OPEN: prima di un'operazione di resharding, uno shard padre è nello stato `OPEN`, il che significa che i record di dati possono essere sia aggiunti che recuperati dallo shard.
- CLOSED: dopo un'operazione di resharding, lo shard padre passa allo stato `CLOSED`. Ciò significa che i record di dati non sono più aggiunti allo shard. I record di dati che dovevano essere aggiunti a questo shard sono ora aggiunti allo shard figlio. Tuttavia, i record di dati possono essere ancora recuperati dallo shard per un periodo di tempo limitato.
- EXPIRED: dopo la scadenza del periodo di conservazione del flusso, tutti i record di dati nello shard padre sono scaduti e non sono più accessibili. In questo momento, lo stesso shard passa a uno stato `EXPIRED`. Le chiamate a `getStreamDescription().getShards` per enumerare gli shard nel flusso non includono gli shard `EXPIRED` nell'elenco degli shard restituiti. Per altre informazioni sul periodo di conservazione di un flusso, vedere Modifica il periodo di conservazione dei dati.

Dopo che è stato eseguito il resharding e il flusso è nuovamente nello stato `ACTIVE`, è possibile iniziare immediatamente a leggere i dati dagli shard figlio. Tuttavia, gli shard principali che rimangono dopo la reshard potrebbero ancora contenere dati che non hai ancora letto e che sono stati aggiunti allo stream prima del reshard. Se si leggono i dati dagli shard figlio prima di aver letto tutti i dati dagli shard padre, è possibile leggere i dati per una determinata chiave hash che non rientra nell'ordine fornito dai numeri di sequenza dei record di dati. Pertanto, presupponendo che l'ordine dei dati è importante, è necessario, dopo il resharding, continuare sempre a leggere i dati dagli shard padre finché non è scaduto. Solo dopo si può iniziare a leggere i dati dagli shard figlio. Quando `getRecordsResult.getNextShardIterator` restituisce `null`, indica che sono stati letti tutti i dati nello shard padre.

Modifica il periodo di conservazione dei dati

Flusso di dati Amazon Kinesis supporta le modifiche al periodo di conservazione dei record di dati del tuo flusso di dati. Un flusso di dati Kinesis è una sequenza ordinata di record di dati che deve essere utilizzata per la scrittura e la lettura in tempo reale. I record di dati vengono quindi memorizzati negli shard del flusso temporaneamente. Il periodo di tempo dall'aggiunta di un record all'inaccessibilità viene chiamato il periodo di conservazione. Un flusso di dati Kinesis archivia i record da 24 ore per impostazione predefinita, fino a 8.760 ore (365 giorni).

È possibile aggiornare il periodo di conservazione tramite la console Kinesis Data Streams o IncreaseStreamRetentionPeriodutilizzando e le DecreaseStreamRetentionPerioodoperazioni. Con la console del flusso di dati Kinesis, puoi modificare in blocco il periodo di conservazione di più di

un flusso di dati contemporaneamente. È possibile aumentare il periodo di conservazione fino a un massimo di 8760 ore (365 giorni) utilizzando l'IncreaseStreamRetentionPeriodoperazione o la console Kinesis Data Streams. È possibile ridurre il periodo di conservazione fino a un minimo di 24 ore utilizzando l'DecreaseStreamRetentionPeriodoperazione o la console Kinesis Data Streams. La sintassi di richiesta per entrambe le operazioni include il nome del flusso e il periodo di conservazione in ore. Infine, è possibile controllare l'attuale periodo di conservazione di un flusso chiamando l'operazione DescribeStream.

Di seguito è riportato un esempio di modifica del periodo di conservazione utilizzando la AWS CLI:

```
aws kinesis increase-stream-retention-period --stream-name retentionPeriodDemo --
retention-period-hours 72
```

Il flusso di dati Kinesis smette di rendere inaccessibili i record al vecchio periodo di conservazione entro alcuni minuti dall'aumentare del periodo di conservazione. Ad esempio, modificando il periodo di conservazione da 24 ore a 48 ore significa che i record aggiunti al flusso 23 ore 55 minuti prima sono ancora disponibili dopo 24 ore.

Il flusso di dati Kinesis quasi immediatamente rende i record più vecchi del nuovo periodo di conservazione inaccessibili diminuendo il periodo di conservazione. Pertanto, fare attenzione nel richiamare l'operazione DecreaseStreamRetentionPeriod.

Impostare il periodo di conservazione dei dati per assicurarsi che i consumatori siano in grado di leggere i dati prima della scadenza, se sorgono problemi. È consigliabile valutare attentamente tutte le possibilità, ad esempio un problema con la logica di elaborazione dei record o una dipendenza downstream inattiva per un lungo periodo di tempo. Il periodo di conservazione deve essere considerato come una rete di sicurezza che consente più tempo ai consumatori di dati per il recupero. Le operazioni API del periodo di conservazione consentono di eseguire questa operazione in modo proattivo o di rispondere a eventi operativi in modo reattivo.

Sono applicabili costi aggiuntivi per flussi con un periodo di conservazione impostato sopra le 24 ore. Per ulteriori informazioni, consulta Prezzi dei flussi di dati per Amazon Kinesis.

Etichetta le tue risorse Amazon Kinesis Data Streams

Puoi assegnare i tuoi metadati agli stream e ai fan-out consumer avanzati che crei in Amazon Kinesis Data Streams sotto forma di tag. Un tag è un valore-chiave che definisci per un flusso. L'uso dei tag è un modo semplice ma efficace per gestire le AWS risorse e organizzare i dati, inclusi i dati di fatturazione.

Indice

- Rivedi le nozioni di base sui tag
- Tieni traccia dei costi utilizzando i tag
- Comprendi le restrizioni relative ai tag
- Etichetta i flussi utilizzando la console Kinesis Data Streams
- Contrassegna gli stream utilizzando AWS CLI
- Etichetta i flussi utilizzando Kinesis Data Streams APIs
- Tagga i consumatori utilizzando il AWS CLI
- Tagga i consumatori utilizzando Kinesis Data Streams APIs

Rivedi le nozioni di base sui tag

Le risorse Kinesis Data Streams che puoi etichettare includono flussi di dati e consumatori fan-out avanzati. Utilizzi la AWS CLI console Kinesis Data Streams o l'API Kinesis Data Streams per completare le seguenti attività:

- Crea una risorsa con tag
- Aggiungere tag a una risorsa
- Elenca i tag per le tue risorse
- Rimuovi i tag da una risorsa

> **Note**
>
> Non puoi applicare tag ai consumatori con fan-out avanzato utilizzando la console Kinesis Data Streams. Per applicare i tag ai consumatori, usa l' AWS CLI API Kinesis Data Streams.

Puoi utilizzare i tag per categorizzare le risorse . Ad esempio, puoi categorizzarle in base a scopo, proprietario o ambiente. Poiché definisci una chiave e un valore per ogni tag, puoi creare un set di categorie personalizzate per soddisfare esigenze specifiche. Ad esempio, potresti definire un set di tag che ti aiuti a tenere traccia delle risorse per proprietario e applicazione associata. Di seguito sono riportati vari esempi di tag:

- Progetto: nome del progetto

- Proprietario: nome
- Scopo: test di carico
- Applicazione: nome dell'applicazione
- Ambiente: produzione

> **Important**
>
> - Per aggiungere tag durante la creazione di uno stream, devi includere le autorizzazioni `kinesis:CreateStream` e le `kinesis:AddTagsToStream` autorizzazioni relative a tale stream. Non puoi utilizzare l'`kinesis:TagResource`autorizzazione per taggare gli stream durante la creazione.
> - Per aggiungere tag durante la registrazione dei consumatori, devi includere le `kinesis:RegisterStreamConsumer` autorizzazioni `kinesis:TagResource` e.

Tieni traccia dei costi utilizzando i tag

Puoi utilizzare i tag per classificare e tenere traccia AWS dei costi. Quando applichi i tag alle tue risorse Kinesis Data Streams AWS , il report sull'allocazione dei costi include l'utilizzo e i costi aggregati per tag. Puoi applicare tag che rappresentano categorie aziendali, come centri di costo, nomi delle applicazioni o proprietari, per organizzare i costi su più servizi. Per ulteriori informazioni, consulta Utilizzo dei tag per l'allocazione dei costi ai fini dei report di fatturazione personalizzati nella AWS Billing User Guide (Guida per l'utente di Amazon API Gateway).

Comprendi le restrizioni relative ai tag

Ai tag si applicano le limitazioni seguenti:

Limitazioni di base

- Il numero massimo di tag per ogni risorsa è 50.
- Per le chiavi e i valori dei tag viene fatta la distinzione tra maiuscole e minuscole.
- Non è possibile cambiare o modificare i tag di una risorsa eliminata.

Limitazioni applicate alle chiavi di tag

- Ogni chiave di tag deve essere univoca. Se aggiungi un tag con una chiave già in uso, il nuovo tag sovrascrive la coppia chiave-valore esistente.
- Non puoi iniziare una chiave di tag con `aws:` perché questo prefisso è riservato all'uso di AWS. AWS crea tag che iniziano con questo prefisso per tuo conto, ma non puoi modificarli o eliminarli.
- Le chiavi di tag devono avere una lunghezza compresa tra 1 e 128 caratteri Unicode.
- Le chiavi di tag devono contenere i seguenti caratteri: lettere Unicode, cifre, spazio e i seguenti caratteri speciali: `_ . / = + - @`.

Limitazioni applicate ai valori dei tag

- I valori dei tag devono avere una lunghezza compresa tra 0 e 255 caratteri Unicode.
- I valori dei tag possono essere vuoti. In caso contrario, devono contenere i seguenti caratteri: lettere Unicode, cifre, spazio e i seguenti caratteri speciali: `_ . / = + - @`.

Etichetta i flussi utilizzando la console Kinesis Data Streams

Puoi aggiungere, aggiornare, elencare e rimuovere tag dai tuoi stream utilizzando la console Kinesis Data Streams.

Visualizzazione di tag per un flusso

1. Accedi AWS Management Console e apri la console Kinesis all'indirizzo https://console.aws.amazon.com /kinesis.
2. Nel riquadro di navigazione a sinistra, scegli Flussi di dati.
3. Nella pagina Flussi di dati, scegli lo stream a cui vuoi taggare.
4. Nella pagina dei dettagli dello stream, scegli Configurazione.
5. Nella sezione Tag, visualizza i tag applicati allo stream.

Per creare un flusso di dati con un tag

1. Apri la console del flusso di dati Kinesis.
2. Nel riquadro di navigazione a sinistra, scegli Flussi di dati.
3. Selezionare Create data stream (Crea flusso di dati).

4. Nella pagina Crea flusso di dati, inserisci un nome per il tuo flusso di dati.
5. Per la capacità del flusso di dati, scegli la modalità di capacità On-demand o Provisioned.

   Per ulteriori informazioni sulle modalità di capacità, vedere. Scegli la modalità di capacità del flusso di dati
6. Nella sezione Tag, effettuate le seguenti operazioni:

   a. Scegli Aggiungi nuovo tag.

   b. Per Chiave, inserite il tag e, facoltativamente, specificate un valore nel campo Valore.

      Se visualizzi un errore, significa che la chiave o il valore del tag che hai specificato non soddisfano le restrizioni relative ai tag. Per ulteriori informazioni, consulta Comprendi le restrizioni relative ai tag.
7. Selezionare Create data stream (Crea flusso di dati).

Per aggiungere o aggiornare un tag in uno stream

1. Apri la console del flusso di dati Kinesis.
2. Nel riquadro di navigazione a sinistra, scegli Flussi di dati.
3. Nella pagina Flussi di dati, scegli lo stream a cui desideri aggiungere o aggiornare i tag.
4. Nella pagina dei dettagli dello stream, scegli Configurazione.
5. Nella sezione Tag scegli Gestisci tag.
6. In Tag, esegui una delle seguenti operazioni:

   - Per aggiungere un nuovo tag, scegli Aggiungi nuovo tag, quindi inserisci i dati Chiave e Valore del tag. Ripetere questa operazione tutte le volte necessarie.

     Il numero massimo di tag che puoi aggiungere per ogni stream è 50.
   - Per aggiornare un tag esistente, inserisci un nuovo valore di tag nel campo Valore della chiave di quel tag.

   Se visualizzi un errore, la chiave o il valore del tag che hai specificato non soddisfa le restrizioni del tag. Per ulteriori informazioni, consulta Comprendi le restrizioni relative ai tag.
7. Scegli Salva modifiche.

Rimozione di un tag da un flusso

1. Apri la console del flusso di dati Kinesis.
2. Nel riquadro di navigazione a sinistra, scegli Flussi di dati.
3. Nella pagina Flussi di dati, scegli il flusso da cui desideri rimuovere i tag.
4. Nella pagina dei dettagli dello stream, scegli Configurazione.
5. Nella sezione Tag scegli Gestisci tag.
6. Trova la coppia di tag Chiave e Valore che desideri rimuovere. Quindi, scegli Rimuovi.
7. Scegli Save changes (Salva modifiche).

Contrassegna gli stream utilizzando AWS CLI

Puoi aggiungere, elencare e rimuovere tag dai tuoi stream utilizzando. AWS CLI Per alcuni esempi, consultare la seguente documentazione.

crea-stream

Crea uno stream con tag.

add-tags-to-stream

Aggiunge o aggiorna i tag per il flusso specificato.

list-tags-for-stream

Elenca i tag per il flusso specificato.

remove-tags-from-stream

Rimuove i tag dal flusso specificato.

Etichetta i flussi utilizzando Kinesis Data Streams APIs

Puoi aggiungere, elencare e rimuovere tag dai tuoi stream utilizzando Kinesis Data Streams. APIs Per alcuni esempi, consultare la seguente documentazione:

CreateStream

Crea uno stream con tag.

AddTagsToStream

Aggiunge o aggiorna i tag per il flusso specificato.

ListTagsForStream

Elenca i tag per il flusso specificato.

RemoveTagsFromStream

Rimuove i tag dal flusso specificato.

Tagga i consumatori utilizzando il AWS CLI

Puoi aggiungere, elencare e rimuovere tag sui tuoi consumatori utilizzando il AWS CLI. Per alcuni esempi, consultare la seguente documentazione:

register-stream-consumer

Registra un consumatore per un flusso di dati Kinesis con tag.

tag-resource

Aggiunge o aggiorna i tag per la risorsa Kinesis specificata.

list-tags-for-resource

Elenca i tag per la risorsa Kinesis specificata.

untag-resource

Rimuove i tag dalla risorsa Kinesis specificata.

Tagga i consumatori utilizzando Kinesis Data Streams APIs

Puoi aggiungere, elencare e rimuovere tag sui tuoi consumatori utilizzando Kinesis APIs Data Streams. Per alcuni esempi, consultare la seguente documentazione:

RegisterStreamConsumer

Registra un consumatore per un flusso di dati Kinesis con tag.

TagResource

Aggiunge o aggiorna i tag per la risorsa Kinesis specificata.

[ListTagsForResource](#)

Elenca i tag per la risorsa Kinesis specificata.

[UntagResource](#)

Rimuove i tag dalla risorsa Kinesis specificata.

Scrittura di dati su Amazon Kinesis Data Streams

Un producer è un'applicazione che scrive dati su Flusso di dati Amazon Kinesis. È possibile creare produttori per Kinesis Data Streams AWS SDK per Java utilizzando e la Kinesis Producer Library (KPL).

Se sei nuovo per il flusso di dati Kinesis, ti consigliamo di familiarizzare prima con i concetti chiave e la terminologia introdotti in Cos'è Amazon Kinesis Data Streams? e Utilizza il AWS CLI per eseguire operazioni di Amazon Kinesis Data Streams.

> **Important**
>
> Il flusso di dati Kinesis supporta le modifiche al periodo di conservazione dei record di dati del tuo flusso di dati. Per ulteriori informazioni, consulta Modifica il periodo di conservazione dei dati.

Per inserire dati nel flusso, devi specificare il nome del flusso, una chiave di partizione e il blob di dati da aggiungere al flusso. La chiave di partizione viene utilizzata per determinare a quale shard nel flusso viene aggiunto il record di dati.

Tutti i dati nello shard vengono inviati allo stesso lavoratore che elabora lo shard. Quale chiave di partizione utilizzi dipende dalla logica dell'applicazione. Il numero di chiavi di partizione normalmente deve essere molto superiore al numero di shard. Questo perché la chiave di partizione viene utilizzata per determinare come mappare un record di dati a un determinato shard. Se hai un numero sufficiente di chiavi di partizione, i dati possono essere distribuiti in modo uniforme negli shard in un flusso.

Argomenti

- Sviluppa produttori utilizzando Amazon Kinesis Producer Library (KPL)
- Sviluppa i produttori utilizzando l'API Amazon Kinesis Data Streams con AWS SDK per Java
- Scrivi su Amazon Kinesis Data Streams utilizzando Kinesis Agent
- Scrivi su Kinesis Data AWS Streams utilizzando altri servizi
- Scrivi su Kinesis Data Streams utilizzando integrazioni di terze parti
- Risolvi i problemi dei produttori di Amazon Kinesis Data Streams
- Ottimizza i produttori di Kinesis Data Streams

Sviluppa produttori utilizzando Amazon Kinesis Producer Library (KPL)

Un producer del flusso di dati Amazon Kinesis è qualsiasi applicazione che inserisce record di dati utente in un flusso di dati Kinesis (anche detto importazione dei dati). Amazon Kinesis Producer Library (KPL) semplifica lo sviluppo di applicazioni di produzione, permettendo agli sviluppatori di ottenere un throughput di scrittura elevato su un flusso di dati Kinesis.

Puoi monitorare il KPL con Amazon CloudWatch. Per ulteriori informazioni, consulta Monitoraggio della Kinesis Client Library con Amazon CloudWatch.

Argomenti

- Rivedi il ruolo del KPL
- Scopri i vantaggi dell'utilizzo di KPL
- Comprendi quando non usare KPL
- Installa il KPL
- Esegui la migrazione da KPL 0.x a KPL 1.x
- Transizione ai certificati Amazon Trust Services (ATS) per la KPL
- Piattaforme supportate da KPL
- Concetti chiave di KPL
- Integra il KPL con il codice del produttore
- Scrivi nel tuo flusso di dati Kinesis utilizzando KPL
- Configurazione della libreria Amazon Kinesis Producer
- Implementa la deaggregazione dei consumatori
- Usa KPL con Amazon Data Firehose
- Utilizzate il KPL con lo Schema Registry AWS Glue
- Configura la configurazione del proxy KPL
- Politica del ciclo di vita della versione KPL

> **Note**
>
> Consigliamo di eseguire l'aggiornamento alla versione KPL più recente. La KCL viene regolarmente aggiornata con versioni più recenti che includono le ultime patch di dipendenza

e sicurezza, correzioni di bug e nuove funzionalità retrocompatibili. Per maggiori informazioni, consulta https://github.com/awslabs/amazon-kinesis-producer/releases/.

Rivedi il ruolo del KPL

KPL è una easy-to-use libreria altamente configurabile che ti aiuta a scrivere su un flusso di dati Kinesis. Funge da intermediario tra il codice dell'applicazione producer e le operazioni API del flusso di dati Kinesis. La KPL esegue le seguenti attività primarie:

- Scrive su uno o più flussi di dati Kinesis con un meccanismo di tentativi automatico e configurabile
- Raccoglie record e utilizza `PutRecords` per scrivere più record in più shard per richiesta
- Unisce i record degli utenti per aumentare la dimensione del payload e migliorare il throughput
- Si integra perfettamente con la Kinesis Client Library (KCL) per disaggregare i record raggruppati nel consumer
- Invia i CloudWatch parametri di Amazon per tuo conto per fornire visibilità sulle prestazioni dei produttori

Tieni presente che KPL è diverso dall'API Kinesis Data Streams disponibile in. AWS SDKs L'API del flusso di dati Kinesis consente di gestire molti aspetti del flusso di dati Kinesis (inclusa la creazione di flussi, il ripartizionamento e l'inserimento e l'estrazione di record), mentre la KPL fornisce un livello di astrazione specificamente per l'importazione dei dati. Per ulteriori informazioni sulle API del flusso di dati Kinesis, consulta la Documentazione di riferimento delle API di Amazon Kinesis.

Scopri i vantaggi dell'utilizzo di KPL

L'elenco seguente rappresenta alcuni dei principali vantaggi dell'utilizzo della KPL per lo sviluppo di producer del flusso di dati Kinesis.

La KPL può essere utilizzata in casi di utilizzo sincroni o asincroni. Suggeriamo di utilizzare le prestazioni più elevate dell'interfaccia asincrona, a meno che non vi sia un motivo specifico per l'utilizzo del comportamento sincrono. Per ulteriori informazioni su questi due casi d'uso e sul codice di esempio, consulta Scrivi nel tuo flusso di dati Kinesis utilizzando KPL.

Vantaggi in termini di prestazioni

La KPL può aiutare a creare producer ad alte prestazioni. Prendi in considerazione una situazione in cui le tue EC2 istanze Amazon fungono da proxy per raccogliere eventi da 100 byte da

centinaia o migliaia di dispositivi a basso consumo e scrivere record in un flusso di dati Kinesis. Ciascuna di queste EC2 istanze deve scrivere migliaia di eventi al secondo nel tuo flusso di dati. Per ottenere il throughput necessario, i produttori devono implementare una logica complicata, ad esempio esecuzione di batch o multithreading, in aggiunta al tentativo di disaggregare la logica e i record da parte del consumer. La KPL esegue tutte queste attività per tuo conto.

Consumer-Side Ease of Use (Facilità utilizzo lato consumer)

Per gli sviluppatori lato consumer che utilizzano la KCL in Java, la KPL si integra senza sforzo aggiuntivo. Quando la KCL recupera un record del flusso di dati Kinesis aggregato composto da più record utente della KPL, automaticamente richiama la KPL per estrarre i record utente individuali prima di restituirli all'utente.

Per gli sviluppatori lato consumer che non utilizzano la KCL ma invece utilizzano l'operazione API `GetRecords` direttamente, una libreria Java della KPL è disponibile per estrarre i record utente individuali prima di restituirli all'utente.

Monitoraggio producer

Puoi raccogliere, monitorare e analizzare i tuoi produttori di Kinesis Data Streams utilizzando CloudWatch Amazon e KPL. Il KPL emette velocità effettiva, errori e altre metriche per tuo CloudWatch conto ed è configurabile per il monitoraggio a livello di stream, shard o produttore.

Asynchronous Architecture (Architettura asincrona)

Poiché il KPL può memorizzare nel buffer i record prima di inviarli a Kinesis Data Streams, non impone all'applicazione chiamante di bloccarsi e attendere la conferma che il record è arrivato al server prima di continuare l'esecuzione. Una chiamata per inserire un record nella KPL restituisce sempre immediatamente e non attende l'invio del record né di ricevere una risposta dal server. Al contrario, viene creato un oggetto `Future` che riceve il risultato di inviare il record al flusso di dati Kinesis in un secondo momento. Questo è lo stesso comportamento dei client asincroni nell'SDK. AWS

Comprendi quando non usare KPL

La KPL può subire un ulteriore ritardo di elaborazione fino a `RecordMaxBufferedTime` all'interno della libreria (configurabile dall'utente). Valori più elevati di `RecordMaxBufferedTime` risultano in creazione di pacchetti più veloce e prestazioni migliori. Le applicazioni che non possono tollerare questo ritardo aggiuntivo potrebbero dover utilizzare direttamente l' AWS SDK. Per ulteriori informazioni sull'utilizzo dell' AWS SDK con Kinesis Data Streams, consulta. Sviluppa i produttori

utilizzando l'API Amazon Kinesis Data Streams con AWS SDK per Java Per ulteriori informazioni su `RecordMaxBufferedTime` e altre proprietà configurabili dall'utente della KPL, consulta Configurazione della libreria Amazon Kinesis Producer.

Installa il KPL

Amazon fornisce file binari predefiniti della Amazon Kinesis Producer Library (KPL) C++ per macOS, Windows e distribuzioni Linux recenti (per i dettagli sulla piattaforma supportata, consulta la sezione successiva). Questi file binari vengono confezionati come parte dei file.jar Java e vengono automaticamente richiamati e utilizzati se utilizzi Maven per installare il pacchetto. Per individuare le versioni più recenti della KPL e della KCL, utilizza i seguenti collegamenti di ricerca Maven:

- KPL
- KCL

I file binari Linux sono stati compilati con GNU Compiler Collection (GCC) e collegati staticamente a libstdc++ su Linux. Dovrebbero funzionare su qualsiasi distribuzione Linux a 64 bit, che include una versione glibc 2.5 o successiva.

Gli utenti di distribuzioni Linux precedenti possono creare il KPL utilizzando le istruzioni di compilazione fornite insieme al codice sorgente. GitHub Per scaricare il KPL da GitHub, consulta Amazon Kinesis Producer Library.

> **Important**
>
> Amazon Kinesis Producer Library (KPL) 0.x sarà disponibile il 30 end-of-support gennaio 2026. Ti consigliamo vivamente di migrare le tue applicazioni KPL utilizzando la versione 0.x all'ultima versione KPL prima del 30 gennaio 2026. Per trovare la versione KPL più recente, consulta la pagina KPL su Github. Per informazioni sulla migrazione da KPL 0.x a KPL 1.x, consulta. Esegui la migrazione da KPL 0.x a KPL 1.x

Esegui la migrazione da KPL 0.x a KPL 1.x

Questo argomento fornisce step-by-step istruzioni per migrare il consumatore da KPL 0.x a KPL 1.x. KPL 1.x introduce il supporto per la versione AWS SDK per Java 2.x pur mantenendo la compatibilità dell'interfaccia con le versioni precedenti. Non è necessario aggiornare la logica di elaborazione dei dati di base per migrare a KPL 1.x.

1. Assicurati di avere i seguenti prerequisiti:

   - Java Development Kit (JDK) 8 o versione successiva
   - AWS SDK per Java 2.x
   - Maven o Gradle per la gestione delle dipendenze

2. Aggiungi dipendenze

   Se utilizzi Maven, aggiungi la seguente dipendenza al tuo file pom.xml. Assicurati di aver aggiornato GroupId `com.amazonaws` da `software.amazon.kinesis` a e la `1.x.x` versione all'ultima versione KPL.

   ```
   <dependency>
       <groupId>software.amazon.kinesis</groupId>
       <artifactId>amazon-kinesis-producer</artifactId>
       <version>1.x.x</version> <!-- Use the latest version -->
   </dependency>
   ```

   Se stai usando Gradle, aggiungi quanto segue al tuo file. `build.gradle` Assicurati di sostituirlo `1.x.x` con l'ultima versione di KPL.

   ```
   implementation 'software.amazon.kinesis:amazon-kinesis-producer:1.x.x'
   ```

   Puoi verificare la versione più recente di KPL sul Maven Central Repository.

3. Aggiorna le dichiarazioni di importazione per KPL

   KPL 1.x utilizza la versione AWS SDK per Java 2.x e utilizza un nome di pacchetto aggiornato che inizia con`software.amazon.kinesis`, rispetto al nome del pacchetto nel KPL precedente che inizia con. `com.amazonaws.services.kinesis`

   Sostituisci l'importazione per con. `com.amazonaws.services.kinesis` `software.amazon.kinesis` La tabella seguente elenca le importazioni che è necessario sostituire.

Importa sostituzioni

Sostituisci:	Con:
import com.amazonaws.services.kinesis.producer.Attempt;	importa software.amazon.kinesis.Producer.tentment;
importa com.amazonaws.services.kinesis.producer. BinaryToHexConverter;	importare software.amazon.kinesis.producer.BinaryToHexConverter;
importare com.amazonaws.services.kinesis.producer. CertificateExtractor;	importare software.amazon.kinesis.producer.CertificateExtractor;
importare com.amazonaws.services.kinesis.producer.daemon;	importare software.amazon.kinesis.Producer.daemon;
importa com.amazonaws.services.kinesis.producer. DaemonException;	importare software.amazon.kinesis.producer.DaemonException;
importare com.amazonaws.services.kinesis.producer. FileAgeManager;	importare software.amazon.kinesis.producer.FileAgeManager;
importare com.amazonaws.services.kinesis.producer. FutureTimedOutException;	importare software.amazon.kinesis.producer.FutureTimedOutException;
importare com.amazonaws.services.kinesis.producer. GlueSchemaRegistrySerializerInstance;	importare software.amazon.kinesis.producer.GlueSchemaRegistrySerializerInstance;
importare com.amazonaws.services.kinesis.producer. HashedFileCopier;	importare software.amazon.kinesis.producer.HashedFileCopier;
importare com.amazonaws.services.kinesis.producer. IKinesisproduttore;	importa software.amazon.kinesis.producer.IKinesisproduttore;
importa com.amazonaws.services.kinesis.producer. IrrecoverableError;	importare software.amazon.kinesis.producer.IrrecoverableError;

Sostituisci:	Con:
importare com.amazonaws.services.kinesis.producer. KinesisProducer;	importare software.amazon.kinesis.producer. KinesisProducer;
importare com.amazonaws.services.kinesis.producer. KinesisProducerConfiguration;	importare software.amazon.kinesis.producer. KinesisProducerConfiguration;
importare com.amazonaws.services.kinesis.producer. LogInputStreamReader;	importare software.amazon.kinesis.producer. LogInputStreamReader;
importare com.amazonaws.services.kinesis.producer.metric;	importare software.amazon.kinesis.producer.metric;
importa com.amazonaws.services.kinesis.producer. ProcessFailureBehavior;	importare software.amazon.kinesis.producer. ProcessFailureBehavior;
importare com.amazonaws.services.kinesis.producer. UnexpectedMessageException;	importare software.amazon.kinesis.producer. UnexpectedMessageException;
importare com.amazonaws.services.kinesis.producer. UserRecord;	importare software.amazon.kinesis.producer. UserRecord;
importare com.amazonaws.services.kinesis.producer. UserRecordFailedException;	importare software.amazon.kinesis.producer. UserRecordFailedException;
importare com.amazonaws.services.kinesis.producer. UserRecordResult;	importare software.amazon.kinesis.producer. UserRecordResult;
importare com.amazonaws.services.kinesis.producer.protobuf.messages;	importare software.amazon.kinesis.producer.protobuf.messages;
importare com.amazonaws.services.kinesis.producer.protobuf.config;	importare software.amazon.kinesis.producer.protobuf.config;

4. Aggiorna le dichiarazioni di importazione per le classi del fornitore di credenziali AWS

   Durante la migrazione a KPL 1.x, è necessario aggiornare i pacchetti e le classi importati nel codice dell'applicazione KPL basati sulla versione AWS SDK per Java 1.x con quelli corrispondenti basati sulla 2.x. AWS SDK per Java Le importazioni comuni nell'applicazione

KPL sono le classi di fornitori di credenziali. Consulta le modifiche al provider di credenziali nella documentazione della guida alla migrazione AWS SDK per Java 2.x per l'elenco completo delle modifiche al provider di credenziali. Ecco le comuni modifiche di importazione che potresti dover apportare alle tue applicazioni KPL.

Importa in KPL 0.x

```
import com.amazonaws.auth.DefaultAWSCredentialsProviderChain;
```

Importazione in KPL 1.x

```
import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
```

Se importi altri provider di credenziali basati sulla versione AWS SDK per Java 1.x, devi aggiornarli con quelli equivalenti alla AWS SDK per Java versione 2.x. Se non hai importato nessuna classe/pacchetto dalla versione AWS SDK per Java 1.x, puoi ignorare questo passaggio.

5. Aggiorna la configurazione del provider di credenziali nella configurazione KPL

La configurazione del provider di credenziali in KPL 1.x richiede i provider di credenziali 2.x. AWS SDK per Java Se stai passando i provider di credenziali per la versione AWS SDK per Java 1.x `KinesisProducerConfiguration` sovrascrivendo il provider di credenziali predefinito, devi aggiornarlo con i provider di credenziali 2.x. AWS SDK per Java Vedi Modifiche al provider di credenziali nella documentazione della guida alla migrazione AWS SDK per Java 2.x per l'elenco completo delle modifiche al provider di credenziali. Se non hai sovrascritto il provider di credenziali predefinito nella configurazione KPL, puoi ignorare questo passaggio.

Ad esempio, se stai sovrascrivendo il provider di credenziali predefinito per KPL con il codice seguente:

```
KinesisProducerConfiguration config = new KinesisProducerConfiguration();
// SDK v1 default credentials provider
config.setCredentialsProvider(new DefaultAWSCredentialsProviderChain());
```

È necessario aggiornarli con il codice seguente per utilizzare il provider di credenziali AWS SDK per Java 2.x:

```
KinesisProducerConfiguration config = new KinesisProducerConfiguration();
```

```
// New SDK v2 default credentials provider
config.setCredentialsProvider(DefaultCredentialsProvider.create());
```

Transizione ai certificati Amazon Trust Services (ATS) per la KPL

Il 9 febbraio 2018, alle 9.00 PST, Flusso di dati Amazon Kinesis ha installato i certificati ATS. Per continuare a scrivere record su Kinesis Data Streams utilizzando Amazon Kinesis Producer Library (KPL), è necessario aggiornare l'installazione di KPL alla versione 0.12.6 o successiva. Questa modifica riguarda tutte le regioni. AWS

Per informazioni sul passaggio ad ATS, consulta How to Prepare for AWS's Move to Its Own Certificate Authority.

In caso di problemi e se hai bisogno di supporto tecnico, crea un caso utilizzando il Centro di supporto AWS .

Piattaforme supportate da KPL

Amazon Kinesis Producer Library (KPL) è scritta in C++ e viene eseguita come processo secondario rispetto al processo utente principale. I file binari nativi precompilati a 64 bit sono nel rilascio di Java e sono gestiti dal wrapper Java.

Il pacchetto Java viene eseguito senza la necessità di installare librerie aggiuntive per i seguenti sistemi operativi:

- Distribuzioni Linux con kernel 2.6.18 (settembre 2006) e versioni successive
- Apple iOS X 10.9 e versioni successive
- Windows Server 2008 e versione successiva

> **Important**
>
> Windows Server 2008 e versioni successive è supportato per tutte le versioni KPL fino alla versione 0.14.0.
> La piattaforma Windows NON è supportata a partire dalla versione KPL 0.14.0 o successiva.

Tenere presente che la KPL è solo a 64 bit.

Codice sorgente

Se i file binari forniti nell'installazione della KPL non sono sufficienti per l'ambiente, il core della KPL sarà scritto come modulo C++. Il codice sorgente per il modulo C++ e l'interfaccia Java sono rilasciati con Amazon Public License e sono disponibili GitHub presso Amazon Kinesis Producer Library. Sebbene la KPL possa essere utilizzata su qualsiasi piattaforma per la quale sono disponibili un compilatore C++ conforme agli standard e JRE, Amazon non supporta ufficialmente alcuna piattaforma che non si trova nell'elenco delle piattaforme supportate.

Concetti chiave di KPL

Le seguenti sezioni contengono i concetti e la terminologia necessari per comprendere e sfruttare la Amazon Kinesis Producer Library (KPL).

Argomenti

- Registri
- Batching
- Aggregazione
- Raccolta

Registri

In questa guida, si distingue tra record utente della KPL e record del flusso di dati Kinesis. Quando utilizziamo il termine record senza un qualificatore, ci riferiamo a un record utente dell KPL. Quando facciamo riferimento a un record del flusso di dati Kinesis, diciamo esplicitamente record del flusso di dati Kinesis.

Un record utente della KPL è un blob di dati che ha un significato particolare per l'utente. Alcuni esempi includono un blob JSON che rappresenta un evento di interfaccia utente su un sito Web o una voce di log da un server Web.

Un record del flusso di dati Kinesis è un'istanza della struttura dati `Record` definita dall'API del servizio del flusso di dati Kinesis. Contiene una chiave di partizione, un numero di sequenza e un blob di dati.

Batching

Batching si riferisce all'esecuzione di una singola operazione su più elementi anziché eseguire ripetutamente l'operazione su ogni singolo elemento.

In questo contesto, un "elemento" è un record e l'operazione è inviarlo al flusso di dati Kinesis. In una situazione non di batching, è necessario posizionare ogni record in un record del flusso di dati Kinesis separato ed effettuare una richiesta HTTP per inviarlo al flusso di dati Kinesis. Con batching, ogni richiesta HTTP è in grado di eseguire più record invece di uno solo.

La KPL supporta due tipi di batching:

- Aggregazione: l'archiviazione di più record in un singolo record del flusso di dati Kinesis.
- Raccolta: l'utilizzo dell'operazione API `PutRecords` per inviare più record del flusso di dati Kinesis a una o più partizioni nel flusso di dati Kinesis.

I due tipi di batching KPL sono stati progettati per coesistere e possono essere attivati o disattivati in modo indipendente. Come impostazione predefinita, entrambi sono abilitati.

Aggregazione

Aggregazione si riferisce all'archiviazione di più record in un record del flusso di dati Kinesis. L'aggregazione consente ai clienti di aumentare il numero di record inviati per chiamata API e ciò aumenta anche il throughput producer.

Le partizioni del flusso di dati Kinesis supportano fino a 1.000 record del flusso di dati Kinesis al secondo o 1 MB di velocità di trasmissione effettiva. I record al secondo del flusso di dati Kinesis vincola i clienti a record di dimensioni inferiori a 1 KB. L'aggregazione dei record consente ai clienti di combinare più record in un singolo record del flusso di dati Kinesis. Questo consente ai clienti di migliorare il loro throughput per shard.

Si consideri il caso di uno shard nella regione us-east-1 attualmente in esecuzione a una velocità costante di 1.000 record al secondo, con record di 512 byte ciascuno. Con l'aggregazione della KPL, è possibile impacchettare 1.000 record in soli 10 record del flusso di dati Kinesis, riducendo le richieste al secondo (RPS) a 10 (a 50 KB ognuna).

Raccolta

Raccolta si riferisce alla creazione di batch di più record del flusso di dati Kinesis e al loro invio in una singola richiesta HTTP con una chiamata all'operazione API `PutRecords`, invece che inviare ogni record del flusso di dati Kinesis nella propria richiesta HTTP.

Questo aumenta il throughput rispetto all'utilizzo di nessuna raccolta perché riduce i costi di molte richieste HTTP separate. Infatti, `PutRecords` è stato specificamente progettato per questo scopo.

La raccolta differisce dall'aggregazione in quanto lavora con gruppi di record del flusso di dati Kinesis. I record del flusso di dati Kinesis raccolti possono comunque contenere più record da parte dell'utente. La relazione può essere visualizzata come segue:

```
record 0 --|
record 1   |        [ Aggregation ]
    ...    |--> Amazon Kinesis record 0 --|
    ...    |                              |
record A --|                              |
                                           |
    ...                  ...               |
                                           |
record K --|                              |
record L   |                              |      [ Collection ]
    ...    |--> Amazon Kinesis record C --|--> PutRecords Request
    ...    |                              |
record S --|                              |
                                           |
    ...                  ...               |
                                           |
record AA--|                              |
record BB  |                              |
    ...    |--> Amazon Kinesis record M --|
    ...    |
record ZZ--|
```

Integra il KPL con il codice del produttore

Amazon Kinesis Producer Library (KPL) viene eseguita in un processo separato e comunica con il processo utente principale tramite IPC. Questa architettura talvolta viene definita un microservizio ed è scelta per due motivi:

1) Il processo utente non si interrompe anche se la KPL riporta un arresto anomalo

Il processo può avere attività non correlate al flusso di dati Kinesis e può essere in grado di continuare l'operazione anche se la KPL si arresta in maniera anomala. È anche possibile che il processo dell'utente principale riavvi la KPL e passi a uno stato di funzionamento completo (questa funzionalità è nei wrapper ufficiali).

Un esempio è un server Web che invia i parametri al flusso di dati Kinesis; il server può continuare a servire le pagine anche se la parte del flusso di dati Kinesis ha smesso di funzionare. Interrompere l'intero server a causa di un bug nella KPL provocherebbe interruzioni inutili.

2) I client arbitrari possono essere supportati

Ci sono sempre clienti che utilizzano linguaggi diversi da quelli ufficialmente supportati. Questi clienti dovrebbero essere in grado di utilizzare facilmente la KPL.

Matrice di utilizzo consigliata

La seguente matrice di utilizzo elenca le impostazioni consigliate per i diversi utenti e consiglia se e come utilizzare il KPL. Ricorda che se l'aggregazione è abilitata, la disaggregazione deve essere utilizzata per estrarre i record nel lato consumer.

Linguaggio lato producer	Linguaggio lato consumer	Versione della KCL	Logica di checkpoint	Puoi utilizzare la KPL?	Avvertenze
Tutto tranne Java	*	*	*	No	N/D
Java	Java	Utilizza Java SDK direttamente	N/D	Sì	Se l'aggregazione viene utilizzata, è necessario utilizzare la libreria di disaggregazione fornita dopo le chiamate `GetRecords` .
Java	Tutto tranne Java	Utilizza SDK direttamente	N/D	Sì	È necessario disabilitare l'aggregazione.
Java	Java	1.3.x	N/D	Sì	È necessario disabilit

Linguaggio lato producer	Linguaggio lato consumer	Versione della KCL	Logica di checkpoint	Puoi utilizzare la KPL?	Avvertenze
					are l'aggregazione.
Java	Java	1.4.x	Chiama il checkpoint senza alcun argomento	Sì	Nessuno
Java	Java	1.4.x	Chiama il checkpoint con un numero di sequenza esplicito	Sì	Disabilita l'aggregazione o modifica il codice per utilizzare numeri di sequenza estesi per il checkpoint.
Java	Tutto tranne Java	1.3.x + daemon multilingua + wrapper specifici della lingua	N/D	Sì	È necessario disabilitare l'aggregazione.

Scrivi nel tuo flusso di dati Kinesis utilizzando KPL

Le sezioni seguenti mostrano un esempio di codice in una progressione dal produttore più semplice al codice completamente asincrono.

Codice del produttore Barebones

Il codice seguente è tutto quello che serve per scrivere un producer che lavora al minimo. I record utente di Amazon Kinesis Producer Library (KPL) vengono elaborati in background.

```
// KinesisProducer gets credentials automatically like
// DefaultAWSCredentialsProviderChain.
// It also gets region automatically from the EC2 metadata service.
KinesisProducer kinesis = new KinesisProducer();
// Put some records
for (int i = 0; i < 100; ++i) {
    ByteBuffer data = ByteBuffer.wrap("myData".getBytes("UTF-8"));
    // doesn't block
    kinesis.addUserRecord("myStream", "myPartitionKey", data);
}
// Do other stuff ...
```

Rispondi ai risultati in modo sincrono

Nell'esempio precedente, il codice non ha controllato se i record utente della KPL sono riusciti. La KPL esegue qualsiasi tentativo necessario per tenere conto degli errori. Tuttavia, se desideri controllare i risultati, puoi esaminarli utilizzando gli oggetti `Future` restituiti da `addUserRecord`, come nell'esempio seguente (esempio precedente mostrato per contesto):

```
KinesisProducer kinesis = new KinesisProducer();

// Put some records and save the Futures
List<Future<UserRecordResult>> putFutures = new
 LinkedList<Future<UserRecordResult>>();
for (int i = 0; i < 100; i++) {
    ByteBuffer data = ByteBuffer.wrap("myData".getBytes("UTF-8"));
    // doesn't block
    putFutures.add(
        kinesis.addUserRecord("myStream", "myPartitionKey", data));
}

// Wait for puts to finish and check the results
for (Future<UserRecordResult> f : putFutures) {
    UserRecordResult result = f.get(); // this does block
    if (result.isSuccessful()) {
        System.out.println("Put record into shard " +
                           result.getShardId());
```

```
    } else {
        for (Attempt attempt : result.getAttempts()) {
            // Analyze and respond to the failure
        }
    }
}
```

Rispondi ai risultati in modo asincrono

L'esempio precedente è la chiamata `get()` a un `Future` oggetto, che blocca il runtime. Se non vuoi bloccare il runtime, puoi usare un callback asincrono, come mostrato nell'esempio seguente:

```
KinesisProducer kinesis = new KinesisProducer();

FutureCallback<UserRecordResult> myCallback = new FutureCallback<UserRecordResult>() {

    @Override public void onFailure(Throwable t) {
        /* Analyze and respond to the failure  */
    };
    @Override public void onSuccess(UserRecordResult result) {
        /* Respond to the success */
    };
};

for (int i = 0; i < 100; ++i) {
    ByteBuffer data = ByteBuffer.wrap("myData".getBytes("UTF-8"));
    ListenableFuture<UserRecordResult> f = kinesis.addUserRecord("myStream",
 "myPartitionKey", data);
    // If the Future is complete by the time we call addCallback, the callback will be
 invoked immediately.
    Futures.addCallback(f, myCallback);
}
```

Configurazione della libreria Amazon Kinesis Producer

Nonostante le impostazioni predefinite dovrebbero funzionare perfettamente per la maggior parte dei casi d'uso, è consigliabile modificare alcune delle impostazioni predefinite per adattare il comportamento di `KinesisProducer` alle proprie esigenze. Un'istanza di classe `KinesisProducerConfiguration` può essere passata al costruttore `KinesisProducer` per fare ciò, per esempio:

```
KinesisProducerConfiguration config = new KinesisProducerConfiguration()
```

```
        .setRecordMaxBufferedTime(3000)
        .setMaxConnections(1)
        .setRequestTimeout(60000)
        .setRegion("us-west-1");

final KinesisProducer kinesisProducer = new KinesisProducer(config);
```

Puoi anche caricare una configurazione da un file proprietà:

```
KinesisProducerConfiguration config =
 KinesisProducerConfiguration.fromPropertiesFile("default_config.properties");
```

Puoi sostituire qualsiasi percorso e nome di file al quale ha accesso il processo utente. Inoltre puoi chiamare metodi impostati sull'istanza `KinesisProducerConfiguration` creata in questo modo per personalizzare la configurazione.

Il file delle proprietà deve specificare i parametri usando i loro nomi in PascalCase.
I nomi corrispondono a quelli utilizzati nei metodi impostati nella classe `KinesisProducerConfiguration`. Per esempio:

```
RecordMaxBufferedTime = 100
MaxConnections = 4
RequestTimeout = 6000
Region = us-west-1
```

Per ulteriori informazioni sulle regole di utilizzo dei parametri di configurazione e sui limiti di valore, vedere il file di esempio delle proprietà di configurazione su GitHub.

Nota che dopo che `KinesisProducer` è stato inizializzato, modificare l'istanza `KinesisProducerConfiguration` che è stata utilizzata non ha un effetto ulteriore. `KinesisProducer` non supporta al momento la riconfigurazione automatica.

Implementa la deaggregazione dei consumatori

A partire dalla versione 1.4.0, la KCL supporta la disaggregazione automatica di record utente della KPL. Il codice dell'applicazione consumer scritto con versioni precedenti della KCL verrà compilato senza alcuna modifica dopo l'aggiornamento della KCL. Tuttavia, se l'aggregazione della KPL viene utilizzata nel lato producer, bisogna tenere conto di quanto segue riguardo i checkpoint: tutti i sottorecord in un record aggregato hanno lo stesso numero di sequenza, quindi i dati aggiuntivi

devono essere memorizzati nel checkpoint se è necessario distinguere tra sottorecord. Questi dati supplementari vengono definiti come numero sottosuccessione.

Opzioni

- Esegui la migrazione dalle versioni precedenti di KCL
- Usa le estensioni KCL per la disaggregazione KPL
- Usa direttamente GetRecords

Esegui la migrazione dalle versioni precedenti di KCL

Non è necessario modificare le chiamate esistenti per eseguire il checkpoint con aggregazione. È comunque garantito che è possibile recuperare tutti i record archiviati nel flusso di dati Kinesis. Il KCL offre ora due nuove operazioni di checkpoint per supportare casi d'uso particolari, descritti di seguito.

Se il codice esistente è stato scritto per KCL prima del supporto KPL e l'operazione di checkpoint viene chiamata senza argomenti, equivale a controllare il numero di sequenza dell'ultimo record utente KPL del batch. Se l'operazione di checkpoint viene chiamata con una stringa di numero di sequenza, è equivalente all'esecuzione del checkpoint nel numero di sequenza del batch fornito assieme al numero di sottosuccessione 0 (zero).

Chiamare la nuova operazione di checkpoint della KCL `checkpoint()` senza alcun argomento è equivalente dal punto di vista semantico all'esecuzione di checkpoint nel numero di sequenza dell'ultima chiamata `Record` nel batch, assieme al numero di sottosuccessione implicito 0 (zero).

Chiamare la nuova operazione di checkpoint della KCL `checkpoint(Record record)` è equivalente dal punto di vista semantico all'esecuzione di checkpoint nel numero di sequenza del `Record` fornito, assieme al numero di sottosuccessione implicito 0 (zero). Se la chiamata `Record` è effettivamente un `UserRecord`, sul numero di sequenza `UserRecord` e sul numero di sottosuccessione viene eseguito il checkpoint.

Se si richiama la nuova operazione checkpoint della KCL `checkpoint(String sequenceNumber, long subSequenceNumber)` sul numero di sequenza assieme al numero fornito di sottosuccessione viene eseguito il checkpoint.

In questi casi, dopo che il checkpoint è stato memorizzato nella tabella di checkpoint di Amazon DynamoDB, la KCL può riprendere correttamente i record di recupero anche quando l'applicazione si interrompe e riavvia. Se più record sono contenuti nella sequenza, il recupero viene eseguito a partire dal prossimo record di numero di sottosuccessione nel record con la sequenza sulla quale è

stato eseguito il checkpoint più recentemente. Se il checkpoint più recente includeva l'ultimo record di numero di sottosuccessione del precedente record di numero di sequenza, il recupero parte con il record con il prossimo numero di sequenza.

La sezione successiva illustra i dettagli del checkpoint di sequenza e sequenza per i consumatori, che devono evitare di saltare e duplicare i record. Se il salto (o duplicazione) di record quando interrompi e riavvii l'elaborazione del record del consumer non è importante, puoi eseguire il codice esistente senza alcuna modifica.

Usa le estensioni KCL per la disaggregazione KPL

La disaggregazione KPL può comportare il checkpoint di sottosequenza. Per facilitare l'utilizzo del checkpoint della sottosuccessione, una classe `UserRecord` è stata aggiunta alla KCL:

```
public class UserRecord extends Record {
    public long getSubSequenceNumber() {
    /* ... */
    }
    @Override
    public int hashCode() {
    /* contract-satisfying implementation */
    }
    @Override
    public boolean equals(Object obj) {
    /* contract-satisfying implementation */
    }
}
```

Questa classe viene ora utilizzata invece di `Record`. Questo non interrompe il codice esistente perché è una sottoclasse di `Record`. La classe `UserRecord` rappresenta entrambi i sottorecord effettivi e i record standard, non aggregati. I record non aggregati non possono essere considerati come record aggregati con un solo sottorecord.

Inoltre, due operazioni nuove vengono aggiunte a `IRecordProcessorCheckpointer`:

```
public void checkpoint(Record record);
public void checkpoint(String sequenceNumber, long subSequenceNumber);
```

Per iniziare a utilizzare il checkpoint del numero di sottosuccessione, puoi eseguire la seguente conversione. Modifica il seguente codice di modulo:

```
checkpointer.checkpoint(record.getSequenceNumber());
```

Nuovo codice di modulo:

```
checkpointer.checkpoint(record);
```

È consigliabile utilizzare il modulo `checkpoint(Record record)` per il checkpoint della sottosuccessione. Tuttavia, se stai già memorizzando `sequenceNumbers` in stringhe da utilizzare per il checkpoint, devi anche memorizzare `subSequenceNumber`, come nell'esempio seguente:

```
String sequenceNumber = record.getSequenceNumber();
long subSequenceNumber = ((UserRecord) record).getSubSequenceNumber();  // ... do other
 processing
checkpointer.checkpoint(sequenceNumber, subSequenceNumber);
```

Il cast from to ha `UserRecord` sempre successo `Record` perché l'implementazione utilizza sempre. `UserRecord` Se non è necessario eseguire calcoli aritmetici sui numeri di sequenza, questo approccio non è consigliato.

Durante l'elaborazione dei record utente della KPL, la KCL scrive il numero di sottosequenza in Amazon DynamoDB come un campo extra per ogni riga. Le versioni precedenti della KCL hanno utilizzato `AFTER_SEQUENCE_NUMBER` per recuperare record durante la ripresa dei checkpoint. L'attuale KCL con il supporto KPL utilizza invece `AT_SEQUENCE_NUMBER`. Quando viene recuperato al numero di sequenza sul quale è stato eseguito il checkpoint, viene controllato il numero di sequenza sul quale è stato eseguito il checkpoint e i sottorecord vengono interrotti, in base alle esigenze (potenzialmente tutti se l'ultimo sottorecord è quello sul quale è stato eseguito il checkpoint). I record non aggregati possono essere considerati come record non aggregati con un sottorecord singolo, perciò lo stesso algoritmo funziona sia per i record aggregati sia non aggregati.

Usa direttamente GetRecords

Puoi anche scegliere di non utilizzare la KCL, ma invece richiamare l'operazione API `GetRecords` direttamente per recuperare i record del flusso di dati Kinesis. Per decomprimere questi record recuperati nei record utente della KPL originali, richiama una delle seguenti operazioni statiche in `UserRecord.java`:

```
public static List<Record> deaggregate(List<Record> records)
```

```
public static List<UserRecord> deaggregate(List<UserRecord> records, BigInteger
 startingHashKey, BigInteger endingHashKey)
```

La prima operazione utilizza il valore predefinito `0` (zero) per `startingHashKey` e il valore predefinito `2^128 -1` per `endingHashKey`.

Ognuna di queste operazioni disaggrega l'elenco fornito di record del flusso di dati Kinesis in un elenco di record utente della KPL. Qualsiasi record utente KPL la cui chiave hash esplicita o chiave di partizione non rientra nell'intervallo di `startingHashKey` (incluso) e `endingHashKey`(incluso) viene eliminato dalla lista dei record restituiti.

Usa KPL con Amazon Data Firehose

Se utilizzi la Kinesis Producer Library (KPL) per scrivere i dati su un flusso di dati Kinesis, puoi utilizzare l'aggregazione per abbinare i record che scrivi al flusso di dati Kinesis. Se poi utilizzate quel flusso di dati come fonte per il flusso di distribuzione Firehose, Firehose disaggrega i record prima di consegnarli alla destinazione. Se configuri il flusso di distribuzione per trasformare i dati, Firehose disaggrega i record prima di inviarli a. AWS Lambda Per ulteriori informazioni, consulta Scrittura su Amazon Firehose tramite il flusso di dati Kinesis.

Utilizzate il KPL con lo Schema Registry AWS Glue

Puoi integrare i tuoi flussi di dati Kinesis con lo Schema Registry. AWS Glue Lo AWS Glue Schema Registry ti consente di scoprire, controllare ed evolvere centralmente gli schemi, garantendo al contempo che i dati prodotti siano convalidati continuamente da uno schema registrato. Uno schema definisce la struttura e il formato di un registro di dati. Uno schema è una specifica con versioni per la pubblicazione, il consumo o l'archiviazione dei dati in modo affidabile. Lo AWS Glue Schema Registry consente di migliorare la qualità e la governance end-to-end dei dati all'interno delle applicazioni di streaming. Per ulteriori informazioni, consulta Registro degli schemi di AWS Glue. Uno dei modi per configurare questa integrazione è utilizzare le librerie KPL e Kinesis Client Library (KCL) in Java.

> **Important**
>
> Attualmente, l'integrazione di Kinesis Data AWS Glue Streams e del registro dello schema è supportata solo per i flussi di dati Kinesis che utilizzano produttori KPL implementati in Java. Il supporto multilingue non viene fornito.

Per istruzioni dettagliate su come configurare l'integrazione di Kinesis Data Streams con Schema Registry utilizzando KPL, consulta la sezione «Interazione con i dati utilizzando le librerie KPL/KCL» in Caso d'uso: integrazione di Amazon Kinesis Data Streams con il registro dello schema Glue. AWS

Configura la configurazione del proxy KPL

Per le applicazioni che non possono connettersi direttamente a Internet, tutti i client AWS SDK supportano l'uso di proxy HTTP o HTTPS. In un ambiente aziendale tipico, tutto il traffico di rete in uscita deve passare attraverso server proxy. Se l'applicazione utilizza Kinesis Producer Library (KPL) per raccogliere e inviare dati AWS in un ambiente che utilizza server proxy, l'applicazione richiederà la configurazione del proxy KPL. KPL è una libreria di alto livello basata sull'SDK AWS Kinesis. È suddivisa in un processo nativo e un wrapper. Il processo nativo esegue tutti i processi di elaborazione e invio dei record, mentre il wrapper gestisce il processo nativo e comunica con esso. Per ulteriori informazioni, consulta Implementazione di producer efficienti e affidabili con la Amazon Kinesis Producer Library.

Il wrapper è scritto in Java e il processo nativo è scritto in C++ con l'uso dell'SDK Kinesis. La versione 0.14.7 e successive della KPL ora supportano la configurazione proxy nel wrapper Java, che può passare tutte le configurazioni proxy al processo nativo. Per ulteriori informazioni, consulta https://github.com/awslabs/amazon-kinesis-producer/0.14.7. releases/tag/v

Puoi utilizzare il codice seguente per aggiungere configurazioni proxy alle applicazioni KPL.

```
KinesisProducerConfiguration configuration = new KinesisProducerConfiguration();
// Next 4 lines used to configure proxy
configuration.setProxyHost("10.0.0.0"); // required
configuration.setProxyPort(3128); // default port is set to 443
configuration.setProxyUserName("username"); // no default
configuration.setProxyPassword("password"); // no default

KinesisProducer kinesisProducer = new KinesisProducer(configuration);
```

Politica del ciclo di vita della versione KPL

Questo argomento descrive la politica del ciclo di vita delle versioni per Amazon Kinesis Producer Library (KPL). AWS fornisce regolarmente nuove versioni per le versioni KPL per supportare nuove

funzionalità e miglioramenti, correzioni di bug, patch di sicurezza e aggiornamenti delle dipendenze. Ti consigliamo di rimanere up-to-date con le versioni KPL per stare al passo con le funzionalità più recenti, gli aggiornamenti di sicurezza e le dipendenze sottostanti. Non consigliamo l'uso continuato di una versione KPL non supportata.

Il ciclo di vita delle principali versioni KPL è costituito dalle tre fasi seguenti:

- Disponibilità generale (GA): durante questa fase, la versione principale è completamente supportata. AWS fornisce regolarmente versioni secondarie e patch che includono il supporto per nuove funzionalità o aggiornamenti delle API per Kinesis Data Streams, oltre a correzioni di bug e sicurezza.
- Modalità di manutenzione: AWS limita il rilascio delle versioni delle patch per risolvere solo le correzioni di bug critici e i problemi di sicurezza. La versione principale non riceverà aggiornamenti per le nuove funzionalità o APIs per Kinesis Data Streams.
- E nd-of-support — La versione principale non riceverà più aggiornamenti o versioni. Le versioni pubblicate in precedenza continueranno a essere disponibili tramite gestori di pacchetti pubblici e il codice rimarrà attivo GitHub. L'uso di una versione raggiunta end-of-support viene effettuato a discrezione dell'utente. Ti consigliamo di eseguire l'aggiornamento alla versione principale più recente.

Versione principale	Fase attuale	Data di rilascio	Data della modalità di manutenzione	End-of-support data
KPL 0.x	modalità di manutenzione	2015-06-02	2025-04-17	2026-01-30
KPL 1.x	Disponibilità generale	2024-12-15	--	--

Sviluppa i produttori utilizzando l'API Amazon Kinesis Data Streams con AWS SDK per Java

Puoi sviluppare produttori utilizzando l'API Amazon Kinesis Data Streams AWS con l'SDK for Java. Se sei nuovo per il flusso di dati Kinesis, ti consigliamo di familiarizzare prima con i concetti chiave e

la terminologia introdotti in Cos'è Amazon Kinesis Data Streams? e Utilizza il AWS CLI per eseguire operazioni di Amazon Kinesis Data Streams.

Questi esempi parlano dell'API del flusso di dati Kinesis e utilizzano l'SDK AWS per Java per aggiungere (inserire) dati a un flusso. Tuttavia, per la maggior parte dei casi d'uso, è preferibile fare riferimento alla libreria KPL del flusso di dati Kinesis. Per ulteriori informazioni, consulta Sviluppa produttori utilizzando Amazon Kinesis Producer Library (KPL).

Il codice di esempio Java in questo capitolo illustra come eseguire le operazioni API del flusso di dati Kinesis di base ed è suddiviso logicamente in tipo di operazioni. Questi esempi non rappresentano codici pronti per la produzione, poiché non eseguono un controllo per tutte le possibili eccezioni o spiegano tutte le possibili considerazioni relative alle prestazioni e alla sicurezza. Inoltre, è possibile chiamare l'API di SDK utilizzando altri linguaggi di programmazione. Per ulteriori informazioni su tutto ciò che è disponibile AWS SDKs, consulta Inizia a sviluppare con Amazon Web Services.

Ogni attività ha dei requisiti preliminari; ad esempio, non è possibile aggiungere dati a un flusso se non si è creato un flusso, che a sua volta presuppone la creazione di un client. Per ulteriori informazioni, consulta Crea e gestisci flussi di dati Kinesis.

Argomenti

- Aggiungi dati a uno stream
- Interagisci con i dati utilizzando lo AWS Glue Schema Registry

Aggiungi dati a uno stream

Una volta che il flusso è stato creato, è possibile aggiungere dati sotto forma di record. Un record è una struttura di dati che contiene i dati da elaborare sotto forma di un blob di dati. Una volta che i dati sono stati archiviati nel record, il flusso di dati Kinesis non li ispeziona, interpreta o modifica in alcun modo. Ogni record, inoltre, è associato a un numero di sequenza e a una chiave di partizione.

Sono disponibili due diverse operazioni nell'API del flusso di dati Kinesis che consentono di aggiungere dati a un flusso, `PutRecords` e `PutRecord`. L'operazione `PutRecords` invia più record al flusso in ogni richiesta HTTP e l'operazione unica `PutRecord` invia i record al flusso uno alla volta (ogni record necessita di una richiesta HTTP separata). È preferibile utilizzare l'operazione `PutRecords` per la maggior parte delle applicazioni perché questa raggiunge un maggiore throughput per producer dati. Per ulteriori informazioni su ciascuna di queste operazioni, vedere le sottosezioni separate di seguito.

Argomenti

- Aggiungi più record con PutRecords
- Aggiungi un singolo record con PutRecord

È necessario tenere sempre a mente che mentre l'applicazione origine aggiunge dati al flusso utilizzando l'API del flusso di dati Kinesis è probabile che ci siano una o più applicazioni consumer che elaborano simultaneamente i dati dal flusso. Per informazioni su come i consumer ottengono i dati utilizzando l'API del flusso di dati Kinesis, consulta Ottieni dati da un flusso.

> **Important**
>
> Modifica il periodo di conservazione dei dati

Aggiungi più record con PutRecords

L'operazione `PutRecords` invia più record a SDK in una singola richiesta. Utilizzando `PutRecords`, i producer possono ottenere maggiori livelli di velocità di trasmissione effettiva durante l'invio di dati al loro flusso di dati Kinesis. Ogni richiesta `PutRecords` può supportare fino a 500 record. Ciascun record nella richiesta può avere una dimensione massima pari a 1 MB, con un limite a 5 MB, per l'intera richiesta, incluse le chiavi di partizione. Come con l'operazione singola `PutRecord` descritta di seguito, `PutRecords` utilizza numeri di sequenza e chiavi di partizione. Tuttavia, il parametro `PutRecord SequenceNumberForOrdering` non è incluso in una chiamata `PutRecords`. L'operazione `PutRecords` tenta di elaborare tutti i record secondo l'ordine naturale della richiesta.

Ogni record di dati dispone di un numero di sequenza univoco. Il numero di sequenza viene assegnato dal flusso di dati Kinesis dopo aver chiamato `client.putRecords` per aggiungere i record di dati al flusso. I numeri di sequenza per la stessa chiave di partizione di solito aumentano nel tempo; più è lungo il periodo di tempo tra le richieste `PutRecords`, più aumentano i numeri di sequenza.

> **Note**
>
> i numeri di sequenza non possono essere utilizzati come indici per set di dati all'interno dello stesso flusso. Per separare i set di dati logicamente, utilizza le chiavi di partizione o crea un flusso separato per ogni set di dati.

Una richiesta `PutRecords` può includere record con diverse chiavi di partizione. L'ambito della richiesta è un flusso; ogni richiesta può includere qualsiasi combinazione di chiavi di partizione e record fino al raggiungimento dei limiti della richiesta. Le richieste effettuate con molte diverse chiavi di partizione a flussi con diversi shard sono in genere più veloci delle richieste con un piccolo numero di chiavi di partizione su un esiguo numero di shard. Il numero di chiavi di partizione deve essere molto più grande del numero di shard per ridurre la latenza e massimizzare il throughput.

PutRecords Esempio

Il codice seguente crea 100 record di dati con chiavi di partizione sequenziali e li mette in un flusso chiamato `DataStream`.

```
       AmazonKinesisClientBuilder clientBuilder =
AmazonKinesisClientBuilder.standard();

       clientBuilder.setRegion(regionName);
       clientBuilder.setCredentials(credentialsProvider);
       clientBuilder.setClientConfiguration(config);

       AmazonKinesis kinesisClient = clientBuilder.build();

       PutRecordsRequest putRecordsRequest  = new PutRecordsRequest();
       putRecordsRequest.setStreamName(streamName);
       List <PutRecordsRequestEntry> putRecordsRequestEntryList  = new ArrayList<>();
       for (int i = 0; i < 100; i++) {
           PutRecordsRequestEntry putRecordsRequestEntry  = new
PutRecordsRequestEntry();

putRecordsRequestEntry.setData(ByteBuffer.wrap(String.valueOf(i).getBytes()));
           putRecordsRequestEntry.setPartitionKey(String.format("partitionKey-%d",
i));
           putRecordsRequestEntryList.add(putRecordsRequestEntry);
       }

       putRecordsRequest.setRecords(putRecordsRequestEntryList);
       PutRecordsResult putRecordsResult  =
kinesisClient.putRecords(putRecordsRequest);
       System.out.println("Put Result" + putRecordsResult);
```

La risposta `PutRecords` include una matrice di risposta `Records`. Ogni record nella matrice di risposta è direttamente correlata a un record nella matrice di richiesta in base all'ordine naturale,

dall'alto al basso della richiesta e della risposta. La matrice di risposta `Records` include sempre lo stesso numero di record della matrice di richiesta.

Gestisci gli errori durante l'utilizzo PutRecords

Per impostazione predefinita, l'errore nei singoli record all'interno di una richiesta non blocca l'elaborazione dei record successivi in una richiesta `PutRecords`. Ciò significa che una matrice `Records` di risposta include record elaborati e non. È necessario rilevare i record non elaborati correttamente e includerli nella chiamata successiva.

I record elaborati correttamente includono valori `SequenceNumber` e `ShardID`, mentre quelli non elaborati correttamente includono valori `ErrorCode` e `ErrorMessage`. Il parametro `ErrorCode` riflette il tipo di errore e può coincidere con uno dei seguenti valori: `ProvisionedThroughputExceededException` o `InternalFailure`. `ErrorMessage` offre informazioni più dettagliate sull'eccezione `ProvisionedThroughputExceededException`, incluso l'ID dell'account, il nome del flusso e l'ID dello shard del record oggetto di throttling. L'esempio seguente ha tre record in una richiesta `PutRecords`. Il secondo record ha generato un errore che si riflette nella risposta.

Example PutRecords Sintassi della richiesta

```
{
    "Records": [
        {
     "Data": "XzxkYXRhPl8w",
     "PartitionKey": "partitionKey1"
        },
        {
     "Data": "AbceddeRFfg12asd",
     "PartitionKey": "partitionKey1"
        },
        {
     "Data": "KFpcd98*7nd1",
     "PartitionKey": "partitionKey3"
        }
    ],
    "StreamName": "myStream"
}
```

Example PutRecords Sintassi di risposta

```
{
    "FailedRecordCount”: 1,
    "Records": [
        {
     "SequenceNumber": "21269319989900637946712965403778482371",
     "ShardId": "shardId-000000000001"

        },
        {
     “ErrorCode":”ProvisionedThroughputExceededException”,
     “ErrorMessage": "Rate exceeded for shard shardId-000000000001 in stream
 exampleStreamName under account 111111111111."

        },
        {
     "SequenceNumber": "21269319989999637946712965403778482985",
     "ShardId": "shardId-000000000002"
        }
    ]
}
```

I record non elaborati correttamente possono essere inclusi nelle richieste `PutRecords` successive. In primo luogo, controlla il parametro `FailedRecordCount` in `putRecordsResult` per confermare la presenza di record non elaborati. In questo caso, ogni `putRecordsEntry` che ha un `ErrorCode` che non è `null` deve essere aggiunta a una richiesta successiva. Per un esempio di questo tipo di gestore, fai riferimento al seguente codice.

Example PutRecords gestore degli errori

```
PutRecordsRequest putRecordsRequest = new PutRecordsRequest();
putRecordsRequest.setStreamName(myStreamName);
List<PutRecordsRequestEntry> putRecordsRequestEntryList = new ArrayList<>();
for (int j = 0; j < 100; j++) {
    PutRecordsRequestEntry putRecordsRequestEntry = new PutRecordsRequestEntry();
    putRecordsRequestEntry.setData(ByteBuffer.wrap(String.valueOf(j).getBytes()));
    putRecordsRequestEntry.setPartitionKey(String.format("partitionKey-%d", j));
    putRecordsRequestEntryList.add(putRecordsRequestEntry);
}

putRecordsRequest.setRecords(putRecordsRequestEntryList);
```

```
PutRecordsResult putRecordsResult = amazonKinesisClient.putRecords(putRecordsRequest);

while (putRecordsResult.getFailedRecordCount() > 0) {
    final List<PutRecordsRequestEntry> failedRecordsList = new ArrayList<>();
    final List<PutRecordsResultEntry> putRecordsResultEntryList =
 putRecordsResult.getRecords();
    for (int i = 0; i < putRecordsResultEntryList.size(); i++) {
        final PutRecordsRequestEntry putRecordRequestEntry =
 putRecordsRequestEntryList.get(i);
        final PutRecordsResultEntry putRecordsResultEntry =
 putRecordsResultEntryList.get(i);
        if (putRecordsResultEntry.getErrorCode() != null) {
            failedRecordsList.add(putRecordRequestEntry);
        }
    }
    putRecordsRequestEntryList = failedRecordsList;
    putRecordsRequest.setRecords(putRecordsRequestEntryList);
    putRecordsResult = amazonKinesisClient.putRecords(putRecordsRequest);
}
```

Aggiungi un singolo record con PutRecord

Ogni chiamata a `PutRecord` opera su un singolo record. È preferibile usare l'operazione `PutRecords` descritta in Aggiungi più record con PutRecords a meno che l'applicazione in uso non abbia specificamente bisogno di inviare sempre record singoli per ogni richiesta, o nel caso in cui per altre ragioni l'operazione `PutRecords` non può essere utilizzata.

Ogni record di dati dispone di un numero di sequenza univoco. Il numero di sequenza viene assegnato dal flusso di dati Kinesis dopo aver chiamato `client.putRecord` per aggiungere i record di dati al flusso. I numeri di sequenza per la stessa chiave di partizione di solito aumentano nel tempo; più è lungo il periodo di tempo tra le richieste `PutRecord`, più aumentano i numeri di sequenza.

Quando le immissioni si verificano in rapida successione, non è garantito che i numeri di sequenza restituiti aumentino perché le operazioni di introduzione appaiono sostanzialmente come simultanee al flusso di dati Kinesis. Per garantire un aumento rigoroso dei numeri di sequenza per la stessa chiave di partizione, utilizzare il parametro `SequenceNumberForOrdering`, come mostrato nel codice di esempio PutRecord Esempio.

Che venga utilizzato o meno il `SequenceNumberForOrdering`, i record che il flusso di dati Kinesis riceve tramite una chiamata `GetRecords` sono rigorosamente ordinati in base al numero di sequenza.

> **Note**
>
> i numeri di sequenza non possono essere utilizzati come indici per set di dati all'interno dello stesso flusso. Per separare i set di dati logicamente, utilizza le chiavi di partizione o crea un flusso separato per ogni set di dati.

Per raggruppare i dati nel flusso viene utilizzata una chiave di partizione. Un record di dati viene assegnato a un shard all'interno del flusso in base alla sua chiave di partizione. Nello specifico, il flusso di dati Kinesis utilizza la chiave di partizione come input a una funzione hash che mappi la chiave di partizione (e i dati associati) a una determinata partizione.

Come conseguenza di questo meccanismo di hashing, tutti i record di dati con la stessa chiave di partizione vengono mappati allo stesso shard all'interno del flusso. Tuttavia, se il numero di chiavi di partizione supera il numero di shard, alcuni shard necessariamente conterranno record con chiavi di partizione diverse. Dal punto di vista di progettazione, per assicurare che tutti gli shard siano ben utilizzati, il numero di shard (specificato dal metodo `setShardCount` di `CreateStreamRequest`) deve essere notevolmente inferiore al numero di chiavi di partizione univoche e la quantità di flusso di dati verso un'unica chiave di partizione deve essere notevolmente inferiore alla capacità dello shard.

PutRecord Esempio

Il codice seguente crea dieci record di dati, distribuiti su due chiavi di partizione, e li mette in un flusso chiamato `myStreamName`.

```
for (int j = 0; j < 10; j++)
{
  PutRecordRequest putRecordRequest = new PutRecordRequest();
  putRecordRequest.setStreamName( myStreamName );
  putRecordRequest.setData(ByteBuffer.wrap( String.format( "testData-%d",
 j ).getBytes() ));
  putRecordRequest.setPartitionKey( String.format( "partitionKey-%d", j/5 ));
  putRecordRequest.setSequenceNumberForOrdering( sequenceNumberOfPreviousRecord );
  PutRecordResult putRecordResult = client.putRecord( putRecordRequest );
  sequenceNumberOfPreviousRecord = putRecordResult.getSequenceNumber();
}
```

Il codice di esempio precedente usa `setSequenceNumberForOrdering` per garantire un ordinamento in aumento rigoroso all'interno di ciascuna chiave di partizione. Per usare questo parametro efficacemente, impostare il `SequenceNumberForOrdering` del record corrente (record n) sul numero di sequenza del record precedente (record n-1). Per ottenere il numero di sequenza di un record che è stato aggiunto al flusso, chiamare `getSequenceNumber` sul risultato di `putRecord`.

Il parametro `SequenceNumberForOrdering` garantisce garantisce un numero di sequenza strettamente crescente per la stessa chiave di partizione. `SequenceNumberForOrdering` non fornisce ordinazione di record su più chiavi di partizione.

Interagisci con i dati utilizzando lo AWS Glue Schema Registry

Puoi integrare i tuoi flussi di dati Kinesis con lo Schema Registry. AWS Glue Lo AWS Glue Schema Registry ti consente di scoprire, controllare ed evolvere centralmente gli schemi, garantendo al contempo che i dati prodotti siano convalidati continuamente da uno schema registrato. Uno schema definisce la struttura e il formato di un registro di dati. Uno schema è una specifica con versioni per la pubblicazione, il consumo o l'archiviazione dei dati in modo affidabile. Lo AWS Glue Schema Registry consente di migliorare la qualità e la governance end-to-end dei dati all'interno delle applicazioni di streaming. Per ulteriori informazioni, consulta Registro degli schemi di AWS Glue. Uno dei modi per configurare questa integrazione è tramite `PutRecord` Kinesis APIs Data Streams disponibile AWS in Java SDK. `PutRecords`

Per istruzioni dettagliate su come configurare l'integrazione di Kinesis Data Streams con il registro dello schema PutRecords utilizzando Kinesis Data Streams, consulta PutRecord la sezione «Interazione con i dati utilizzando APIs Kinesis Data Streams» in Caso d'uso: integrazione di Amazon Kinesis Data APIs Streams con il registro dello schema Glue. AWS

Scrivi su Amazon Kinesis Data Streams utilizzando Kinesis Agent

L'agente Kinesis è un'applicazione software Java autonoma che offre un modo semplice per raccogliere e inviare dati al flusso di dati Kinesis. L'agente controlla costantemente un set di file e invia i nuovi dati al flusso. L'agente gestisce la rotazione dei file, i checkpoint e i tentativi in caso di errori. Fornisce tutti i dati in un modo affidabile, tempestivo e semplice. Inoltre, emette i CloudWatch parametri di Amazon per aiutarti a monitorare e risolvere meglio il processo di streaming.

Come impostazione predefinita, i record vengono analizzati da ciascun file in base alla nuova riga di caratteri (`'\n'`). Tuttavia, l'agente può anche essere configurato per analizzare record a più righe (consulta Specificare le impostazioni di configurazione dell'agente).

Puoi installare l'agente su ambienti server basati su Linux, come server Web, server di log e server di database. Dopo aver installato l'agente, configurarlo specificando i file da monitorare e il flusso per i dati. Dopo che l'agente è configurato, raccoglie i dati dal file in modo durevole e li invia al flusso in modo affidabile.

Argomenti

- Completa i prerequisiti per Kinesis Agent
- Scarica e installa l'agente
- Configura e avvia l'agente
- Specificare le impostazioni di configurazione dell'agente
- Monitora più directory di file e scrivi su più flussi
- Usa l'agente per pre-elaborare i dati
- Usa i comandi CLI dell'agente
- Domande frequenti

Completa i prerequisiti per Kinesis Agent

- Il sistema operativo deve essere Amazon Linux AMI con versione 2015.09 o successiva o Red Hat Enterprise Linux versione 7 o successiva.
- Se utilizzi Amazon EC2 per eseguire il tuo agente, avvia l' EC2 istanza.
- Gestisci AWS le tue credenziali utilizzando uno dei seguenti metodi:
  - Specificate un ruolo IAM all'avvio dell' EC2 istanza.
  - Specificate AWS le credenziali quando configurate l'agente (vedi awsAccessKeyID e awsSecretAccesschiave).
  - Modifica `/etc/sysconfig/aws-kinesis-agent` per specificare la regione e le chiavi di AWS accesso.
  - Se l' EC2 istanza si trova in un AWS account diverso, crea un ruolo IAM per fornire l'accesso al servizio Kinesis Data Streams e specifica quel ruolo quando configuri l'agente (vedi assumeRoleExternal AssumeroLearn e Id). Utilizza uno dei metodi precedenti per specificare le AWS credenziali di un utente nell'altro account che ha il permesso di assumere questo ruolo.
- Il ruolo o AWS le credenziali IAM specificati devono disporre dell'autorizzazione per eseguire l'operazione Kinesis Data PutRecordsStreams affinché l'agente possa inviare dati al tuo stream. Se abiliti il CloudWatch monitoraggio per l'agente, è necessaria anche l'autorizzazione a eseguire

l' CloudWatch PutMetricDataoperazione. Per ulteriori informazioni, vedere Controllo dell'accesso alle risorse del flusso di dati Amazon Kinesis tramite IAMMonitoraggio dell'integrità dell'agente del flusso di dati Kinesis con Amazon CloudWatch, e Controllo degli CloudWatch accessi.

Scarica e installa l'agente

Innanzitutto connettiti all'istanza, Per ulteriori informazioni, consulta Connect to Your Instance nella Amazon EC2 User Guide. Se hai problemi di connessione, consulta Risoluzione dei problemi di connessione alla tua istanza nella Amazon EC2 User Guide.

Per configurare l'agente utilizzando l'AMI Amazon Linux

Utilizzare il seguente comando per scaricare e installare l'agente:

```
sudo yum install -y aws-kinesis-agent
```

Per configurare l'agente utilizzando Red Hat Enterprise Linux

Utilizzare il seguente comando per scaricare e installare l'agente:

```
sudo yum install -y https://s3.amazonaws.com/streaming-data-agent/aws-kinesis-agent-
latest.amzn2.noarch.rpm
```

Per configurare l'agente utilizzando GitHub

1. Scarica l'agente da awlabs/ amazon-kinesis-agent.
2. Installare l'agente spostandosi nella directory di download ed eseguendo il comando seguente:

   ```
   sudo ./setup --install
   ```

Configurazione dell'agente in un container Docker

L'agente Kinesis può essere eseguito anche in un container tramite la base container amazonlinux. Utilizza il seguente Dockerfile e poi esegui `docker build`.

```
FROM amazonlinux
```

```
RUN yum install -y aws-kinesis-agent which findutils
COPY agent.json /etc/aws-kinesis/agent.json

CMD ["start-aws-kinesis-agent"]
```

Configura e avvia l'agente

Configurazione e avvio dell'agente

1. Aprire e modificare il file di configurazione (come superutente se vengono utilizzate le autorizzazioni predefinite di accesso al file): `/etc/aws-kinesis/agent.json`

   In questo file di configurazione, specificare i file (`"filePattern"`) dai quali l'agente raccoglie i dati e il nome del flusso (`"kinesisStream"`) al quale l'agente invia i dati. Si noti che il nome del file è un modello e l'agente riconosce le rotazioni dei file. Puoi ruotare i file o creare nuovi file non più di una volta al secondo. L'agente usa il timestamp di creazione di file per stabilire quale file tracciare e inserire nel flusso; la creazione di nuovi file o la rotazione di file più frequentemente di una volta al secondo non consente all'agente di distinguerli in modo corretto.

   ```
   {
      "flows": [
           {
               "filePattern": "/tmp/app.log*",
               "kinesisStream": "yourkinesisstream"
           }
      ]
   }
   ```

2. Avvia l'agente manualmente:

   ```
   sudo service aws-kinesis-agent start
   ```

3. (Facoltativo) Configurare l'agente per iniziare l'avvio del sistema:

   ```
   sudo chkconfig aws-kinesis-agent on
   ```

L'agente è ora in esecuzione come servizio di sistema in background. Controlla costantemente i file specificati e invia i dati al flusso specificato. L'attività dell'agente viene registrata in `/var/log/aws-kinesis-agent/aws-kinesis-agent.log`.

Specificare le impostazioni di configurazione dell'agente

L'agente supporta le due impostazioni di configurazione obbligatorie `filePattern` e `kinesisStream`, più impostazioni di configurazione opzionali per funzionalità aggiuntive. Puoi specificare la configurazione obbligatoria e opzionale in `/etc/aws-kinesis/agent.json`.

Quando modifichi il file di configurazione, devi arrestare e avviare l'agente, utilizzando i comandi seguenti:

```
sudo service aws-kinesis-agent stop
sudo service aws-kinesis-agent start
```

In alternativa, potresti utilizzare il comando seguente:

```
sudo service aws-kinesis-agent restart
```

Seguono le impostazioni di configurazione generali.

Impostazione di configurazione	Descrizione
`assumeRoleARN`	L'ARN del ruolo da assegnare all'utente. Per ulteriori informazioni, consulta Delegare l'accesso tra AWS account utilizzando i ruoli IAM nella Guida per l'utente IAM.
`assumeRoleExternalId`	Si è verificato un identificatore opzionale che determina chi può assumere il ruolo. Per ulteriori informazioni, consulta Come utilizzare un ID esterno nella Guida per l'utente di IAM.
`awsAccessKeyId`	AWS ID della chiave di accesso che sostituisce le credenziali predefini te. Questa impostazione ha la precedenza su tutti gli altri provider di credenziali.
`awsSecretAccessKey`	AWS chiave segreta che sostituisce le credenziali predefinite. Questa impostazione ha la precedenza su tutti gli altri provider di credenziali.
`cloudwatch.emitMetrics`	Consente all'agente di emettere metriche su CloudWatch if set (true). Impostazione predefinita: true

Impostazione di configurazione	Descrizione
`cloudwatch.endpoint`	L'endpoint regionale per. CloudWatch Impostazione predefinita: `monitoring.us-east-1.amazonaws.com`
`kinesis.endpoint`	L'endpoint regionale per il flusso di dati Kinesis. Impostazione predefinita: `kinesis.us-east-1.amazonaws.com`

Seguono le impostazioni di configurazione del flusso.

Impostazione di configurazione	Descrizione
`dataProcessingOptions`	L'elenco delle opzioni di elaborazione applicate a ciascun record analizzato prima dell'invio al flusso. Le opzioni di elaborazione vengono eseguite nell'ordine specificato. Per ulteriori informazioni, consulta Usa l'agente per pre-elaborare i dati.
`kinesisStream`	[Obbligatorio] Il nome del flusso.
`filePattern`	[Obbligatorio] La directory e il modello di file che devono corrispondere per essere rilevati dall'agente. Per tutti i file che corrispondono a questo modello, le autorizzazioni di lettura devono essere concesse a `aws-kinesis-agent-user` . Per la directory contenente i file, le autorizzazioni di lettura ed esecuzione devono essere concesse `aws-kinesis-agent-user` .
`initialPosition`	La posizione iniziale dalla quale è iniziata l'analisi del file. I valori validi sono `START_OF_FILE` e `END_OF_FILE` . Impostazione predefinita: `END_OF_FILE`
`maxBufferAgeMillis`	Il tempo massimo, in millisecondi, in cui l'agente effettua il buffer dati prima di inviarli al flusso.

Impostazione di configurazione	Descrizione
	Intervallo valore: da 1.000 a 900.000 (da 1 secondo a 15 minuti) Impostazione predefinita: 60.000 (1 minuto)
`maxBufferSizeBytes`	Le dimensioni massime, in byte, in cui l'agente effettua il buffer dati prima di inviarli al flusso. Intervallo valore: da 1 a 4.194.304 (4 MB) Impostazione predefinita: 4.194.304 (4 MB)
`maxBufferSizeRecords`	Il numero massimo di record in cui l'agente effettua il buffer dati prima di inviarli al flusso. Intervallo valore: da 1 a 500 Impostazione predefinita: 500
`minTimeBetweenFilePollsMillis`	L'intervallo di tempo, in millisecondi, in cui l'agente esegue il polling e analizza i dati nuovi nei file monitorati. Intervallo valore: 1 o più Impostazione predefinita: 100
`multiLineStartPattern`	Il modello per identificare l'inizio di un record. Un record è composto da una riga corrispondente al modello e da tutte le righe successive non corrispondenti al modello. I valori validi sono espressioni regolari. Come impostazione predefinita, ogni nuova riga nei file di log viene analizzata come un record.
`partitionKeyOption`	Il metodo per generare la chiave di partizione. I valori validi sono `RANDOM` (integer generato casualmente) e `DETERMINISTIC` (un valore hash calcolato dai dati). Impostazione predefinita: `RANDOM`

Impostazione di configurazione	Descrizione
`skipHeaderLines`	Il numero di righe necessarie perché l'agente salti l'analisi all'inizio dei file monitorati. Intervallo valore: 0 o più Impostazione predefinita: 0 (zero)
`truncatedRecordTerminator`	La stringa che l'agente utilizza per troncare un record analizzato quando le dimensioni del record eccedono il limite di dimensioni del record . (1.000 KB) Impostazione predefinita: `'\n'` (nuova riga)

Monitora più directory di file e scrivi su più flussi

Specificando più impostazioni di configurazione del flusso, puoi configurare l'agente in modo che monitori più directory di file e invii dati a più flussi. Nel seguente esempio di configurazione, l'agente monitora due directory di file e invia i dati rispettivamente a un flusso Kinesis e a un flusso di distribuzione Firehose. Tieni presente che puoi specificare endpoint diversi per Kinesis Data Streams e Firehose in modo che Kinesis stream e Firehose non debbano necessariamente trovarsi nella stessa regione.

```
{
    "cloudwatch.emitMetrics": true,
    "kinesis.endpoint": "https://your/kinesis/endpoint",
    "firehose.endpoint": "https://your/firehose/endpoint",
    "flows": [
        {
            "filePattern": "/tmp/app1.log*",
            "kinesisStream": "yourkinesisstream"
        },
        {
            "filePattern": "/tmp/app2.log*",
            "deliveryStream": "yourfirehosedeliverystream"
        }
    ]
```

```
}
```

Per informazioni più dettagliate sull'utilizzo dell'agente con Firehose, consulta Writing to Amazon Data Firehose with Kinesis Agent.

Usa l'agente per pre-elaborare i dati

L'agente può preelaborare i record analizzati dai file monitorati prima di inviarli al flusso. È possibile abilitare questa funzionalità aggiungendo le impostazioni di configurazione `dataProcessingOptions` al flusso di file. Una o più opzioni di elaborazione possono essere aggiunte e saranno eseguite nell'ordine specificato.

L'agente supporta le seguenti opzioni di elaborazione elencate. Poiché l'agente è open source, è possibile sviluppare ulteriormente e ampliare le opzioni di elaborazione. Puoi scaricare l'agente dall'agente Kinesis.

Opzioni di elaborazione

`SINGLELINE`

Converte un record a più righe in un record a riga singola rimuovendo i caratteri di nuova riga, gli spazi iniziali e finali.

```
{
    "optionName": "SINGLELINE"
}
```

`CSVTOJSON`

Converte un record da un formato delimitatore separato a un formato JSON.

```
{
    "optionName": "CSVTOJSON",
    "customFieldNames": [ "field1", "field2", ... ],
    "delimiter": "yourdelimiter"
}
```

`customFieldNames`

[Obbligatorio] I nomi di campo utilizzati come chiavi in ciascuna coppia chiave-valore JSON. Ad esempio, se specifichi `["f1", "f2"]`, il record "v1, v2" sarà convertito in `{"f1":"v1","f2":"v2"}`.

`delimiter`

La stringa utilizzata come delimitatore nel record. L'impostazione predefinita è una virgola (,).

`LOGTOJSON`

Converte un record da un formato log a un formato JSON. I formati di log supportati sono Apache Common Log, Apache Combined Log, Apache Error Log e RFC3164 Syslog.

```
{
    "optionName": "LOGTOJSON",
    "logFormat": "logformat",
    "matchPattern": "yourregexpattern",
    "customFieldNames": [ "field1", "field2", … ]
}
```

`logFormat`

[Obbligatorio] Il formato di inserimento dei log. I seguenti sono i valori possibili:

- `COMMONAPACHELOG` - Il formato Apache Common Log. Ogni voce di log ha il seguente modello come impostazione predefinita: "`%{host} %{ident} %{authuser} [%{datetime}] \"%{request}\" %{response} %{bytes}`".
- `COMBINEDAPACHELOG`: il formato Apache Combined Log. Ogni voce di log ha il seguente modello come impostazione predefinita: "`%{host} %{ident} %{authuser} [%{datetime}] \"%{request}\" %{response} %{bytes} %{referrer} %{agent}`".
- `APACHEERRORLOG`: il formato Apache Error Log. Ogni voce di log ha il seguente modello come impostazione predefinita: "`[%{timestamp}] [%{module}:%{severity}] [pid %{processid}:tid %{threadid}] [client: %{client}] %{message}`".
- `SYSLOG`— Il formato RFC3164 Syslog. Ogni voce di log ha il seguente modello come impostazione predefinita: "`%{timestamp} %{hostname} %{program} [%{processid}]: %{message}`".

`matchPattern`

Il modello di espressione regolare utilizzato per estrarre valori dalle voci di log. Questa impostazione viene utilizzata se la tua voce di log non è in uno dei formati di log predefiniti. Se questa impostazione viene utilizzata, devi specificare anche `customFieldNames`.

`customFieldNames`

I nomi di campo obbligatori utilizzati come chiavi in ciascuna coppia chiave-valore JSON. Puoi utilizzare questa impostazione per definire i nomi dei campi per i valori estratti da `matchPattern` oppure sovrascrivere i nomi dei campi predefiniti dei formati di log predefiniti.

Example : configurazione LOGTOJSON

Questo è un esempio di una configurazione `LOGTOJSON` per una voce Apache Common Log convertita in formato JSON:

```
{
    "optionName": "LOGTOJSON",
    "logFormat": "COMMONAPACHELOG"
}
```

Prima della conversione:

```
64.242.88.10 - - [07/Mar/2004:16:10:02 -0800] "GET /mailman/listinfo/hsdivision
 HTTP/1.1" 200 6291
```

Dopo la conversione:

```
{"host":"64.242.88.10","ident":null,"authuser":null,"datetime":"07/
Mar/2004:16:10:02 -0800","request":"GET /mailman/listinfo/hsdivision
 HTTP/1.1","response":"200","bytes":"6291"}
```

Example : configurazione LOGTOJSON con campi personalizzati

Ecco un altro esempio di configurazione `LOGTOJSON`:

```
{
    "optionName": "LOGTOJSON",
    "logFormat": "COMMONAPACHELOG",
    "customFieldNames": ["f1", "f2", "f3", "f4", "f5", "f6", "f7"]
}
```

Con questa impostazione di configurazione, la stessa voce Apache Common Log dall'esempio precedente viene convertita in formato JSON come segue:

```
{"f1":"64.242.88.10","f2":null,"f3":null,"f4":"07/Mar/2004:16:10:02 -0800","f5":"GET /
mailman/listinfo/hsdivision HTTP/1.1","f6":"200","f7":"6291"}
```

Example : convertire la voce Apache Common Log

La seguente configurazione di flusso converte una voce Apache Common Log in record a riga singola in formato JSON:

```
{
    "flows": [
        {
            "filePattern": "/tmp/app.log*",
            "kinesisStream": "my-stream",
            "dataProcessingOptions": [
                {
                    "optionName": "LOGTOJSON",
                    "logFormat": "COMMONAPACHELOG"
                }
            ]
        }
    ]
}
```

Example : convertire record a più righe

La seguente configurazione del flusso analizza i record a più righe la cui prima riga inizia con "`[SEQUENCE=`". Ogni record viene convertito in un record a riga singola. Quindi, i valori vengono estratti dal record in base a un delimitatore di schede. I valori estratti sono mappati in valori `customFieldNames` specificati per formare un record a riga singola in formato JSON.

```
{
    "flows": [
        {
            "filePattern": "/tmp/app.log*",
            "kinesisStream": "my-stream",
            "multiLineStartPattern": "\\[SEQUENCE=",
            "dataProcessingOptions": [
                {
                    "optionName": "SINGLELINE"
                },
                {
```

```
                    "optionName": "CSVTOJSON",
                    "customFieldNames": [ "field1", "field2", "field3" ],
                    "delimiter": "\\t"
                }
            ]
        }
    ]
}
```

Example : configurazione LOGTOJSON con modello corrispondente

Questo è un esempio di una configurazione LOGTOJSON per una voce Apache Common Log convertita in formato JSON, con l'ultimo campo (byte) omesso:

```
{
    "optionName": "LOGTOJSON",
    "logFormat": "COMMONAPACHELOG",
    "matchPattern": "^([\\d.]+) (\\S+) (\\S+) \\[([\\w:/]+\\s[+\\-]\\d{4})\\] \"(.
+?)\" (\\d{3})",
    "customFieldNames": ["host", "ident", "authuser", "datetime", "request",
 "response"]
}
```

Prima della conversione:

```
123.45.67.89 - - [27/Oct/2000:09:27:09 -0400] "GET /java/javaResources.html HTTP/1.0"
 200
```

Dopo la conversione:

```
{"host":"123.45.67.89","ident":null,"authuser":null,"datetime":"27/Oct/2000:09:27:09
 -0400","request":"GET /java/javaResources.html HTTP/1.0","response":"200"}
```

Usa i comandi CLI dell'agente

Avviare automaticamente l'agente all'avvio del sistema:

```
sudo chkconfig aws-kinesis-agent on
```

Controlla lo stato dell'agente:

```
sudo service aws-kinesis-agent status
```

Interrompi l'agente:

```
sudo service aws-kinesis-agent stop
```

Leggi il file di log dell'agente da questa posizione:

```
/var/log/aws-kinesis-agent/aws-kinesis-agent.log
```

Disinstalla l'agente:

```
sudo yum remove aws-kinesis-agent
```

Domande frequenti

Esiste un agente Kinesis per Windows?

L'agente Kinesis per Windows è un software diverso dall'agente Kinesis per piattaforme Linux.

Perché l'agente Kinesis rallenta e/o **`RecordSendErrors`** aumenta?

Di solito ciò è dovuto alla limitazione di Kinesis. Controlla la `WriteProvisionedThroughputExceeded` metrica per Kinesis Data Streams o la `ThrottledRecords` metrica per Firehose Delivery Streams. Qualsiasi aumento rispetto a 0 di questi parametri indica che è necessario aumentare i limiti dei flussi. Per ulteriori informazioni, consulta Limiti del flusso di dati Kinesis e Flussi di consegna di Amazon Firehose.

Una volta esclusa la limitazione, verifica se Kinesis Agent è configurato in modo da monitorare grandi quantità di file di piccole dimensioni. Si verifica un ritardo nel momento in cui Kinesis Agent esegue il tail di un nuovo file, quindi Kinesis Agent dovrebbe eseguire la coda su una piccola quantità di file più grandi. Prova a consolidare i tuoi file di log in file più grandi.

Perché ricevo delle **`java.lang.OutOfMemoryError`** eccezioni?

Kinesis Agent non dispone di memoria sufficiente per gestire il carico di lavoro corrente. Prova ad aumentare, `JAVA_START_HEAP` inserire `/usr/bin/start-aws-kinesis-agent` e `JAVA_MAX_HEAP` riavviare l'agente.

Perché **`IllegalStateException : connection pool shut down`** ricevo delle eccezioni?

Kinesis Agent non dispone di connessioni sufficienti per gestire il carico di lavoro corrente. Prova ad aumentare `maxConnections` e `maxSendingThreads` a inserire le impostazioni generali della configurazione dell'agente su. `/etc/aws-kinesis/agent.json` Il valore predefinito per questi campi è 12 volte superiore ai processori di runtime disponibili. Consulta AgentConfiguration.java per ulteriori informazioni sulle impostazioni avanzate delle configurazioni degli agenti.

Come posso eseguire il debug di un altro problema con Kinesis Agent?

`DEBUG`i log di livello possono essere abilitati in. `/etc/aws-kinesis/log4j.xml`

Come devo configurare Kinesis Agent?

Più piccolo è`maxBufferSizeBytes`, più frequentemente Kinesis Agent invierà i dati. Ciò può essere utile in quanto riduce i tempi di consegna dei record, ma aumenta anche le richieste al secondo a Kinesis.

Perché Kinesis Agent invia record duplicati?

Ciò si verifica a causa di un'errata configurazione nella coda dei file. Assicurati che ognuno corrisponda `fileFlow's filePattern` a un solo file. Ciò può verificarsi anche se la `logrotate` modalità utilizzata è `copytruncate` attiva. Prova a passare alla modalità predefinita o crea per evitare duplicazioni. Per ulteriori informazioni sulla gestione dei record duplicati, vedere Gestione dei record duplicati.

Scrivi su Kinesis Data AWS Streams utilizzando altri servizi

I seguenti AWS servizi possono integrarsi direttamente con Amazon Kinesis Data Streams per scrivere dati su flussi di dati Kinesis. Consulta le informazioni relative a ciascun servizio che ti interessa e consulta i riferimenti forniti.

Argomenti

- Scrivi su Kinesis Data Streams utilizzando AWS Amplify
- Scrivi su Kinesis Data Streams utilizzando Amazon Aurora
- Scrivi su Kinesis Data Streams utilizzando Amazon CloudFront

- Scrivi su Kinesis Data Streams CloudWatch utilizzando Amazon Logs
- Scrivi su Kinesis Data Streams utilizzando Amazon Connect
- Scrivi su Kinesis Data Streams utilizzando AWS Database Migration Service
- Scrivi su Kinesis Data Streams utilizzando Amazon DynamoDB
- Scrivi su Kinesis Data Streams utilizzando Amazon EventBridge
- Scrivi su Kinesis Data Streams utilizzando AWS IoT Core
- Scrivi su Kinesis Data Streams utilizzando Amazon Relational Database Service
- Scrivi su Kinesis Data Streams utilizzando Amazon Pinpoint
- Scrivi su Kinesis Data Streams utilizzando Amazon Quantum Ledger Database (Amazon QLDB)

Scrivi su Kinesis Data Streams utilizzando AWS Amplify

Puoi utilizzare Amazon Kinesis Data Streams per lo streaming di dati dalle tue applicazioni mobili create AWS con Amplify per l'elaborazione in tempo reale. Puoi quindi creare pannelli di controllo in tempo reale, acquisire eccezioni e generare avvisi, generare consigli e prendere altre decisioni aziendali o operative in tempo reale. Puoi anche inviare dati ad altri servizi come Amazon Simple Storage Service, Amazon DynamoDB e Amazon Redshift.

Per ulteriori informazioni, consulta Utilizzo di Amazon Kinesis nel Centro per sviluppatori di AWS Amplify.

Scrivi su Kinesis Data Streams utilizzando Amazon Aurora

Puoi utilizzare Flusso di dati Amazon Kinesis per monitorare le attività sui cluster di database Amazon Aurora. Con i flussi di attività di database, il cluster di database Aurora invia le attività a un Flusso di dati Amazon Kinesis in tempo reale. È quindi possibile creare applicazioni per la gestione della conformità che utilizzano queste attività, le controllano e generano avvisi. Puoi utilizzare Amazon Firehose anche per archiviare i dati.

Per ulteriori informazioni, consulta Flussi di attività del database nella Guida per gli sviluppatori di Amazon Aurora.

Scrivi su Kinesis Data Streams utilizzando Amazon CloudFront

Puoi usare Amazon Kinesis Data CloudFront Streams con log in tempo reale e ottenere informazioni sulle richieste fatte a una distribuzione in tempo reale. Puoi quindi creare il tuo consumer Kinesis

Data Stream o utilizzare Amazon Data Firehose per inviare i dati di log ad Amazon S3, Amazon Redshift, Amazon Service o a un servizio di elaborazione dei log di terze OpenSearch parti.

Per ulteriori informazioni, consulta la sezione Log in tempo reale nella Amazon CloudFront Developer Guide.

Scrivi su Kinesis Data Streams CloudWatch utilizzando Amazon Logs

Puoi utilizzare CloudWatch gli abbonamenti per accedere a un feed di eventi di registro in tempo reale da Amazon CloudWatch Logs e inviarlo a un flusso di dati Kinesis per l'elaborazione, l'analisi e il caricamento su altri sistemi.

Per ulteriori informazioni, consulta Elaborazione in tempo reale dei dati di log con abbonamenti nella Amazon CloudWatch Logs User Guide.

Scrivi su Kinesis Data Streams utilizzando Amazon Connect

Puoi utilizzare il flusso di dati Kinesis per esportare i record dei contatti e gli eventi degli agenti in tempo reale dalla tua istanza Amazon Connect. Puoi anche abilitare lo streaming di dati dai profili dei clienti di Amazon Connect per ricevere automaticamente aggiornamenti a un flusso di dati Kinesis sulla creazione di nuovi profili o modifiche a quelli esistenti.

Puoi quindi creare applicazioni consumer per elaborare e analizzare i dati in tempo reale. Ad esempio, utilizzando i record di contatto e i dati del profilo cliente, è possibile conservare i dati dei sistemi di origine, come CRMs gli strumenti di automazione del marketing, up-to-date con le informazioni più recenti. Utilizzando i dati sugli eventi degli agenti, puoi creare pannelli di controllo che visualizzano informazioni ed eventi sugli agenti e attivare notifiche personalizzate di attività specifiche degli agenti.

Per ulteriori informazioni, consulta Streaming di dati per la tua istanza, Configurazione dell'esportazione in tempo reale e Flussi di eventi degli agenti nella Guida per l'amministratore di Amazon Connect.

Scrivi su Kinesis Data Streams utilizzando AWS Database Migration Service

Puoi utilizzarlo AWS Database Migration Service per migrare i dati verso un flusso di dati Kinesis. Puoi quindi creare applicazioni consumer per elaborare e analizzare i record di dati in tempo reale. Puoi anche inviare facilmente dati downstream ad altri servizi come Amazon Simple Storage Service, Amazon DynamoDB e Amazon Redshift

Per ulteriori informazioni, consulta Utilizzo del flusso di dati Kinesis nella Guida per l'utente di AWS Database Migration Service .

Scrivi su Kinesis Data Streams utilizzando Amazon DynamoDB

Utilizza Flusso di dati Amazon Kinesis per acquisire le modifiche apportate a Amazon DynamoDB. Kinesis Data Streams acquisisce le modifiche a livello di elemento in qualsiasi tabella DynamoDB e le replica in un flusso dei dati Kinesis. Le tue applicazioni consumer possono accedere a questo flusso per visualizzare le modifiche a livello di elemento in tempo reale e fornire tali modifiche a valle o agire in base al contenuto.

Per ulteriori informazioni, consulta Funzionamento del flusso di dati Kinesis con DynamoDB nella Guida per gli sviluppatori di Amazon DynamoDB.

Scrivi su Kinesis Data Streams utilizzando Amazon EventBridge

Utilizzando Kinesis Data Streams, AWS puoi inviare EventBridge eventi di chiamata API in un flusso, creare applicazioni consumer ed elaborare grandi quantità di dati. Puoi anche utilizzare Kinesis Data Streams come EventBridge destinazione in Pipes e fornire record in un flusso da una delle fonti disponibili dopo il filtraggio e l'arricchimento opzionali.

Per ulteriori informazioni, consulta Send events to an Amazon Kinesis stream and EventBridge Pipes nella Amazon EventBridge User Guide.

Scrivi su Kinesis Data Streams utilizzando AWS IoT Core

È possibile scrivere dati in tempo reale dai messaggi MQTT in AWS IoT Core utilizzando le azioni AWS IoT Rule. È quindi possibile creare applicazioni che elaborano i dati, ne analizzano il contenuto e generano avvisi e li distribuiscono in applicazioni di analisi o altri servizi AWS ,

Per ulteriori informazioni, consultal flusso di dati Kinesishttps://docs.aws.amazon.com/iot/latest/developerguide/kinesis-rule-action.html nella Guida per gli sviluppatori di AWS IoT Core.

Scrivi su Kinesis Data Streams utilizzando Amazon Relational Database Service

Puoi utilizzare Flusso di dati Amazon Kinesis per monitorare le attività sulle istanze Amazon RDS. Utilizzando Database Activity Streams, Amazon RDS invia le attività a un flusso di dati Kinesis in

tempo reale. È quindi possibile creare applicazioni per la gestione della conformità che utilizzano queste attività, le controllano e generano avvisi. Puoi anche utilizzare Amazon Data Firehose per archiviare i dati.

Per ulteriori informazioni, consulta Flussi di attività del database nella Guida per gli sviluppatori di Amazon RDS.

Scrivi su Kinesis Data Streams utilizzando Amazon Pinpoint

È possibile configurare Amazon Pinpoint per l'invio dei dati degli eventi a Flusso di dati Amazon Kinesis. Amazon Pinpoint può inviare dati sugli eventi per campagne, percorsi e messaggi e-mail e SMS transazionali. Puoi quindi importare i dati in applicazioni di analisi o creare applicazioni personalizzate per i consumer che intraprendono azioni in base al contenuto degli eventi.

Per ulteriori informazioni, consulta Eventi di streaming nella Guida per gli sviluppatori di Amazon Pinpoint.

Scrivi su Kinesis Data Streams utilizzando Amazon Quantum Ledger Database (Amazon QLDB)

Puoi creare uno stream in Amazon QLDB che acquisisce ogni revisione del documento inserita nel tuo journal e fornisce questi dati ad Amazon Kinesis Data Streams in tempo reale. Un flusso QLDB è un flusso continuo di dati dal diario del registro a una risorsa del flusso di dati Kinesis. Quindi, puoi utilizzare la piattaforma di streaming Kinesis o la Kinesis Client Library per consumare lo streaming, elaborare i record di dati e analizzare il contenuto dei dati. Un flusso QLDB scrive i dati sul flusso di dati Kinesis in tre tipi di record: `control`, `block summary` e `revision details`.

Per ulteriori informazioni, consulta Flussi nella Guida per gli sviluppatori di Amazon QLDB.

Scrivi su Kinesis Data Streams utilizzando integrazioni di terze parti

È possibile scrivere dati su Kinesis Data Streams utilizzando una delle seguenti opzioni di terze parti che si integrano con Kinesis Data Streams. Seleziona l'opzione su cui desideri saperne di più e trova risorse e link alla documentazione pertinente.

Argomenti

- Apache Flink
- Fluentd

- Debezium
- Oracle GoldenGate
- Connessione Kafka
- Adobe Experience
- Striim

Apache Flink

Apache Flink è un diffuso framework open source e motore di elaborazione distribuito per calcoli con stato su flussi di dati illimitati e limitati. Per ulteriori informazioni sulla scrittura sul flusso di dati Kinesis da Apache Flink, consulta Connettore del flusso di dati Amazon Kinesis.

Fluentd

Fluentd è un raccoglitore di dati open source per un livello di registrazione unificato. Per ulteriori informazioni sulla scrittura sul flusso di dati Kinesis da Fluentd. Per ulteriori informazioni, consulta Elaborazione dei flussi con Kinesis.

Debezium

Debezium è una piattaforma open source distribuita per l'acquisizione dei dati relativi alle modifiche. Per ulteriori informazioni sulla scrittura sul flusso di dati Kinesis da Debezium, consulta Streaming delle modifiche ai dati MySQL su Amazon Kinesis.

Oracle GoldenGate

Oracle GoldenGate è un prodotto software che consente di replicare, filtrare e trasformare i dati da un database a un altro database. Per ulteriori informazioni sulla scrittura su Kinesis Data Streams GoldenGate da Oracle, consulta Replica dei dati su Kinesis Data Stream utilizzando Oracle. GoldenGate

Connessione Kafka

Kafka Connect è uno strumento per lo streaming scalabile e affidabile di dati tra Apache Kafka e altri sistemi. Per ulteriori informazioni sulla scrittura di dati da Apache Kafka al flusso di dati Kinesis, consulta Connettore Kafka per Kinesis.

Adobe Experience

La piattaforma Adobe Experience consente alle organizzazioni di centralizzare e standardizzare i dati dei clienti da qualsiasi sistema. Quindi applica data science e machine learning per migliorare notevolmente la progettazione e la distribuzione di esperienze ricche e personalizzate. Per ulteriori informazioni sulla scrittura di dati dalla piattaforma Adobe Experience al flusso di dati Kinesis, consulta Come creare una connessione ad Amazon Kinesis.

Striim

Striim è una piattaforma completa in memoria per la raccolta end-to-end, il filtraggio, la trasformazione, l'arricchimento, l'aggregazione, l'analisi e la fornitura di dati in tempo reale. Per ulteriori informazioni su come scrivere dati sul flusso di dati Kinesis da Striim, consulta Writer Kinesis.

Risolvi i problemi dei produttori di Amazon Kinesis Data Streams

I seguenti argomenti offrono soluzioni a problemi comuni con i produttori di Amazon Kinesis Data Streams:

- La mia applicazione di produzione sta scrivendo a una velocità inferiore al previsto
- Ricevo un errore di autorizzazione della chiave master KMS non autorizzata
- Risolvi altri problemi comuni per i produttori

La mia applicazione di produzione sta scrivendo a una velocità inferiore al previsto

I motivi più comuni per cui la velocità di scrittura è più lenta del previsto sono:

- Limiti di servizio superati
- Voglio ottimizzare il mio produttore
- Uso improprio delle operazioni flushSync()

Limiti di servizio superati

Per scoprire se vengono superate le restrizioni dei servizi, controlla se il producer sta generando eccezioni di throughput dal servizio e convalida le operazioni API sulle quali viene effettuato il throttling. Ricorda che ci sono limiti diversi in base alla chiamata, consulta Quote e limiti. Ad esempio,

in aggiunta ai limiti a livello di shard per le operazioni di lettura e scrittura più comunemente noti, ci sono i seguenti limiti a livello di flusso:

- CreateStream
- DeleteStream
- ListStreams
- GetShardIterator
- MergeShards
- DescribeStream
- DescribeStreamSummary

Le operazioni `CreateStream`, `DeleteStream`, `ListStreams`, `GetShardIterator`, e `MergeShards` sono limitate a 5 chiamate al secondo. L'operazione `DescribeStream` è limitata a 10 chiamate al secondo. L'operazione `DescribeStreamSummary` è limitata a 20 chiamate al secondo.

Se queste chiamate non sono il problema, assicurati di aver selezionato una chiave di partizione che consente di distribuire in modo uniforme le operazioni put e che non disponi di una determinata chiave di partizione che va contro le restrizioni dei servizi quando il resto non lo fa. Ciò richiede di misurare i picchi di throughput e considerare il numero di shard nel flusso. Per ulteriori informazioni sulla gestione dei flussi, consulta Crea e gestisci flussi di dati Kinesis.

> **Tip**
>
> Ricordati di arrotondare al kilobyte più vicino nei calcoli di throttling del throughput quando utilizzi l'operazione di record singolo PutRecord, mentre l'operazione multi-record PutRecords arrotonda la somma cumulativa dei record in ciascuna chiamata. Ad esempio, su una richiesta `PutRecords` di 600 record di 1,1 KB non verrà effettuato il throttling.

Voglio ottimizzare il mio produttore

Prima di iniziare a ottimizzare il tuo produttore, completa le seguenti attività chiave. In primo luogo, identifica il throughput maggiore desiderato in termini di dimensioni di record e record al secondo. Quindi, scarta la capacità di streaming come fattore di limitazione (Limiti di servizio superati). Se hai escluso la capacità di streaming, utilizza i seguenti suggerimenti per la risoluzione dei problemi e linee guida sull'ottimizzazione per due tipi di produttori comuni.

Produttore di grandi dimensioni

Un produttore di grandi dimensioni di solito viene eseguito da un server locale o da un' EC2 istanza Amazon. Ai clienti che necessitano di un throughput maggiore da un producer di grandi dimensioni in genere interessa la latenza per record. Le strategie per gestire la latenza includono: se il cliente è in grado di creare record in microbatch/buffer, utilizza la Amazon Kinesis Producer Library (che dispone di una logica di aggregazione avanzata), l'operazione multi-record o aggrega i record in un file più grande prima di utilizzare l'operazione PutRecordsa record singolo. PutRecord Se non sei in grado di eseguire un batch/buffer, utilizza più thread per scrivere nel servizio di Flussi di dati Kinesis nello stesso momento. Questi AWS SDK per Java e altri SDKs includono client asincroni che possono eseguire questa operazione con pochissimo codice.

Produttore di piccole dimensioni

Un producer di piccole dimensioni solitamente è un'applicazione mobile, un dispositivo IoT o un client Web. Se si tratta di un'app per dispositivi mobili, ti consigliamo di utilizzare l'`PutRecords`operazione o il Kinesis Recorder nel dispositivo mobile. AWS SDKs Per ulteriori informazioni, consulta la Guida AWS Mobile SDK for Android introduttiva e la Guida AWS Mobile SDK for iOS introduttiva. Le applicazioni mobili devono gestire le connessioni intermittenti intrinsecamente e hanno bisogno di un batch di put, ad esempio `PutRecords`. Se non sei in grado di eseguire un batch per qualsiasi motivo, consulta le informazioni sul Produttore di grandi dimensioni qui sopra. Se il producer è un browser, la quantità di dati generati è generalmente inferiore. Tuttavia, stai avviando le operazioni put nel percorso critico dell'applicazione, che non è consigliabile.

Uso improprio delle operazioni **`flushSync()`**

Un utilizzo `flushSync()` errato può influire in modo significativo sulle prestazioni di scrittura. L'`flushSync()`operazione è progettata per scenari di arresto per garantire che tutti i record memorizzati nel buffer vengano inviati prima del termine dell'applicazione KPL. Se hai implementato questa operazione dopo ogni operazione di scrittura, può aggiungere una notevole latenza aggiuntiva, circa 500 ms per scrittura. Assicurati di aver implementato `flushSync()` solo l'arresto dell'applicazione per evitare inutili ritardi aggiuntivi nelle prestazioni di scrittura.

Ricevo un errore di autorizzazione della chiave master KMS non autorizzata

Questo errore si verifica quando un'applicazione producer scrive in un flusso crittografato senza autorizzazioni sulla chiave master KMS. Per assegnare le autorizzazioni a un'applicazione per accedere a una chiave KMS, consulta Utilizzo delle policy della chiave in AWS KMS e Utilizzo delle policy IAM con AWS KMS.

Risolvi altri problemi comuni per i produttori

- Perché il mio flusso di dati Kinesis restituisce un errore interno del server 500?
- Come posso risolvere gli errori di timeout durante la scrittura da Flink al flusso di dati Kinesis?
- Come posso risolvere gli errori di limitazione nel flusso di dati Kinesis?
- Perché il mio flusso di dati Kinesis è limitato?
- Come posso inserire i record di dati in un flusso di dati Kinesis utilizzando la KPL?

Ottimizza i produttori di Kinesis Data Streams

Puoi ottimizzare ulteriormente i tuoi produttori di Amazon Kinesis Data Streams in base al comportamento specifico che vedi. Consulta i seguenti argomenti per identificare le soluzioni.

Argomenti

- Personalizza i tentativi KPL e il comportamento dei limiti di velocità
- Applica le migliori pratiche all'aggregazione KPL

Personalizza i tentativi KPL e il comportamento dei limiti di velocità

Quando aggiungi record utente di Amazon Kinesis Producer Library (KPL) utilizzando l'`addUserRecord()`operazione KPL, a un record viene assegnato un timestamp e aggiunto a un buffer con una scadenza impostata dal parametro di configurazione. `RecordMaxBufferedTime` Questa combinazione timestamp/scadenza imposta la priorità di buffer. I record vengono svuotati dal buffer in base ai seguenti criteri:

- Priorità buffer
- Configurazione aggregazione
- Configurazione della raccolta

I parametri di configurazione di aggregazione e raccolta che influiscono sul comportamento del buffer sono i seguenti:

- `AggregationMaxCount`
- `AggregationMaxSize`
- `CollectionMaxCount`

- `CollectionMaxSize`

I record scaricati vengono quindi inviati al flusso di dati Kinesis come record del flusso di dati Amazon Kinesis utilizzando una chiamata all'operazione dell'API del flusso di dati Kinesis `PutRecords`. L'operazione `PutRecords` invia le richieste al tuo flusso che occasionalmente restituiscono errori totali o parziali. I record con esito negativo vengono automaticamente aggiunti al buffer della KPL. La nuova scadenza è impostata sul minimo di questi due valori:

- Metà della configurazione `RecordMaxBufferedTime` corrente
- Il valore del record time-to-live

Questa strategia consente di includere i record utente KPL riprovati nelle successive chiamate all'API Kinesis Data Streams, per migliorare il throughput e ridurre la complessità, rafforzando al contempo il valore del record Kinesis Data Streams. time-to-live Non esiste un algoritmo di backoff, pertanto questa strategia di nuovi tentativi è relativamente aggressiva. Lo spamming causato da un numero di nuovi tentativi eccessivo viene impedito dalla limitazione della frequenza, discussa nella sezione successiva.

Limitazione della velocità

La KPL include una funzione di limitazione della frequenza, che limita la velocità di trasmissione effettiva della partizione inviata da un singolo producer. La limitazione della frequenza viene implementata utilizzando un algoritmo bucket di token con bucket separati sia per i byte sia per i record . Ogni scrittura riuscita in un flusso di dati Kinesis aggiunge uno o più token a ciascun bucket, fino a una determinata soglia. Questa soglia è configurabile ma, per impostazione predefinita, è impostata su un valore del 50% superiore rispetto al limite di shard effettivo, per permettere la saturazione dello shard da un singolo producer.

È possibile ridurre questo limite per ridurre lo spamming dovuto a un numero eccessivo di nuovi tentativi. Tuttavia, la best practice per ogni producer è riprovare per ottenere il massimo rendimento in modo aggressivo e gestire qualsiasi throttling risultante considerato eccessivo espandendo la capacità del flusso e implementando una strategia di chiave di partizione appropriata.

Applica le migliori pratiche all'aggregazione KPL

Sebbene lo schema dei numeri di sequenza dei record Amazon Kinesis Data Streams risultanti rimanga lo stesso, l'aggregazione fa sì che l'indicizzazione dei record utente di Amazon Kinesis Producer Library (KPL) contenuti in un record Kinesis Data Streams aggregato inizi da 0 (zero);

tuttavia, purché non si faccia affidamento sui numeri di sequenza per identificare in modo univoco i record utente KPL, il codice puoi ignorarlo, come l'aggregazione (dei tuoi record utente KPL in un record Kinesis Data Streams) e la successiva disaggregazione (di un record Kinesis Data Streams nei tuoi record utente KPL) se ne occupa automaticamente per te. Questo vale sia che il consumatore utilizzi KCL o l' AWS SDK. Per utilizzare questa funzionalità di aggregazione, dovrai inserire la parte Java di KPL nella tua build se il tuo consumatore è stato scritto utilizzando l'API fornita nell'SDK. AWS

Per utilizzare i numeri di sequenza come identificatori univoci dei tuoi record utente della KPL, ti consigliamo di usare le operazioni `public int hashCode()` e `public boolean equals(Object obj)` fornite in `Record` e `UserRecord` per abilitare il confronto dei record utente della KPL. Inoltre, per esaminare il numero di sequenza secondaria del tuo record utente &KPL;, è possibile trasmetterlo a un'istanza `UserRecord` e recuperare il suo numero di sequenza secondaria.

Per ulteriori informazioni, consulta Implementa la deaggregazione dei consumatori.

Leggi i dati da Amazon Kinesis Data Streams

Un consumer è un'applicazione che elabora i dati da un flusso di dati Kinesis. Quando un consumer utilizza l'espansione fan-out avanzato, ottiene la propria assegnazione di 2 MiB/sec di rendimento di lettura, consentendo a più utenti di leggere i dati dallo stesso flusso in parallelo, senza contendersi il rendimento di lettura con altri consumatori. Per utilizzare la funzionalità avanzata di smistamento degli shard, consulta Sviluppa consumatori con fan-out migliorati con un throughput dedicato.

Puoi creare utenti per Kinesis Data Streams utilizzando Kinesis Client Library (KCL) o. AWS SDK per Java Puoi anche sviluppare consumatori utilizzando altri AWS servizi come AWS Lambda Amazon Managed Service for Apache Flink e Amazon Data Firehose. Kinesis Data Streams supporta integrazioni AWS con altri servizi come Amazon EMR, Amazon e Amazon Redshift. Supporta anche integrazioni di terze parti tra cui Apache Flink EventBridge AWS Glue, Adobe Experience Platform, Apache Druid, Apache Spark, Databricks, Confluent Platform, Kinesumer e Talend.

Argomenti

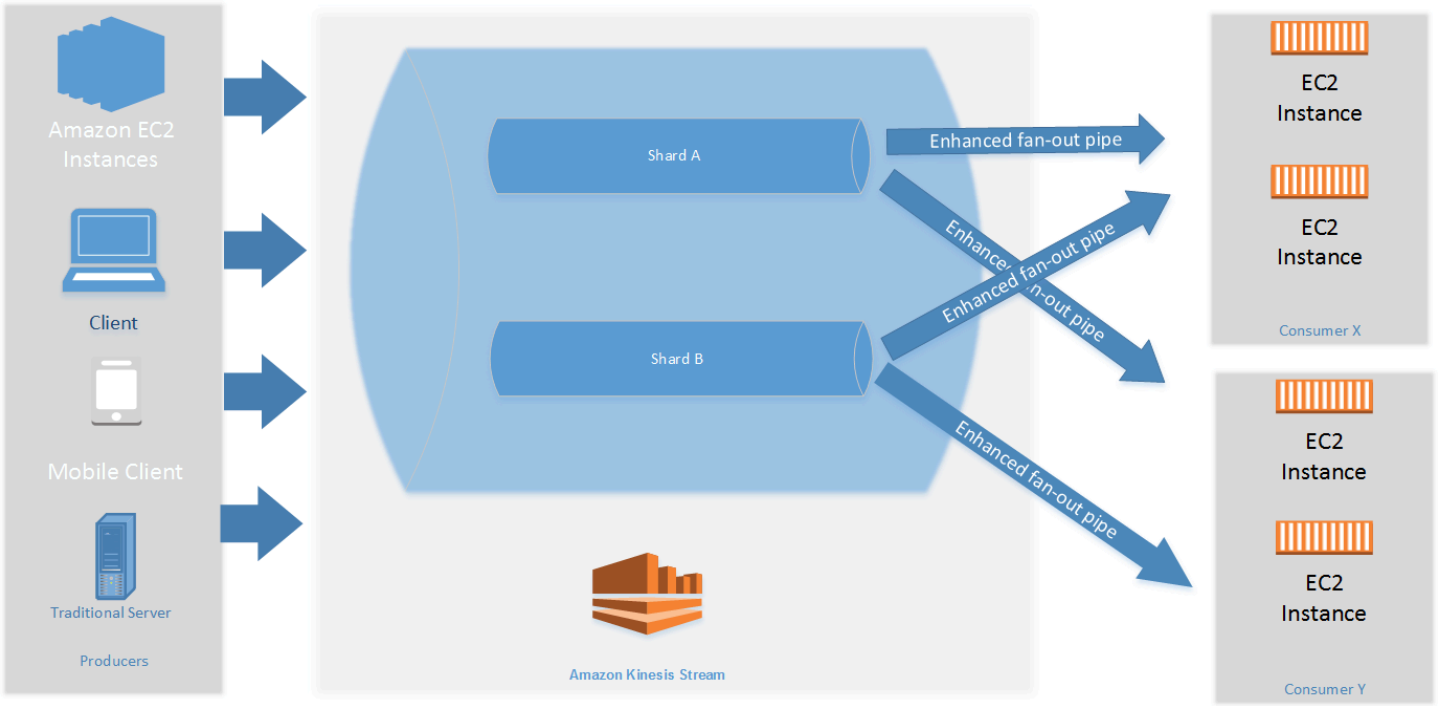
- Sviluppa consumatori con fan-out migliorati con un throughput dedicato
- Usa il Data Viewer nella console Kinesis
- Interroga i tuoi flussi di dati nella console Kinesis
- Usa la libreria client Kinesis
- Sviluppa i consumatori con AWS SDK per Java
- Sviluppa i consumatori utilizzando AWS Lambda
- Sviluppa i consumatori utilizzando Amazon Managed Service per Apache Flink
- Sviluppa i consumatori utilizzando Amazon Data Firehose
- Leggi i dati da Kinesis Data AWS Streams utilizzando altri servizi
- Leggi da Kinesis Data Streams utilizzando integrazioni di terze parti
- Risolvi i problemi dei consumatori di Kinesis Data Streams
- Ottimizza i consumatori di Amazon Kinesis Data Streams

Sviluppa consumatori con fan-out migliorati con un throughput dedicato

In Flusso di dati Amazon Kinesis, è possibile creare applicazioni consumer che utilizzano la funzione di fan-out avanzato. Questa funzionalità consente ai consumatori di ricevere record da uno stream con un throughput fino a 2 MB di dati al secondo per shard. Questa velocità di trasmissione effettiva è dedicata, il che significa che le applicazioni consumer che utilizzano il fan-out avanzato non devono convivere con altre applicazioni consumer che stanno ricevendo dati dal flusso. Il flusso di dati Kinesis inserisce i record di dati dal flusso alle applicazioni consumer che utilizzano il fan-out avanzato. Pertanto, per queste applicazioni consumer non è necessario eseguire il polling per i dati.

> **Important**
>
> È possibile registrare fino a venti applicazioni consumer per flusso per l'utilizzo del fan-out avanzato.

Il diagramma seguente illustra l'architettura del fan-out avanzato. Se si utilizza la versione 2.0 o una successiva di Amazon Kinesis Client Library (KCL) per creare un'applicazione consumer, la KCL imposterà l'applicazione per utilizzare il fan-out avanzato e ricevere i dati provenienti da tutte le partizioni del flusso. Se si utilizza l'API per creare un'applicazione consumer che utilizza il fan-out avanzato, è possibile abbonarsi a singoli shard.



Il diagramma mostra:

- Un flusso con due shard.
- Due applicazioni consumer che utilizzano il fan-out avanzato per ricevere i dati dal flusso: Consumer X e Consumer Y. Ognuna delle due applicazioni consumer è abbonata a tutti gli shard e a tutti i record del flusso. Se si utilizza la versione 2.0 o una successiva della KCL per creare un'applicazione consumer, la KCL sottoscriverà automaticamente l'applicazione a tutte le partizioni del flusso. D'altra parte, se si utilizza l'API per creare un'applicazione consumer è possibile abbonarsi a singoli shard.
- Frecce che rappresentano i canali di diffusione del fan-out che le applicazioni consumer utilizzano per ricevere i dati dal flusso. Un canale di fan-out avanzato fornisce fino a 2 MiB/sec di dati per shard, indipendentemente da qualsiasi altro canale o dal numero totale di applicazioni consumer.

Argomenti

- Differenze tra una soluzione condivisa tra tutti i consumatori e una versione estesa a tutti i consumatori
- Gestisci al meglio i consumatori con il fan-out con o AWS CLI APIs

Differenze tra una soluzione condivisa tra tutti i consumatori e una versione estesa a tutti i consumatori

La tabella seguente mette a confronto i consumatori predefiniti con throughput condiviso con i consumatori con fan-out avanzato. Il ritardo di propagazione dei messaggi è definito come il tempo impiegato, in millisecondi, da un payload inviato utilizzando il payload-dispatching (like and) per raggiungere l'applicazione consumer tramite il APIs payload-consuma (like and). `PutRecord` `PutRecords` APIs `GetRecords` `SubscribeToShard`

Questa tabella mette a confronto i consumatori con throughput condiviso con quelli dei consumatori che utilizzano la tecnologia fan-out più evoluta

Caratteristiche	Consumatori con throughput condiviso senza fan-out avanzato	Consumatori con fan-out migliorato
Throughput di lettura	Risolto per un totale di 2. MB/sec per shard. If there are multiple consumers reading from the same shard, they all share this throughput. The sum of the throughputs they receive from the shard doesn't exceed 2 MB/sec	Adatta man mano che consumatori si registrano per utilizzare il rendimento avanzato. Ogni consumer registrato per l'utilizzo di un fan-out avanzato riceve il proprio throughput di lettura per shard, fino a 2 MiB/sec, indipendentemente dagli altri consumer.
Ritardo di propagazione dei messaggi	Una media di circa 200 ms in presenza di un consumer che legge dal flusso. Tale media sale fino a circa 1000 ms in presenza di cinque consumer.	In genere, la media è pari a 70 ms, in presenza di uno o di 5 consumer.
Costo	Non applicabile	Non esistono costi di recupero dati né costi orari per shard-consumer. Per ulteriori informazioni, consulta Prezzi

Caratteristiche	Consumatori con throughput condiviso senza fan-out avanzato	Consumatori con fan-out migliorato
		dei flussi di dati per Amazon Kinesis.
Modello di distribuzione record	Recupera il modello tramite HTTP utilizzando GetRecords.	Kinesis Data Streams invia i record all'utente tramite HTTP/2 utilizzando. Subscribe ToShard

Gestisci al meglio i consumatori con il fan-out con o AWS CLI APIs

Le applicazioni consumer che utilizzano il fan-out avanzato in Flusso di dati Amazon Kinesis possono ricevere record da un flusso di dati con velocità di trasmissione effettiva dedicata fino a 2 MiB di dati al secondo per partizione. Per ulteriori informazioni, consulta Sviluppa consumatori con fan-out migliorati con un throughput dedicato.

Puoi utilizzare AWS CLI o Kinesis APIs Data Streams per registrare, descrivere, elencare e annullare la registrazione di un consumatore che utilizza il fan-out avanzato in Kinesis Data Streams.

Gestisci i consumatori utilizzando il AWS CLI

Puoi registrare, descrivere, elencare e annullare la registrazione dei consumatori avanzati con fan-out utilizzando il. AWS CLI Per alcuni esempi, consultare la seguente documentazione.

register-stream-consumer

Registra un consumatore per un flusso di dati Kinesis. Puoi applicare i tag durante la registrazione del consumatore.

describe-stream-consumer

Ottiene la descrizione di un consumatore registrato con l'ARN del consumatore o la combinazione di nome del consumatore e ARN di streaming.

list-stream-consumers

Elenca i consumatori registrati per ricevere dati da uno stream utilizzando il fan-out avanzato.

deregister-stream-consumer

Annulla la registrazione di un consumatore con l'ARN del consumatore o il nome del consumatore e trasmetti in streaming la combinazione ARN.

Gestisci i consumatori utilizzando Kinesis Data Streams APIs

Puoi registrare, descrivere, elencare e annullare la registrazione dei consumatori fan-out avanzati utilizzando Kinesis Data Streams. APIs Per alcuni esempi, consultare la seguente documentazione.

RegisterStreamConsumer

Registra un consumatore per un flusso di dati Kinesis con tag. Puoi applicare i tag durante la registrazione del consumatore.

DescribeStreamConsumer

Ottiene la descrizione di un consumatore registrato con l'ARN del consumatore o la combinazione di nome del consumatore e ARN di streaming.

ListStreamConsumers

Elenca i consumatori registrati per ricevere dati da uno stream utilizzando il fan-out avanzato.

DeregisterStreamConsumer

Annulla la registrazione di un consumatore con l'ARN del consumatore o il nome del consumatore e trasmetti in streaming la combinazione ARN.

Etichettare i consumatori

Puoi assegnare i tuoi metadati agli stream e ai fan-out consumer avanzati che crei in Kinesis Data Streams sotto forma di tag. Puoi utilizzare i tag per classificare e tenere traccia dei costi dei tuoi consumatori. Puoi anche controllare l'accesso ai consumatori utilizzando tag con controllo degli accessi basato sugli attributi (ABAC). Per ulteriori informazioni, consulta Etichetta le tue risorse Amazon Kinesis Data Streams.

Usa il Data Viewer nella console Kinesis

Il Data Viewer nella Kinesis Management Console consente di visualizzare i record di dati all'interno dello shard specificato del flusso di dati senza dover sviluppare un'applicazione consumer. Per utilizzare il Data Viewer, procedi come segue:

1. Accedi AWS Management Console e apri la console Kinesis all'indirizzo https://console.aws.amazon.com /kinesis.
2. Scegli il flusso di dati attivo di cui desideri visualizzare i record con il Data Viewer, quindi scegli la scheda Data Viewer.
3. Nella scheda Data Viewer per il flusso di dati attivo selezionato, scegli la partizione di cui desideri visualizzare i record, scegli la posizione iniziale e quindi fai clic su Ottieni record. È possibile impostare la posizione iniziale su uno dei seguenti valori:

   - Al numero di sequenza: mostra i record dalla posizione indicata dal numero di sequenza specificato nel campo del numero di sequenza.
   - Dopo il numero di sequenza: mostra i record dalla posizione indicata dal numero di sequenza specificato nel campo del numero di sequenza.
   - Al timestamp: mostra i record dalla posizione indicata dal timestamp specificato nel campo timestamp.
   - Orizzonte di taglio: mostra i record dell'ultimo record non tagliato della partizione, che è il record di dati più vecchio della partizione.
   - Più recente: mostra i record subito dopo il record più recente nella partizione, in modo da leggere sempre i dati più recenti nella partizione.

     I record di dati generati che corrispondono all'ID della partizione e alla posizione iniziale specificati vengono quindi visualizzati in una tabella dei record nella console. Vengono visualizzati al massimo 50 record alla volta. Per visualizzare il set di record successivo, fare clic sul pulsante Avanti.
4. Fare clic su un singolo record per visualizzare il payload del record in dati non elaborati o in formato JSON in una finestra separata.

Tieni presente che quando fai clic sui pulsanti Ottieni record o Avanti in Data Viewer, viene richiamata l'GetRecordsAPI e ciò si applica al limite GetRecordsAPI di 5 transazioni al secondo.

Interroga i tuoi flussi di dati nella console Kinesis

La scheda Data Analytics nella console Kinesis Data Streams consente di interrogare i flussi di dati utilizzando SQL. Per utilizzare questa funzionalità, segui questi passaggi:

1. Accedi AWS Management Console e apri la console Kinesis all'indirizzo https://console.aws.amazon.com /kinesis.

2. Scegli il flusso di dati attivo su cui desideri interrogare con SQL, quindi scegli la scheda Analisi dei dati.
3. Nella scheda Analisi dei dati, puoi eseguire l'ispezione e la visualizzazione del flusso con un notebook Apache Flink Studio gestito. Puoi eseguire query SQL ad hoc per ispezionare il flusso di dati e visualizzare i risultati in pochi secondi utilizzando Apache Zeppelin. Nella scheda Analisi dei dati, scegli Accetto, quindi scegli Crea notebook per creare un taccuino.
4. Dopo aver creato il taccuino, scegli Apri in Apache Zeppelin. Questo aprirà il tuo taccuino in una nuova scheda. Un notebook è un'interfaccia interattiva in cui è possibile inviare le query SQL. Scegli la nota che contiene il nome del tuo stream.
5. Vedrai una nota con una `SELECT` query di esempio per l'output dei dati nello stream già in esecuzione. Ciò consente di visualizzare lo schema per il flusso di dati.
6. Per provare altre query, ad esempio finestre ribaltabili o scorrevoli, scegli Visualizza interrogazioni di esempio nella scheda Analisi dei dati. Copia la query, modificala per adattarla allo schema del flusso di dati, quindi eseguila in un nuovo paragrafo della tua nota Zeppelin.

Usa la libreria client Kinesis

Cos'è Kinesis Client Library?

Kinesis Client Library (KCL) è una libreria software Java autonoma progettata per semplificare il processo di consumo ed elaborazione dei dati da Amazon Kinesis Data Streams. KCL gestisce molte delle attività complesse associate all'elaborazione distribuita, permettendo agli sviluppatori di concentrarsi sull'implementazione della logica aziendale per l'elaborazione dei dati. Gestisce attività come il bilanciamento del carico tra più lavoratori, la risposta agli errori dei lavoratori, il checkpoint dei record elaborati e la risposta alle variazioni del numero di shard nel flusso.

KCL viene aggiornato frequentemente per incorporare versioni più recenti delle librerie sottostanti, miglioramenti della sicurezza e correzioni di bug. Ti consigliamo di utilizzare l'ultima versione di KCL per evitare problemi noti e beneficiare di tutti gli ultimi miglioramenti. Per trovare l'ultima versione di KCL, consulta KCL Github.

> **Important**
>
> - Ti consigliamo di utilizzare l'ultima versione di KCL per evitare bug e problemi noti. Se utilizzi KCL 2.6.0 o una versione precedente, esegui l'upgrade a KCL 2.6.1 o versione

successiva per evitare una rara condizione che può bloccare l'elaborazione degli shard quando la capacità del flusso cambia.

- KCL è una libreria Java. Il supporto per linguaggi diversi da Java viene fornito utilizzando un demone basato su Java chiamato. MultiLangDaemon MultiLangDaemoninteragisce con l'applicazione KCL tramite STDIN e STDOUT. Per ulteriori informazioni su on, vedere. MultiLangDaemon GitHub Sviluppa i consumatori con KCL in linguaggi non Java
- Non utilizzare le AWS SDK per Java versioni da 2.27.19 a 2.27.23 con KCL 3.x. Queste versioni includono un problema che causa un errore di eccezione relativo all'utilizzo di DynamoDB da parte di KCL. Si consiglia di utilizzare la AWS SDK per Java versione 2.28.0 o successiva per evitare questo problema.

Caratteristiche e vantaggi principali di KCL

Di seguito sono riportate le caratteristiche principali e i relativi vantaggi del KCL:

- Scalabilità: KCL consente alle applicazioni di scalare dinamicamente distribuendo il carico di elaborazione tra più lavoratori. È possibile ampliare o ridurre l'applicazione, manualmente o con l'auto-scaling, senza preoccuparsi della ridistribuzione del carico.
- Bilanciamento del carico: KCL bilancia automaticamente il carico di elaborazione tra i lavoratori disponibili, garantendo una distribuzione uniforme del lavoro tra i lavoratori.
- Checkpointing: KCL gestisce il checkpoint dei record elaborati, consentendo alle applicazioni di riprendere l'elaborazione dall'ultima posizione elaborata con successo.
- Tolleranza ai guasti: KCL fornisce meccanismi integrati di tolleranza agli errori, che assicurano che l'elaborazione dei dati continui anche in caso di fallimento dei singoli lavoratori. KCL fornisce at-least-once anche la consegna.
- Gestione delle modifiche a livello di flusso: KCL si adatta alle divisioni e alle unioni degli shard che potrebbero verificarsi a causa di variazioni nel volume dei dati. Mantiene l'ordine assicurandosi che i frammenti secondari vengano elaborati solo dopo che lo shard principale è stato completato e controllato.
- Monitoraggio: KCL si integra con Amazon CloudWatch per il monitoraggio a livello di consumatore.
- Supporto multilingue: KCL supporta nativamente Java e consente di utilizzare più linguaggi di programmazione non Java. MultiLangDaemon

Concetti KCL

Questa sezione spiega i concetti e le interazioni principali di Kinesis Client Library (KCL). Questi concetti sono fondamentali per lo sviluppo e la gestione delle applicazioni consumer KCL.

- Applicazione consumer KCL: un'applicazione personalizzata progettata per leggere ed elaborare i record dai flussi di dati Kinesis utilizzando la Kinesis Client Library.
- Worker: le applicazioni consumer KCL sono generalmente distribuite, con uno o più lavoratori in esecuzione contemporaneamente. KCL coordina i lavoratori affinché utilizzino i dati del flusso in modo distribuito e bilancia il carico in modo uniforme tra più lavoratori.
- Scheduler: una classe di alto livello utilizzata da un lavoratore KCL per iniziare l'elaborazione dei dati. Ogni lavoratore KCL ha uno scheduler. Lo scheduler inizializza e supervisiona varie attività, tra cui la sincronizzazione delle informazioni sugli shard dai flussi di dati Kinesis, il monitoraggio delle assegnazioni degli shard tra i lavoratori e l'elaborazione dei dati dal flusso in base agli shard assegnati al lavoratore. Scheduler può adottare varie configurazioni che influiscono sul comportamento dello scheduler, come il nome del flusso da elaborare e le credenziali. AWS Scheduler avvia la consegna dei record di dati dallo stream ai processori di record.
- Processore di record: definisce la logica con cui l'applicazione consumer KCL elabora i dati che riceve dai flussi di dati. È necessario implementare la propria logica di elaborazione dei dati personalizzata nel processore di registrazione. Un lavoratore KCL crea un'istanza di uno scheduler. Lo scheduler crea quindi un'istanza di un processore di record per ogni shard a cui appartiene un contratto di locazione. Un lavoratore può eseguire più processori di record.
- Leasing: definisce l'assegnazione tra un lavoratore e uno shard. Le applicazioni consumer di KCL utilizzano i contratti di locazione per distribuire l'elaborazione dei record di dati tra più lavoratori. Ogni frammento è vincolato a un solo lavoratore mediante un contratto di locazione alla volta e ogni lavoratore può detenere uno o più contratti di locazione contemporaneamente. Quando un lavoratore smette di essere titolare di un contratto di locazione a causa di un arresto o di un fallimento, KCL assegna a un altro lavoratore il compito di sottoscrivere il contratto di locazione. Per ulteriori informazioni sul contratto di locazione, consulta la documentazione di Github: Lease Lifecycle.
- Tabella di leasing: è una tabella Amazon DynamoDB unica utilizzata per tenere traccia di tutti i leasing per l'applicazione consumer KCL. Ogni applicazione consumer KCL crea la propria tabella di leasing. La tabella dei contratti di locazione viene utilizzata per mantenere lo stato tra tutti i lavoratori e coordinare l'elaborazione dei dati. Per ulteriori informazioni, consulta Tabelle di metadati DynamoDB e bilanciamento del carico in KCL.

- Checkpointing: è il processo di memorizzazione persistente della posizione dell'ultimo record elaborato con successo in uno shard. KCL gestisce il checkpoint per garantire che l'elaborazione possa essere ripresa dall'ultima posizione di checkpoint se un lavoratore fallisce o l'applicazione si riavvia. I checkpoint vengono archiviati nella tabella di lease di DynamoDB come parte dei metadati del leasing. Ciò consente ai lavoratori di continuare l'elaborazione dal punto in cui si era fermato il lavoratore precedente.

Tabelle di metadati DynamoDB e bilanciamento del carico in KCL

KCL gestisce i metadati dei lavoratori, come i leasing e le metriche di utilizzo della CPU. KCL tiene traccia di questi metadati utilizzando le tabelle DynamoDB. Per ogni applicazione Amazon Kinesis Data Streams, KCL crea tre tabelle DynamoDB per gestire i metadati: lease table, worker metrics table e coordinator state table.

> **Note**
>
> KCL 3.x ha introdotto due nuove tabelle di metadati: le metriche dei lavoratori e le tabelle degli stati dei coordinatori.

> **Important**
>
> È necessario aggiungere le autorizzazioni appropriate per le applicazioni KCL per creare e gestire tabelle di metadati in DynamoDB. Per informazioni dettagliate, consultare Autorizzazioni IAM richieste per le applicazioni consumer KCL.
> L'applicazione consumer KCL non rimuove automaticamente queste tre tabelle di metadati DynamoDB. Assicurati di rimuovere queste tabelle di metadati DynamoDB create dall'applicazione consumer KCL quando disattivi l'applicazione consumer per evitare costi inutili.

Tabella di leasing

Una tabella di lease è una tabella Amazon DynamoDB unica utilizzata per tenere traccia degli shard presi in leasing ed elaborati dagli scheduler dell'applicazione consumer KCL. Ogni applicazione consumer KCL crea la propria tabella di leasing. Per impostazione predefinita, KCL utilizza il nome dell'applicazione consumer per il nome della tabella di leasing. È possibile impostare un nome di

tabella personalizzato utilizzando la configurazione. KCL crea anche un indice secondario globale sulla tabella di leasing con la chiave di partizione di LeaseOwner per un'efficiente individuazione del leasing. L'indice secondario globale rispecchia l'attributo LeaseKey della tabella di leasing di base. Se la tabella di leasing per l'applicazione consumer KCL non esiste all'avvio dell'applicazione, uno dei lavoratori crea la tabella di leasing per l'applicazione.

È possibile visualizzare la tabella di lease utilizzando la console Amazon DynamoDB mentre l'applicazione è in esecuzione.

> **Important**
>
> - Il nome di ogni applicazione consumer KCL deve essere univoco per evitare la duplicazione del nome della tabella di leasing.
> - Il tuo account sarà addebitato per i costi associati alla tabella DynamoDB, oltre ai costi associati al flusso di dati Kinesis stesso.

Ogni riga della tabella di leasing rappresenta uno shard che viene elaborato dagli scheduler dell'applicazione consumer. I campi chiave includono quanto segue:

- LeaseKey: per l'elaborazione a flusso singolo, questo è l'ID dello shard. Per l'elaborazione multi-stream con KCL, è strutturata come. `account-id:StreamName:streamCreationTimestamp:ShardId` leaseKey è la chiave di partizione della tabella di leasing. Per ulteriori informazioni sull'elaborazione multi-stream, consulta. Elaborazione multi-stream con KCL
- checkpoint: il numero di sequenza di checkpoint più recente per lo shard.
- checkpointSubSequenceNumero: quando si utilizza la funzione di aggregazione della Kinesis Producer Library, si tratta di un'estensione del checkpoint che tiene traccia dei record dei singoli utenti all'interno del record Kinesis.
- LeaseCounter: utilizzato per verificare se un lavoratore sta attualmente elaborando attivamente il contratto di locazione. LeaseCounter aumenta se la proprietà del leasing viene trasferita a un altro lavoratore.
- LeaseOwner: il lavoratore attuale che detiene questo contratto di locazione.
- ownerSwitchesSinceCheckpoint: quante volte questo contratto di locazione ha cambiato lavoratori dall'ultimo checkpoint.

- parentShardId: ID del genitore di questo shard. Si assicura che lo shard principale sia completamente elaborato prima che inizi l'elaborazione sugli shard secondari, mantenendo l'ordine di elaborazione dei record corretto.
- childShardId: Elenco degli shard secondari IDs risultanti dalla divisione o dall'unione di questo shard. Utilizzato per tracciare la derivazione degli shard e gestire l'ordine di elaborazione durante le operazioni di resharding.
- startingHashKey: Il limite inferiore dell'intervallo di chiavi hash per questo shard.
- endingHashKey: Il limite superiore dell'intervallo di chiavi hash per questo shard.

Se utilizzi l'elaborazione multi-stream con KCL, nella tabella di leasing vengono visualizzati i due campi aggiuntivi seguenti. Per ulteriori informazioni, consulta Elaborazione multi-stream con KCL .

- shardID: l'ID della partizione.
- streamName: l'identificatore del flusso di dati nel seguente formato:. `account-id:StreamName:streamCreationTimestamp`

Tabella delle metriche dei lavoratori

La tabella Worker Metrics è una tabella Amazon DynamoDB unica per ogni applicazione KCL e viene utilizzata per registrare i parametri di utilizzo della CPU di ogni lavoratore. Queste metriche verranno utilizzate da KCL per eseguire assegnazioni di leasing efficienti al fine di garantire un utilizzo equilibrato delle risorse tra i lavoratori. Per impostazione predefinita, KCL utilizza `KCLApplicationName-WorkerMetricStats` per il nome della tabella delle metriche dei lavoratori.

Tabella degli stati del coordinatore

Una tabella di stato del coordinatore è una tabella Amazon DynamoDB unica per ogni applicazione KCL e viene utilizzata per memorizzare informazioni sullo stato interno per i lavoratori. Ad esempio, la tabella degli stati del coordinatore memorizza i dati relativi all'elezione del leader o i metadati associati alla migrazione in atto da KCL 2.x a KCL 3.x. Per impostazione predefinita, KCL utilizza `KCLApplicationName-CoordinatorState` per il nome del coordinatore la tabella degli stati.

Modalità di capacità DynamoDB per le tabelle di metadati create da KCL

Per impostazione predefinita, la Kinesis Client Library (KCL) crea tabelle di metadati DynamoDB come la tabella di lease, la tabella delle metriche dei lavoratori e la tabella dello stato del coordinatore

utilizzando la modalità di capacità su richiesta. Questa modalità ridimensiona automaticamente la capacità di lettura e scrittura per adattarsi al traffico senza richiedere la pianificazione della capacità. Ti consigliamo vivamente di mantenere la modalità di capacità come modalità on-demand per un funzionamento più efficiente di queste tabelle di metadati.

Se decidi di passare dalla tabella di leasing alla modalità di capacità assegnata, segui queste best practice:

- Analizza i modelli di utilizzo:
  - Monitora i modelli e gli utilizzi di lettura e scrittura della tua applicazione (RCU, WCU) utilizzando i parametri di Amazon. CloudWatch
  - Comprendi i requisiti di velocità di picco e media.
- Calcola la capacità richiesta:
  - Stima le unità di capacità di lettura (RCUs) e le unità di capacità di scrittura (WCUs) in base all'analisi.
  - Prendi in considerazione fattori come il numero di frammenti, la frequenza dei checkpoint e il numero di lavoratori.
- Implementa il ridimensionamento automatico:
  - Utilizza la scalabilità automatica di DynamoDB per regolare automaticamente la capacità fornita e impostare limiti di capacità minimi e massimi appropriati.
  - La scalabilità automatica di DynamoDB ti aiuterà a evitare che la tabella di metadati KCL raggiunga il limite di capacità e venga limitata.
- Monitoraggio e ottimizzazione regolari:
  - Monitora continuamente le CloudWatch metriche per`ThrottledRequests`.
  - Regola la capacità man mano che il carico di lavoro cambia nel tempo.

Se riscontri un problema `ProvisionedThroughputExceededException` nelle tabelle DynamoDB dei metadati per la tua applicazione consumer KCL, devi aumentare la capacità di throughput assegnata della tabella DynamoDB. Se imposti un certo livello di unità di capacità di lettura (RCU) e di unità di capacità di scrittura (WCU) quando crei per la prima volta l'applicazione consumer, potrebbe non essere sufficiente man mano che l'utilizzo aumenta. Ad esempio, se la tua applicazione consumer KCL esegue checkpoint frequenti o funziona su uno stream con molti shard, potresti aver bisogno di più unità di capacità. Per informazioni sulla velocità effettiva assegnata in DynamoDB, consulta la capacità di throughput di DynamoDB e l'aggiornamento di una tabella nella Amazon DynamoDB Developer Guide.

In che modo KCL assegna i contratti di locazione ai lavoratori e bilancia il carico

KCL raccoglie e monitora continuamente i parametri di utilizzo della CPU dagli host di elaborazione che eseguono i worker per garantire una distribuzione uniforme del carico di lavoro. Queste metriche di utilizzo della CPU sono memorizzate nella tabella delle metriche dei lavoratori in DynamoDB. Se KCL rileva che alcuni lavoratori mostrano tassi di utilizzo della CPU più elevati rispetto ad altri, riassegnerà i contratti di locazione tra i lavoratori per ridurre il carico sui lavoratori più utilizzati. L'obiettivo è bilanciare il carico di lavoro in modo più uniforme su tutto il parco di applicazioni consumer, evitando che ogni singolo lavoratore si sovraccarichi. Poiché KCL distribuisce l'utilizzo della CPU in tutto il parco di applicazioni consumer, è possibile dimensionare correttamente la capacità del parco di applicazioni consumer scegliendo il numero giusto di dipendenti o utilizzare la scalabilità automatica per gestire in modo efficiente la capacità di elaborazione e ottenere costi inferiori.

> **Important**
>
> KCL può raccogliere i parametri di utilizzo della CPU dai lavoratori solo se vengono soddisfatti determinati prerequisiti. Per informazioni dettagliate, consultare Prerequisiti. Se KCL non è in grado di raccogliere i parametri di utilizzo della CPU dai lavoratori, KCL ricorrerà alla velocità effettiva per lavoratore per assegnare i contratti di locazione e bilanciare il carico tra i lavoratori della flotta. KCL monitorerà la produttività che ogni lavoratore riceve in un determinato momento e riassegnerà i leasing per assicurarsi che ogni lavoratore ottenga un livello di produttività totale simile dai leasing assegnati.

Sviluppa i consumatori con KCL

Puoi utilizzare la Kinesis Client Library (KCL) per creare applicazioni consumer che elaborano i dati dai tuoi flussi di dati Kinesis.

KCL è disponibile in più lingue. Questo argomento spiega come sviluppare utenti KCL in linguaggi Java e non Java.

- Per visualizzare il riferimento Javadoc di Kinesis Client Library, consulta Amazon Kinesis Client Library Javadoc.
- Per scaricare KCL per Java da GitHub, consulta Amazon Kinesis Client Library for Java.
- Per individuare KCL per Java su Apache Maven, consulta il KCL Maven Central Repository.

Argomenti

- Sviluppa i consumatori con KCL in Java
- Sviluppa i consumatori con KCL in linguaggi non Java

Sviluppa i consumatori con KCL in Java

Prerequisiti

Prima di iniziare a utilizzare KCL 3.x, assicurati di avere quanto segue:

- Java Development Kit (JDK) 8 o versione successiva
- AWS SDK per Java 2.x
- Maven o Gradle per la gestione delle dipendenze

KCL raccoglie i parametri di utilizzo della CPU, come l'utilizzo della CPU, dall'host di elaborazione su cui lavorano i lavoratori per bilanciare il carico e raggiungere un livello di utilizzo uniforme delle risorse tra i lavoratori. Per consentire a KCL di raccogliere i parametri di utilizzo della CPU dai lavoratori, è necessario soddisfare i seguenti prerequisiti:

Amazon Elastic Compute Cloud(Amazon EC2)

- Il sistema operativo deve essere Linux OS.
- È necessario IMDSv2abilitarlo nella propria EC2 istanza.

Amazon Elastic Container Service (Amazon ECS) su Amazon EC2

- Il sistema operativo deve essere Linux OS.
- È necessario abilitare l'endpoint ECS Task Metadata Endpoint versione 4.
- La versione dell'agente container Amazon ECS deve essere 1.39.0 o successiva.

Amazon ECS su AWS Fargate

- È necessario abilitare l'endpoint dei metadati delle attività Fargate versione 4. Se si utilizza la versione 1.4.0 o successiva della piattaforma Fargate, questa opzione è abilitata per impostazione predefinita.
- Piattaforma Fargate versione 1.4.0 o successiva.

Amazon Elastic Kubernetes Service (Amazon EKS) su Amazon EC2

- Il sistema operativo deve essere Linux OS.

Amazon EKS su AWS Fargate

- Piattaforma Fargate 1.3.0 o versione successiva.

> **Important**
>
> Se KCL non è in grado di raccogliere i parametri di utilizzo della CPU dai lavoratori, ricorrerà alla velocità effettiva per lavoratore per assegnare i contratti di locazione e bilanciare il carico tra i lavoratori della flotta. Per ulteriori informazioni, consulta In che modo KCL assegna i contratti di locazione ai lavoratori e bilancia il carico.

Installa e aggiungi dipendenze

Se usi Maven, aggiungi la seguente dipendenza al tuo file. `pom.xml` Assicurati di aver sostituito 3.x.x con l'ultima versione di KCL.

```
<dependency>
    <groupId>software.amazon.kinesis</groupId>
    <artifactId>amazon-kinesis-client</artifactId>
    <version>3.x.x</version> <!-- Use the latest version -->
</dependency>
```

Se stai usando Gradle, aggiungi quanto segue al tuo file. `build.gradle` Assicurati di aver sostituito 3.x.x con l'ultima versione di KCL.

```
implementation 'software.amazon.kinesis:amazon-kinesis-client:3.x.x'
```

Puoi verificare la versione più recente di KCL sul Maven Central Repository.

Implementa il consumatore

Un'applicazione consumer KCL è composta dai seguenti componenti chiave:

Componenti chiave

- RecordProcessor
- RecordProcessorFactory
- Pianificatore
- Applicazione principale per i consumatori

RecordProcessor

RecordProcessor è il componente principale in cui risiede la logica aziendale per l'elaborazione dei record del flusso di dati Kinesis. Definisce in che modo l'applicazione elabora i dati che riceve dal flusso Kinesis.

Responsabilità chiave:

- Inizializza l'elaborazione per uno shard
- Elabora batch di record dal flusso Kinesis
- Arresta l'elaborazione di uno shard (ad esempio, quando lo shard si divide o si fonde o il leasing viene ceduto a un altro host)
- Gestisci i checkpoint per tenere traccia dei progressi

Di seguito viene mostrato un esempio di implementazione:

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.slf4j.MDC;
import software.amazon.kinesis.exceptions.InvalidStateException;
import software.amazon.kinesis.exceptions.ShutdownException;
import software.amazon.kinesis.lifecycle.events.*;
import software.amazon.kinesis.processor.ShardRecordProcessor;

public class SampleRecordProcessor implements ShardRecordProcessor {
    private static final String SHARD_ID_MDC_KEY = "ShardId";
    private static final Logger log =
 LoggerFactory.getLogger(SampleRecordProcessor.class);
    private String shardId;

    @Override
    public void initialize(InitializationInput initializationInput) {
        shardId = initializationInput.shardId();
        MDC.put(SHARD_ID_MDC_KEY, shardId);
```

```
        try {
            log.info("Initializing @ Sequence: {}",
initializationInput.extendedSequenceNumber());
        } finally {
            MDC.remove(SHARD_ID_MDC_KEY);
        }
    }

    @Override
    public void processRecords(ProcessRecordsInput processRecordsInput) {
        MDC.put(SHARD_ID_MDC_KEY, shardId);
        try {
            log.info("Processing {} record(s)", processRecordsInput.records().size());
            processRecordsInput.records().forEach(r ->
                log.info("Processing record pk: {} -- Seq: {}", r.partitionKey(),
r.sequenceNumber())
            );

            // Checkpoint periodically
            processRecordsInput.checkpointer().checkpoint();
        } catch (Throwable t) {
            log.error("Caught throwable while processing records. Aborting.", t);
        } finally {
            MDC.remove(SHARD_ID_MDC_KEY);
        }
    }

    @Override
    public void leaseLost(LeaseLostInput leaseLostInput) {
        MDC.put(SHARD_ID_MDC_KEY, shardId);
        try {
            log.info("Lost lease, so terminating.");
        } finally {
            MDC.remove(SHARD_ID_MDC_KEY);
        }
    }

    @Override
    public void shardEnded(ShardEndedInput shardEndedInput) {
        MDC.put(SHARD_ID_MDC_KEY, shardId);
        try {
            log.info("Reached shard end checkpointing.");
            shardEndedInput.checkpointer().checkpoint();
        } catch (ShutdownException | InvalidStateException e) {
```

```
            log.error("Exception while checkpointing at shard end. Giving up.", e);
        } finally {
            MDC.remove(SHARD_ID_MDC_KEY);
        }
    }

    @Override
    public void shutdownRequested(ShutdownRequestedInput shutdownRequestedInput) {
        MDC.put(SHARD_ID_MDC_KEY, shardId);
        try {
            log.info("Scheduler is shutting down, checkpointing.");
            shutdownRequestedInput.checkpointer().checkpoint();
        } catch (ShutdownException | InvalidStateException e) {
            log.error("Exception while checkpointing at requested shutdown. Giving
 up.", e);
        } finally {
            MDC.remove(SHARD_ID_MDC_KEY);
        }
    }
}
```

Di seguito è riportata una spiegazione dettagliata di ogni metodo utilizzato nell'esempio:

inizializza (InitializationInputinitializationInput)

- Scopo: impostare le risorse o lo stato necessari per l'elaborazione dei record.
- Quando viene chiamato: una volta, quando KCL assegna uno shard a questo registratore.
- Punti chiave:
  - `initializationInput.shardId()`: L'ID dello shard che questo processore gestirà.
  - `initializationInput.extendedSequenceNumber()`: Il numero di sequenza da cui iniziare l'elaborazione.

ProcessRecords () ProcessRecordsInput processRecordsInput

- Scopo: Elaborare i record in entrata e, facoltativamente, controllare l'avanzamento del checkpoint.
- Quando viene chiamato: ripetutamente, purché il processore di dischi detenga il contratto di locazione dello shard.
- Punti chiave:
  - `processRecordsInput.records()`: Elenco dei record da elaborare.

- `processRecordsInput.checkpointer()`: utilizzato per controllare lo stato di avanzamento.
- Assicurati di aver gestito tutte le eccezioni durante l'elaborazione per evitare che KCL fallisca.
- Questo metodo dovrebbe essere idempotente, poiché lo stesso record può essere elaborato più di una volta in alcuni scenari, ad esempio dati che non sono stati sottoposti a checkpoint prima che un lavoratore si blocchi o riavvii imprevisti.
- Svuota sempre tutti i dati memorizzati nel buffer prima del checkpoint per garantire la coerenza dei dati.

LeaseLostInput leaseLostInputleaseLost ()

- Scopo: ripulire tutte le risorse specifiche per l'elaborazione di questo frammento.
- Quando viene chiamato: quando un altro Scheduler assume il contratto di locazione di questo shard.
- Punti chiave:
  - Il checkpoint non è consentito con questo metodo.

shardEnded () ShardEndedInput shardEndedInput

- Scopo: completare l'elaborazione di questo frammento e di questo checkpoint.
- Quando viene chiamato: quando lo shard si divide o si fonde, indica che tutti i dati relativi allo shard sono stati elaborati.
- Punti chiave:
  - `shardEndedInput.checkpointer()`: utilizzato per eseguire il checkpoint finale.
  - Il checkpoint in questo metodo è obbligatorio per completare l'elaborazione.
  - Il mancato scarico dei dati e del checkpoint in questo campo può causare la perdita dei dati o un'elaborazione duplicata alla riapertura dello shard.

ShutdownRequestedInput shutdownRequestedInputShutdownRequested ()

- Scopo: controllare e ripulire le risorse quando KCL si spegne.
- Quando viene chiamato: quando KCL si spegne, ad esempio quando l'applicazione è in fase di chiusura).
- Punti chiave:

- `shutdownRequestedInput.checkpointer()`: utilizzato per eseguire il checkpoint prima dello spegnimento.
- Assicurati di aver implementato il checkpoint nel metodo in modo che i progressi vengano salvati prima che l'applicazione si fermi.
- Il mancato ripristino dei dati e del checkpoint in questo campo può comportare la perdita dei dati o la rielaborazione dei record al riavvio dell'applicazione.

> **Important**
>
> KCL 3.x garantisce un minor numero di ritrattamenti dei dati quando il contratto di locazione viene ceduto da un lavoratore a un altro lavoratore effettuando un checkpoint prima che il lavoratore precedente venga chiuso. Se non implementate la logica di checkpoint nel `shutdownRequested()` metodo, non vedrete questo vantaggio. Assicurati di aver implementato una logica di checkpoint all'interno del metodo. `shutdownRequested()`

RecordProcessorFactory

RecordProcessorFactory è responsabile della creazione di nuove RecordProcessor istanze. KCL utilizza questa factory per crearne una nuova RecordProcessor per ogni shard che l'applicazione deve elaborare.

Responsabilità chiave:

- Crea nuove RecordProcessor istanze su richiesta
- Assicurati che ognuna RecordProcessor sia inizializzata correttamente

Di seguito è riportato un esempio di implementazione:

```
import software.amazon.kinesis.processor.ShardRecordProcessor;
import software.amazon.kinesis.processor.ShardRecordProcessorFactory;

public class SampleRecordProcessorFactory implements ShardRecordProcessorFactory {
    @Override
    public ShardRecordProcessor shardRecordProcessor() {
        return new SampleRecordProcessor();
    }
}
```

In questo esempio, la factory ne crea una nuova SampleRecordProcessor ogni volta che viene chiamata shardRecordProcessor (). È possibile estenderlo per includere qualsiasi logica di inizializzazione necessaria.

Pianificatore

Scheduler è un componente di alto livello che coordina tutte le attività dell'applicazione KCL. È responsabile dell'orchestrazione complessiva dell'elaborazione dei dati.

Responsabilità chiave:

- Gestire il ciclo di vita di RecordProcessors
- Gestisci la gestione del leasing per i frammenti
- Coordina il checkpoint
- Bilanciate il carico di elaborazione degli shard tra i diversi worker della vostra applicazione
- Gestisci segnali di spegnimento e terminazione delle applicazioni senza interruzioni

Scheduler viene in genere creato e avviato nell'applicazione principale. È possibile controllare l'esempio di implementazione di Scheduler nella sezione seguente, Main Consumer Application.

Applicazione principale per i consumatori

L'applicazione principale per i consumatori collega tutti i componenti. È responsabile della configurazione del consumatore KCL, della creazione dei client necessari, della configurazione dello Scheduler e della gestione del ciclo di vita dell'applicazione.

Responsabilità chiave:

- Configurare i client AWS di servizio (Kinesis, DynamoDB,) CloudWatch
- Configurare l'applicazione KCL
- Crea e avvia lo Scheduler
- Gestisci l'arresto dell'applicazione

Di seguito è riportato un esempio di implementazione:

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cloudwatch.CloudWatchAsyncClient;
import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;
import software.amazon.awssdk.services.kinesis.KinesisAsyncClient;
```

```
import software.amazon.kinesis.common.ConfigsBuilder;
import software.amazon.kinesis.common.KinesisClientUtil;
import software.amazon.kinesis.coordinator.Scheduler;
import java.util.UUID;

public class SampleConsumer {
    private final String streamName;
    private final Region region;
    private final KinesisAsyncClient kinesisClient;

    public SampleConsumer(String streamName, Region region) {
        this.streamName = streamName;
        this.region = region;
        this.kinesisClient =
 KinesisClientUtil.createKinesisAsyncClient(KinesisAsyncClient.builder().region(this.region));
    }

    public void run() {
        DynamoDbAsyncClient dynamoDbAsyncClient =
 DynamoDbAsyncClient.builder().region(region).build();
        CloudWatchAsyncClient cloudWatchClient =
 CloudWatchAsyncClient.builder().region(region).build();

        ConfigsBuilder configsBuilder = new ConfigsBuilder(
            streamName,
            streamName,
            kinesisClient,
            dynamoDbAsyncClient,
            cloudWatchClient,
            UUID.randomUUID().toString(),
            new SampleRecordProcessorFactory()
        );

        Scheduler scheduler = new Scheduler(
            configsBuilder.checkpointConfig(),
            configsBuilder.coordinatorConfig(),
            configsBuilder.leaseManagementConfig(),
            configsBuilder.lifecycleConfig(),
            configsBuilder.metricsConfig(),
            configsBuilder.processorConfig(),
            configsBuilder.retrievalConfig()
        );

        Thread schedulerThread = new Thread(scheduler);
```

```
        schedulerThread.setDaemon(true);
        schedulerThread.start();
    }

    public static void main(String[] args) {
        String streamName = "your-stream-name"; // replace with your stream name
        Region region = Region.US_EAST_1; // replace with your region
        new SampleConsumer(streamName, region).run();
    }
}
```

Per impostazione predefinita, KCL crea un consumatore Enhanced Fan-out (EFO) con throughput dedicato. Per ulteriori informazioni su Enhanced Fan-out, consulta. Sviluppa consumatori con fan-out migliorati con un throughput dedicato Se hai meno di 2 consumatori o non hai bisogno di ritardi di propagazione in lettura inferiori a 200 ms, devi impostare la seguente configurazione nell'oggetto scheduler per utilizzare i consumatori a throughput condiviso:

```
configsBuilder.retrievalConfig().retrievalSpecificConfig(new PollingConfig(streamName,
 kinesisClient))
```

Il codice seguente è un esempio di creazione di un oggetto scheduler che utilizza consumatori a throughput condiviso:

Importazioni:

```
import software.amazon.kinesis.retrieval.polling.PollingConfig;
```

Codice:

```
Scheduler scheduler = new Scheduler(
            configsBuilder.checkpointConfig(),
            configsBuilder.coordinatorConfig(),
            configsBuilder.leaseManagementConfig(),
            configsBuilder.lifecycleConfig(),
            configsBuilder.metricsConfig(),
            configsBuilder.processorConfig(),
            configsBuilder.retrievalConfig().retrievalSpecificConfig(new
 PollingConfig(streamName, kinesisClient))
        );/
```

Sviluppa i consumatori con KCL in linguaggi non Java

Questa sezione tratta l'implementazione dei consumatori che utilizzano Kinesis Client Library (KCL) in Python, Node.js, .NET e Ruby.

KCL è una libreria Java. Il supporto per linguaggi diversi da Java viene fornito utilizzando un'interfaccia multilingue chiamata. `MultiLangDaemon` Questo demone è basato su Java e viene eseguito in background quando si utilizza un KCL con un linguaggio diverso da Java. Pertanto, se installi KCL per linguaggi non Java e scrivi la tua app consumer interamente in linguaggi non Java, avrai comunque bisogno che Java sia installato sul tuo sistema a causa di. `MultiLangDaemon` Inoltre, `MultiLangDaemon` presenta alcune impostazioni predefinite che potresti dover personalizzare per il tuo caso d'uso (ad esempio, la regione AWS a cui si connette). Per ulteriori informazioni su `MultiLangDaemon` on GitHub, consulta il MultiLangDaemon progetto KCL.

Sebbene i concetti fondamentali rimangano gli stessi in tutte le lingue, ci sono alcune considerazioni e implementazioni specifiche per ciascuna lingua. Per i concetti fondamentali sullo sviluppo di KCL per i consumatori, vedi. Sviluppa i consumatori con KCL in Java Per informazioni più dettagliate su come sviluppare consumatori KCL in Python, Node.js, .NET e Ruby e sugli ultimi aggiornamenti, fai riferimento ai seguenti repository: GitHub

- Python: amazon-kinesis-client-python
- Node.js: amazon-kinesis-client-nodejs
- .NET: amazon-kinesis-client-net
- Rubino: amazon-kinesis-client-ruby

> **Important**
>
> Non utilizzare le seguenti versioni della libreria KCL non Java se utilizzi JDK 8. Queste versioni contengono una dipendenza (logback) incompatibile con JDK 8.
>
> - KCL Python 3.0.2 e 2.2.0
> - KCL Node.js 2.3.0
> - KCL.NET 3.1.0
> - KCL Ruby 2.2.0

Si consiglia di utilizzare versioni rilasciate prima o dopo queste versioni interessate quando si lavora con JDK 8.

Elaborazione multi-stream con KCL

Questa sezione descrive le modifiche richieste in KCL che consentono di creare applicazioni consumer KCL in grado di elaborare più di un flusso di dati contemporaneamente.

Important

- L'elaborazione multi-stream è supportata solo in KCL 2.3 o versioni successive.
- L'elaborazione multi-stream non è supportata per i consumatori KCL scritti in linguaggi non Java che funzionano con. `multilangdaemon`
- L'elaborazione multi-stream non è supportata in nessuna versione di KCL 1.x.

- MultistreamTracker interfaccia
  - Per creare un'applicazione consumer in grado di elaborare più flussi contemporaneamente, è necessario implementare una nuova interfaccia denominata MultistreamTracker. Questa interfaccia include il metodo `streamConfigList` che restituisce l'elenco dei flussi di dati e le relative configurazioni che devono essere elaborati dall'applicazione consumer KCL. Si noti che i flussi di dati in fase di elaborazione possono essere modificati durante il runtime dell'applicazione consumer. `streamConfigList`viene chiamato periodicamente da KCL per conoscere le modifiche nei flussi di dati da elaborare.
  - Compila `streamConfigList` l'elenco. StreamConfig

```
package software.amazon.kinesis.common;

import lombok.Data;
import lombok.experimental.Accessors;

@Data
@Accessors(fluent = true)
public class StreamConfig {
    private final StreamIdentifier streamIdentifier;
```

```
    private final InitialPositionInStreamExtended initialPositionInStreamExtended;
    private String consumerArn;
}
```

- I campi `StreamIdentifier` e `InitialPositionInStreamExtended` sono obbligatori, mentre `consumerArn` è facoltativo. È necessario fornire i dati `consumerArn` solo se si utilizza KCL per implementare un'applicazione utente fan-out avanzata.
- Per ulteriori informazioni su`StreamIdentifier`, vedere https://github.com/awslabs/amazon-kinesis-client/blob/v2.5.8/amazon-kinesis-client/src/main/java/software/amazon/kinesis/common/StreamIdentifier.java #L129. Per creare un`StreamIdentifier`, ti consigliamo di creare un'istanza multistream da `streamArn` and the `streamCreationEpoch` disponibile in KCL 2.5.0 o versioni successive. In KCL v2.3 e v2.4, che non supportano`streamArm`, crea un'istanza multistream utilizzando il formato. `account-id:StreamName:streamCreationTimestamp` Questo formato sarà obsoleto e non sarà più supportato a partire dalla prossima versione principale.
- MultistreamTracker include anche una strategia per eliminare i leasing di vecchi stream nella tabella dei lease (). formerStreamsLeases DeletionStrategy Si noti che la strategia NON PUÒ essere modificata durante il runtime dell'applicazione consumer. Per ulteriori informazioni, vedete https://github.com/awslabs/ amazon-kinesis-client / blob/0c5042dadf794fe988438436252a5a8fe70b6b0b/ .java. amazon-kinesis-client src/main/java/ software/amazon/kinesis/processor/FormerStreamsLeasesDeletionStrategy

•

Oppure puoi inizializzare ConfigsBuilder con `MultiStreamTracker` se desideri implementare un'applicazione consumer KCL che elabora più flussi contemporaneamente.

```
* Constructor to initialize ConfigsBuilder with MultiStreamTracker
     * @param multiStreamTracker
     * @param applicationName
     * @param kinesisClient
     * @param dynamoDBClient
     * @param cloudWatchClient
     * @param workerIdentifier
     * @param shardRecordProcessorFactory
     */
```

```
   public ConfigsBuilder(@NonNull MultiStreamTracker multiStreamTracker, @NonNull
String applicationName,
           @NonNull KinesisAsyncClient kinesisClient, @NonNull DynamoDbAsyncClient
dynamoDBClient,
           @NonNull CloudWatchAsyncClient cloudWatchClient, @NonNull String
workerIdentifier,
           @NonNull ShardRecordProcessorFactory shardRecordProcessorFactory) {
       this.appStreamTracker = Either.left(multiStreamTracker);
       this.applicationName = applicationName;
       this.kinesisClient = kinesisClient;
       this.dynamoDBClient = dynamoDBClient;
       this.cloudWatchClient = cloudWatchClient;
       this.workerIdentifier = workerIdentifier;
       this.shardRecordProcessorFactory = shardRecordProcessorFactory;
   }
```

- Con il supporto multi-stream implementato per la tua applicazione consumer KCL, ogni riga della tabella di lease dell'applicazione ora contiene l'ID dello shard e il nome del flusso dei molteplici flussi di dati elaborati da questa applicazione.
- Quando viene implementato il supporto multi-stream per la tua applicazione consumer KCL, LeaseKey assume la seguente struttura:. `account-id:StreamName:streamCreationTimestamp:ShardId` Ad esempio, `111111111:multiStreamTest-1:12345:shardId-000000000336`.

> **Important**
>
> Quando la tua applicazione consumer KCL esistente è configurata per elaborare solo un flusso di dati, `leaseKey` (che è la chiave di partizione per la tabella di leasing) è lo shard ID. Se riconfigurate un'applicazione consumer KCL esistente per elaborare più flussi di dati, la tabella di leasing si interrompe, perché la `leaseKey` struttura deve essere la seguente: per supportare il multi-stream. `account-id:StreamName:StreamCreationTimestamp:ShardId`

Usa il registro Schema con KCL AWS Glue

È possibile integrare Kinesis Data Streams AWS Glue con il registro Schema. Il registro AWS Glue Schema consente di scoprire, controllare ed evolvere centralmente gli schemi, garantendo al

contempo che i dati prodotti siano continuamente convalidati da uno schema registrato. Uno schema definisce la struttura e il formato di un registro di dati. Uno schema è una specifica con versioni per la pubblicazione, il consumo o l'archiviazione dei dati in modo affidabile. Il registro AWS Glue Schema consente di migliorare la qualità e la governance end-to-end dei dati all'interno delle applicazioni di streaming. Per ulteriori informazioni, consulta Registro degli schemi di AWS Glue. Uno dei modi per configurare questa integrazione è tramite KCL per Java.

> **Important**
>
> - AWS Glue L'integrazione del registro Schema per Kinesis Data Streams è supportata solo in KCL 2.3 o versioni successive.
> - AWS Glue L'integrazione del registro dello schema per Kinesis Data Streams non è supportata per gli utenti KCL scritti in linguaggi non Java che funzionano con. `multilangdaemon`
> - AWS Glue L'integrazione del registro dello schema per Kinesis Data Streams non è supportata in nessuna versione di KCL 1.x.

Per istruzioni dettagliate su come configurare l'integrazione di Kinesis Data Streams con il registro AWS Glue Schema utilizzando KCL, consulta la sezione «Interazione con i dati utilizzando le librerie KPL/KCL» in Caso d'uso: integrazione di Amazon Kinesis Data Streams con lo Schema Registry. AWS Glue

Autorizzazioni IAM richieste per le applicazioni consumer KCL

Devi aggiungere le seguenti autorizzazioni al ruolo o all'utente IAM associato alla tua applicazione consumer KCL.

Le migliori pratiche di sicurezza per AWS imporre l'uso di autorizzazioni granulari per controllare l'accesso a diverse risorse. AWS Identity and Access Management (IAM) consente di gestire gli utenti e le autorizzazioni degli utenti in. AWS Una policy IAM elenca in modo esplicito le operazioni consentite e le risorse per le quali sono applicabili le operazioni.

La tabella seguente mostra le autorizzazioni IAM minime generalmente richieste per le applicazioni consumer KCL:

Autorizzazioni IAM minime per le applicazioni consumer KCL

Servizio	Azioni	Risorse () ARNs	Scopo

Flusso di dati Amazon Kinesis	`DescribeStream`
	
	`DescribeStreamSummary`
	
	`RegisterStreamConsumer`
		Flusso di dati Kinesis da cui l'applicazione KCL elaborerà i dati.
		
		`arn:aws:kinesis:region:account:stream/StreamName`
			Prima di leggere i record, il consumer verifica se il flusso di dati esiste, è attivo e contiene il flusso di dati.
			
			Registra i consumatori su uno shard.

Flusso di dati Amazon Kinesis	`GetRecords`
	
	`GetShardIterator`
	
	`ListShards`
		Flusso di dati Kinesis da cui l'applicazione KCL elaborerà i dati.
		
		`arn:aws:kinesis:region:account:stream/StreamName`
			Legge record da uno shard.

Flusso di dati Amazon Kinesis	`SubscribeToShard`
	
	`DescribeStreamConsumer`
		Flusso di dati Kinesis da cui l'applicazione KCL elaborerà i dati. Aggiungi questa azione solo se utilizzi utenti Enhanced Fan-Out (EFO).
		
		`arn:aws:kinesis:region:account:stream/StreamName/consumer/*`
			Si abbona a uno shard per consumatori Enhanced Fan-out (EFO).

Servizio	Azioni	Risorse () ARNs	Scopo

Amazon DynamoDB	`CreateTable`
	`DescribeTable`
	`UpdateTable`
	`Scan`
	`GetItem`
	`PutItem`
	`UpdateItem`
	`DeleteItem`
		Tabella di leasing (tabella dei metadati) in DynamoDB creata da KCL.
		`arn:aws:dynamodb:region:account:table/KCLApplicationName`
			Queste azioni sono necessarie per consentire a KCL di gestire la tabella di lease creata in DynamoDB.

Servizio	Azioni	Risorse () ARNs	Scopo

Amazon DynamoDB	`CreateTable`
	
	`DescribeTable`
	
	`Scan`
	
	`GetItem`
	
	`PutItem`
	
	`UpdateItem`
	
	`DeleteItem`
		Metriche dei lavoratori e tabella dello stato del coordinatore (tabelle di metadati in DynamoDB) creata da KCL.
		
		`arn:aws:dynamodb:region:account:table/KCLApplicationName-WorkerMetricStats`
		
		`arn:aws:dynamodb:region:account:table/KCLApplicationName-CoordinatorState`
			Queste azioni sono necessarie per consentire a KCL di gestire le metriche dei lavoratori e le tabelle dei metadati dello stato del coordinatore in DynamoDB.

Servizio	Azioni	Risorse () ARNs	Scopo
Amazon DynamoDB	`Query`	Indice secondari o globale nella tabella dei contratti di locazione. `arn:aws:dynamodb:region:account:table/KCLApplicationName/index/*`	Questa azione è necessaria affinché KCL legga l'indice secondario globale della tabella di lease creata in DynamoDB.
Amazon CloudWatch	`PutMetricData`	*	Carica metriche utili per monitorar e l'applicazione. CloudWatch L'asterisco (*) viene utilizzato perché non esiste una risorsa specifica in CloudWatch cui viene richiamata l'azione. `PutMetricData`

> **Note**
>
> Sostituisci «region», «account»StreamName, "» e "KCLApplicationName» ARNs con il tuo Account AWS numero Regione AWS, il nome del flusso di dati Kinesis e il nome dell'applicazione KCL rispettivamente. KCL 3.x crea altre due tabelle di metadati in DynamoDB. Per informazioni dettagliate sulle tabelle di metadati DynamoDB create da KCL, vedere. Tabelle di metadati DynamoDB e bilanciamento del carico in KCL Se utilizzi configurazioni per personalizzare i nomi delle tabelle di metadati create da KCL, usa i nomi di tabella specificati anziché il nome dell'applicazione KCL.

Di seguito è riportato un esempio di documento politico per un'applicazione consumer KCL.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kinesis:DescribeStream",
                "kinesis:DescribeStreamSummary",
                "kinesis:RegisterStreamConsumer",
                "kinesis:GetRecords",
                "kinesis:GetShardIterator",
                "kinesis:ListShards"
            ],
            "Resource": "arn:aws:kinesis:REGION:ACCOUNT_ID:stream/STREAM_NAME"
        },
        {
            "Effect": "Allow",
            "Action": [
                "kinesis:SubscribeToShard",
                "kinesis:DescribeStreamConsumer"
            ],
            "Resource": "arn:aws:kinesis:REGION:ACCOUNT_ID:stream/STREAM_NAME/consumer/
*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:CreateTable",
                "dynamodb:DescribeTable",
                "dynamodb:UpdateTable",
                "dynamodb:GetItem",
                "dynamodb:UpdateItem",
                "dynamodb:PutItem",
                "dynamodb:DeleteItem",
                "dynamodb:Scan"
            ],
            "Resource": [
                "arn:aws:dynamodb:REGION:ACCOUNT_ID:table/KCL_APPLICATION_NAME"
            ]
        },
        {
            "Effect": "Allow",
```

```
            "Action": [
                "dynamodb:CreateTable",
                "dynamodb:DescribeTable",
                "dynamodb:GetItem",
                "dynamodb:UpdateItem",
                "dynamodb:PutItem",
                "dynamodb:DeleteItem",
                "dynamodb:Scan"
            ],
            "Resource": [
                "arn:aws:dynamodb:REGION:ACCOUNT_ID:table/KCL_APPLICATION_NAME-
WorkerMetricStats",
                "arn:aws:dynamodb:REGION:ACCOUNT_ID:table/KCL_APPLICATION_NAME-
CoordinatorState"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:Query"
            ],
            "Resource": [
                "arn:aws:dynamodb:REGION:ACCOUNT_ID:table/KCL_APPLICATION_NAME/index/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:PutMetricData"
            ],
            "Resource": "*"
        }
    ]
}
```

Prima di utilizzare questa politica di esempio, controlla i seguenti elementi:

- Sostituisci REGION con il tuo Regione AWS (ad esempio, us-east-1).
- Sostituisci ACCOUNT_ID con il tuo ID. Account AWS
- Sostituisci STREAM_NAME con il nome del tuo flusso di dati Kinesis.
- Sostituisci CONSUMER_NAME con il nome del tuo consumatore, in genere il nome dell'applicazione quando usi KCL.

- Sostituisci KCL_APPLICATION_NAME con il nome della tua applicazione KCL.

Configurazioni KCL

Puoi impostare le proprietà di configurazione per personalizzare la funzionalità di Kinesis Client Library per soddisfare i tuoi requisiti specifici. La tabella seguente descrive le proprietà e le classi di configurazione.

> **Important**
>
> In KCL 3.x, l'algoritmo di bilanciamento del carico mira a ottenere un utilizzo uniforme della CPU tra i lavoratori, non un numero uguale di leasing per lavoratore. Se impostato su un valore `maxLeasesForWorker` troppo basso, potresti limitare la capacità di KCL di bilanciare efficacemente il carico di lavoro. Se utilizzi la `maxLeasesForWorker` configurazione, valuta la possibilità di aumentarne il valore per consentire la migliore distribuzione del carico possibile.

Questa tabella mostra le proprietà di configurazione per KCL

Proprietà di configurazione	Classe di configurazione	Descrizione	Valore predefinito
`applicationName`	ConfigsBuilder	Il nome per l'applicazione della KCL. Utilizzato come predefinito per `tableName` e `consumerName` .	Non applicabile
`tableName`	ConfigsBuilder	Consente di ignorare il nome della tabella utilizzato per la tabella di lease di Amazon DynamoDB.	Non applicabile

Proprietà di configurazione	Classe di configurazione	Descrizione	Valore predefinito
`streamName`	ConfigsBuilder	Il nome del flusso dal quale l'applicazione elabora i record.	Non applicabile
`workerIdentifier`	ConfigsBuilder	Un identificatore univoco che rappresenta la creazione dell'elaboratore di applicazione. Deve essere univoco.	Non applicabile
`failoverTimeMillis`	LeaseManagementConfig	Il numero di millisecondi che devono passare prima di poter considerare un proprietario di lease come fallito. Per le applicazioni che hanno un numero elevato di shard, questo può essere impostato su un numero più alto per ridurre il numero di IOPS DynamoDB necessari per il tracciamento dei leasing.	10.000 (10 secondi)
`shardSyncIntervalMillis`	LeaseManagementConfig	Il periodo di tempo tra le chiamate di sincronizzazione dello shard.	60.000 (60 secondi)

Proprietà di configurazione	Classe di configurazione	Descrizione	Valore predefinito
`cleanupLeasesUponShardCompletion`	LeaseManagementConfig	Quando impostati, i lease vengono rimossi non appena i lease figlio hanno iniziato l'elaborazione.	TRUE
`ignoreUnexpectedChildShards`	LeaseManagementConfig	Quando impostato, i shard figlio che hanno un shard aperto vengono ignorati. Questo è principalmente per DynamoDB Streams.	FALSE

Proprietà di configurazione	Classe di configurazione	Descrizione	Valore predefinito
`maxLeasesForWorker`	LeaseManagementConfig	Il numero massimo di contratti di locazione che un singolo lavoratore deve accettare. Un valore troppo basso può causare la perdita di dati se i lavoratori non sono in grado di elaborare tutti gli shard e portare a un'assegnazione del leasing non ottimale tra i lavoratori. Al momento della configurazione, tenete conto del numero totale di shard, del numero di lavoratori e della capacità di elaborazione dei worker.	Illimitato
`maxLeaseRenewalThreads`	LeaseManagementConfig	Controlla le dimensioni del pool di thread di rinnovo del lease. Quanto maggiori sono i lease che può richiedere l'applicazione, tanto più grande deve essere questo pool.	20

Proprietà di configurazione	Classe di configurazione	Descrizione	Valore predefinito
`billingMode`	LeaseManagementConfig	Determina la modalità di capacità della tabella di leasing creata in DynamoDB. Sono disponibili due opzioni: modalità on-demand (PAY_PER_REQUEST) e modalità provisioned. Si consiglia di utilizzare l'impostazione predefinita della modalità on demand, poiché si ridimensiona automaticamente per adattarsi al carico di lavoro senza la necessità di pianificare la capacità.	PAY_PER_REQUEST (modalità su richiesta)

Proprietà di configurazione	Classe di configurazione	Descrizione	Valore predefinito
`initialLeaseTableReadCapacity`	LeaseManagementConfig	La capacità di lettura di DynamoDB utilizzata se la Kinesis Client Library deve creare una nuova tabella di lease DynamoDB con modalità di capacità fornita. È possibile ignorare questa configurazione se si utilizza la modalità di capacità on-demand predefinita nella configurazione. `billingMode`	10
`initialLeaseTableWriteCapacity`	LeaseManagementConfig	La capacità di lettura di DynamoDB utilizzata se la Kinesis Client Library deve creare una nuova tabella di lease DynamoDB. È possibile ignorare questa configurazione se si utilizza la modalità di capacità on-demand predefinita nella configurazione. `billingMode`	10

Proprietà di configurazione	Classe di configurazione	Descrizione	Valore predefinito
`initialPositionInStreamExtended`	LeaseManagementConfig	La posizione iniziale nel flusso nella quale l'applicazione dovrebbe iniziare. Questo viene utilizzato soltanto durante la creazione del lease iniziale.	InitialPositionInStream.TRIM_HORIZON
`reBalanceThresholdPercentage`	LeaseManagementConfig	Un valore percentuale che determina quando l'algoritmo di bilanciamento del carico deve prendere in considerazione la riassegnazione degli shard tra i lavoratori. Questa è una nuova configurazione introdotta in KCL 3.x.	10
`dampeningPercentage`	LeaseManagementConfig	Un valore percentuale utilizzato per attenuare la quantità di carico che verrà spostata dal lavoratore sovraccarico in un'unica operazione di ribilanciamento. Questa è una nuova configurazione introdotta in KCL 3.x.	60

Proprietà di configurazione	Classe di configurazione	Descrizione	Valore predefinito
`allowThroughputOvershoot`	LeaseManagementConfig	Determina se è ancora necessario richiedere un leasing aggiuntivo al lavoratore sovraccarico, anche se ciò fa sì che la quantità totale della produzione di leasing utilizzata superi la quantità di produzione desiderata. Questa è una nuova configurazione introdotta in KCL 3.x.	TRUE

Proprietà di configurazione	Classe di configurazione	Descrizione	Valore predefinito
`disableWorkerMetrics`	LeaseManagementConfig	Determina se KCL deve ignorare le metriche relative alle risorse fornite dai lavoratori (come l'utilizzo della CPU) durante la riassegnazione dei leasing e il bilanciamento del carico. Impostalo su TRUE se vuoi impedire a KCL di bilanciare il carico in base all'utilizzo della CPU. Questa è una nuova configurazione introdotta in KCL 3.x.	FALSE
`maxThroughputPerHostKBps`	LeaseManagementConfig	Importo della produttività massima da assegnare a un lavoratore durante l'assegnazione del leasing. Questa è una nuova configurazione introdotta in KCL 3.x.	Illimitato

Proprietà di configurazione	Classe di configurazione	Descrizione	Valore predefinito
`isGracefulLeaseHandoffEnabled`	LeaseManagementConfig	Controlla il comportamento della cessione del contratto di locazione tra i lavoratori. Se impostato su true, KCL tenterà di trasferire i contratti di locazione in modo corretto concedendo allo shard il tempo RecordProcessor sufficiente per completare l'elaborazione prima di cedere il contratto di locazione a un altro lavoratore. Ciò può contribuire a garantire l'integrità dei dati e le transizioni fluide, ma può aumentare i tempi di consegna. Se impostato su false, il contratto di locazione verrà ceduto immediatamente senza attendere che RecordProcessor venga chiuso correttamente. Ciò può portare a	TRUE

Proprietà di configura zione	Classe di configura zione	Descrizione	Valore predefinito
		consegne più rapide, ma può comportare il rischio di un'elabor azione incompleta. Nota: il checkpoin ting deve essere implementato all'inter no del metodo shutdownRequested () di RecordProcessor per trarre vantaggio dalla funzional ità graceful lease handoff. Questa è una nuova configurazione introdotta in KCL 3.x.	

Proprietà di configurazione	Classe di configurazione	Descrizione	Valore predefinito
`gracefulLeaseHandoffTimeoutMillis`	LeaseManagementConfig	Speciifica il tempo minimo (in millisecondi) di attesa che lo shard corrente si spenga correttamente prima di RecordProcessor trasferire forzatamente il lease al proprietario successivo. Se il metodo ProcessRecords in genere dura più a lungo del valore predefinito, è consigliabile aumentare questa impostazione. Ciò garantisce che RecordProcessor disponga di tempo sufficiente per completare l'elaborazione prima che avvenga il trasferimento del leasing. Questa è una nuova configurazione introdotta in KCL 3.x.	30.000 (30 secondi)

Proprietà di configurazione	Classe di configurazione	Descrizione	Valore predefinito
`maxRecords`	PollingConfig	Consente di impostare il numero massimo di record restituiti da Kinesis.	10.000
`retryGetRecordsInSeconds`	PollingConfig	Configura il ritardo tra i GetRecords tentativi di errore.	Nessuno
`maxGetRecordsThreadPool`	PollingConfig	La dimensione del pool di thread utilizzato per. GetRecords	Nessuno
`idleTimeBetweenReadsInMillis`	PollingConfig	Determina il tempo di attesa di KCL tra una GetRecords chiamata e l'altra per estrarre i dati dai flussi di dati. L'unità è in millisecondi.	1.500
`callProcessRecordsEvenForEmptyRecordList`	ProcessorConfig	Quando impostato, l'elaboratore di record viene chiamato anche quando nessun record è stato fornito da .	FALSE
`parentShardPollIntervalMillis`	CoordinatorConfig	Con quale frequenza un elaboratore di record deve eseguire il polling per vedere se il shard padre è stata completato. L'unità è in millisecondi.	10.000 (10 secondi)

Proprietà di configurazione	Classe di configurazione	Descrizione	Valore predefinito
`skipShardSyncAtWorkerInitializationIfLeaseExist`	CoordinatorConfig	Disabilita la sincronizzazione dei dati shard se la tabella di lease contiene lease esistenti.	FALSE
`shardPrioritization`	CoordinatorConfig	Quale prioritizzazione shard utilizzare.	NoOpShardPrioritization
`ClientVersionConfig`	CoordinatorConfig	Determina in quale modalità di compatibilità della versione KCL verrà eseguita l'applicazione. Questa configurazione è solo per la migrazione e dalle versioni precedenti di KCL. Quando si esegue la migrazione a 3.x, è necessario impostare questa configurazione su. `CLIENT_VERSION_CONFIG_COMPATIBLE_WITH_2X` È possibile rimuovere questa configurazione una volta completata la migrazione.	CLIENT_VERSION_CONFIG_3X

Proprietà di configurazione	Classe di configurazione	Descrizione	Valore predefinito
`taskBackoffTimeMillis`	LifecycleConfig	Il tempo di attesa per riprovare le attività KCL non riuscite. L'unità è in millisecondi.	500 (0,5 secondi)
`logWarningForTaskAfterMillis`	LifecycleConfig	Quanto tempo bisogna attendere prima che venga registrato un avviso se un'attività non è stata completata.	Nessuno
`listShardsBackoffTimeInMillis`	RetrievalConfig	Il numero di millisecondi di attesa tra le chiamate in `ListShards` quando si verificano errori. L'unità è in millisecondi.	1.500 (1,5 secondi)
`maxListShardsRetryAttempts`	RetrievalConfig	Il numero massimo di volte che `ListShards` effettua nuovi tentativi prima di desistere.	50
`metricsBufferTimeMillis`	MetricsConfig	Specifica la durata massima (in millisecondi) per memorizzare le metriche nel buffer prima di pubblicarle su. CloudWatch	10.000 (10 secondi)

Proprietà di configurazione	Classe di configurazione	Descrizione	Valore predefinito
`metricsMaxQueueSize`	MetricsConfig	Specifica il numero massimo di metriche da inserire nel buffer prima della pubblicazione. CloudWatch	10.000
`metricsLevel`	MetricsConfig	Specifica il livello di granularità delle CloudWatch metriche da abilitare e pubblicare. Valori possibili: NONE, SUMMARY, DETAILED.	MetricsLevel.DETTAGLIATO
`metricsEnabledDimensions`	MetricsConfig	Controlla le dimensioni consentite per le CloudWatch metriche.	Tutte le dimensioni

Configurazioni fuori produzione in KCL 3.x

Le seguenti proprietà di configurazione non sono più disponibili in KCL 3.x:

La tabella mostra le proprietà di configurazione interrotte per KCL 3.x

Proprietà di configurazione	Classe di configurazione	Descrizione
`maxLeasesToStealAtOneTime`	LeaseManagementConfig	Il numero massimo di lease che un'applicazione deve tentare di intercettare simultaneamente. KCL 3.x ignorerà questa configurazione e riassegnerà i leasing

Proprietà di configurazione	Classe di configurazione	Descrizione
		in base all'utilizzo delle risorse dei lavoratori.
`enablePriorityLeaseAssignment`	LeaseManagementConfig	Controlla se i lavoratori devono dare la priorità ai contratti di locazione scaduti (contratti di locazione non rinnovati per il triplo del tempo di failover) e ai nuovi contratti di locazione condivisi , indipendentemente dal numero di leasing previsto, ma rispettando comunque i limiti massimi di leasing. KCL 3.x ignorerà questa configurazione e distribuirà sempre i leasing scaduti tra i lavoratori.

> **Important**
>
> È comunque necessario disporre delle proprietà di configurazione interrotte durante la migrazione dalle versioni precedenti di KCL a KCL 3.x. Durante la migrazione, il lavoratore KCL inizierà innanzitutto con la modalità compatibile con KCL 2.x e passerà alla modalità di funzionalità KCL 3.x quando rileva che tutti i lavoratori KCL dell'applicazione sono pronti per eseguire KCL 3.x. Queste configurazioni interrotte sono necessarie mentre i lavoratori KCL eseguono la modalità compatibile con KCL 2.x.

Politica del ciclo di vita della versione KCL

Questo argomento descrive la politica del ciclo di vita delle versioni per Amazon Kinesis Client Library (KCL). AWS fornisce regolarmente nuove versioni per le versioni di KCL per supportare nuove funzionalità e miglioramenti, correzioni di bug, patch di sicurezza e aggiornamenti delle dipendenze. Ti consigliamo di rimanere up-to-date con le versioni KCL per tenerti aggiornato sulle funzionalità

più recenti, sugli aggiornamenti di sicurezza e sulle dipendenze sottostanti. Non consigliamo l'uso continuato di una versione KCL non supportata.

Il ciclo di vita delle principali versioni di KCL è costituito dalle tre fasi seguenti:

- Disponibilità generale (GA): durante questa fase, la versione principale è completamente supportata. AWS fornisce regolarmente versioni secondarie e patch che includono il supporto per nuove funzionalità o aggiornamenti delle API per Kinesis Data Streams, oltre a correzioni di bug e sicurezza.
- Modalità di manutenzione: AWS limita il rilascio delle versioni delle patch per risolvere solo le correzioni di bug critici e i problemi di sicurezza. La versione principale non riceverà aggiornamenti per le nuove funzionalità o APIs per Kinesis Data Streams.
- E nd-of-support — La versione principale non riceverà più aggiornamenti o versioni. Le versioni pubblicate in precedenza continueranno a essere disponibili tramite gestori di pacchetti pubblici e il codice rimarrà attivo GitHub. L'uso di una versione raggiunta end-of-support viene effettuato a discrezione dell'utente. Ti consigliamo di eseguire l'aggiornamento alla versione principale più recente.

Versione principale	Fase attuale	Data di rilascio	Data della modalità di manutenzione	End-of-support data
KCL 1.x	modalità di manutenzione	19-12-2013	2025-04-17	2026-01-30
KCL 2.x	Disponibilità generale	2018-08-02	--	--
KCL 3.x	Disponibilità generale	2024-11-06	--	--

Esegui la migrazione dalle versioni precedenti di KCL

Questo argomento spiega come migrare dalle versioni precedenti della Kinesis Client Library (KCL).

Cosa c'è di nuovo in KCL 3.0?

Kinesis Client Library (KCL) 3.0 introduce diversi importanti miglioramenti rispetto alle versioni precedenti:

- Riduce i costi di elaborazione per le applicazioni consumer ridistribuendo automaticamente il lavoro dai lavoratori sovrautilizzati ai lavoratori sottoutilizzati del parco applicazioni consumer. Questo nuovo algoritmo di bilanciamento del carico garantisce l'utilizzo della CPU distribuito in modo uniforme tra i lavoratori ed elimina la necessità di sovraccaricare i lavoratori.
- Riduce i costi di DynamoDB associati a KCL ottimizzando le operazioni di lettura sulla tabella di leasing.
- Riduce al minimo la rielaborazione dei dati quando i contratti di locazione vengono riassegnati a un altro lavoratore, permettendo al lavoratore corrente di completare il checkpoint dei record che ha elaborato.
- Viene utilizzato AWS SDK for Java 2.x per migliorare le prestazioni e le funzionalità di sicurezza, eliminando completamente la dipendenza dalla versione 1.x. AWS SDK per Java

Per ulteriori informazioni, consulta la nota di rilascio di KCL 3.0.

Argomenti

- Migrazione da KCL 2.x a KCL 3.x
- Torna alla versione precedente di KCL
- Passa a KCL 3.x dopo un rollback
- Procedure consigliate per la tabella di leasing con modalità di capacità assegnata
- Migrazione da KCL 1.x a KCL 3.x

Migrazione da KCL 2.x a KCL 3.x

Questo argomento fornisce step-by-step istruzioni per migrare il consumatore da KCL 2.x a KCL 3.x. KCL 3.x supporta la migrazione in loco dei consumatori KCL 2.x. Puoi continuare a utilizzare i dati del tuo flusso di dati Kinesis mentre migri i tuoi lavoratori in modo continuativo.

> **Important**
>
> KCL 3.x mantiene le stesse interfacce e gli stessi metodi di KCL 2.x. Pertanto non è necessario aggiornare il codice di elaborazione dei record durante la migrazione. Tuttavia,

è necessario impostare la configurazione corretta e verificare i passaggi richiesti per la migrazione. Ti consigliamo vivamente di seguire i seguenti passaggi di migrazione per un'esperienza di migrazione senza intoppi.

Fase 1: prerequisiti

Prima di iniziare a utilizzare KCL 3.x, assicurati di disporre di quanto segue:

- Java Development Kit (JDK) 8 o versione successiva
- AWS SDK per Java 2.x
- Maven o Gradle per la gestione delle dipendenze

Important

Non utilizzare la AWS SDK per Java versione da 2.27.19 a 2.27.23 con KCL 3.x. Queste versioni includono un problema che causa un errore di eccezione relativo all'utilizzo di DynamoDB da parte di KCL. Si consiglia di utilizzare la AWS SDK per Java versione 2.28.0 o successiva per evitare questo problema.

Passaggio 2: aggiungere dipendenze

Se utilizzi Maven, aggiungi la seguente dipendenza al tuo file. `pom.xml` Assicurati di aver sostituito 3.x.x con l'ultima versione di KCL.

```
<dependency>
    <groupId>software.amazon.kinesis</groupId>
    <artifactId>amazon-kinesis-client</artifactId>
    <version>3.x.x</version> <!-- Use the latest version -->
</dependency>
```

Se stai usando Gradle, aggiungi quanto segue al tuo file. `build.gradle` Assicurati di aver sostituito 3.x.x con l'ultima versione di KCL.

```
implementation 'software.amazon.kinesis:amazon-kinesis-client:3.x.x'
```

Puoi verificare la versione più recente di KCL sul Maven Central Repository.

Passaggio 3: configurare la configurazione relativa alla migrazione

Per migrare da KCL 2.x a KCL 3.x, è necessario impostare il seguente parametro di configurazione:

- CoordinatorConfig. clientVersionConfig: Questa configurazione determina in quale modalità di compatibilità della versione KCL verrà eseguita l'applicazione. Quando si esegue la migrazione da KCL 2.x a 3.x, è necessario impostare questa configurazione su. `CLIENT_VERSION_CONFIG_COMPATIBLE_WITH_2X` Per impostare questa configurazione, aggiungi la seguente riga durante la creazione dell'oggetto scheduler:

```
configsBuilder.coordiantorConfig().clientVersionConfig(ClientVersionConfig.CLIENT_VERSION_CONFI
```

Di seguito è riportato un esempio di come impostare la `CoordinatorConfig.clientVersionConfig` migrazione da KCL 2.x a 3.x. Puoi regolare altre configurazioni secondo necessità in base alle tue esigenze specifiche:

```
Scheduler scheduler = new Scheduler(
    configsBuilder.checkpointConfig(),

 configsBuilder.coordiantorConfig().clientVersionConfig(ClientVersionConfig.CLIENT_VERSION_CONF
    configsBuilder.leaseManagementConfig(),
    configsBuilder.lifecycleConfig(),
    configsBuilder.metricsConfig(),
    configsBuilder.processorConfig(),
    configsBuilder.retrievalConfig()
);
```

È importante che tutti gli operatori dell'applicazione consumer utilizzino lo stesso algoritmo di bilanciamento del carico in un determinato momento, poiché KCL 2.x e 3.x utilizzano algoritmi di bilanciamento del carico diversi. L'utilizzo di lavoratori con algoritmi di bilanciamento del carico diversi può causare una distribuzione del carico non ottimale poiché i due algoritmi funzionano in modo indipendente.

Questa impostazione di compatibilità KCL 2.x consente all'applicazione KCL 3.x di funzionare in una modalità compatibile con KCL 2.x e di utilizzare l'algoritmo di bilanciamento del carico per KCL 2.x fino a quando tutti i worker dell'applicazione consumer non saranno stati aggiornati a KCL 3.x. Una volta completata la migrazione, KCL passerà automaticamente alla modalità di funzionalità KCL 3.x completa e inizierà a utilizzare un nuovo algoritmo di bilanciamento del carico KCL 3.x per tutti i lavoratori in esecuzione.

> **Important**
>
> Se non stai utilizzando `ConfigsBuilder` ma stai creando un `LeaseManagementConfig` oggetto per impostare le configurazioni, devi aggiungere un altro parametro chiamato `applicationName` nella versione KCL 3.x o successiva. Per i dettagli, vedi Errore di compilazione con il costruttore. LeaseManagementConfig Si consiglia di `ConfigsBuilder` utilizzarlo per impostare le configurazioni KCL. `ConfigsBuilder`offre un modo più flessibile e gestibile per configurare l'applicazione KCL.

Fase 4: Segui le migliori pratiche per l'implementazione del metodo shutdownRequested ()

KCL 3.x introduce una funzionalità chiamata graceful lease handoff per ridurre al minimo la rielaborazione dei dati quando un contratto di locazione viene ceduto a un altro lavoratore come parte del processo di riassegnazione del contratto di locazione. Ciò si ottiene selezionando l'ultimo numero di sequenza elaborato nella tabella di locazione prima della cessione del contratto di locazione. Per assicurarvi che il grazioso comando di lease funzioni correttamente, dovete assicurarvi di richiamare l'oggetto all'interno del metodo della classe. `checkpointer` `shutdownRequested` `RecordProcessor` Se non state invocando l'`checkpointer`oggetto all'interno del `shutdownRequested` metodo, potete implementarlo come illustrato nell'esempio seguente.

> **Important**
>
> - Il seguente esempio di implementazione è un requisito minimo per la consegna del leasing grazioso. È possibile estenderlo per includere una logica aggiuntiva relativa al checkpoint, se necessario. Se stai eseguendo un'elaborazione asincrona, assicurati che tutti i record consegnati al sistema a valle siano stati elaborati prima di richiamare il checkpoint.
> - Sebbene la consegna gradita del leasing riduca in modo significativo la probabilità di ritrattamento dei dati durante i trasferimenti di leasing, non elimina completamente questa possibilità. Per preservare l'integrità e la coerenza dei dati, progetta le tue applicazioni consumer downstream in modo che siano idempotenti. Ciò significa che dovrebbero essere in grado di gestire potenziali elaborazioni di record duplicati senza effetti negativi sull'intero sistema.

```
/**
```

```
 * Invoked when either Scheduler has been requested to gracefully shutdown
 * or lease ownership is being transferred gracefully so the current owner
 * gets one last chance to checkpoint.
 *
 * Checkpoints and logs the data a final time.
 *
 * @param shutdownRequestedInput Provides access to a checkpointer, allowing a record
 processor to checkpoint
 *                               before the shutdown is completed.
 */
public void shutdownRequested(ShutdownRequestedInput shutdownRequestedInput) {
    try {
       // Ensure that all delivered records are processed
       // and has been successfully flushed to the downstream before calling
       // checkpoint
       // If you are performing any asynchronous processing or flushing to
       // downstream, you must wait for its completion before invoking
       // the below checkpoint method.
        log.info("Scheduler is shutting down, checkpointing.");
        shutdownRequestedInput.checkpointer().checkpoint();
    } catch (ShutdownException | InvalidStateException e) {
        log.error("Exception while checkpointing at requested shutdown. Giving up.",
 e);
    }
}
```

Fase 5: Verifica i prerequisiti di KCL 3.x per la raccolta delle metriche dei lavoratori

KCL 3.x raccoglie i parametri di utilizzo della CPU, come l'utilizzo della CPU, dai lavoratori per bilanciare il carico tra i lavoratori in modo uniforme. I consumer application worker possono essere eseguiti su Amazon EC2, Amazon ECS, Amazon EKS o AWS Fargate. KCL 3.x può raccogliere i parametri di utilizzo della CPU dai lavoratori solo quando vengono soddisfatti i seguenti prerequisiti:

Amazon Elastic Compute Cloud(Amazon EC2)

- Il sistema operativo deve essere Linux OS.
- È necessario IMDSv2abilitarlo nella propria EC2 istanza.

Amazon Elastic Container Service (Amazon ECS) su Amazon EC2

- Il sistema operativo deve essere Linux OS.
- È necessario abilitare l'endpoint ECS Task Metadata Endpoint versione 4.

- La versione dell'agente container Amazon ECS deve essere 1.39.0 o successiva.

Amazon ECS su AWS Fargate

- È necessario abilitare l'endpoint dei metadati delle attività Fargate versione 4. Se si utilizza la versione 1.4.0 o successiva della piattaforma Fargate, questa opzione è abilitata per impostazione predefinita.
- Piattaforma Fargate versione 1.4.0 o successiva.

Amazon Elastic Kubernetes Service (Amazon EKS) su Amazon EC2

- Il sistema operativo deve essere Linux OS.

Amazon EKS su AWS Fargate

- Piattaforma Fargate 1.3.0 o versione successiva.

> **Important**
>
> Se KCL 3.x non è in grado di raccogliere i parametri di utilizzo della CPU dai lavoratori perché i prerequisiti non sono soddisfatti, riequilibrerà il carico in base al livello di throughput per leasing. Questo meccanismo di ribilanciamento alternativo assicurerà che tutti i lavoratori ottengano livelli di produttività totale simili dai contratti di locazione assegnati a ciascun lavoratore. Per ulteriori informazioni, consulta In che modo KCL assegna i contratti di locazione ai lavoratori e bilancia il carico.

Fase 6: Aggiornamento delle autorizzazioni IAM per KCL 3.x

Devi aggiungere le seguenti autorizzazioni al ruolo o alla policy IAM associata alla tua applicazione consumer KCL 3.x. Ciò comporta l'aggiornamento della politica IAM esistente utilizzata dall'applicazione KCL. Per ulteriori informazioni, consulta Autorizzazioni IAM richieste per le applicazioni consumer KCL.

Important

Le tue applicazioni KCL esistenti potrebbero non avere le seguenti azioni e risorse IAM aggiunte nella policy IAM perché non erano necessarie in KCL 2.x. Assicurati di averle aggiunte prima di eseguire l'applicazione KCL 3.x:

- Azioni: `UpdateTable`
  - Risorse (ARNs): `arn:aws:dynamodb:region:account:table/KCLApplicationName`
- Azioni: `Query`
  - Risorse (ARNs): `arn:aws:dynamodb:region:account:table/KCLApplicationName/Index/*`
- Azioni: `CreateTableDescribeTable,,Scan,GetItem,PutItem,UpdateItem,DeleteItem`
  - Risorse (ARNs):`arn:aws:dynamodb:region:account:table/KCLApplicationName-WorkerMetricStats`, `arn:aws:dynamodb:region:account:table/KCLApplicationName-CoordinatorState`

Sostituisci «regione», «account» e "KCLApplicationNome» rispettivamente ARNs con il tuo Regione AWS Account AWS numero e il nome dell'applicazione KCL. Se utilizzi configurazioni per personalizzare i nomi delle tabelle di metadati create da KCL, usa i nomi di tabella specificati anziché il nome dell'applicazione KCL.

Passaggio 7: distribuisci il codice KCL 3.x ai tuoi lavoratori

Dopo aver impostato la configurazione richiesta per la migrazione e completato tutte le liste di controllo sulla migrazione precedenti, puoi creare e distribuire il codice ai tuoi lavoratori.

Note

Se vedi un errore di compilazione con il `LeaseManagementConfig` costruttore, vedi Errore di compilazione con il costruttore per informazioni sulla risoluzione dei LeaseManagementConfig problemi.

Fase 8: Completare la migrazione

Durante la distribuzione del codice KCL 3.x, KCL continua a utilizzare l'algoritmo di assegnazione del leasing di KCL 2.x. Dopo aver distribuito con successo il codice KCL 3.x a tutti i lavoratori, KCL lo rileva automaticamente e passa al nuovo algoritmo di assegnazione del leasing basato sull'utilizzo delle risorse dei lavoratori. Per ulteriori dettagli sul nuovo algoritmo di assegnazione del leasing, vedere. In che modo KCL assegna i contratti di locazione ai lavoratori e bilancia il carico

Durante la distribuzione, è possibile monitorare il processo di migrazione emettendo le seguenti metriche a. CloudWatch È possibile monitorare le metriche durante l'operazione. `Migration` Tutte le metriche sono per-KCL-application metriche e impostate sul livello metrico. `SUMMARY` Se la `Sum` statistica della `CurrentState:3xWorker` metrica corrisponde al numero totale di lavoratori nell'applicazione KCL, indica che la migrazione a KCL 3.x è stata completata con successo.

> **Important**
>
> KCL impiega almeno 10 minuti per passare al nuovo algoritmo di assegnazione del leasee dopo che tutti i lavoratori sono pronti a eseguirlo.

CloudWatch metriche per il processo di migrazione KCL

Metriche	Descrizione
`CurrentState:3xWorker`	Il numero di lavoratori KCL è migrato con successo a KCL 3.x e ha eseguito il nuovo algoritmo di assegnazione del leasing. Se il `Sum` conteggio di questa metrica corrisponde al numero totale dei lavoratori, indica che la migrazione a KCL 3.x è stata completata con successo. • Livello parametro: Summary • Unità: numero • Statistiche: la statistica più utile è `Sum`
`CurrentState:2xCompatibleWorker`	Il numero di lavoratori KCL in esecuzione in modalità compatibile con KCL 2.x durante il processo di migrazione. Un valore diverso da

Metriche	Descrizione
	zero per questa metrica indica che la migrazion e è ancora in corso. • Livello parametro: Summary • Unità: numero • Statistiche: la statistica più utile è `Sum`
`Fault`	Il numero di eccezioni riscontrate durante il processo di migrazione. La maggior parte di queste eccezioni sono errori transitori e KCL 3.x riproverà automaticamente a completar e la migrazione. Se osservi un valore `Fault` metrico persistente, esamina i log del periodo di migrazione per un'ulteriore risoluzione dei problemi. Se il problema persiste, contatta. Supporto • Livello parametro: Summary • Unità: numero • Statistiche: la statistica più utile è `Sum`
`GsiStatusReady`	Lo stato della creazione dell'indice secondari o globale (GSI) nella tabella di leasing. Questa metrica indica se il GSI nella tabella di leasing è stato creato, un prerequisito per eseguire KCL 3.x. Il valore è 0 o 1, dove 1 indica che la creazione è riuscita. Durante uno stato di rollback, questa metrica non verrà emessa. Dopo aver eseguito nuovamente il rollforward, puoi riprendere a monitorare questa metrica. • Livello parametro: Summary • Unità: numero • Statistiche: la statistica più utile è `Sum`

Metriche	Descrizione
`workerMetricsReady`	Stato delle emissioni delle metriche dei lavoratori da parte di tutti i lavoratori. Le metriche indicano se tutti i lavoratori emettono parametri come l'utilizzo della CPU. Il valore è 0 o 1, dove 1 indica che tutti i lavoratori stanno emettendo correttamente le metriche e sono pronti per il nuovo algoritmo di assegnazione del leasing. Durante uno stato di rollback, questa metrica non verrà emessa. Dopo aver eseguito nuovamente il rollforward, puoi riprendere a monitorare questa metrica. • Livello parametro: Summary • Unità: numero • Statistiche: la statistica più utile è `Sum`

KCL fornisce funzionalità di rollback alla modalità compatibile con 2.x durante la migrazione. Una volta completata la migrazione a KCL 3.x, ti consigliamo di rimuovere l'`CoordinatorConfig.clientVersionConfig`impostazione `CLIENT_VERSION_CONFIG_COMPATIBLE_WITH_2X` se il rollback non è più necessario. La rimozione di questa configurazione interrompe l'emissione di metriche relative alla migrazione dall'applicazione KCL.

> **Note**
>
> Ti consigliamo di monitorare le prestazioni e la stabilità dell'applicazione per un periodo durante la migrazione e dopo il completamento della migrazione. Se si riscontrano problemi, è possibile ripristinare i lavoratori affinché utilizzino la funzionalità compatibile con KCL 2.x utilizzando lo strumento di migrazione KCL.

Torna alla versione precedente di KCL

Questo argomento spiega i passaggi per riportare il consumatore alla versione precedente. Quando è necessario eseguire il rollback, è prevista una procedura in due fasi:

1. Esegui lo strumento di migrazione KCL.
2. Ridistribuisci il codice della versione precedente di KCL (opzionale).

Fase 1: Esegui lo strumento di migrazione KCL

Quando è necessario tornare alla versione precedente di KCL, è necessario eseguire lo strumento di migrazione KCL. Il KCL Migration Tool svolge due compiti importanti:

- Rimuove una tabella di metadati denominata tabella delle metriche dei lavoratori e indice secondario globale nella tabella di lease in DynamoDB. Questi due artefatti sono creati da KCL 3.x ma non sono necessari quando si torna alla versione precedente.
- Consente a tutti i lavoratori di funzionare in una modalità compatibile con KCL 2.x e di iniziare a utilizzare l'algoritmo di bilanciamento del carico utilizzato nelle versioni precedenti di KCL. Se hai problemi con il nuovo algoritmo di bilanciamento del carico in KCL 3.x, questo ridurrà immediatamente il problema.

> **Important**
>
> La tabella degli stati del coordinatore in DynamoDB deve esistere e non deve essere eliminata durante il processo di migrazione, rollback e rollforward.

> **Note**
>
> È importante che tutti i lavoratori dell'applicazione consumer utilizzino lo stesso algoritmo di bilanciamento del carico in un determinato momento. Il KCL Migration Tool assicura che tutti i lavoratori dell'applicazione consumer KCL 3.x passino alla modalità compatibile con KCL 2.x, in modo che tutti i lavoratori eseguano lo stesso algoritmo di bilanciamento del carico durante il roll-depayment alla versione precedente di KCL.

Puoi scaricare lo strumento di migrazione KCL nella directory degli script del repository KCL. GitHub Lo script può essere eseguito da qualsiasi lavoratore o da qualsiasi host che disponga delle autorizzazioni necessarie per scrivere nella tabella degli stati del coordinatore, eliminare la tabella delle metriche dei lavoratori e aggiornare la tabella dei leasing. Puoi fare riferimento all'autorizzazione IAM richiesta Autorizzazioni IAM richieste per le applicazioni consumer KCL per eseguire lo script. È

necessario eseguire lo script solo una volta per applicazione KCL. È possibile eseguire lo strumento di migrazione KCL con il seguente comando:

```
python3 ./KclMigrationTool.py --region <region> --mode rollback [--
application_name <applicationName>] [--lease_table_name <leaseTableName>] [--
coordinator_state_table_name <coordinatorStateTableName>] [--worker_metrics_table_name
 <workerMetricsTableName>]
```

Parametri

- --region: sostituisci `<region>` con il tuo. Regione AWS
- --application_name: Questo parametro è obbligatorio se utilizzi nomi predefiniti per le tabelle dei metadati di DynamoDB (lease table, coordinator state table e worker metrics table). Se hai specificato nomi personalizzati per queste tabelle, puoi omettere questo parametro. `<applicationName>`Sostituiscilo con il nome effettivo dell'applicazione KCL. Lo strumento utilizza questo nome per derivare i nomi delle tabelle predefiniti se non vengono forniti nomi personalizzati.
- --lease_table_name (opzionale): questo parametro è necessario quando hai impostato un nome personalizzato per la tabella di leasing nella tua configurazione KCL. Se stai usando il nome di tabella predefinito, puoi omettere questo parametro. `leaseTableName`Sostituiscilo con il nome della tabella personalizzata che hai specificato per la tabella di leasing.
- --coordinator_state_table_name (opzionale): questo parametro è necessario quando hai impostato un nome personalizzato per la tabella degli stati del coordinatore nella tua configurazione KCL. Se stai usando il nome di tabella predefinito, puoi omettere questo parametro. `<coordinatorStateTableName>`Sostituiscilo con il nome di tabella personalizzato che hai specificato per la tabella degli stati del coordinatore.
- --worker_metrics_table_name (opzionale): questo parametro è necessario quando hai impostato un nome personalizzato per la tabella delle metriche dei lavoratori nella tua configurazione KCL. Se stai usando il nome di tabella predefinito, puoi omettere questo parametro. `<workerMetricsTableName>`Sostituiscilo con il nome della tabella personalizzata che hai specificato per la tabella delle metriche dei lavoratori.

Passaggio 2: ridistribuisci il codice con la versione precedente di KCL (opzionale)

Dopo aver eseguito lo strumento di migrazione KCL per un rollback, vedrai uno di questi messaggi:

- Messaggio 1: «Rollback completato. L'applicazione KCL eseguiva la modalità compatibile con KCL 2.x. Se non vedi alcuna mitigazione di alcuna regressione, torna ai file binari dell'applicazione precedente distribuendo il codice con la versione KCL precedente.»
  - Azione richiesta: Ciò significa che i tuoi lavoratori funzionavano nella modalità compatibile con KCL 2.x. Se il problema persiste, ridistribuisci il codice con la versione precedente di KCL ai tuoi lavoratori.
- Messaggio 2: «Rollback completato. L'applicazione KCL eseguiva la modalità di funzionalità KCL 3.x. Il rollback ai file binari dell'applicazione precedente non è necessario, a meno che non si veda alcuna mitigazione del problema entro 5 minuti. Se il problema persiste, ripristina i file binari dell'applicazione precedente distribuendo il codice con la versione precedente di KCL.»
  - Azione richiesta: Ciò significa che i tuoi lavoratori lavoravano in modalità KCL 3.x e lo strumento di migrazione KCL ha portato tutti i lavoratori alla modalità compatibile con KCL 2.x. Se il problema viene risolto, non è necessario ridistribuire il codice con la versione precedente di KCL. Se il problema persiste, ridistribuisci il codice con la versione precedente di KCL ai tuoi worker.

Passa a KCL 3.x dopo un rollback

Questo argomento spiega i passaggi per riportare il consumatore a KCL 3.x dopo un rollback. Quando è necessario eseguire il rollforward, è necessario eseguire una procedura in due fasi:

1. Esegui lo strumento di migrazione KCL.
2. Distribuisci il codice con KCL 3.x.

Passaggio 1: eseguire lo strumento di migrazione KCL

Esegui lo strumento di migrazione KCL. Strumento di migrazione KCL con il seguente comando per passare a KCL 3.x:

```
python3 ./KclMigrationTool.py --region <region> --mode rollforward [--application_name
 <applicationName>] [--coordinator_state_table_name <coordinatorStateTableName>]
```

Parametri

- --region: sostituisci con il tuo. `<region>` Regione AWS
- --application_name: questo parametro è obbligatorio se utilizzi nomi predefiniti per la tabella degli stati del coordinatore. Se hai specificato nomi personalizzati per la tabella degli stati del

coordinatore, puoi omettere questo parametro. `<applicationName>`Sostituiscilo con il nome effettivo dell'applicazione KCL. Lo strumento utilizza questo nome per derivare i nomi delle tabelle predefiniti se non vengono forniti nomi personalizzati.

- --coordinator_state_table_name (opzionale): questo parametro è necessario quando hai impostato un nome personalizzato per la tabella degli stati del coordinatore nella tua configurazione KCL. Se stai usando il nome di tabella predefinito, puoi omettere questo parametro. `<coordinatorStateTableName>`Sostituiscilo con il nome di tabella personalizzato che hai specificato per la tabella degli stati del coordinatore.

Dopo aver eseguito lo strumento di migrazione in modalità roll-forward, KCL crea le seguenti risorse DynamoDB necessarie per KCL 3.x:

- Un indice secondario globale nella tabella dei contratti di locazione
- Una tabella delle metriche dei lavoratori

Fase 2: Implementa il codice con KCL 3.x

Dopo aver eseguito KCL Migration Tool per un rollforward, distribuisci il codice con KCL 3.x ai tuoi lavoratori. Segui Fase 8: Completare la migrazione per completare la migrazione.

Procedure consigliate per la tabella di leasing con modalità di capacità assegnata

Se la tabella di leasing dell'applicazione KCL è passata alla modalità di capacità fornita, KCL 3.x crea un indice secondario globale nella tabella di leasing con la modalità di fatturazione assegnata e le stesse unità di capacità di lettura (RCU) e unità di capacità di scrittura (WCU) della tabella di leasing di base. Quando viene creato l'indice secondario globale, consigliamo di monitorare l'utilizzo effettivo dell'indice secondario globale nella console DynamoDB e di regolare le unità di capacità, se necessario. Per una guida più dettagliata sulla modifica della modalità di capacità delle tabelle di metadati DynamoDB create da KCL, vedere. Modalità di capacità DynamoDB per le tabelle di metadati create da KCL

> **Note**
>
> Per impostazione predefinita, KCL crea tabelle di metadati come la tabella di leasing, la tabella delle metriche dei lavoratori e la tabella degli stati dei coordinatori e l'indice secondario globale nella tabella di leasing utilizzando la modalità di capacità su richiesta. Ti consigliamo

di utilizzare la modalità di capacità su richiesta per regolare automaticamente la capacità in base alle modifiche all'utilizzo.

Migrazione da KCL 1.x a KCL 3.x

Questo argomento spiega le istruzioni per migrare il consumatore da KCL 1.x a KCL 3.x. KCL 1.x utilizza classi e interfacce diverse rispetto a KCL 2.x e KCL 3.x. È necessario prima migrare il record processor, il record processor factory e le classi worker al formato compatibile con KCL 2.x/3.x e seguire i passaggi di migrazione per la migrazione da KCL 2.x a KCL 3.x. È possibile eseguire l'aggiornamento direttamente da KCL 1.x a KCL 3.x.

- Passaggio 1: migrare il processore di registrazione

  Segui la sezione Migrare il processore di registrazione nella pagina Migrare i consumatori da KCL 1.x a KCL 2.x.
- Fase 2: Migrazione della fabbrica del processore di registrazione

  Segui la sezione Migrare la fabbrica del processore di registrazione nella pagina Migrare i consumatori da KCL 1.x a KCL 2.x.
- Fase 3: Migrare il lavoratore

  Segui la sezione Migrare il lavoratore nella pagina Migrare i consumatori da KCL 1.x a KCL 2.x.
- Fase 4: Migrazione della configurazione di KCL 1.x

  Segui la sezione Configura il client Amazon Kinesis nella pagina Migrare i consumatori da KCL 1.x a KCL 2.x.
- Passaggio 5: verifica la rimozione dei periodi di inattività e le rimozioni della configurazione del client

  Segui le sezioni Rimozione dei tempi di inattività e Rimozione della configurazione del client nella pagina Migrare i consumatori da KCL 1.x a KCL 2.x.
- Passaggio 6: segui le step-by-step istruzioni nella guida alla migrazione da KCL 2.x a KCL 3.x

  Segui le istruzioni sulla Migrazione da KCL 2.x a KCL 3.x pagina per completare la migrazione. Se devi tornare alla versione precedente di KCL o passare a KCL 3.x dopo un rollback, consulta e. Torna alla versione precedente di KCL Passa a KCL 3.x dopo un rollback

> **Important**
>
> Non utilizzare la AWS SDK per Java versione da 2.27.19 a 2.27.23 con KCL 3.x. Queste versioni includono un problema che causa un errore di eccezione relativo all'utilizzo di DynamoDB da parte di KCL. Si consiglia di utilizzare la AWS SDK per Java versione 2.28.0 o successiva per evitare questo problema.

Documentazione della versione precedente di KCL

I seguenti argomenti sono stati archiviati. Per visualizzare la documentazione corrente di Kinesis Client Library, vedere. Usa la libreria client Kinesis

> **Important**
>
> Le versioni 1.x e 2.x di Amazon Kinesis Client Library (KCL) sono obsolete. KCL 1.x sarà disponibile il 30 gennaio 2026. end-of-support Ti consigliamo vivamente di migrare le tue applicazioni KCL utilizzando la versione 1.x all'ultima versione di KCL prima del 30 gennaio 2026. Per trovare la versione più recente di KCL, consulta la pagina Amazon Kinesis Client Library su. GitHub Per informazioni sulle ultime versioni di KCL, consulta. Usa la libreria client Kinesis Per informazioni sulla migrazione da KCL 1.x a KCL 3.x, consulta. Migrazione da KCL 1.x a KCL 3.x

Documentazione ritirata

- Informazioni su KCL 1.x e 2.x
- Sviluppa consumatori personalizzati con un throughput condiviso
- Migra i consumatori da KCL 1.x a KCL 2.x

Informazioni su KCL 1.x e 2.x

> **Important**
>
> Le versioni 1.x e 2.x di Amazon Kinesis Client Library (KCL) sono obsolete. KCL 1.x sarà disponibile il 30 gennaio 2026. end-of-support Ti consigliamo vivamente di migrare le tue applicazioni KCL utilizzando la versione 1.x all'ultima versione di KCL prima del 30 gennaio

2026. Per trovare la versione più recente di KCL, consulta la pagina Amazon Kinesis Client Library su. GitHub Per informazioni sulle ultime versioni di KCL, consulta. Usa la libreria client Kinesis Per informazioni sulla migrazione da KCL 1.x a KCL 3.x, consulta. Migrazione da KCL 1.x a KCL 3.x

Uno dei metodi per sviluppare applicazioni consumer personalizzate in grado di elaborare i dati dai flussi di dati KDS consiste nell'utilizzare la Kinesis Client Library (KCL).

Argomenti

- Informazioni su KCL (versioni precedenti)
- Versioni precedenti di KCL
- Concetti KCL (versioni precedenti)
- Utilizza una tabella di leasing per tenere traccia degli shard elaborati dall'applicazione consumer KCL
- Elabora più flussi di dati con la stessa applicazione consumer KCL 2.x per Java
- Usa KCL con lo Schema Registry AWS Glue

Note

Sia per KCL 1.x che per KCL 2.x, si consiglia di eseguire l'aggiornamento alla versione più recente di KCL 1.x o KCL 2.x, a seconda dello scenario di utilizzo. Sia KCL 1.x che KCL 2.x vengono regolarmente aggiornate con versioni più recenti che includono le ultime patch di dipendenza e sicurezza, correzioni di bug e nuove funzionalità retrocompatibili. Per ulteriori informazioni, consulta /releases. https://github.com/awslabs/ amazon-kinesis-client

Informazioni su KCL (versioni precedenti)

KCL ti aiuta a consumare ed elaborare i dati da un flusso di dati Kinesis occupandoti di molte delle attività complesse associate al calcolo distribuito. Queste includono il bilanciamento del carico su più istanze di applicazioni consumer, la risposta agli errori delle istanze delle applicazioni consumer, il checkpoint dei record elaborati e la reazione al ripartizionamento. La KCL si occupa di tutte queste attività secondarie in modo che tu possa concentrare i tuoi sforzi sulla scrittura della tua logica di elaborazione dei record personalizzata.

Il KCL è diverso dai Kinesis Data APIs Streams disponibili in. AWS SDKs Kinesis APIs Data Streams ti aiuta a gestire molti aspetti di Kinesis Data Streams, tra cui la creazione di stream, il resharding e l'inserimento e l'acquisizione di record. La KCL fornisce un livello di astrazione su tutte queste sottoattività, in particolare per consentirti di concentrarti sulla logica di elaborazione dei dati personalizzata dell'applicazione consumer. Per ulteriori informazioni sulle API del flusso di dati Kinesis, consulta la Documentazione di riferimento delle API di Amazon Kinesis.

> **Important**
>
> La KCL è una libreria Java. Il supporto per linguaggi diversi da Java viene fornito utilizzando un'interfaccia multilingue chiamata. MultiLangDaemon Questo daemon è basato su Java e viene eseguito in background quando si utilizza un linguaggio KCL diverso da Java. Ad esempio, se installi KCL per Python e scrivi la tua applicazione consumer interamente in Python, avrai comunque bisogno che Java sia installato sul tuo sistema a causa di. MultiLangDaemon Inoltre, MultiLangDaemon ha alcune impostazioni predefinite che potresti dover personalizzare in base al tuo caso d'uso, ad esempio la AWS regione a cui si connette. Per ulteriori informazioni su MultiLangDaemon on GitHub, consulta il MultiLangDaemon progetto KCL.

La KCL funge da intermediario tra la tua logica di elaborazione di record e il flusso di dati Kinesis.

Versioni precedenti di KCL

Al momento per creare applicazioni consumer personalizzate puoi utilizzare una delle seguenti versioni supportate di KCL:

- KCL 1.x

  Per ulteriori informazioni, consulta Sviluppa i consumatori di KCL 1.x
- KCL 2.x

  Per ulteriori informazioni, consulta Sviluppa KCL 2.x Consumers

È possibile utilizzare KCL 1.x o KCL 2.x per creare applicazioni consumer che utilizzano una velocità di trasmissione effettiva condivisa. Per ulteriori informazioni, consulta Sviluppa consumatori personalizzati con un throughput condiviso utilizzando KCL.

Per creare applicazioni consumer che utilizzano una velocità di trasmissione effettiva dedicata (utenti fan-out avanzati), è possibile utilizzare solo KCL 2.x. Per ulteriori informazioni, consulta Sviluppa consumatori con fan-out migliorati con un throughput dedicato.

Per informazioni sulle differenze tra KCL 1.x e KCL 2.x e le istruzioni su come passare da KCL 1.x a KCL 2.x, consulta Migra i consumatori da KCL 1.x a KCL 2.x.

Concetti KCL (versioni precedenti)

- Applicazione consumer KCL: un'applicazione personalizzata che utilizza KCL e progettata per leggere ed elaborare i record dai flussi di dati.
- Istanza di applicazioni consumer: le applicazioni consumer KCL sono generalmente distribuite, con una o più istanze applicative eseguite contemporaneamente per coordinarsi in caso di guasti e bilanciare dinamicamente l'elaborazione dei record di dati.
- Worker: una classe di livello superiore utilizzata da un'istanza di applicazione consumer KCL per iniziare l'elaborazione dei dati.

> **Important**
>
> Ogni istanza dell'applicazione consumer KCL ha un worker.

Il worker inizializza e supervisiona varie attività, tra cui la sincronizzazione delle informazioni sulle partizioni e sui lease, il monitoraggio delle assegnazioni delle partizioni e l'elaborazione dei dati dalle partizioni. Un worker fornisce a KCL le informazioni di configurazione per l'applicazione consumer, ad esempio il nome del flusso di dati i cui record di dati verranno elaborati dall'applicazione consumer KCL e le AWS credenziali necessarie per accedere a questo flusso di dati. Il worker avvia inoltre quella specifica istanza dell'applicazione consumer KCL per fornire i record di dati dal flusso di dati ai processori di record.

> **Important**
>
> In KCL 1.x questa classe si chiama Worker. Per ulteriori informazioni, (questi sono i repository Java KCL), vedere/.java. https://github.com/awslabs/ amazon-kinesis-client blob/v1.x/src/main/java/com/amazonaws/services/kinesis/clientlibrary/lib/worker/Worker In KCL 2.x questa classe si chiama Scheduler. Lo scopo di Scheduler in KCL 2.x è identico allo scopo di Worker in KCL 1.x. Per ulteriori informazioni sulla classe Scheduler in KCL

2.x, vedete/.java. https://github.com/awslabs/ amazon-kinesis-client blob/master/amazon-kinesis-client/src/main/java/software/amazon/kinesis/coordinator/Scheduler

- Lease: dati che definiscono l'associazione tra un worker e una partizione. Le applicazioni consumer KCL distribuite utilizzano i lease per suddividere l'elaborazione dei record di dati tra un parco istanze di worker. In qualsiasi momento, ogni partizione di record di dati è legato a un determinato worker tramite un lease identificato dalla variabile leaseKey.

  Per impostazione predefinita, un lavoratore può detenere uno o più contratti di locazione (in base al valore della variabile maxLeasesForWorker) contemporaneamente.

  > **Important**
  >
  > Ogni worker si impegna a detenere tutti i lease disponibili per tutte le partizioni disponibili in un flusso di dati. Ma solo un worker alla volta si aggiudicherà con successo ogni lease.

  Ad esempio, se si dispone di un'istanza di applicazione consumer A con worker A che elabora un flusso di dati con 4 partizioni, il worker A può detenere i lease per le partizioni 1, 2, 3 e 4 contemporaneamente. Tuttavia, se si dispone di due istanze di applicazioni consumer, A e B, con worker A e worker B, e queste istanze elaborano un flusso di dati con 4 partizioni, il worker A e il worker B non possono entrambi detenere il lease per la partizione 1 contemporaneamente. Un worker detiene il lease di una particolare partizione finché non è pronto a interrompere l'elaborazione dei record di dati della partizione o fino a quando non si verifica un guasto. Quando un worker smette di detenere il lease, un altro worker lo riprende e lo mantiene.

  Per ulteriori informazioni (questi sono i repository Java KCL), vedere https://github.com/awslabs/amazon-kinesis-client/blob/v1.x/src/main/java/com/amazonaws/services/kinesis/leases/impl/Lease.java per KCL 1.x e https://github.com/awslabs/amazon-kinesis-client/.java per KCL 2.x. blob/master/amazon-kinesis-client/src/main/java/software/amazon/kinesis/leases/Lease

- Tabella di lease: una tabella univoca di Amazon DynamoDB utilizzata per tenere traccia delle partizioni in un flusso di dati KDS sottoposte a lease ed elaborate dai worker dell'applicazione consumer KCL. La tabella di lease deve rimanere sincronizzata (all'interno di un worker e tra tutti i worker) con le ultime informazioni sulle partizioni provenienti dal flusso di dati mentre l'applicazione consumer KCL è in esecuzione. Per ulteriori informazioni, consulta Utilizza una tabella di leasing per tenere traccia degli shard elaborati dall'applicazione consumer KCL.

- Processore di record: la logica che definisce il modo in cui l'applicazione consumer KCL elabora i dati che riceve dai flussi di dati. Durante il runtime, un'istanza dell'applicazione consumer KCL istanzia un worker e questo worker istanzia un processore di record per ogni partizione su cui detiene un lease.

Utilizza una tabella di leasing per tenere traccia degli shard elaborati dall'applicazione consumer KCL

Argomenti

- Cos'è una tabella di leasing
- Prestazioni
- Come viene sincronizzata una tabella di leasing con gli shard in un flusso di dati Kinesis

Cos'è una tabella di leasing

Tabella di lease: per ogni applicazione Flusso di dati Amazon Kinesis, la KCL utilizza una tabella di lease univoca (archiviata in una tabella di Amazon DynamoDB) per tenere traccia delle partizioni in un flusso di dati KDS sottoposte a lease ed elaborate dai worker dell'applicazione consumer KCL.

> **Important**
>
> La KCL utilizza il nome dell'applicazione consumer per creare il nome della tabella di lease utilizzata da questa applicazione consumer, pertanto il nome di ogni applicazione consumer deve essere univoco.

È possibile visualizzare la tabella di lease utilizzando la console Amazon DynamoDB mentre l'applicazione è in esecuzione.

Se la tabella di lease per l'applicazione consumer KCL non esiste all'avvio dell'applicazione, uno dei worker la crea per l'applicazione.

> **Important**
>
> Il tuo account sarà addebitato per i costi associati alla tabella DynamoDB, oltre ai costi associati al flusso di dati Kinesis stesso.

Ogni riga della tabella di lease rappresenta una partizione che viene elaborata dalla tua applicazione consumer. Quando l'applicazione consumer KCL esistente è configurata per elaborare un solo flusso di dati, allora `leaseKey` (che è la chiave hash per la tabella di lease) è l'ID della partizione. Se sei Elabora più flussi di dati con la stessa applicazione consumer KCL 2.x per Java, allora la struttura di leaseKey avrà il seguente aspetto: `account-id:StreamName:streamCreationTimestamp:ShardId`. Ad esempio, `111111111:multiStreamTest-1:12345:shardId-000000000336`.

Oltre all'ID dello shard, ogni riga include anche i seguenti dati:

- checkpoint: il numero di sequenza di checkpoint più recente per lo shard. Questo valore è univoco per tutte le partizioni nel flusso di dati.
- checkpointSubSequenceNumero: quando si utilizza la funzione di aggregazione della Kinesis Producer Library, si tratta di un'estensione del checkpoint che tiene traccia dei record dei singoli utenti all'interno del record Kinesis.
- leaseCounter: utilizzato per la funzione Versioni multiple del lease, in modo tale da permettere ai lavoratori di rilevare che il loro lease è stato preso da un altro lavoratore.
- leaseKey: un identificatore univoco per un lease. Ogni lease è specifico di una partizione nel flusso di dati ed è detenuto da un worker alla volta.
- leaseOwner: il lavoratore che detiene questo lease.
- ownerSwitchesSinceCheckpoint: quante volte questo contratto di locazione ha cambiato lavoratori dall'ultima volta che è stato scritto un checkpoint.
- parentShardId: Utilizzato per garantire che lo shard principale sia completamente elaborato prima che inizi l'elaborazione sui frammenti secondari. In questo modo, ci si assicura che i record siano elaborati nello stesso ordine in cui sono stati introdotti nel flusso.
- hashrange: utilizzato dalla `PeriodicShardSyncManager` per eseguire sincronizzazioni periodiche per trovare le partizioni mancanti nella tabella di lease e creare lease per esse, se necessario.

> **Note**
>
> Questi dati sono presenti nella tabella di lease per ogni partizione a partire da KCL 1.14 e KCL 2.3. Per ulteriori informazioni su `PeriodicShardSyncManager` e sulla sincronizzazione periodica tra lease e partizioni, consulta Come viene sincronizzata una tabella di leasing con gli shard in un flusso di dati Kinesis.

- childshards: utilizzato da `LeaseCleanupManager` per esaminare lo stato di elaborazione della partizione secondaria e decidere se la partizione principale può essere eliminata dalla tabella di lease.

  > **Note**
  >
  > Questi dati sono presenti nella tabella di lease per ogni partizione a partire da KCL 1.14 e KCL 2.3.

- shardID: l'ID della partizione.

  > **Note**
  >
  > Questi dati sono presenti nella tabella dei lease solo se sei Elabora più flussi di dati con la stessa applicazione consumer KCL 2.x per Java. È supportato solo in KCL 2.x per Java, a partire da KCL 2.3 per Java e versioni successive.

- nome del flusso: l'identificatore del flusso di dati nel seguente formato: `account-id:StreamName:streamCreationTimestamp`.

  > **Note**
  >
  > Questi dati sono presenti nella tabella dei lease solo se sei Elabora più flussi di dati con la stessa applicazione consumer KCL 2.x per Java. È supportato solo in KCL 2.x per Java, a partire da KCL 2.3 per Java e versioni successive.

Prestazioni

Se l'applicazione Flusso di dati Amazon Kinesis riceve eccezioni di velocità di trasmissione effettiva assegnata, dovrai aumentare la velocità di trasmissione effettiva assegnata per la tabella DynamoDB. La KCL crea la tabella con una velocità di trasmissione effettiva assegnata di 10 letture al secondo e 10 scritture al secondo, ma questo potrebbe non essere sufficiente per l'applicazione. Ad esempio, se la tua applicazione Flusso di dati Amazon Kinesis crea frequentemente dei checkpoint o opera in un flusso che è composto da molte partizioni, potrebbe essere necessaria una velocità di trasmissione effettiva maggiore.

Per informazioni sulla velocità di trasmissione effettiva assegnata in DynamoDB, consulta Modalità capacità di lettura/scrittura e Utilizzo di tabelle e dati nella Guida per gli sviluppatori di Amazon DynamoDB.

Come viene sincronizzata una tabella di leasing con gli shard in un flusso di dati Kinesis

I worker delle applicazioni consumer KCL utilizzano i lease per elaborare le partizioni di un determinato flusso di dati. Le informazioni su quale worker sta eseguendo il lease di una partizione in un dato momento vengono archiviate in una tabella di lease. La tabella di lease deve rimanere sincronizzata con le ultime informazioni sulle partizioni provenienti dal flusso di dati mentre l'applicazione consumer KCL è in esecuzione. La KCL sincronizza la tabella di lease con le informazioni sulle partizioni acquisite dal servizio del flusso di dati Kinesis durante l'avvio dell'applicazione consumer (quando l'applicazione consumer viene inizializzata o riavviata) e anche ogni volta che una partizione in fase di elaborazione raggiunge il suo termine (ripartizionamento). In altre parole, i worker o un'applicazione consumer KCL vengono sincronizzati con il flusso di dati che stanno elaborando durante l'avvio iniziale dell'applicazione consumer e ogni volta che l'applicazione consumer incontra un evento di ripartizionamento del flusso di dati.

Argomenti

- Sincronizzazione in KCL 1.0 - 1.13 e KCL 2.0 - 2.2
- Sincronizzazione in KCL 2.x, a partire da KCL 2.3 e versioni successive
- Sincronizzazione in KCL 1.x, a partire da KCL 1.14 e versioni successive

Sincronizzazione in KCL 1.0 - 1.13 e KCL 2.0 - 2.2

In KCL 1.0 - 1.13 e KCL 2.0 - 2.2, durante l'avvio dell'applicazione consumer e anche durante ogni evento di reshard del flusso di dati, KCL sincronizza la tabella di lease con le informazioni sugli shard acquisite dal servizio Kinesis Data Streams richiamando l'or the discovery. `ListShards` `DescribeStream` APIs In tutte le versioni KCL sopra elencate, ogni worker di un'applicazione consumer KCL completa i seguenti passaggi per eseguire il processo di sincronizzazione lease/partizione durante l'avvio dell'applicazione consumer e in occasione di ogni evento di ripartizionamento del flusso:

- Recupera tutte le partizioni per i dati elaborati dal flusso
- Recupera tutte i lease delle partizioni dalla tabella di lease
- Filtra ogni partizione aperta che non ha un lease nella tabella di lease

- Itera su tutte le partizioni aperte trovate e per ogni partizione aperta senza un elemento principale aperto:
  - Attraversa l'albero gerarchico lungo il percorso dei suoi antenati per determinare se la partizione è un discendente. Una partizione è considerata un discendente se una partizione antenata è in fase di elaborazione (la voce di lease relativa alla partizione antenata esiste nella tabella di lease) o se è necessario elaborare una partizione antenata (ad esempio, se la posizione iniziale è `TRIM_HORIZON` o `AT_TIMESTAMP`)
  - Se la partizione aperta nel contesto è un discendente, la KCL controlla la partizione in base alla posizione iniziale e crea dei lease per i suoi elementi principali, se necessario

Sincronizzazione in KCL 2.x, a partire da KCL 2.3 e versioni successive

A partire dalle ultime versioni supportate di KCL 2.x (KCL 2.3) e successive, la libreria ora supporta le seguenti modifiche al processo di sincronizzazione. Queste modifiche alla sincronizzazione lease/partizione riducono significativamente il numero di chiamate API effettuate dalle applicazioni consumer KCL al servizio del flusso di dati Kinesis e ottimizzano la gestione dei lease nell'applicazione consumer KCL.

- Durante l'avvio dell'applicazione, se la tabella di lease è vuota, la KCL utilizza l'opzione di filtro dell'API `ListShard` (il parametro di richiesta `ShardFilter` facoltativo) per recuperare e creare lease solo per uno snapshot di partizioni aperte nel momento specificato dal parametro `ShardFilter`. Il parametro `ShardFilter` consente di filtrare la risposta dell'API `ListShards`. L'unica proprietà richiesta del parametro `ShardFilter` è `Type`. KCL utilizza la proprietà di filtro `Type` e i seguenti valori validi per identificare e restituire uno snapshot delle partizioni aperte che potrebbero richiedere nuovi lease:
  - `AT_TRIM_HORIZON`: la risposta include tutte le partizioni che erano aperte in `TRIM_HORIZON`.
  - `AT_LATEST`: la risposta include solo le partizioni del flusso di dati correntemente aperte.
  - `AT_TIMESTAMP`: la risposta include tutte le partizioni il cui timestamp di inizio è inferiore o uguale al timestamp specificato e il timestamp di fine è maggiore o uguale al timestamp specificato o sono ancora aperte.

  `ShardFilter` viene utilizzato durante la creazione di lease per una tabella di lease vuota per inizializzare i lease per uno snapshot delle partizioni specificate in `RetrievalConfig#initialPositionInStreamExtended`.

  Per ulteriori informazioni su `ShardFilter`, consulta https://docs.aws.amazon.com/kinesis/latest/APIReference/API_ShardFilter.html.

- Invece che tutti i lavoratori eseguano la sincronizzazione. lease/shard synchronization to keep the lease table up to date with the latest shards in the data stream, a single elected worker leader performs the lease/shard
- KCL 2.3 utilizza il parametro `ChildShards` return di `GetRecords` and the `SubscribeToShard` APIs per eseguire la sincronizzazione lease/shard che avviene per gli shard chiusi, consentendo a un worker KCL di creare leasing solo `SHARD_END` per i frammenti secondari dello shard che ha terminato l'elaborazione. Per le applicazioni consumer con velocità di trasmissione effettiva condivisa, questa ottimizzazione della sincronizzazione lease/partizione utilizza il parametro `ChildShards` dell'API `GetRecords`. Per le applicazioni consumer (fan-out migliorato) con velocità di trasmissione effettiva condivisa, questa ottimizzazione della sincronizzazione lease/partizione utilizza il parametro `ChildShards` dell'API `SubscribeToShard`. Per ulteriori informazioni, consulta GetRecords, SubscribeToShards e ChildShard.
- Con le modifiche di cui sopra, il comportamento della KCL sta passando dal modello in cui tutti i worker apprendono tutte le partizioni esistenti a un modello in cui i worker apprendono solo le partizioni secondarie delle partizioni di proprietà di ogni worker. Pertanto, oltre alla sincronizzazione che avviene durante gli eventi di avvio e ripartizionamento delle applicazioni consumer, la KCL ora esegue anche ulteriori scansioni periodiche di partizione/lease per identificare eventuali potenziali buchi nella tabella di lease (in altre parole, per conoscere tutte le nuove partizioni) per garantire l'elaborazione dell'intervallo hash completo del flusso di dati e creare i lease, se necessario. `PeriodicShardSyncManager` è il componente responsabile dell'esecuzione di scansioni periodiche di lease/partizione.

  Per ulteriori informazioni su KCL 2.3, vedere/.java #L201 -L213. `PeriodicShardSyncManager` https://github.com/awslabs/ amazon-kinesis-client blob/master/amazon-kinesis-client/src/main/java/software/amazon/kinesis/leases/LeaseManagementConfig

  In KCL 2.3, sono disponibili nuove opzioni di configurazione per configurare `PeriodicShardSyncManager` in `LeaseManagementConfig`:

Nome	Valore predefinito	Descrizione
leasesRecoveryAuditorExecutionFrequencyMillis	120.000 (2 minuti)	La frequenza (in millisecondi) del lavoro del revisore per la ricerca di lease parziali nella

Nome	Valore predefinito	Descrizione
		tabella di lease. Se il revisore rileva un buco nei lease relativi a uno stream, attiva la sincroniz zazione delle partizioni in base a `leasesRec overyAudi torIncons istencyCo nfidenceT hreshold` .

Nome	Valore predefinito	Descrizione
leasesRecoveryAuditorInconsistencyConfidenceThreshold	3	Soglia di confidenza per il lavoro periodico del revisore volto a determinare se i lease per un flusso di dati nella tabella di lease non sono coerenti. Se il revisore rileva lo stesso insieme di incongruenze consecutivamente per un flusso di dati per questo numero di volte, attiva una sincronizzazione delle partizioni.

Ora vengono inoltre emesse nuove CloudWatch metriche per monitorare lo stato di. `PeriodicShardSyncManager` Per ulteriori informazioni, consulta PeriodicShardSyncManager.

- Inclusa un'ottimizzazione a `HierarchicalShardSyncer` per creare lease solo per un livello di partizioni.

Sincronizzazione in KCL 1.x, a partire da KCL 1.14 e versioni successive

A partire dalle ultime versioni supportate di KCL 1.x (KCL 1.14) e successive, la libreria ora supporta le seguenti modifiche al processo di sincronizzazione. Queste modifiche alla sincronizzazione lease/partizione riducono significativamente il numero di chiamate API effettuate

dalle applicazioni consumer KCL al servizio del flusso di dati Kinesis e ottimizzano la gestione dei lease nell'applicazione consumer KCL.

- Durante l'avvio dell'applicazione, se la tabella di lease è vuota, la KCL utilizza l'opzione di filtro dell'API `ListShard` (il parametro di richiesta `ShardFilter` facoltativo) per recuperare e creare lease solo per uno snapshot di partizioni aperte nel momento specificato dal parametro `ShardFilter`. Il parametro `ShardFilter` consente di filtrare la risposta dell'API `ListShards`. L'unica proprietà richiesta del parametro `ShardFilter` è `Type`. KCL utilizza la proprietà di filtro `Type` e i seguenti valori validi per identificare e restituire uno snapshot delle partizioni aperte che potrebbero richiedere nuovi lease:
  - `AT_TRIM_HORIZON`: la risposta include tutte le partizioni che erano aperte in `TRIM_HORIZON`.
  - `AT_LATEST`: la risposta include solo le partizioni del flusso di dati correntemente aperte.
  - `AT_TIMESTAMP`: la risposta include tutte le partizioni il cui timestamp di inizio è inferiore o uguale al timestamp specificato e il timestamp di fine è maggiore o uguale al timestamp specificato o sono ancora aperte.

  `ShardFilter` viene utilizzato durante la creazione di lease per una tabella di lease vuota per inizializzare i lease per uno snapshot delle partizioni specificate in `KinesisClientLibConfiguration#initialPositionInStreamExtended`.

  Per ulteriori informazioni su `ShardFilter`, consulta https://docs.aws.amazon.com/kinesis/latest/APIReference/API_ShardFilter.html.
- Invece che tutti i lavoratori eseguano la sincronizzazione. lease/shard synchronization to keep the lease table up to date with the latest shards in the data stream, a single elected worker leader performs the lease/shard
- KCL 1.14 utilizza il parametro `ChildShards` return di `GetRecords` and the `SubscribeToShard` APIs per eseguire la sincronizzazione lease/shard che avviene per gli shard chiusi, consentendo a un worker KCL di creare leasing solo `SHARD_END` per i frammenti secondari dello shard che ha terminato l'elaborazione. Per ulteriori informazioni, consultare GetRecords e ChildShard.
- Con le modifiche di cui sopra, il comportamento della KCL sta passando dal modello in cui tutti i worker apprendono tutte le partizioni esistenti a un modello in cui i worker apprendono solo le partizioni secondarie delle partizioni di proprietà di ogni worker. Pertanto, oltre alla sincronizzazione che avviene durante gli eventi di avvio e ripartizionamento delle applicazioni consumer, la KCL ora esegue anche ulteriori scansioni periodiche di partizione/lease per identificare eventuali potenziali buchi nella tabella di lease (in altre parole, per conoscere tutte le nuove partizioni) per garantire l'elaborazione dell'intervallo hash completo del flusso di dati e creare i lease, se necessario.

`PeriodicShardSyncManager` è il componente responsabile dell'esecuzione di scansioni periodiche di lease/partizione.

Quando `KinesisClientLibConfiguration#shardSyncStrategyType` è impostato su `ShardSyncStrategyType.SHARD_END`, `PeriodicShardSync` `leasesRecoveryAuditorInconsistencyConfidenceThreshold` viene utilizzato per determinare la soglia per il numero di scansioni consecutive contenenti buchi nella tabella di lease, dopodiché imporre la sincronizzazione delle partizioni. Quando `KinesisClientLibConfiguration#shardSyncStrategyType` è impostato su `ShardSyncStrategyType.PERIODIC`, `leasesRecoveryAuditorInconsistencyConfidenceThreshold` viene ignorato.

Per ulteriori informazioni su KCL 1.14, vedere/.java #L987 -L999. `PeriodicShardSyncManager` https://github.com/awslabs/ amazon-kinesis-client blob/v1.x/src/main/java/com/amazonaws/ services/kinesis/clientlibrary/lib/worker/KinesisClientLibConfiguration

In KCL 1.14, è disponibile una nuova opzione di configurazione per configurare `PeriodicShardSyncManager` in `LeaseManagementConfig`:

Nome	Valore predefinito	Descrizione
leasesRecoveryAuditorInconsistencyConfidenceThreshold	3	Soglia di confidenza per il lavoro periodico del revisore volto a determinare se i lease per un flusso di dati nella tabella di lease non sono coerenti. Se il revisore rileva lo stesso insieme di incongruenze consecutivamente per un flusso di dati per questo numero di volte, attiva una sincronizzazione delle partizioni.

Ora vengono inoltre emesse nuove CloudWatch metriche per monitorare lo stato di. `PeriodicShardSyncManager` Per ulteriori informazioni, consulta PeriodicShardSyncManager.

- KCL 1.14 ora supporta anche la pulizia differita dei lease. I lease vengono eliminati in modo asincrono da `LeaseCleanupManager` quando viene raggiunto `SHARD_END` o quando una partizione è scaduta, superando il periodo di conservazione del flusso di dati o è stata chiusa a seguito di un'operazione di ripartizionamento.

  Sono disponibili nuove opzioni di configurazione per configurare `LeaseCleanupManager`.

Nome	Valore predefinito	Descrizione
leaseClea nupIntervalMillis	1 minuto	Intervallo in cui eseguire il thread di pulizia dei lease.
completed LeaseClea nupIntervalMillis	5 minuti	Intervallo in cui verificare se un lease è stato completato o meno.
garbageLe aseCleanu pIntervalMillis	30 minuti	Intervallo durante il quale verificare se un lease è inutile (ossia se è stato interrotto oltre il periodo di conservazione del flusso di dati) o meno.

- Inclusa un'ottimizzazione a `KinesisShardSyncer` per creare lease solo per un livello di partizioni.

Elabora più flussi di dati con la stessa applicazione consumer KCL 2.x per Java

Questa sezione descrive le seguenti modifiche a KCL 2.x per Java che consentono di creare applicazioni consumer KCL in grado di elaborare più di un flusso di dati contemporaneamente.

> **Important**
>
> L'elaborazione multistream è supportata solo in KCL 2.x per Java, a partire da KCL 2.3 per Java e versioni successive.

L'elaborazione multistream NON è supportata per altri linguaggi in cui è possibile implementare KCL 2.x.
L'elaborazione multistream NON è supportata in nessuna versione di KCL 1.x.

- MultistreamTracker interfaccia

  Per creare un'applicazione consumer in grado di elaborare più flussi contemporaneamente, è necessario implementare una nuova interfaccia denominata MultistreamTracker. Questa interfaccia include il metodo `streamConfigList` che restituisce l'elenco dei flussi di dati e le relative configurazioni che devono essere elaborati dall'applicazione consumer KCL. Si noti che i flussi di dati in fase di elaborazione possono essere modificati durante il runtime dell'applicazione consumer. `streamConfigList` viene chiamato periodicamente dalla KCL per conoscere le modifiche nei flussi di dati da elaborare.

  Il `streamConfigList` metodo compila l'elenco. StreamConfig

```
package software.amazon.kinesis.common;

import lombok.Data;
import lombok.experimental.Accessors;

@Data
@Accessors(fluent = true)
public class StreamConfig {
    private final StreamIdentifier streamIdentifier;
    private final InitialPositionInStreamExtended initialPositionInStreamExtended;
    private String consumerArn;
}
```

  Nota che `StreamIdentifier` e `InitialPositionInStreamExtended` sono obbligatori, mentre `consumerArn` è facoltativo. È necessario fornire `consumerArn` solo se si utilizza KCL 2.x per implementare un'applicazione consumer fan-out migliorata.

  Per ulteriori informazioni su`StreamIdentifier`, vedere https://github.com/awslabs/amazon-kinesis-client/blob/v2.5.8/amazon-kinesis-client/src/main/java/software/amazon/kinesis/common/StreamIdentifier.java #L129. Per creare un`StreamIdentifier`, ti consigliamo di creare un'istanza multistream da `streamArn` and the disponibile nella versione `streamCreationEpoch` 2.5.0

e successive. In KCL v2.3 e v2.4, che non supportano`streamArm`, crea un'istanza multistream utilizzando il formato. `account-id:StreamName:streamCreationTimestamp` Questo formato sarà obsoleto e non sarà più supportato a partire dalla prossima versione principale.

`MultistreamTracker` include anche una strategia per eliminare i lease di vecchi flussi nella tabella dei lease (`formerStreamsLeasesDeletionStrategy`). Si noti che la strategia NON PUÒ essere modificata durante il runtime dell'applicazione consumer. Per ulteriori informazioni, consulta https://github.com/awslabs/ amazon-kinesis-client / blob/0c5042dadf794fe988438436252a5a8fe70b6b0b/ .java amazon-kinesis-client src/main/java/ software/amazon/kinesis/processor/FormerStreamsLeasesDeletionStrategy

- ConfigsBuilderè una classe a livello di applicazione che puoi utilizzare per specificare tutte le impostazioni di configurazione di KCL 2.x da utilizzare durante la creazione dell'applicazione consumer KCL. `ConfigsBuilder`la classe ora supporta l'interfaccia. `MultistreamTracker` Puoi inizializzarli ConfigsBuilder entrambi con il nome dell'unico flusso di dati da cui consumare i record da:

```
/**
    * Constructor to initialize ConfigsBuilder with StreamName
    * @param streamName
    * @param applicationName
    * @param kinesisClient
    * @param dynamoDBClient
    * @param cloudWatchClient
    * @param workerIdentifier
    * @param shardRecordProcessorFactory
    */
   public ConfigsBuilder(@NonNull String streamName, @NonNull String
applicationName,
           @NonNull KinesisAsyncClient kinesisClient, @NonNull DynamoDbAsyncClient
dynamoDBClient,
           @NonNull CloudWatchAsyncClient cloudWatchClient, @NonNull String
workerIdentifier,
           @NonNull ShardRecordProcessorFactory shardRecordProcessorFactory) {
       this.appStreamTracker = Either.right(streamName);
       this.applicationName = applicationName;
       this.kinesisClient = kinesisClient;
       this.dynamoDBClient = dynamoDBClient;
       this.cloudWatchClient = cloudWatchClient;
       this.workerIdentifier = workerIdentifier;
       this.shardRecordProcessorFactory = shardRecordProcessorFactory;
```

```
    }
```

Oppure puoi inizializzare ConfigsBuilder con `MultiStreamTracker` se desideri implementare un'applicazione consumer KCL che elabora più flussi contemporaneamente.

```
* Constructor to initialize ConfigsBuilder with MultiStreamTracker
     * @param multiStreamTracker
     * @param applicationName
     * @param kinesisClient
     * @param dynamoDBClient
     * @param cloudWatchClient
     * @param workerIdentifier
     * @param shardRecordProcessorFactory
     */
    public ConfigsBuilder(@NonNull MultiStreamTracker multiStreamTracker, @NonNull
 String applicationName,
            @NonNull KinesisAsyncClient kinesisClient, @NonNull DynamoDbAsyncClient
 dynamoDBClient,
            @NonNull CloudWatchAsyncClient cloudWatchClient, @NonNull String
 workerIdentifier,
            @NonNull ShardRecordProcessorFactory shardRecordProcessorFactory) {
        this.appStreamTracker = Either.left(multiStreamTracker);
        this.applicationName = applicationName;
        this.kinesisClient = kinesisClient;
        this.dynamoDBClient = dynamoDBClient;
        this.cloudWatchClient = cloudWatchClient;
        this.workerIdentifier = workerIdentifier;
        this.shardRecordProcessorFactory = shardRecordProcessorFactory;
    }
```

- Con il supporto multistream implementato per la tua applicazione consumer KCL, ogni riga della tabella di lease dell'applicazione ora contiene l'ID della partizione e il nome del flusso dei molteplici flussi di dati elaborati da questa applicazione.
- Quando viene implementato il supporto multistream per la tua applicazione consumer KCL, leaseKey assume la seguente struttura: `account-id:StreamName:streamCreationTimestamp:ShardId`. Ad esempio, `111111111:multiStreamTest-1:12345:shardId-000000000336`.

> **Important**
>
> Quando l'applicazione consumer KCL esistente è configurata per elaborare un solo flusso di dati, leaseKey (che è la chiave hash per la tabella di lease) è l'ID della partizione. Se riconfiguri questa applicazione consumer KCL esistente per elaborare più flussi di dati, la tabella di lease viene interrotta perché con il supporto multistream, la struttura leaseKey deve essere `account-id:StreamName:StreamCreationTimestamp:ShardId`.

Usa KCL con lo Schema Registry AWS Glue

Puoi integrare i tuoi flussi di dati Kinesis con lo Schema Registry. AWS Glue Lo AWS Glue Schema Registry ti consente di scoprire, controllare ed evolvere centralmente gli schemi, garantendo al contempo che i dati prodotti siano convalidati continuamente da uno schema registrato. Uno schema definisce la struttura e il formato di un registro di dati. Uno schema è una specifica con versioni per la pubblicazione, il consumo o l'archiviazione dei dati in modo affidabile. Lo AWS Glue Schema Registry consente di migliorare la qualità e la governance end-to-end dei dati all'interno delle applicazioni di streaming. Per ulteriori informazioni, consulta Registro degli schemi di AWS Glue. Uno dei modi per configurare questa integrazione è tramite la KCL in Java.

> **Important**
>
> Attualmente, l'integrazione tra Kinesis Data AWS Glue Streams e Schema Registry è supportata solo per i flussi di dati Kinesis che utilizzano i consumatori KCL 2.3 implementati in Java. Il supporto multilingue non viene fornito. I consumer KCL 1.0 non sono supportati. I consumer KCL 2.x precedenti a KCL 2.3 non sono supportati.

Per istruzioni dettagliate su come configurare l'integrazione di Kinesis Data Streams con Schema Registry utilizzando KCL, consulta la sezione «Interazione con i dati utilizzando le librerie KPL/KCL» in Caso d'uso: integrazione di Amazon Kinesis Data Streams con il registro dello schema Glue. AWS

Sviluppa consumatori personalizzati con un throughput condiviso

> **Important**
>
> Le versioni 1.x e 2.x di Amazon Kinesis Client Library (KCL) sono obsolete. KCL 1.x sarà disponibile il 30 gennaio 2026. end-of-support Ti consigliamo vivamente di migrare le tue

applicazioni KCL utilizzando la versione 1.x all'ultima versione di KCL prima del 30 gennaio 2026. Per trovare la versione più recente di KCL, consulta la pagina Amazon Kinesis Client Library su. GitHub Per informazioni sulle ultime versioni di KCL, consulta. Usa la libreria client Kinesis Per informazioni sulla migrazione da KCL 1.x a KCL 3.x, consulta. Migrazione da KCL 1.x a KCL 3.x

Se non è necessaria una velocità di trasmissione effettiva dedicata durante la ricezione dei dati dal flusso di dati Kinesis e se non sono necessari ritardi di propagazione della lettura inferiori a 200 ms, puoi creare applicazioni consumater come descritto nei seguenti argomenti. È possibile utilizzare la Kinesis Client Library (KCL) o il. AWS SDK per Java

Argomenti

- Sviluppa consumatori personalizzati con un throughput condiviso utilizzando KCL

Per ulteriori informazioni sulla creazione di consumer che possono ricevere record dal flusso di dati Kinesis con velocità di trasmissione effettiva dedicata, consulta Sviluppa consumatori con fan-out migliorati con un throughput dedicato.

Sviluppa consumatori personalizzati con un throughput condiviso utilizzando KCL

Important

Le versioni 1.x e 2.x di Amazon Kinesis Client Library (KCL) sono obsolete. KCL 1.x sarà disponibile il 30 gennaio 2026. end-of-support Ti consigliamo vivamente di migrare le tue applicazioni KCL utilizzando la versione 1.x all'ultima versione di KCL prima del 30 gennaio 2026. Per trovare la versione più recente di KCL, consulta la pagina Amazon Kinesis Client Library su. GitHub Per informazioni sulle ultime versioni di KCL, consulta. Usa la libreria client Kinesis Per informazioni sulla migrazione da KCL 1.x a KCL 3.x, consulta. Migrazione da KCL 1.x a KCL 3.x

Uno dei metodi per sviluppare un'applicazione consumer personalizzata con velocità di trasmissione effettiva condivisa consiste nell'utilizzare la Kinesis Client Library (KCL).

Scegli tra i seguenti argomenti per la versione di KCL che stai utilizzando.

Argomenti

- Sviluppa i consumatori di KCL 1.x
- Sviluppa KCL 2.x Consumers

Sviluppa i consumatori di KCL 1.x

> **Important**
>
> Le versioni 1.x e 2.x di Amazon Kinesis Client Library (KCL) sono obsolete. KCL 1.x sarà disponibile il 30 gennaio 2026. end-of-support Ti consigliamo vivamente di migrare le tue applicazioni KCL utilizzando la versione 1.x all'ultima versione di KCL prima del 30 gennaio 2026. Per trovare la versione più recente di KCL, consulta la pagina Amazon Kinesis Client Library su. GitHub Per informazioni sulle ultime versioni di KCL, consulta. Usa la libreria client Kinesis Per informazioni sulla migrazione da KCL 1.x a KCL 3.x, consulta. Migrazione da KCL 1.x a KCL 3.x

È possibile sviluppare un'applicazione consumer per flusso di dati Amazon Kinesis utilizzando la Kinesis Client Library (KCL).

Per ulteriori informazioni su KCL, consulta Informazioni su KCL (versioni precedenti).

Scegli tra i seguenti argomenti a seconda dell'opzione che desideri utilizzare.

Indice

- Sviluppa un utente di Kinesis Client Library in Java
- Sviluppa un utente della Kinesis Client Library in Node.js
- Sviluppa un utente di Kinesis Client Library in .NET
- Sviluppa un utente della Kinesis Client Library in Python
- Sviluppa un utente di Kinesis Client Library in Ruby

Sviluppa un utente di Kinesis Client Library in Java

> **Important**
>
> Le versioni 1.x e 2.x di Amazon Kinesis Client Library (KCL) sono obsolete. KCL 1.x sarà disponibile il 30 gennaio 2026. end-of-support Ti consigliamo vivamente di migrare le tue applicazioni KCL utilizzando la versione 1.x all'ultima versione di KCL prima del 30 gennaio

2026. Per trovare la versione più recente di KCL, consulta la pagina Amazon Kinesis Client Library su. GitHub Per informazioni sulle ultime versioni di KCL, consulta. Usa la libreria client Kinesis Per informazioni sulla migrazione da KCL 1.x a KCL 3.x, consulta. Migrazione da KCL 1.x a KCL 3.x

È possibile utilizzare la Kinesis Client Library (KCL) per creare applicazioni che elaborano dati dai tuoi flussi di dati Kinesis. La Kinesis Client Library è disponibile in più linguaggi. In questo argomento viene discusso Java. Per visualizzare il riferimento a Javadoc, consultate l'argomento Javadoc per Class.AWS AmazonKinesisClient

Per scaricare Java KCL da GitHub, vai a Kinesis Client Library (Java). Per individuare la KCL Java su Apache Maven, vai alla pagina Risultati di ricerca di KCL. Per scaricare il codice di esempio per un'applicazione consumer Java KCL da GitHub, vai alla pagina del progetto di esempio KCL for Java su. GitHub

L'applicazione di esempio utilizza Apache Commons Logging. È possibile modificare la configurazione di registro nel metodo statico `configure` definito nel file `AmazonKinesisApplicationSample.java`. Per ulteriori informazioni su come utilizzare Apache Commons Logging con Log4j e le applicazioni AWS Java, consulta Logging with Log4j nella Developer Guide.AWS SDK per Java

È necessario completare le seguenti attività durante l'implementazione di un'applicazione consumer KCL in Java:

Attività

- Implementa i metodi del processore IRecord
- Implementa una fabbrica di classi per l'interfaccia IRecord Processor
- Crea un lavoratore
- Modificare le proprietà di configurazione
- Esegui la migrazione alla versione 2 dell'interfaccia del processore di registrazione

Implementa i metodi del processore IRecord

La KCL supporta attualmente due versioni dell'interfaccia `IRecordProcessor`: l'interfaccia originale è disponibile con la prima versione della KCL e la versione 2 è disponibile a partire da KCL versione 1.5.0. Entrambe le interfacce sono completamente supportate. La scelta dipende dai tuoi requisiti

specifici di scenario. Fai riferimento ai tuoi Javadocs locali o al codice sorgente per visualizzare tutte le differenze. Le seguenti sezioni delineano l'implementazione minima per iniziare.

IRecordVersioni del processore

- Interfaccia originale (Versione 1)
- Interfaccia aggiornata (versione 2)

Interfaccia originale (Versione 1)

L'interfaccia `IRecordProcessor` originale (`package com.amazonaws.services.kinesis.clientlibrary.interfaces`) espone i seguenti metodi di processore del record che il tuo consumer deve implementare. L'esempio fornisce implementazioni che è possibile utilizzare come punto di partenza (consulta `AmazonKinesisApplicationSampleRecordProcessor.java`).

```
public void initialize(String shardId)
public void processRecords(List<Record> records, IRecordProcessorCheckpointer
 checkpointer)
public void shutdown(IRecordProcessorCheckpointer checkpointer, ShutdownReason reason)
```

initialize

La KCL chiama il metodo `initialize` quando viene creata un'istanza del processore di record, passando un ID della partizione specifico come parametro. Questo processore di record elabora esclusivamente questo shard e, in genere, è vero anche il contrario (questo shard è elaborato solo da questo processore di record). Tuttavia, il tuo consumer deve tenere conto della possibilità che un record di dati possa essere elaborato più di una volta. Il flusso di dati Kinesis ha una semantica almeno una volta, il che significa che ogni record di dati da una partizione viene elaborato almeno una volta da un worker nel tuo consumer. Per ulteriori informazioni sui casi in cui un determinato shard può essere elaborato da più di un lavoratore, consulta Usa il resharding, lo scaling e l'elaborazione parallela per modificare il numero di shard.

```
public void initialize(String shardId)
```

processRecords

La KCL chiama il metodo `processRecords` e passa un elenco di record di dati dalla partizione specificata dal metodo `initialize(shardId)`. Il processore di record elabora i dati in questi

record in base alla semantica del consumer. Ad esempio, il worker potrebbe eseguire una trasformazione dei dati e, successivamente, archiviare il risultato in un bucket Amazon Simple Storage Service (Amazon S3).

```
public void processRecords(List<Record> records, IRecordProcessorCheckpointer
 checkpointer)
```

Oltre ai dati stessi, il record contiene anche un numero di sequenza e una chiave di partizione. Il lavoratore può utilizzare questi valori quando elabora i dati. Ad esempio, il lavoratore può scegliere il bucket S3 in cui archiviare i dati in base al valore della chiave di partizione. La classe `Record` espone i seguenti metodi che forniscono l'accesso ai dati del record, al numero di sequenza e alla chiave di partizione.

```
record.getData()
record.getSequenceNumber()
record.getPartitionKey()
```

Nell'esempio, il metodo privato `processRecordsWithRetries` ha un codice che mostra in che modo un lavoratore può accedere ai dati del record, al numero di sequenza e alla chiave di partizione.

Il flusso di dati Kinesis richiede che il processore di record tenga traccia dei record che sono già stati elaborati in una partizione. La KCL si occupa di questo monitoraggio per te, passando un checkpointer (`IRecordProcessorCheckpointer`) a `processRecords`. Il processore di record chiama il metodo `checkpoint` in questa interfaccia per comunicare alla KCL quanto si è progredito nell'elaborazione dei record nella partizione. In caso di errore del worker, la KCL utilizza queste informazioni per riavviare l'elaborazione della partizione nell'ultimo record elaborato conosciuto.

Per le operazioni di divisione o unione, la KCL non avvierà l'elaborazione delle nuove partizioni fino a quando i processori delle partizioni originali non avranno chiamato `checkpoint` per segnalare che l'intera elaborazione delle partizioni originali è completa.

Se non viene passato un parametro, la KCL suppone che la chiamata a `checkpoint` significa che tutti i record sono stati elaborati, fino all'ultimo record passato al processore di record. Pertanto, il processore di record deve chiamare `checkpoint` solo dopo aver elaborato tutti i record nell'elenco passato al processore. I processori di record non devono chiamare `checkpoint` in ciascuna chiamata a `processRecords`. Un processore potrebbe, per esempio, chiamare `checkpoint` in ogni terza chiamata a `processRecords`. Puoi specificare, in modo facoltativo, il numero di

sequenza esatto di un record come parametro per `checkpoint`. In questo caso, la KCL presuppone che tutti i record siano stati elaborati esclusivamente fino a tale record.

Nell'esempio, il metodo privato `checkpoint` mostra come effettuare la chiamata a `IRecordProcessorCheckpointer.checkpoint` utilizzando la gestione delle eccezioni e la logica dei nuovi tentativi appropriate.

La KCL si basa su `processRecords` per gestire eventuali eccezioni generate dall'elaborazione dei record di dati. Se viene generata un'eccezione da `processRecords`, la KCL omette i record di dati passati prima dell'eccezione. Ciò significa che questi record non sono inviati nuovamente al processore di record che ha generato l'eccezione o a qualsiasi altro processore di record nel consumer.

shutdown

La KCL chiama il metodo `shutdown` sia al termine dell'elaborazione (il motivo dell'arresto è `TERMINATE`) che quando il worker non risponde più (il motivo dell'arresto è `ZOMBIE`).

```
public void shutdown(IRecordProcessorCheckpointer checkpointer, ShutdownReason reason)
```

L'elaborazione termina quando il processore di record non riceve ulteriori record dallo shard, perché lo shard è stato frazionato o fuso o perché il flusso è stato eliminato.

La KCL trasferisce inoltre un'interfaccia `IRecordProcessorCheckpointer` a `shutdown`. Se il motivo dell'arresto è `TERMINATE`, il processore di record deve terminare l'elaborazione di qualsiasi record di dati e, di seguito, chiamare il metodo `checkpoint` in questa interfaccia.

Interfaccia aggiornata (versione 2)

L'interfaccia `IRecordProcessor` aggiornata (`package com.amazonaws.services.kinesis.clientlibrary.interfaces.v2`) espone i seguenti metodi di processore del record che il tuo consumer deve implementare:

```
void initialize(InitializationInput initializationInput)
void processRecords(ProcessRecordsInput processRecordsInput)
void shutdown(ShutdownInput shutdownInput)
```

Tutti gli argomenti dalla versione originale dell'interfaccia sono accessibili tramite metodi get negli oggetti del container. Ad esempio, per recuperare l'elenco dei record in `processRecords()`, è possibile utilizzare `processRecordsInput.getRecords()`.

A partire dalla versione 2 di questa interfaccia (KCL 1.5.0 e versioni successive), i seguenti nuovi input sono disponibili in aggiunta agli input forniti dall'interfaccia originale:

Numero di sequenza di partenza

Nell'oggetto `InitializationInput` passato all'operazione `initialize()`, il numero di sequenza iniziale a partire da cui i record verrebbero forniti all'istanza del processore di record. Questo è l'ultimo numero di sequenza in cui è stato eseguito il checkpoint dall'istanza del processore di record che aveva precedentemente elaborato lo stesso shard. Questi dati sono forniti nel caso in cui la tua applicazione necessiti di queste informazioni.

Numero di sequenza di checkpoint in sospeso

Nell'oggetto `InitializationInput` passato all'operazione `initialize()`, il numero di sequenza di checkpoint in sospeso (se del caso) che non è stato possibile confermare prima dell'arresto dell'istanza precedente del processore di record.

Implementa una fabbrica di classi per l'interfaccia IRecord Processor

È inoltre necessario implementare un generatore per la classe che implementa i metodi del processore di record. Quando il tuo consumer avvia un'istanza del lavoratore, passa un riferimento a questo generatore.

Il campione implementa il generatore di classe nel file `AmazonKinesisApplicationSampleRecordProcessorFactory.java` utilizzando l'interfaccia del processore di record originale. Se si desidera che il generatore di classe crei processori di record della versione 2, utilizzi il nome del pacchetto `com.amazonaws.services.kinesis.clientlibrary.interfaces.v2`.

```
public class SampleRecordProcessorFactory implements IRecordProcessorFactory {
    /**
    * Constructor.
    */
    public SampleRecordProcessorFactory() {
        super();
    }
    /**
    * {@inheritDoc}
    */
    @Override
```

```
    public IRecordProcessor createProcessor() {
        return new SampleRecordProcessor();
    }
}
```

Crea un lavoratore

Come discusso nella Implementa i metodi del processore IRecord, ci sono due versioni dell'interfaccia del processore di record KCL da cui scegliere; ciò influenza il modo in cui è possibile creare un worker. L'interfaccia del processore di record originale utilizza la seguente struttura di codice per creare un lavoratore:

```
final KinesisClientLibConfiguration config = new KinesisClientLibConfiguration(...)
final IRecordProcessorFactory recordProcessorFactory = new RecordProcessorFactory();
final Worker worker = new Worker(recordProcessorFactory, config);
```

Con la versione 2 dell'interfaccia del processore di record, è possibile utilizzare `Worker.Builder` per creare un lavoratore senza dover preoccuparsi di quale costruttore utilizzare e dell'ordine degli argomenti. L'interfaccia del processore di record aggiornata utilizza la seguente struttura di codice per creare un lavoratore:

```
final KinesisClientLibConfiguration config = new KinesisClientLibConfiguration(...)
final IRecordProcessorFactory recordProcessorFactory = new RecordProcessorFactory();
final Worker worker = new Worker.Builder()
    .recordProcessorFactory(recordProcessorFactory)
    .config(config)
    .build();
```

Modificare le proprietà di configurazione

L'esempio fornisce valori di default per le proprietà di configurazione. Questi dati di configurazione per il lavoratore sono poi consolidati in un oggetto `KinesisClientLibConfiguration`. Questo oggetto e un riferimento al generatore di classe per `IRecordProcessor` sono passati nella chiamata che avvia un'istanza del lavoratore. È possibile sostituire una qualsiasi di queste proprietà con i tuoi valori utilizzando un file di proprietà Java (consulta `AmazonKinesisApplicationSample.java`).

Nome applicazione

La KCL richiede un nome dell'applicazione univoco per tutte le applicazioni e per tutte le tabelle Amazon DynamoDB nella stessa Regione. La biblioteca utilizza il valore di configurazione del nome dell'applicazione nei seguenti modi:

- Si suppone che tutti i lavoratori associati con questo nome dell'applicazione stiano lavorando insieme nello stesso flusso. Questi lavoratori potrebbero essere distribuiti su più istanze. Se si esegue un'istanza aggiuntiva dello stesso codice dell'applicazione, ma con un nome dell'applicazione diverso, la KCL tratta la seconda istanza come un'applicazione completamente separata che opera anch'essa nello stesso flusso.
- La KCL crea una tabella DynamoDB con il nome dell'applicazione e la utilizza per mantenere le informazioni sullo stato (ad esempio, checkpoint e mappatura worker-partizione) per l'applicazione. Ogni applicazione ha la propria tabella DynamoDB. Per ulteriori informazioni, consulta Utilizza una tabella di leasing per tenere traccia degli shard elaborati dall'applicazione consumer KCL.

Configurare le credenziali

È necessario rendere disponibili le AWS credenziali a uno dei provider di credenziali della catena di provider di credenziali predefinita. Ad esempio, se esegui il tuo consumer su un' EC2 istanza, ti consigliamo di avviare l'istanza con un ruolo IAM. AWS le credenziali che riflettono le autorizzazioni associate a questo ruolo IAM vengono rese disponibili alle applicazioni sull'istanza tramite i relativi metadati dell'istanza. Questo è il modo più sicuro per gestire le credenziali per un consumatore che esegue un'istanza. EC2

L'applicazione di esempio prova prima a recuperare le credenziali IAM dai metadati dell'istanza:

```
credentialsProvider = new InstanceProfileCredentialsProvider();
```

Se l'applicazione di esempio non è in grado di ottenere le credenziali dai metadati dell'istanza, tenta di recuperare le credenziali da un file proprietà:

```
credentialsProvider = new ClasspathPropertiesFileCredentialsProvider();
```

Per ulteriori informazioni sui metadati delle istanze, consulta Instance Metadata nella Amazon EC2 User Guide.

Usa l'ID del lavoratore per più istanze

L'esempio di codice di inizializzazione crea un ID per il lavoratore, `workerId`utilizzando il nome del computer locale e aggiungendo un identificatore univoco globale come illustrato nel seguente frammento di codice. Questo approccio supporta lo scenario di più istanze dell'applicazione di consumo in esecuzione in un singolo computer.

```
String workerId = InetAddress.getLocalHost().getCanonicalHostName() + ":" +
 UUID.randomUUID();
```

Esegui la migrazione alla versione 2 dell'interfaccia del processore di registrazione

Se si desidera migrare il codice che utilizza l'interfaccia originale, in aggiunta ai passaggi descritti in precedenza, sono necessari i seguenti passaggi:

1. Cambia la classe del tuo processore di record per importare la versione 2 dell'interfaccia del processore di record:

   ```
   import com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor;
   ```

2. Cambia i riferimenti per gli input per utilizzare i metodi `get` negli oggetti del container. Ad esempio, nell'operazione `shutdown()`, cambia "`checkpointer`" con "`shutdownInput.getCheckpointer()`".
3. Cambia la classe del generatore del processore di record per importare la versione 2 dell'interfaccia del generatore del processore di record:

   ```
   import
    com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessorFactory;
   ```

4. Cambia la costruzione del lavoratore per utilizzare `Worker.Builder`. Per esempio:

   ```
   final Worker worker = new Worker.Builder()
       .recordProcessorFactory(recordProcessorFactory)
       .config(config)
       .build();
   ```

Sviluppa un utente della Kinesis Client Library in Node.js

> **Important**
>
> Le versioni 1.x e 2.x di Amazon Kinesis Client Library (KCL) sono obsolete. KCL 1.x sarà disponibile il 30 gennaio 2026. end-of-support Ti consigliamo vivamente di migrare le tue applicazioni KCL utilizzando la versione 1.x all'ultima versione di KCL prima del 30 gennaio 2026. Per trovare la versione più recente di KCL, consulta la pagina Amazon Kinesis Client Library su. GitHub Per informazioni sulle ultime versioni di KCL, consulta. Usa la libreria client Kinesis Per informazioni sulla migrazione da KCL 1.x a KCL 3.x, consulta. Migrazione da KCL 1.x a KCL 3.x

È possibile utilizzare la Kinesis Client Library (KCL) per creare applicazioni che elaborano dati dai tuoi flussi di dati Kinesis. La Kinesis Client Library è disponibile in più linguaggi. In questo argomento viene discusso Node.js.

KCL è una libreria Java; il supporto per linguaggi diversi da Java viene fornito utilizzando un'interfaccia multilingue chiamata. MultiLangDaemon Questo daemon è basato su Java e viene eseguito in background quando si utilizza un linguaggio KCL diverso da Java. Pertanto, se installi KCL per Node.js e scrivi la tua app consumer interamente in Node.js, avrai comunque bisogno che Java sia installato sul tuo sistema a causa di. MultiLangDaemon Inoltre, MultiLangDaemon presenta alcune impostazioni predefinite che potresti dover personalizzare in base al tuo caso d'uso, ad esempio la AWS regione a cui si connette. Per ulteriori informazioni su MultiLangDaemon on GitHub, vai alla pagina del MultiLangDaemon progetto KCL.

Per scaricare il file KCL Node.js da GitHub, vai alla Kinesis Client Library (Node.js).

Download di codice di esempio

Ci sono due esempi di codice disponibili per KCL in Node.js:

- basic-sample

  Viene utilizzato nelle seguenti sezioni per illustrare i concetti fondamentali della costruzione di un'applicazione consumer KCL in Node.js.
- click-stream-sample

Leggermente più avanzato e utilizza uno scenario reale. Da utilizzare dopo avere acquisito familiarità con il codice di esempio di base. Questo esempio non è discusso qui, ma dispone di un file README con ulteriori informazioni.

È necessario completare le seguenti attività durante l'implementazione di un'applicazione consumer KCL in Node.js:

Attività

- Implementa il processore di registrazione
- Modificare le proprietà di configurazione

Implementa il processore di registrazione

Il consumer più semplice possibile che utilizza la KCL per Node.js deve implementare una funzione `recordProcessor`, che a sua volta contiene le funzioni `initialize`, `processRecords` e `shutdown`. L'esempio fornisce un'implementazione che è possibile utilizzare come punto di partenza (consulta `sample_kcl_app.js`).

```
function recordProcessor() {
  // return an object that implements initialize, processRecords and shutdown
 functions.}
```

initialize

La KCL chiama la funzione `initialize` quando il processore di record si avvia. Questo processore di record elabora esclusivamente l'ID dello shard passato come `initializeInput.shardId` e, in genere, è vero anche il contrario (questo shard è elaborato solo da questo processore di record). Tuttavia, il tuo consumer deve tenere conto della possibilità che un record di dati possa essere elaborato più di una volta. Ciò si verifica perché il flusso di dati Kinesis ha una semantica almeno una volta, il che significa che ogni record di dati da una partizione viene elaborato almeno una volta da un worker nel tuo consumer. Per ulteriori informazioni sui casi in cui un determinato shard può essere elaborato da più di un lavoratore, consulta Usa il resharding, lo scaling e l'elaborazione parallela per modificare il numero di shard.

```
initialize: function(initializeInput, completeCallback)
```

processRecords

La KCL chiama questa funzione con input che contiene un elenco di record di dati dalla partizione specificata alla funzione `initialize`. Il processore di record che implementi elabora i dati in questi record in base alla semantica del tuo consumer. Ad esempio, il worker potrebbe eseguire una trasformazione dei dati e, successivamente, archiviare il risultato in un bucket Amazon Simple Storage Service (Amazon S3).

```
processRecords: function(processRecordsInput, completeCallback)
```

Oltre ai dati stessi, il record contiene anche un numero di sequenza e una chiave di partizione, che il lavoratore può utilizzare durante l'elaborazione dei dati. Ad esempio, il lavoratore può scegliere il bucket S3 in cui archiviare i dati in base al valore della chiave di partizione. Il dizionario `record` espone le seguenti coppie chiave-valore per accedere ai dati del record, al numero di sequenza e alla chiave di partizione:

```
record.data
record.sequenceNumber
record.partitionKey
```

Tieni presente che i dati sono codificati in Base64.

Nell'esempio di base, la funzione `processRecords` ha un codice che mostra in che modo un lavoratore può accedere ai dati del record, al numero di sequenza e alla chiave di partizione.

Il flusso di dati Kinesis richiede che il processore di record tenga traccia dei record che sono già stati elaborati in una partizione. La KCL si occupa di questo monitoraggio per un oggetto `checkpointer` passato come `processRecordsInput.checkpointer`. Il tuo processore di record chiama la funzione `checkpointer.checkpoint` per comunicare alla KCL quanto si è progredito nell'elaborazione dei record nella partizione. In caso di errore del worker, la KCL utilizza queste informazioni quando si riavvia l'elaborazione della partizione in modo tale che l'elaborazione continua dall'ultimo record elaborato conosciuto.

Per le operazioni di divisione o unione, la KCL non avvia l'elaborazione delle nuove partizioni fino a quando i processori delle partizioni originali non avranno chiamato `checkpoint` per segnalare che l'intera elaborazione delle partizioni originali è completa.

Se non viene passato il numero di sequenza alla funzione `checkpoint`, la KCL suppone che la chiamata a `checkpoint` significa che tutti i record sono stati elaborati, fino all'ultimo record passato al processore di record. Pertanto, il processore di record deve chiamare `checkpoint` solo dopo aver elaborato tutti i record nell'elenco passato al processore. I processori di record non devono chiamare

`checkpoint` in ciascuna chiamata a `processRecords`. Un processore potrebbe, per esempio, chiamare `checkpoint` a ogni terza chiamata o durante un evento esterno al tuo processore di record, ad esempio un servizio personalizzato di verifica/convalida che hai implementato.

Puoi specificare, in modo facoltativo, il numero di sequenza esatto di un record come parametro per `checkpoint`. In questo caso, la KCL presuppone che tutti i record siano stati elaborati esclusivamente fino a tale record.

L'applicazione di esempio di base mostra la chiamata più semplice possibile alla funzione `checkpointer.checkpoint`. È possibile aggiungere le altre logiche di creazione di checkpoint di cui hai bisogno per il tuo consumer a questo punto della funzione.

shutdown

La KCL chiama la funzione `shutdown` sia al termine dell'elaborazione (`shutdownInput.reason` è `TERMINATE`) che quando il worker non risponde più (`shutdownInput.reason` è `ZOMBIE`).

```
shutdown: function(shutdownInput, completeCallback)
```

L'elaborazione termina quando il processore di record non riceve ulteriori record dallo shard, perché lo shard è stato frazionato o fuso o perché il flusso è stato eliminato.

La KCL trasferisce inoltre un oggetto `shutdownInput.checkpointer` a `shutdown`. Se il motivo dell'arresto è `TERMINATE`, è necessario assicurarsi che il processore di record abbia terminato l'elaborazione di qualsiasi record di dati e, di seguito, chiamare la funzione `checkpoint` in questa interfaccia.

Modificare le proprietà di configurazione

L'esempio fornisce valori di default per le proprietà di configurazione. È possibile sostituire una qualsiasi di queste proprietà con i tuoi valori (consulta `sample.properties` nell'esempio di base).

Nome applicazione

La KCL richiede un nome dell'applicazione univoco per tutte le applicazioni e per tutte le tabelle Amazon DynamoDB nella stessa Regione. La biblioteca utilizza il valore di configurazione del nome dell'applicazione nei seguenti modi:

- Si suppone che tutti i lavoratori associati con questo nome dell'applicazione stiano lavorando insieme nello stesso flusso. Questi lavoratori potrebbero essere distribuiti su più istanze.
  Se si esegue un'istanza aggiuntiva dello stesso codice dell'applicazione, ma con un nome

dell'applicazione diverso, la KCL tratta la seconda istanza come un'applicazione completamente separata che opera anch'essa nello stesso flusso.

- La KCL crea una tabella DynamoDB con il nome dell'applicazione e la utilizza per mantenere le informazioni sullo stato (ad esempio, checkpoint e mappatura worker-partizione) per l'applicazione. Ogni applicazione ha la propria tabella DynamoDB. Per ulteriori informazioni, consulta Utilizza una tabella di leasing per tenere traccia degli shard elaborati dall'applicazione consumer KCL.

Configurare le credenziali

È necessario rendere disponibili le AWS credenziali a uno dei provider di credenziali della catena di provider di credenziali predefinita. Puoi utilizzare la proprietà `AWSCredentialsProvider` per impostare un provider di credenziali. Il file `sample.properties` deve rendere le tue credenziali disponibili per uno dei provider di credenziali nella catena di provider di credenziali di default. Se esegui il tuo consumer su un' EC2istanza Amazon, ti consigliamo di configurare l'istanza con un ruolo IAM. AWS le credenziali che riflettono le autorizzazioni associate a questo ruolo IAM vengono rese disponibili alle applicazioni sull'istanza tramite i relativi metadati dell'istanza. Questo è il modo più sicuro per gestire le credenziali per un'applicazione consumer in esecuzione su un'istanza. EC2

Nell'esempio seguente si configura la KCL per elaborare un flusso di dati Kinesis denominato `kclnodejssample` utilizzando il processore di record fornito in `sample_kcl_app.js`:

```
# The Node.js executable script
executableName = node sample_kcl_app.js
# The name of an Amazon Kinesis stream to process
streamName = kclnodejssample
# Unique KCL application name
applicationName = kclnodejssample
# Use default AWS credentials provider chain
AWSCredentialsProvider = DefaultAWSCredentialsProviderChain
# Read from the beginning of the stream
initialPositionInStream = TRIM_HORIZON
```

Sviluppa un utente di Kinesis Client Library in .NET

> **Important**
>
> Le versioni 1.x e 2.x di Amazon Kinesis Client Library (KCL) sono obsolete. KCL 1.x sarà disponibile il 30 gennaio 2026. end-of-support Ti consigliamo vivamente di migrare le tue applicazioni KCL utilizzando la versione 1.x all'ultima versione di KCL prima del 30 gennaio

2026. Per trovare la versione più recente di KCL, consulta la pagina Amazon Kinesis Client Library su. GitHub Per informazioni sulle ultime versioni di KCL, consulta. Usa la libreria client Kinesis Per informazioni sulla migrazione da KCL 1.x a KCL 3.x, consulta. Migrazione da KCL 1.x a KCL 3.x

È possibile utilizzare la Kinesis Client Library (KCL) per creare applicazioni che elaborano dati dai tuoi flussi di dati Kinesis. La Kinesis Client Library è disponibile in più linguaggi. In questo argomento viene discusso .NET.

KCL è una libreria Java; il supporto per linguaggi diversi da Java viene fornito utilizzando un'interfaccia multilingue chiamata. MultiLangDaemon Questo daemon è basato su Java e viene eseguito in background quando si utilizza un linguaggio KCL diverso da Java. Pertanto, se installi KCL per .NET e scrivi la tua app consumer interamente in .NET, avrai comunque bisogno che Java sia installato sul tuo sistema a causa di. MultiLangDaemon Inoltre, MultiLangDaemon presenta alcune impostazioni predefinite che potresti dover personalizzare in base al tuo caso d'uso, ad esempio la AWS regione a cui si connette. Per ulteriori informazioni su MultiLangDaemon on GitHub, vai alla pagina del MultiLangDaemon progetto KCL.

Per scaricare il.NET KCL da GitHub, vai a Kinesis Client Library (.NET). Per scaricare il codice di esempio per un'applicazione consumer di.NET KCL, vai alla pagina del progetto KCL for .NET sample consumer su. GitHub

È necessario completare le seguenti attività durante l'implementazione di un'applicazione consumer KCL in .NET:

Attività

- Implementa i metodi della classe IRecord Processor
- Modificare le proprietà di configurazione

Implementa i metodi della classe IRecord Processor

Il consumer deve implementare i seguenti metodi per `IRecordProcessor`. Il consumer di esempio fornisce implementazioni che è possibile utilizzare come punto di partenza (consulta la classe `SampleRecordProcessor` in `SampleConsumer/AmazonKinesisSampleConsumer.cs`).

```
public void Initialize(InitializationInput input)
public void ProcessRecords(ProcessRecordsInput input)
```

```
public void Shutdown(ShutdownInput input)
```

Inizializzazione

La KCL chiama questo metodo quando viene creata un'istanza del processore di record, passando un ID della partizione specifico nel parametro `input` (`input.ShardId`). Questo processore di record elabora esclusivamente questo shard e, in genere, è vero anche il contrario (questo shard è elaborato solo da questo processore di record). Tuttavia, il tuo consumer deve tenere conto della possibilità che un record di dati possa essere elaborato più di una volta. Ciò si verifica perché il flusso di dati Kinesis ha una semantica almeno una volta, il che significa che ogni record di dati da una partizione viene elaborato almeno una volta da un worker nel tuo consumer. Per ulteriori informazioni sui casi in cui un determinato shard può essere elaborato da più di un lavoratore, consulta Usa il resharding, lo scaling e l'elaborazione parallela per modificare il numero di shard.

```
public void Initialize(InitializationInput input)
```

ProcessRecords

La KCL chiama questo metodo e passa una lista di record di dati nel parametro `input` (`input.Records`) dalla partizione specificata dal metodo `Initialize`. Il processore di record che implementi elabora i dati in questi record in base alla semantica del tuo consumer. Ad esempio, il worker potrebbe eseguire una trasformazione dei dati e, successivamente, archiviare il risultato in un bucket Amazon Simple Storage Service (Amazon S3).

```
public void ProcessRecords(ProcessRecordsInput input)
```

Oltre ai dati stessi, il record contiene anche un numero di sequenza e una chiave di partizione. Il lavoratore può utilizzare questi valori quando elabora i dati. Ad esempio, il lavoratore può scegliere il bucket S3 in cui archiviare i dati in base al valore della chiave di partizione. La classe `Record` espone quanto segue per accedere ai dati del record, al numero di sequenza e alla chiave di partizione:

```
byte[] Record.Data
string Record.SequenceNumber
string Record.PartitionKey
```

Nell'esempio, il metodo `ProcessRecordsWithRetries` ha un codice che mostra in che modo un lavoratore può accedere ai dati del record, al numero di sequenza e alla chiave di partizione.

Il flusso di dati Kinesis richiede che il processore di record tenga traccia dei record che sono già stati elaborati in una partizione. La KCL si occupa di questo monitoraggio per te, passando un oggetto `Checkpointer` a `ProcessRecords` (`input.Checkpointer`). Il processore di record chiama il metodo`Checkpointer.Checkpoint` per comunicare alla KCL quanto si è progredito nell'elaborazione dei record nella partizione. In caso di errore del worker, la KCL utilizza queste informazioni per riavviare l'elaborazione della partizione nell'ultimo record elaborato conosciuto.

Per le operazioni di divisione o unione, la KCL non avvia l'elaborazione delle nuove partizioni fino a quando i processori delle partizioni originali non avranno chiamato `Checkpointer.Checkpoint` per segnalare che l'intera elaborazione delle partizioni originali è completa.

Se non viene passato un parametro, la KCL suppone che la chiamata a `Checkpointer.Checkpoint` significa che tutti i record sono stati elaborati, fino all'ultimo record passato al processore di record. Pertanto, il processore di record deve chiamare `Checkpointer.Checkpoint` solo dopo aver elaborato tutti i record nell'elenco passato al processore. I processori di record non devono chiamare `Checkpointer.Checkpoint` in ciascuna chiamata a `ProcessRecords`. Un processore potrebbe, per esempio, chiamare `Checkpointer.Checkpoint` in ogni terza o quarta chiamata. Puoi specificare, in modo facoltativo, il numero di sequenza esatto di un record come parametro per `Checkpointer.Checkpoint`. In questo caso, la KCL presuppone che tutti i record siano stati elaborati esclusivamente fino a tale record.

Nell'esempio, il metodo privato `Checkpoint(Checkpointer checkpointer)` mostra come effettuare la chiamata al metodo `Checkpointer.Checkpoint` utilizzando la gestione delle eccezioni e la logica dei nuovi tentativi appropriate.

La KCL per .NET gestisce le eccezioni in modo diverso rispetto alle altre biblioteche di linguaggi KCL, in quanto non gestisce eventuali eccezioni generate dall'elaborazione dei record di dati. Le eccezioni non rilevate dal codice dell'utente determinano l'arresto del programma.

Arresto

La KCL chiama il metodo `Shutdown` sia al termine dell'elaborazione (il motivo dell'arresto è `TERMINATE`) che quando il worker non risponde più (il valore di `input.Reason` dell'arresto è `ZOMBIE`).

```
public void Shutdown(ShutdownInput input)
```

L'elaborazione termina quando il processore di record non riceve ulteriori record dallo shard, perché lo shard è stato frazionato o fuso o perché il flusso è stato eliminato.

La KCL trasferisce inoltre un oggetto `Checkpointer` a `shutdown`. Se il motivo dell'arresto è `TERMINATE`, il processore di record deve terminare l'elaborazione di qualsiasi record di dati e, di seguito, chiamare il metodo `checkpoint` in questa interfaccia.

Modificare le proprietà di configurazione

Il consumer di esempio fornisce valori di default per le proprietà di configurazione. È possibile sostituire una qualsiasi di queste proprietà con i tuoi valori (consulta `SampleConsumer/kcl.properties`).

Nome applicazione

La KCL richiede un nome dell'applicazione univoco per tutte le applicazioni e per tutte le tabelle Amazon DynamoDB nella stessa Regione. La biblioteca utilizza il valore di configurazione del nome dell'applicazione nei seguenti modi:

- Si suppone che tutti i lavoratori associati con questo nome dell'applicazione stiano lavorando insieme nello stesso flusso. Questi lavoratori potrebbero essere distribuiti su più istanze.
  Se si esegue un'istanza aggiuntiva dello stesso codice dell'applicazione, ma con un nome dell'applicazione diverso, la KCL tratta la seconda istanza come un'applicazione completamente separata che opera anch'essa nello stesso flusso.
- La KCL crea una tabella DynamoDB con il nome dell'applicazione e la utilizza per mantenere le informazioni sullo stato (ad esempio, checkpoint e mappatura worker-partizione) per l'applicazione. Ogni applicazione ha la propria tabella DynamoDB. Per ulteriori informazioni, consulta Utilizza una tabella di leasing per tenere traccia degli shard elaborati dall'applicazione consumer KCL.

Configurare le credenziali

È necessario rendere disponibili le AWS credenziali a uno dei provider di credenziali della catena di provider di credenziali predefinita. Puoi utilizzare la proprietà `AWSCredentialsProvider` per impostare un provider di credenziali. Le proprietà di esempio devono rendere le tue credenziali disponibili per uno dei provider di credenziali nella catena di provider di credenziali di default. Se stai eseguendo la tua applicazione consumer su un' EC2 istanza, ti consigliamo di configurare l'istanza con un ruolo IAM. AWS le credenziali che riflettono le autorizzazioni associate a questo ruolo IAM vengono rese disponibili alle applicazioni sull'istanza tramite i relativi metadati dell'istanza. Questo è il modo più sicuro per gestire le credenziali per un consumatore che esegue un'istanza. EC2

Il file di proprietà di esempio configura la KCL per elaborare un flusso di dati Kinesis denominato "words" utilizzando il processore di record fornito in `AmazonKinesisSampleConsumer.cs`.

Sviluppa un utente della Kinesis Client Library in Python

> **Important**
>
> Le versioni 1.x e 2.x di Amazon Kinesis Client Library (KCL) sono obsolete. KCL 1.x sarà disponibile il 30 gennaio 2026. end-of-support Ti consigliamo vivamente di migrare le tue applicazioni KCL utilizzando la versione 1.x all'ultima versione di KCL prima del 30 gennaio 2026. Per trovare la versione più recente di KCL, consulta la pagina Amazon Kinesis Client Library su. GitHub Per informazioni sulle ultime versioni di KCL, consulta. Usa la libreria client Kinesis Per informazioni sulla migrazione da KCL 1.x a KCL 3.x, consulta. Migrazione da KCL 1.x a KCL 3.x

È possibile utilizzare la Kinesis Client Library (KCL) per creare applicazioni che elaborano dati dai tuoi flussi di dati Kinesis. La Kinesis Client Library è disponibile in più linguaggi. In questo argomento viene discusso Python.

KCL è una libreria Java; il supporto per linguaggi diversi da Java viene fornito utilizzando un'interfaccia multilingue chiamata. MultiLangDaemon Questo daemon è basato su Java e viene eseguito in background quando si utilizza un linguaggio KCL diverso da Java. Pertanto, se installi KCL per Python e scrivi la tua app consumer interamente in Python, avrai comunque bisogno che Java sia installato sul tuo sistema a causa di. MultiLangDaemon Inoltre, MultiLangDaemon ha alcune impostazioni predefinite che potresti dover personalizzare in base al tuo caso d'uso, ad esempio la AWS regione a cui si connette. Per ulteriori informazioni su MultiLangDaemon on GitHub, vai alla pagina del MultiLangDaemon progetto KCL.

Per scaricare Python KCL da GitHub, vai alla Kinesis Client Library (Python). Per scaricare il codice di esempio per un'applicazione consumer Python KCL, vai alla pagina del progetto di esempio KCL for Python su. GitHub

È necessario completare le seguenti attività durante l'implementazione di un'applicazione consumer KCL in Python:

Attività

- Implementa i metodi della classe RecordProcessor
- Modificare le proprietà di configurazione

Implementa i metodi della classe RecordProcessor

La classe `RecordProcess` deve estendere la `RecordProcessorBase` per implementare i seguenti metodi. L'esempio fornisce implementazioni che è possibile utilizzare come punto di partenza (consulta `sample_kclpy_app.py`).

```
def initialize(self, shard_id)
def process_records(self, records, checkpointer)
def shutdown(self, checkpointer, reason)
```

initialize

La KCL chiama il metodo `initialize` quando viene creata un'istanza del processore di record, passando un ID della partizione specifico come parametro. Questo processore di record elabora esclusivamente questo shard e, in genere, è vero anche il contrario (questo shard è elaborato solo da questo processore di record). Tuttavia, il tuo consumer deve tenere conto della possibilità che un record di dati possa essere elaborato più di una volta. Ciò si verifica perché il flusso di dati Kinesis ha una semantica almeno una volta, il che significa che ogni record di dati da una partizione viene elaborato almeno una volta da un worker nel tuo consumer. Per ulteriori informazioni sui casi in cui un determinato shard può essere elaborato da più di un lavoratore, consulta Usa il resharding, lo scaling e l'elaborazione parallela per modificare il numero di shard.

```
def initialize(self, shard_id)
```

process_records

La KCL chiama questo metodo e passa un elenco di record di dati dalla partizione specificata dal metodo `initialize`. Il processore di record che implementi elabora i dati in questi record in base alla semantica del tuo consumer. Ad esempio, il worker potrebbe eseguire una trasformazione dei dati e, successivamente, archiviare il risultato in un bucket Amazon Simple Storage Service (Amazon S3).

```
def process_records(self, records, checkpointer)
```

Oltre ai dati stessi, il record contiene anche un numero di sequenza e una chiave di partizione. Il lavoratore può utilizzare questi valori quando elabora i dati. Ad esempio, il lavoratore può scegliere il bucket S3 in cui archiviare i dati in base al valore della chiave di partizione. Il dizionario `record` espone le seguenti coppie chiave-valore per accedere ai dati del record, al numero di sequenza e alla chiave di partizione:

```
record.get('data')
record.get('sequenceNumber')
record.get('partitionKey')
```

Tieni presente che i dati sono codificati in Base64.

Nell'esempio, il metodo `process_records` ha un codice che mostra in che modo un lavoratore può accedere ai dati del record, al numero di sequenza e alla chiave di partizione.

Il flusso di dati Kinesis richiede che il processore di record tenga traccia dei record che sono già stati elaborati in una partizione. La KCL si occupa di questo monitoraggio per te, passando un oggetto `Checkpointer` a `process_records`. Il processore di record chiama il metodo `checkpoint` in questo oggetto per comunicare alla KCL quanto si è progredito nell'elaborazione dei record nella partizione. In caso di errore del worker, la KCL utilizza queste informazioni per riavviare l'elaborazione della partizione nell'ultimo record elaborato conosciuto.

Per le operazioni di divisione o unione, la KCL non avvia l'elaborazione delle nuove partizioni fino a quando i processori delle partizioni originali non avranno chiamato `checkpoint` per segnalare che l'intera elaborazione delle partizioni originali è completa.

Se non viene passato un parametro, la KCL suppone che la chiamata a `checkpoint` significa che tutti i record sono stati elaborati, fino all'ultimo record passato al processore di record. Pertanto, il processore di record deve chiamare `checkpoint` solo dopo aver elaborato tutti i record nell'elenco passato al processore. I processori di record non devono chiamare `checkpoint` in ciascuna chiamata a `process_records`. Un processore potrebbe, per esempio, chiamare `checkpoint` in ogni terza chiamata. Puoi specificare, in modo facoltativo, il numero di sequenza esatto di un record come parametro per `checkpoint`. In questo caso, la KCL presuppone che tutti i record siano stati elaborati esclusivamente fino a tale record.

Nell'esempio, il metodo privato `checkpoint` mostra come effettuare la chiamata al metodo `Checkpointer.checkpoint` utilizzando la gestione delle eccezioni e la logica dei nuovi tentativi appropriate.

La KCL si basa su `process_records` per gestire eventuali eccezioni generate dall'elaborazione dei record di dati. Se viene generata un'eccezione da `process_records`, la KCL omette i record di dati passati a `process_records` prima dell'eccezione. Ciò significa che questi record non sono inviati nuovamente al processore di record che ha generato l'eccezione o a qualsiasi altro processore di record nel consumer.

shutdown

La KCL chiama il metodo `shutdown` sia al termine dell'elaborazione (il motivo dell'arresto è `TERMINATE`) che quando il worker non risponde più (il `reason` l'arresto è `ZOMBIE`).

```
def shutdown(self, checkpointer, reason)
```

L'elaborazione termina quando il processore di record non riceve ulteriori record dallo shard, perché lo shard è stato frazionato o fuso o perché il flusso è stato eliminato.

La KCL trasferisce inoltre un oggetto `Checkpointer` a `shutdown`. Se il `reason` dell'arresto è `TERMINATE`, il processore di record deve terminare l'elaborazione di qualsiasi record di dati e, di seguito, chiamare il metodo `checkpoint` in questa interfaccia.

Modificare le proprietà di configurazione

L'esempio fornisce valori di default per le proprietà di configurazione. È possibile sostituire una qualsiasi di queste proprietà con i tuoi valori (consulta `sample.properties`).

Nome applicazione

La KCL richiede un nome dell'applicazione univoco per tutte le tue applicazioni e per tutte le tabelle Amazon DynamoDB nella stessa Regione. La biblioteca utilizza il valore di configurazione del nome dell'applicazione nei seguenti modi:

- Si suppone che tutti i lavoratori associati con questo nome dell'applicazione stiano lavorando insieme nello stesso flusso. Questi lavoratori possono essere distribuiti su più istanze. Se si esegue un'istanza aggiuntiva dello stesso codice dell'applicazione, ma con un nome dell'applicazione diverso, la KCL tratta la seconda istanza come un'applicazione completamente separata che opera anch'essa nello stesso flusso.
- La KCL crea una tabella DynamoDB con il nome dell'applicazione e la utilizza per mantenere le informazioni sullo stato (ad esempio, checkpoint e mappatura worker-partizione) per l'applicazione. Ogni applicazione ha la propria tabella DynamoDB. Per ulteriori informazioni, consulta Utilizza una tabella di leasing per tenere traccia degli shard elaborati dall'applicazione consumer KCL.

Configurare le credenziali

È necessario rendere disponibili le AWS credenziali a uno dei provider di credenziali della catena di provider di credenziali predefinita. Puoi utilizzare la proprietà `AWSCredentialsProvider` per impostare un provider di credenziali. Le proprietà di esempio devono rendere le tue credenziali

disponibili per uno dei provider di credenziali nella catena di provider di credenziali di default. Se esegui la tua applicazione consumer su un' EC2 istanza Amazon, ti consigliamo di configurare l'istanza con un ruolo IAM. AWS le credenziali che riflettono le autorizzazioni associate a questo ruolo IAM vengono rese disponibili alle applicazioni sull'istanza tramite i relativi metadati dell'istanza. Questo è il modo più sicuro per gestire le credenziali per un'applicazione consumer in esecuzione su un'istanza. EC2

Il file di proprietà di esempio configura la KCL per elaborare un flusso di dati Kinesis denominato "words" utilizzando il processore di record fornito in `sample_kclpy_app.py`.

Sviluppa un utente di Kinesis Client Library in Ruby

> **Important**
>
> Le versioni 1.x e 2.x di Amazon Kinesis Client Library (KCL) sono obsolete. KCL 1.x sarà disponibile il 30 gennaio 2026. end-of-support Ti consigliamo vivamente di migrare le tue applicazioni KCL utilizzando la versione 1.x all'ultima versione di KCL prima del 30 gennaio 2026. Per trovare la versione più recente di KCL, consulta la pagina Amazon Kinesis Client Library su. GitHub Per informazioni sulle ultime versioni di KCL, consulta. Usa la libreria client Kinesis Per informazioni sulla migrazione da KCL 1.x a KCL 3.x, consulta. Migrazione da KCL 1.x a KCL 3.x

È possibile utilizzare la Kinesis Client Library (KCL) per creare applicazioni che elaborano dati dai tuoi flussi di dati Kinesis. La Kinesis Client Library è disponibile in più linguaggi. In questo argomento viene discusso Ruby.

KCL è una libreria Java; il supporto per linguaggi diversi da Java viene fornito utilizzando un'interfaccia multilingue chiamata. MultiLangDaemon Questo daemon è basato su Java e viene eseguito in background quando si utilizza un linguaggio KCL diverso da Java. Perciò, se installi KCL per Ruby e scrivi la tua app consumer interamente in Ruby, avrai comunque bisogno che Java sia installato sul tuo sistema a causa di. MultiLangDaemon Inoltre, MultiLangDaemon ha alcune impostazioni predefinite che potresti dover personalizzare in base al tuo caso d'uso, ad esempio la AWS regione a cui si connette. Per ulteriori informazioni su MultiLangDaemon on GitHub, vai alla pagina del MultiLangDaemon progetto KCL.

Per scaricare Ruby KCL da GitHub, vai alla Kinesis Client Library (Ruby). Per scaricare il codice di esempio per un'applicazione consumer Ruby KCL, vai alla pagina del progetto di esempio KCL for Ruby su. GitHub

Per ulteriori informazioni sulla biblioteca di supporto Ruby di KCL, consulta Documentazione di Ruby Gems KCL.

Sviluppa KCL 2.x Consumers

> **Important**
>
> Le versioni 1.x e 2.x di Amazon Kinesis Client Library (KCL) sono obsolete. KCL 1.x sarà disponibile il 30 gennaio 2026. end-of-support Ti consigliamo vivamente di migrare le tue applicazioni KCL utilizzando la versione 1.x all'ultima versione di KCL prima del 30 gennaio 2026. Per trovare la versione più recente di KCL, consulta la pagina Amazon Kinesis Client Library su. GitHub Per informazioni sulle ultime versioni di KCL, consulta. Usa la libreria client Kinesis Per informazioni sulla migrazione da KCL 1.x a KCL 3.x, consulta. Migrazione da KCL 1.x a KCL 3.x

Questo argomento illustra come utilizzare la versione 2.0 di Kinesis Client Library (KCL).

Per ulteriori informazioni sulla KCL, consulta la panoramica in Sviluppo di app consumer utilizzando Kinesis Client Library 1.x.

Scegli tra i seguenti argomenti a seconda dell'opzione che desideri utilizzare.

Argomenti

- Sviluppa un utente di Kinesis Client Library in Java
- Sviluppa un utente della Kinesis Client Library in Python
- Sviluppa utenti avanzati con fan-out con KCL 2.x

Sviluppa un utente di Kinesis Client Library in Java

> **Important**
>
> Le versioni 1.x e 2.x di Amazon Kinesis Client Library (KCL) sono obsolete. KCL 1.x sarà disponibile il 30 gennaio 2026. end-of-support Ti consigliamo vivamente di migrare le tue applicazioni KCL utilizzando la versione 1.x all'ultima versione di KCL prima del 30 gennaio 2026. Per trovare la versione più recente di KCL, consulta la pagina Amazon Kinesis Client Library su. GitHub Per informazioni sulle ultime versioni di KCL, consulta. Usa la libreria client

Kinesis Per informazioni sulla migrazione da KCL 1.x a KCL 3.x, consulta. Migrazione da KCL 1.x a KCL 3.x

Il seguente codice mostra un esempio di implementazione in Java di `ProcessorFactory` e `RecordProcessor`. Se vuoi sfruttare la funzionalità fan-out avanzato, consulta Utilizzo di app Consumer con il fan-out avanzato.

```
/*
 *  Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 *  Licensed under the Amazon Software License (the "License").
 *  You may not use this file except in compliance with the License.
 *  A copy of the License is located at
 *
 *  http://aws.amazon.com/asl/
 *
 *  or in the "license" file accompanying this file. This file is distributed
 *  on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
 *  express or implied. See the License for the specific language governing
 *  permissions and limitations under the License.
 */


/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 * Licensed under the Apache License, Version 2.0 (the "License").
 * You may not use this file except in compliance with the License.
 * A copy of the License is located at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * or in the "license" file accompanying this file. This file is distributed
 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
 * express or implied. See the License for the specific language governing
 * permissions and limitations under the License.
 */

import java.io.BufferedReader;
import java.io.IOException;
```

```
import java.io.InputStreamReader;
import java.util.UUID;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.ScheduledFuture;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

import org.apache.commons.lang3.ObjectUtils;
import org.apache.commons.lang3.RandomStringUtils;
import org.apache.commons.lang3.RandomUtils;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.slf4j.MDC;

import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cloudwatch.CloudWatchAsyncClient;
import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;
import software.amazon.awssdk.services.kinesis.KinesisAsyncClient;
import software.amazon.awssdk.services.kinesis.model.PutRecordRequest;
import software.amazon.kinesis.common.ConfigsBuilder;
import software.amazon.kinesis.common.KinesisClientUtil;
import software.amazon.kinesis.coordinator.Scheduler;
import software.amazon.kinesis.exceptions.InvalidStateException;
import software.amazon.kinesis.exceptions.ShutdownException;
import software.amazon.kinesis.lifecycle.events.InitializationInput;
import software.amazon.kinesis.lifecycle.events.LeaseLostInput;
import software.amazon.kinesis.lifecycle.events.ProcessRecordsInput;
import software.amazon.kinesis.lifecycle.events.ShardEndedInput;
import software.amazon.kinesis.lifecycle.events.ShutdownRequestedInput;

import software.amazon.kinesis.processor.ShardRecordProcessor;
import software.amazon.kinesis.processor.ShardRecordProcessorFactory;
import software.amazon.kinesis.retrieval.polling.PollingConfig;

/**
 * This class will run a simple app that uses the KCL to read data and uses the AWS SDK
 to publish data.
 * Before running this program you must first create a Kinesis stream through the AWS
 console or AWS SDK.
 */
```

```
public class SampleSingle {

    private static final Logger log = LoggerFactory.getLogger(SampleSingle.class);

    /**
     * Invoke the main method with 2 args: the stream name and (optionally) the region.
     * Verifies valid inputs and then starts running the app.
     */
    public static void main(String... args) {
        if (args.length < 1) {
            log.error("At a minimum, the stream name is required as the first argument.
 The Region may be specified as the second argument.");
            System.exit(1);
        }

        String streamName = args[0];
        String region = null;
        if (args.length > 1) {
            region = args[1];
        }

        new SampleSingle(streamName, region).run();
    }

    private final String streamName;
    private final Region region;
    private final KinesisAsyncClient kinesisClient;

    /**
     * Constructor sets streamName and region. It also creates a KinesisClient object
 to send data to Kinesis.
     * This KinesisClient is used to send dummy data so that the consumer has something
 to read; it is also used
     * indirectly by the KCL to handle the consumption of the data.
     */
    private SampleSingle(String streamName, String region) {
        this.streamName = streamName;
        this.region = Region.of(ObjectUtils.firstNonNull(region, "us-east-2"));
        this.kinesisClient =
 KinesisClientUtil.createKinesisAsyncClient(KinesisAsyncClient.builder().region(this.region));
    }

    private void run() {
```

```
        /**
         * Sends dummy data to Kinesis. Not relevant to consuming the data with the KCL
         */
        ScheduledExecutorService producerExecutor =
Executors.newSingleThreadScheduledExecutor();
        ScheduledFuture<?> producerFuture =
producerExecutor.scheduleAtFixedRate(this::publishRecord, 10, 1, TimeUnit.SECONDS);

        /**
         * Sets up configuration for the KCL, including DynamoDB and CloudWatch
dependencies. The final argument, a
         * ShardRecordProcessorFactory, is where the logic for record processing lives,
and is located in a private
         * class below.
         */
        DynamoDbAsyncClient dynamoClient =
DynamoDbAsyncClient.builder().region(region).build();
        CloudWatchAsyncClient cloudWatchClient =
CloudWatchAsyncClient.builder().region(region).build();
        ConfigsBuilder configsBuilder = new ConfigsBuilder(streamName, streamName,
kinesisClient, dynamoClient, cloudWatchClient, UUID.randomUUID().toString(), new
SampleRecordProcessorFactory());

        /**
         * The Scheduler (also called Worker in earlier versions of the KCL) is the
entry point to the KCL. This
         * instance is configured with defaults provided by the ConfigsBuilder.
         */
        Scheduler scheduler = new Scheduler(
                configsBuilder.checkpointConfig(),
                configsBuilder.coordinatorConfig(),
                configsBuilder.leaseManagementConfig(),
                configsBuilder.lifecycleConfig(),
                configsBuilder.metricsConfig(),
                configsBuilder.processorConfig(),
                configsBuilder.retrievalConfig().retrievalSpecificConfig(new
PollingConfig(streamName, kinesisClient))
        );

        /**
         * Kickoff the Scheduler. Record processing of the stream of dummy data will
continue indefinitely
         * until an exit is triggered.
         */
```

```
        Thread schedulerThread = new Thread(scheduler);
        schedulerThread.setDaemon(true);
        schedulerThread.start();

        /**
         * Allows termination of app by pressing Enter.
         */
        System.out.println("Press enter to shutdown");
        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
        try {
            reader.readLine();
        } catch (IOException ioex) {
            log.error("Caught exception while waiting for confirm. Shutting down.",
ioex);
        }

        /**
         * Stops sending dummy data.
         */
        log.info("Cancelling producer and shutting down executor.");
        producerFuture.cancel(true);
        producerExecutor.shutdownNow();

        /**
         * Stops consuming data. Finishes processing the current batch of data already
received from Kinesis
         * before shutting down.
         */
        Future<Boolean> gracefulShutdownFuture = scheduler.startGracefulShutdown();
        log.info("Waiting up to 20 seconds for shutdown to complete.");
        try {
            gracefulShutdownFuture.get(20, TimeUnit.SECONDS);
        } catch (InterruptedException e) {
            log.info("Interrupted while waiting for graceful shutdown. Continuing.");
        } catch (ExecutionException e) {
            log.error("Exception while executing graceful shutdown.", e);
        } catch (TimeoutException e) {
            log.error("Timeout while waiting for shutdown.  Scheduler may not have
exited.");
        }
        log.info("Completed, shutting down now.");
    }

    /**
```

```
     * Sends a single record of dummy data to Kinesis.
     */
    private void publishRecord() {
        PutRecordRequest request = PutRecordRequest.builder()
                .partitionKey(RandomStringUtils.randomAlphabetic(5, 20))
                .streamName(streamName)
                .data(SdkBytes.fromByteArray(RandomUtils.nextBytes(10)))
                .build();
        try {
            kinesisClient.putRecord(request).get();
        } catch (InterruptedException e) {
            log.info("Interrupted, assuming shutdown.");
        } catch (ExecutionException e) {
            log.error("Exception while sending data to Kinesis. Will try again next
cycle.", e);
        }
    }

    private static class SampleRecordProcessorFactory implements
ShardRecordProcessorFactory {
        public ShardRecordProcessor shardRecordProcessor() {
            return new SampleRecordProcessor();
        }
    }

    /**
     * The implementation of the ShardRecordProcessor interface is where the heart of
the record processing logic lives.
     * In this example all we do to 'process' is log info about the records.
     */
    private static class SampleRecordProcessor implements ShardRecordProcessor {

        private static final String SHARD_ID_MDC_KEY = "ShardId";

        private static final Logger log =
LoggerFactory.getLogger(SampleRecordProcessor.class);

        private String shardId;

        /**
         * Invoked by the KCL before data records are delivered to the
ShardRecordProcessor instance (via
         * processRecords). In this example we do nothing except some logging.
         *
```

```
         * @param initializationInput Provides information related to initialization.
         */
        public void initialize(InitializationInput initializationInput) {
            shardId = initializationInput.shardId();
            MDC.put(SHARD_ID_MDC_KEY, shardId);
            try {
                log.info("Initializing @ Sequence: {}",
initializationInput.extendedSequenceNumber());
            } finally {
                MDC.remove(SHARD_ID_MDC_KEY);
            }
        }

        /**
         * Handles record processing logic. The Amazon Kinesis Client Library will
invoke this method to deliver
         * data records to the application. In this example we simply log our records.
         *
         * @param processRecordsInput Provides the records to be processed as well as
information and capabilities
         *                            related to them (e.g. checkpointing).
         */
        public void processRecords(ProcessRecordsInput processRecordsInput) {
            MDC.put(SHARD_ID_MDC_KEY, shardId);
            try {
                log.info("Processing {} record(s)",
processRecordsInput.records().size());
                processRecordsInput.records().forEach(r -> log.info("Processing record
pk: {} -- Seq: {}", r.partitionKey(), r.sequenceNumber()));
            } catch (Throwable t) {
                log.error("Caught throwable while processing records. Aborting.");
                Runtime.getRuntime().halt(1);
            } finally {
                MDC.remove(SHARD_ID_MDC_KEY);
            }
        }

        /** Called when the lease tied to this record processor has been lost. Once the
lease has been lost,
         * the record processor can no longer checkpoint.
         *
         * @param leaseLostInput Provides access to functions and data related to the
loss of the lease.
         */
```

```
        public void leaseLost(LeaseLostInput leaseLostInput) {
            MDC.put(SHARD_ID_MDC_KEY, shardId);
            try {
                log.info("Lost lease, so terminating.");
            } finally {
                MDC.remove(SHARD_ID_MDC_KEY);
            }
        }

        /**
         * Called when all data on this shard has been processed. Checkpointing must
occur in the method for record
         * processing to be considered complete; an exception will be thrown otherwise.
         *
         * @param shardEndedInput Provides access to a checkpointer method for
completing processing of the shard.
         */
        public void shardEnded(ShardEndedInput shardEndedInput) {
            MDC.put(SHARD_ID_MDC_KEY, shardId);
            try {
                log.info("Reached shard end checkpointing.");
                shardEndedInput.checkpointer().checkpoint();
            } catch (ShutdownException | InvalidStateException e) {
                log.error("Exception while checkpointing at shard end. Giving up.", e);
            } finally {
                MDC.remove(SHARD_ID_MDC_KEY);
            }
        }

        /**
         * Invoked when Scheduler has been requested to shut down (i.e. we decide to
stop running the app by pressing
         * Enter). Checkpoints and logs the data a final time.
         *
         * @param shutdownRequestedInput Provides access to a checkpointer, allowing a
record processor to checkpoint
         *                               before the shutdown is completed.
         */
        public void shutdownRequested(ShutdownRequestedInput shutdownRequestedInput) {
            MDC.put(SHARD_ID_MDC_KEY, shardId);
            try {
                log.info("Scheduler is shutting down, checkpointing.");
                shutdownRequestedInput.checkpointer().checkpoint();
            } catch (ShutdownException | InvalidStateException e) {
```

```
                log.error("Exception while checkpointing at requested shutdown. Giving
 up.", e);
            } finally {
                MDC.remove(SHARD_ID_MDC_KEY);
            }
        }
    }

}
```

Sviluppa un utente della Kinesis Client Library in Python

> **Important**
>
> Le versioni 1.x e 2.x di Amazon Kinesis Client Library (KCL) sono obsolete. KCL 1.x sarà disponibile il 30 gennaio 2026. end-of-support Ti consigliamo vivamente di migrare le tue applicazioni KCL utilizzando la versione 1.x all'ultima versione di KCL prima del 30 gennaio 2026. Per trovare la versione più recente di KCL, consulta la pagina Amazon Kinesis Client Library su. GitHub Per informazioni sulle ultime versioni di KCL, consulta. Usa la libreria client Kinesis Per informazioni sulla migrazione da KCL 1.x a KCL 3.x, consulta. Migrazione da KCL 1.x a KCL 3.x

È possibile utilizzare la Kinesis Client Library (KCL) per creare applicazioni che elaborano dati dai tuoi flussi di dati Kinesis. La Kinesis Client Library è disponibile in più linguaggi. In questo argomento viene discusso Python.

KCL è una libreria Java; il supporto per linguaggi diversi da Java viene fornito utilizzando un'interfaccia multilingue chiamata. MultiLangDaemon Questo daemon è basato su Java e viene eseguito in background quando si utilizza un linguaggio KCL diverso da Java. Pertanto, se installi KCL per Python e scrivi la tua app consumer interamente in Python, avrai comunque bisogno che Java sia installato sul tuo sistema a causa di. MultiLangDaemon Inoltre, MultiLangDaemon ha alcune impostazioni predefinite che potresti dover personalizzare in base al tuo caso d'uso, ad esempio la AWS regione a cui si connette. Per ulteriori informazioni su MultiLangDaemon on GitHub, vai alla pagina del MultiLangDaemon progetto KCL.

Per scaricare Python KCL da GitHub, vai alla Kinesis Client Library (Python). Per scaricare il codice di esempio per un'applicazione consumer Python KCL, vai alla pagina del progetto di esempio KCL for Python su. GitHub

È necessario completare le seguenti attività durante l'implementazione di un'applicazione consumer KCL in Python:

Attività

- Implementa i metodi della classe RecordProcessor
- Modificare le proprietà di configurazione

Implementa i metodi della classe RecordProcessor

La classe `RecordProcess` deve estendere la classe `RecordProcessorBase` per implementare i seguenti metodi:

```
initialize
process_records
shutdown_requested
```

L'esempio fornisce implementazioni che è possibile utilizzare come punto di partenza.

```
#!/usr/bin/env python

# Copyright 2014-2015 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#
# Licensed under the Amazon Software License (the "License").
# You may not use this file except in compliance with the License.
# A copy of the License is located at
#
# http://aws.amazon.com/asl/
#
# or in the "license" file accompanying this file. This file is distributed
# on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
# express or implied. See the License for the specific language governing
# permissions and limitations under the License.

from __future__ import print_function

import sys
import time

from amazon_kclpy import kcl
from amazon_kclpy.v3 import processor
```

```
class RecordProcessor(processor.RecordProcessorBase):
    """
    A RecordProcessor processes data from a shard in a stream. Its methods will be
 called with this pattern:

    * initialize will be called once
    * process_records will be called zero or more times
    * shutdown will be called if this MultiLangDaemon instance loses the lease to this
 shard, or the shard ends due
        a scaling change.
    """
    def __init__(self):
        self._SLEEP_SECONDS = 5
        self._CHECKPOINT_RETRIES = 5
        self._CHECKPOINT_FREQ_SECONDS = 60
        self._largest_seq = (None, None)
        self._largest_sub_seq = None
        self._last_checkpoint_time = None

    def log(self, message):
        sys.stderr.write(message)

    def initialize(self, initialize_input):
        """
        Called once by a KCLProcess before any calls to process_records

        :param amazon_kclpy.messages.InitializeInput initialize_input: Information
 about the lease that this record
            processor has been assigned.
        """
        self._largest_seq = (None, None)
        self._last_checkpoint_time = time.time()

    def checkpoint(self, checkpointer, sequence_number=None, sub_sequence_number=None):
        """
        Checkpoints with retries on retryable exceptions.

        :param amazon_kclpy.kcl.Checkpointer checkpointer: the checkpointer provided to
 either process_records
            or shutdown
        :param str or None sequence_number: the sequence number to checkpoint at.
        :param int or None sub_sequence_number: the sub sequence number to checkpoint
 at.
```

```
        """
        for n in range(0, self._CHECKPOINT_RETRIES):
            try:
                checkpointer.checkpoint(sequence_number, sub_sequence_number)
                return
            except kcl.CheckpointError as e:
                if 'ShutdownException' == e.value:
                    #
                    # A ShutdownException indicates that this record processor should
be shutdown. This is due to
                    # some failover event, e.g. another MultiLangDaemon has taken the
lease for this shard.
                    #
                    print('Encountered shutdown exception, skipping checkpoint')
                    return
                elif 'ThrottlingException' == e.value:
                    #
                    # A ThrottlingException indicates that one of our dependencies is
is over burdened, e.g. too many
                    # dynamo writes. We will sleep temporarily to let it recover.
                    #
                    if self._CHECKPOINT_RETRIES - 1 == n:
                        sys.stderr.write('Failed to checkpoint after {n} attempts,
giving up.\n'.format(n=n))
                        return
                    else:
                        print('Was throttled while checkpointing, will attempt again in
{s} seconds'
                              .format(s=self._SLEEP_SECONDS))
                elif 'InvalidStateException' == e.value:
                    sys.stderr.write('MultiLangDaemon reported an invalid state while
checkpointing.\n')
                else:  # Some other error
                    sys.stderr.write('Encountered an error while checkpointing, error
was {e}.\n'.format(e=e))
            time.sleep(self._SLEEP_SECONDS)

    def process_record(self, data, partition_key, sequence_number,
sub_sequence_number):
        """
        Called for each record that is passed to process_records.

        :param str data: The blob of data that was contained in the record.
        :param str partition_key: The key associated with this recod.
```

```
        :param int sequence_number: The sequence number associated with this record.
        :param int sub_sequence_number: the sub sequence number associated with this
record.
        """
        ####################################
        # Insert your processing logic here
        ####################################
        self.log("Record (Partition Key: {pk}, Sequence Number: {seq}, Subsequence
Number: {sseq}, Data Size: {ds}"
                 .format(pk=partition_key, seq=sequence_number,
sseq=sub_sequence_number, ds=len(data)))

    def should_update_sequence(self, sequence_number, sub_sequence_number):
        """
        Determines whether a new larger sequence number is available

        :param int sequence_number: the sequence number from the current record
        :param int sub_sequence_number: the sub sequence number from the current record
        :return boolean: true if the largest sequence should be updated, false
otherwise
        """
        return self._largest_seq == (None, None) or sequence_number >
self._largest_seq[0] or \
            (sequence_number == self._largest_seq[0] and sub_sequence_number >
self._largest_seq[1])

    def process_records(self, process_records_input):
        """
        Called by a KCLProcess with a list of records to be processed and a
checkpointer which accepts sequence numbers
        from the records to indicate where in the stream to checkpoint.

        :param amazon_kclpy.messages.ProcessRecordsInput process_records_input: the
records, and metadata about the
            records.
        """
        try:
            for record in process_records_input.records:
                data = record.binary_data
                seq = int(record.sequence_number)
                sub_seq = record.sub_sequence_number
                key = record.partition_key
                self.process_record(data, key, seq, sub_seq)
                if self.should_update_sequence(seq, sub_seq):
```

```
                    self._largest_seq = (seq, sub_seq)

            #
            # Checkpoints every self._CHECKPOINT_FREQ_SECONDS seconds
            #
            if time.time() - self._last_checkpoint_time >
 self._CHECKPOINT_FREQ_SECONDS:
                self.checkpoint(process_records_input.checkpointer,
 str(self._largest_seq[0]), self._largest_seq[1])
                self._last_checkpoint_time = time.time()

        except Exception as e:
            self.log("Encountered an exception while processing records. Exception was
 {e}\n".format(e=e))

    def lease_lost(self, lease_lost_input):
        self.log("Lease has been lost")

    def shard_ended(self, shard_ended_input):
        self.log("Shard has ended checkpointing")
        shard_ended_input.checkpointer.checkpoint()

    def shutdown_requested(self, shutdown_requested_input):
        self.log("Shutdown has been requested, checkpointing.")
        shutdown_requested_input.checkpointer.checkpoint()


if __name__ == "__main__":
    kcl_process = kcl.KCLProcess(RecordProcessor())
    kcl_process.run()
```

Modificare le proprietà di configurazione

L'esempio fornisce valori di default per le proprietà di configurazione, come mostra lo script seguente. È possibile sostituire una qualsiasi di queste proprietà con i propri valori.

```
# The script that abides by the multi-language protocol. This script will
# be executed by the MultiLangDaemon, which will communicate with this script
# over STDIN and STDOUT according to the multi-language protocol.
executableName = sample_kclpy_app.py

# The name of an Amazon Kinesis stream to process.
streamName = words
```

```
# Used by the KCL as the name of this application. Will be used as the name
# of an Amazon DynamoDB table which will store the lease and checkpoint
# information for workers with this application name
applicationName = PythonKCLSample

# Users can change the credentials provider the KCL will use to retrieve credentials.
# The DefaultAWSCredentialsProviderChain checks several other providers, which is
# described here:
# http://docs.aws.amazon.com/AWSJavaSDK/latest/javadoc/com/amazonaws/auth/
DefaultAWSCredentialsProviderChain.html
AWSCredentialsProvider = DefaultAWSCredentialsProviderChain

# Appended to the user agent of the KCL. Does not impact the functionality of the
# KCL in any other way.
processingLanguage = python/2.7

# Valid options at TRIM_HORIZON or LATEST.
# See http://docs.aws.amazon.com/kinesis/latest/APIReference/
API_GetShardIterator.html#API_GetShardIterator_RequestSyntax
initialPositionInStream = TRIM_HORIZON

# The following properties are also available for configuring the KCL Worker that is
 created
# by the MultiLangDaemon.

# The KCL defaults to us-east-1
#regionName = us-east-1

# Fail over time in milliseconds. A worker which does not renew it's lease within this
 time interval
# will be regarded as having problems and it's shards will be assigned to other
 workers.
# For applications that have a large number of shards, this msy be set to a higher
 number to reduce
# the number of DynamoDB IOPS required for tracking leases
#failoverTimeMillis = 10000

# A worker id that uniquely identifies this worker among all workers using the same
 applicationName
# If this isn't provided a MultiLangDaemon instance will assign a unique workerId to
 itself.
#workerId =
```

```
# Shard sync interval in milliseconds - e.g. wait for this long between shard sync
 tasks.
#shardSyncIntervalMillis = 60000

# Max records to fetch from Kinesis in a single GetRecords call.
#maxRecords = 10000

# Idle time between record reads in milliseconds.
#idleTimeBetweenReadsInMillis = 1000

# Enables applications flush/checkpoint (if they have some data "in progress", but
 don't get new data for while)
#callProcessRecordsEvenForEmptyRecordList = false

# Interval in milliseconds between polling to check for parent shard completion.
# Polling frequently will take up more DynamoDB IOPS (when there are leases for shards
 waiting on
# completion of parent shards).
#parentShardPollIntervalMillis = 10000

# Cleanup leases upon shards completion (don't wait until they expire in Kinesis).
# Keeping leases takes some tracking/resources (e.g. they need to be renewed,
 assigned), so by default we try
# to delete the ones we don't need any longer.
#cleanupLeasesUponShardCompletion = true

# Backoff time in milliseconds for Amazon Kinesis Client Library tasks (in the event of
 failures).
#taskBackoffTimeMillis = 500

# Buffer metrics for at most this long before publishing to CloudWatch.
#metricsBufferTimeMillis = 10000

# Buffer at most this many metrics before publishing to CloudWatch.
#metricsMaxQueueSize = 10000

# KCL will validate client provided sequence numbers with a call to Amazon Kinesis
 before checkpointing for calls
# to RecordProcessorCheckpointer#checkpoint(String) by default.
#validateSequenceNumberBeforeCheckpointing = true

# The maximum number of active threads for the MultiLangDaemon to permit.
# If a value is provided then a FixedThreadPool is used with the maximum
# active threads set to the provided value. If a non-positive integer or no
```

```
# value is provided a CachedThreadPool is used.
#maxActiveThreads = 0
```

Nome applicazione

La KCL richiede un nome dell'applicazione univoco per tutte le applicazioni e per tutte le tabelle Amazon DynamoDB nella stessa Regione. La biblioteca utilizza il valore di configurazione del nome dell'applicazione nei seguenti modi:

- Si suppone che tutti i lavoratori associati con questo nome dell'applicazione stiano lavorando insieme nello stesso flusso. Questi lavoratori possono essere distribuiti su più istanze. Se si esegue un'istanza aggiuntiva dello stesso codice dell'applicazione, ma con un nome dell'applicazione diverso, la KCL tratta la seconda istanza come un'applicazione completamente separata che opera anch'essa nello stesso flusso.
- La KCL crea una tabella DynamoDB con il nome dell'applicazione e la utilizza per mantenere le informazioni sullo stato (ad esempio, checkpoint e mappatura worker-partizione) per l'applicazione. Ogni applicazione ha la propria tabella DynamoDB. Per ulteriori informazioni, consulta Utilizza una tabella di leasing per tenere traccia degli shard elaborati dall'applicazione consumer KCL.

Credenziali

È necessario rendere disponibili le AWS credenziali a uno dei provider di credenziali della catena di provider di credenziali predefinita. Puoi utilizzare la proprietà `AWSCredentialsProvider` per impostare un provider di credenziali. Se esegui la tua applicazione consumer su un' EC2 istanza Amazon, ti consigliamo di configurare l'istanza con un ruolo IAM. AWS le credenziali che riflettono le autorizzazioni associate a questo ruolo IAM vengono rese disponibili alle applicazioni sull'istanza tramite i relativi metadati dell'istanza. Questo è il modo più sicuro per gestire le credenziali per un'applicazione consumer in esecuzione su un'istanza. EC2

Sviluppa utenti avanzati con fan-out con KCL 2.x

> **Important**
>
> Le versioni 1.x e 2.x di Amazon Kinesis Client Library (KCL) sono obsolete. KCL 1.x sarà disponibile il 30 gennaio 2026. end-of-support Ti consigliamo vivamente di migrare le tue applicazioni KCL utilizzando la versione 1.x all'ultima versione di KCL prima del 30 gennaio 2026. Per trovare la versione più recente di KCL, consulta la pagina Amazon Kinesis Client Library su. GitHub Per informazioni sulle ultime versioni di KCL, consulta. Usa la libreria client

Kinesis Per informazioni sulla migrazione da KCL 1.x a KCL 3.x, consulta. Migrazione da KCL 1.x a KCL 3.x

Le applicazioni consumer che utilizzano il fan-out avanzato in Flusso di dati Amazon Kinesis possono ricevere record da un flusso di dati con velocità di trasmissione effettiva dedicata fino a 2 MiB di dati al secondo per partizione. Questo tipo di applicazioni consumer non sono in competizione con altre applicazioni che ricevono dati dal flusso. Per ulteriori informazioni, consulta Sviluppa consumatori con fan-out migliorati con un throughput dedicato.

È possibile utilizzare la versione 2.0 o una successiva della Kinesis Client Library (KCL) per sviluppare applicazioni che usano il fan-out avanzato per ricevere dati dai flussi. La KCL sottoscrive automaticamente l'applicazione a tutte le partizioni di un flusso e assicura che l'applicazione consumer sia in grado di leggere con un valore di velocità di trasmissione effettiva di 2 MiB/sec per partizione. Se desideri utilizzare la KCL senza attivare la funzionalità fan-out avanzato, consulta Sviluppo di applicazioni Consumer tramite la Kinesis Client Library 2.0.

Argomenti

- Sviluppa utenti fan-out avanzati utilizzando KCL 2.x in Java

Sviluppa utenti fan-out avanzati utilizzando KCL 2.x in Java

⚠ Important

Le versioni 1.x e 2.x di Amazon Kinesis Client Library (KCL) sono obsolete. KCL 1.x sarà disponibile il 30 gennaio 2026. end-of-support Ti consigliamo vivamente di migrare le tue applicazioni KCL utilizzando la versione 1.x all'ultima versione di KCL prima del 30 gennaio 2026. Per trovare la versione più recente di KCL, consulta la pagina Amazon Kinesis Client Library su. GitHub Per informazioni sulle ultime versioni di KCL, consulta. Usa la libreria client Kinesis Per informazioni sulla migrazione da KCL 1.x a KCL 3.x, consulta. Migrazione da KCL 1.x a KCL 3.x

È possibile utilizzare la versione 2.0 o una successiva della Kinesis Client Library (KCL) per sviluppare applicazioni in Flusso di dati Amazon Kinesis per ricevere dati dai flussi tramite il fan-out avanzato. Il seguente codice mostra un esempio di implementazione in Java di `ProcessorFactory` e `RecordProcessor`.

Si consiglia di utilizzare `KinesisClientUtil` per creare `KinesisAsyncClient` e configurare `maxConcurrency` in `KinesisAsyncClient`.

> **Important**
>
> Il client Amazon Kinesis potrebbe presentare un aumento significativo della latenza, a meno che non venga configurato `KinesisAsyncClient` per avere una `maxConcurrency` sufficientemente alta per consentire tutti i canoni più ulteriori utilizzi di `KinesisAsyncClient`.

```
/*
 *  Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 *  Licensed under the Amazon Software License (the "License").
 *  You may not use this file except in compliance with the License.
 *  A copy of the License is located at
 *
 *  http://aws.amazon.com/asl/
 *
 *  or in the "license" file accompanying this file. This file is distributed
 *  on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
 *  express or implied. See the License for the specific language governing
 *  permissions and limitations under the License.
 */

/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 * Licensed under the Apache License, Version 2.0 (the "License").
 * You may not use this file except in compliance with the License.
 * A copy of the License is located at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * or in the "license" file accompanying this file. This file is distributed
 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
 * express or implied. See the License for the specific language governing
 * permissions and limitations under the License.
 */

import java.io.BufferedReader;
```

```
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.UUID;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.ScheduledFuture;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

import org.apache.commons.lang3.ObjectUtils;
import org.apache.commons.lang3.RandomStringUtils;
import org.apache.commons.lang3.RandomUtils;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.slf4j.MDC;

import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cloudwatch.CloudWatchAsyncClient;
import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;
import software.amazon.awssdk.services.kinesis.KinesisAsyncClient;
import software.amazon.awssdk.services.kinesis.model.PutRecordRequest;
import software.amazon.kinesis.common.ConfigsBuilder;
import software.amazon.kinesis.common.KinesisClientUtil;
import software.amazon.kinesis.coordinator.Scheduler;
import software.amazon.kinesis.exceptions.InvalidStateException;
import software.amazon.kinesis.exceptions.ShutdownException;
import software.amazon.kinesis.lifecycle.events.InitializationInput;
import software.amazon.kinesis.lifecycle.events.LeaseLostInput;
import software.amazon.kinesis.lifecycle.events.ProcessRecordsInput;
import software.amazon.kinesis.lifecycle.events.ShardEndedInput;
import software.amazon.kinesis.lifecycle.events.ShutdownRequestedInput;
import software.amazon.kinesis.processor.ShardRecordProcessor;
import software.amazon.kinesis.processor.ShardRecordProcessorFactory;

public class SampleSingle {

    private static final Logger log = LoggerFactory.getLogger(SampleSingle.class);

    public static void main(String... args) {
        if (args.length < 1) {
```

```
            log.error("At a minimum, the stream name is required as the first argument.
The Region may be specified as the second argument.");
            System.exit(1);
        }

        String streamName = args[0];
        String region = null;
        if (args.length > 1) {
            region = args[1];
        }

        new SampleSingle(streamName, region).run();
    }

    private final String streamName;
    private final Region region;
    private final KinesisAsyncClient kinesisClient;

    private SampleSingle(String streamName, String region) {
        this.streamName = streamName;
        this.region = Region.of(ObjectUtils.firstNonNull(region, "us-east-2"));
        this.kinesisClient =
KinesisClientUtil.createKinesisAsyncClient(KinesisAsyncClient.builder().region(this.region));
    }

    private void run() {
        ScheduledExecutorService producerExecutor =
Executors.newSingleThreadScheduledExecutor();
        ScheduledFuture<?> producerFuture =
producerExecutor.scheduleAtFixedRate(this::publishRecord, 10, 1, TimeUnit.SECONDS);

        DynamoDbAsyncClient dynamoClient =
DynamoDbAsyncClient.builder().region(region).build();
        CloudWatchAsyncClient cloudWatchClient =
CloudWatchAsyncClient.builder().region(region).build();
        ConfigsBuilder configsBuilder = new ConfigsBuilder(streamName, streamName,
kinesisClient, dynamoClient, cloudWatchClient, UUID.randomUUID().toString(), new
SampleRecordProcessorFactory());

        Scheduler scheduler = new Scheduler(
                configsBuilder.checkpointConfig(),
                configsBuilder.coordinatorConfig(),
                configsBuilder.leaseManagementConfig(),
                configsBuilder.lifecycleConfig(),
```

```
                configsBuilder.metricsConfig(),
                configsBuilder.processorConfig(),
                configsBuilder.retrievalConfig()
        );

        Thread schedulerThread = new Thread(scheduler);
        schedulerThread.setDaemon(true);
        schedulerThread.start();

        System.out.println("Press enter to shutdown");
        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
        try {
            reader.readLine();
        } catch (IOException ioex) {
            log.error("Caught exception while waiting for confirm. Shutting down.",
ioex);
        }

        log.info("Cancelling producer, and shutting down executor.");
        producerFuture.cancel(true);
        producerExecutor.shutdownNow();

        Future<Boolean> gracefulShutdownFuture = scheduler.startGracefulShutdown();
        log.info("Waiting up to 20 seconds for shutdown to complete.");
        try {
            gracefulShutdownFuture.get(20, TimeUnit.SECONDS);
        } catch (InterruptedException e) {
            log.info("Interrupted while waiting for graceful shutdown. Continuing.");
        } catch (ExecutionException e) {
            log.error("Exception while executing graceful shutdown.", e);
        } catch (TimeoutException e) {
            log.error("Timeout while waiting for shutdown. Scheduler may not have
exited.");
        }
        log.info("Completed, shutting down now.");
    }

    private void publishRecord() {
        PutRecordRequest request = PutRecordRequest.builder()
                .partitionKey(RandomStringUtils.randomAlphabetic(5, 20))
                .streamName(streamName)
                .data(SdkBytes.fromByteArray(RandomUtils.nextBytes(10)))
                .build();
        try {
```

```
            kinesisClient.putRecord(request).get();
        } catch (InterruptedException e) {
            log.info("Interrupted, assuming shutdown.");
        } catch (ExecutionException e) {
            log.error("Exception while sending data to Kinesis. Will try again next
cycle.", e);
        }
    }

    private static class SampleRecordProcessorFactory implements
ShardRecordProcessorFactory {
        public ShardRecordProcessor shardRecordProcessor() {
            return new SampleRecordProcessor();
        }
    }


    private static class SampleRecordProcessor implements ShardRecordProcessor {

        private static final String SHARD_ID_MDC_KEY = "ShardId";

        private static final Logger log =
LoggerFactory.getLogger(SampleRecordProcessor.class);

        private String shardId;

        public void initialize(InitializationInput initializationInput) {
            shardId = initializationInput.shardId();
            MDC.put(SHARD_ID_MDC_KEY, shardId);
            try {
                log.info("Initializing @ Sequence: {}",
initializationInput.extendedSequenceNumber());
            } finally {
                MDC.remove(SHARD_ID_MDC_KEY);
            }
        }

        public void processRecords(ProcessRecordsInput processRecordsInput) {
            MDC.put(SHARD_ID_MDC_KEY, shardId);
            try {
                log.info("Processing {} record(s)",
processRecordsInput.records().size());
                processRecordsInput.records().forEach(r -> log.info("Processing record
pk: {} -- Seq: {}", r.partitionKey(), r.sequenceNumber()));
```

```
            } catch (Throwable t) {
                log.error("Caught throwable while processing records. Aborting.");
                Runtime.getRuntime().halt(1);
            } finally {
                MDC.remove(SHARD_ID_MDC_KEY);
            }
        }

        public void leaseLost(LeaseLostInput leaseLostInput) {
            MDC.put(SHARD_ID_MDC_KEY, shardId);
            try {
                log.info("Lost lease, so terminating.");
            } finally {
                MDC.remove(SHARD_ID_MDC_KEY);
            }
        }

        public void shardEnded(ShardEndedInput shardEndedInput) {
            MDC.put(SHARD_ID_MDC_KEY, shardId);
            try {
                log.info("Reached shard end checkpointing.");
                shardEndedInput.checkpointer().checkpoint();
            } catch (ShutdownException | InvalidStateException e) {
                log.error("Exception while checkpointing at shard end. Giving up.", e);
            } finally {
                MDC.remove(SHARD_ID_MDC_KEY);
            }
        }

        public void shutdownRequested(ShutdownRequestedInput shutdownRequestedInput) {
            MDC.put(SHARD_ID_MDC_KEY, shardId);
            try {
                log.info("Scheduler is shutting down, checkpointing.");
                shutdownRequestedInput.checkpointer().checkpoint();
            } catch (ShutdownException | InvalidStateException e) {
                log.error("Exception while checkpointing at requested shutdown. Giving
up.", e);
            } finally {
                MDC.remove(SHARD_ID_MDC_KEY);
            }
        }
    }
```

```
}
```

Migra i consumatori da KCL 1.x a KCL 2.x

> **Important**
>
> Le versioni 1.x e 2.x di Amazon Kinesis Client Library (KCL) sono obsolete. KCL 1.x sarà disponibile il 30 gennaio 2026. end-of-support Ti consigliamo vivamente di migrare le tue applicazioni KCL utilizzando la versione 1.x all'ultima versione di KCL prima del 30 gennaio 2026. Per trovare la versione più recente di KCL, consulta la pagina Amazon Kinesis Client Library su. GitHub Per informazioni sulle ultime versioni di KCL, consulta. Usa la libreria client Kinesis Per informazioni sulla migrazione da KCL 1.x a KCL 3.x, consulta. Migrazione da KCL 1.x a KCL 3.x

Questo argomento descrive le differenze tra le versioni 1.x e 2.x della Kinesis Client Library (KCL). Viene inoltre illustrato come migrare l'applicazione consumer dalla versione 1.x alla versione 2.x della KCL. Dopo la migrazione, il client avvierà l'elaborazione dei record dall'ultimo punto di controllo verificato.

La versione 2.0 della KCL introduce le seguenti modifiche di interfaccia:

Modifiche di interfaccia KCL

Interfaccia KCL 1.x	Interfaccia KCL 2.0
`com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor`	`software.amazon.kinesis.processor.ShardRecordProcessor`
`com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessorFactory`	`software.amazon.kinesis.processor.ShardRecordProcessorFactory`
`com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IShutdownNotificationAware`	Piegata in `software.amazon.kinesis.processor.ShardRecordProcessor`

Argomenti

- Migrare il processore di registrazione
- Eseguire la migrazione della fabbrica del processore di registrazione
- Migrare il lavoratore
- Configurazione del client Amazon Kinesis
- Rimozione dei tempi di inattività
- Rimozioni della configurazione del client

Migrare il processore di registrazione

L'esempio seguente mostra un elaboratore di record implementato per &KCL; 1.x:

```
package com.amazonaws.kcl;

import com.amazonaws.services.kinesis.clientlibrary.exceptions.InvalidStateException;
import com.amazonaws.services.kinesis.clientlibrary.exceptions.ShutdownException;
import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.IRecordProcessorCheckpointer;
import com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor;
import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IShutdownNotificationAware;
import com.amazonaws.services.kinesis.clientlibrary.lib.worker.ShutdownReason;
import com.amazonaws.services.kinesis.clientlibrary.types.InitializationInput;
import com.amazonaws.services.kinesis.clientlibrary.types.ProcessRecordsInput;
import com.amazonaws.services.kinesis.clientlibrary.types.ShutdownInput;

public class TestRecordProcessor implements IRecordProcessor,
 IShutdownNotificationAware {
    @Override
    public void initialize(InitializationInput initializationInput) {
        //
        // Setup record processor
        //
    }

    @Override
    public void processRecords(ProcessRecordsInput processRecordsInput) {
        //
        // Process records, and possibly checkpoint
        //
    }
```

```
    @Override
    public void shutdown(ShutdownInput shutdownInput) {
        if (shutdownInput.getShutdownReason() == ShutdownReason.TERMINATE) {
            try {
                shutdownInput.getCheckpointer().checkpoint();
            } catch (ShutdownException | InvalidStateException e) {
                throw new RuntimeException(e);
            }
        }
    }

    @Override
    public void shutdownRequested(IRecordProcessorCheckpointer checkpointer) {
        try {
            checkpointer.checkpoint();
        } catch (ShutdownException | InvalidStateException e) {
            //
            // Swallow exception
            //
            e.printStackTrace();
        }
    }
}
```

Per migrare la classe dell'elaboratore di record

1. Modifica le interfacce da `com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor` e `com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IShutdownNotificat` verso `software.amazon.kinesis.processor.ShardRecordProcessor`, come segue:

```
// import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor;
// import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IShutdownNotificationAware;
import software.amazon.kinesis.processor.ShardRecordProcessor;

// public class TestRecordProcessor implements IRecordProcessor,
 IShutdownNotificationAware {
public class TestRecordProcessor implements ShardRecordProcessor {
```

2. Aggiorna le istruzioni `import` per i metodi `initialize` e `processRecords`.

```
// import com.amazonaws.services.kinesis.clientlibrary.types.InitializationInput;
import software.amazon.kinesis.lifecycle.events.InitializationInput;

//import com.amazonaws.services.kinesis.clientlibrary.types.ProcessRecordsInput;
import software.amazon.kinesis.lifecycle.events.ProcessRecordsInput;
```

3. Sostituisci il metodo `shutdown` con i seguenti nuovi metodi: `leaseLost`, `shardEnded` e `shutdownRequested`.

```
//    @Override
//    public void shutdownRequested(IRecordProcessorCheckpointer checkpointer) {
//        //
//        // This is moved to shardEnded(...)
//        //
//        try {
//            checkpointer.checkpoint();
//        } catch (ShutdownException | InvalidStateException e) {
//            //
//            // Swallow exception
//            //
//            e.printStackTrace();
//        }
//    }

    @Override
    public void leaseLost(LeaseLostInput leaseLostInput) {

    }

    @Override
    public void shardEnded(ShardEndedInput shardEndedInput) {
        try {
            shardEndedInput.checkpointer().checkpoint();
        } catch (ShutdownException | InvalidStateException e) {
            //
            // Swallow the exception
            //
            e.printStackTrace();
        }
    }
```

```
//    @Override
//    public void shutdownRequested(IRecordProcessorCheckpointer checkpointer) {
//        //
//        // This is moved to shutdownRequested(ShutdownReauestedInput)
//        //
//        try {
//            checkpointer.checkpoint();
//        } catch (ShutdownException | InvalidStateException e) {
//            //
//            // Swallow exception
//            //
//            e.printStackTrace();
//        }
//    }

    @Override
    public void shutdownRequested(ShutdownRequestedInput shutdownRequestedInput) {
        try {
            shutdownRequestedInput.checkpointer().checkpoint();
        } catch (ShutdownException | InvalidStateException e) {
            //
            // Swallow the exception
            //
            e.printStackTrace();
        }
    }
```

Segue la versione aggiornata della classe dell'elaboratore di record.

```
package com.amazonaws.kcl;

import software.amazon.kinesis.exceptions.InvalidStateException;
import software.amazon.kinesis.exceptions.ShutdownException;
import software.amazon.kinesis.lifecycle.events.InitializationInput;
import software.amazon.kinesis.lifecycle.events.LeaseLostInput;
import software.amazon.kinesis.lifecycle.events.ProcessRecordsInput;
import software.amazon.kinesis.lifecycle.events.ShardEndedInput;
import software.amazon.kinesis.lifecycle.events.ShutdownRequestedInput;
import software.amazon.kinesis.processor.ShardRecordProcessor;

public class TestRecordProcessor implements ShardRecordProcessor {
    @Override
```

```
    public void initialize(InitializationInput initializationInput) {

    }

    @Override
    public void processRecords(ProcessRecordsInput processRecordsInput) {

    }

    @Override
    public void leaseLost(LeaseLostInput leaseLostInput) {

    }

    @Override
    public void shardEnded(ShardEndedInput shardEndedInput) {
        try {
            shardEndedInput.checkpointer().checkpoint();
        } catch (ShutdownException | InvalidStateException e) {
            //
            // Swallow the exception
            //
            e.printStackTrace();
        }
    }

    @Override
    public void shutdownRequested(ShutdownRequestedInput shutdownRequestedInput) {
        try {
            shutdownRequestedInput.checkpointer().checkpoint();
        } catch (ShutdownException | InvalidStateException e) {
            //
            // Swallow the exception
            //
            e.printStackTrace();
        }
    }
}
```

Eseguire la migrazione della fabbrica del processore di registrazione

La fabbrica dell'elaboratore di record è responsabile per la creazione di elaboratori di record quando un lease è acquisito. Di seguito è illustrato un esempio di una fabbrica &KCL; 1.x.

```
package com.amazonaws.kcl;

import com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor;
import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessorFactory;

public class TestRecordProcessorFactory implements IRecordProcessorFactory {
    @Override
    public IRecordProcessor createProcessor() {
        return new TestRecordProcessor();
    }
}
```

Per migrare la fabbrica dell'elaboratore di record

1. Modifica l'interfaccia implementata da `com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessorFa` a `software.amazon.kinesis.processor.ShardRecordProcessorFactory`, come segue.

   ```
   // import
    com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor;
   import software.amazon.kinesis.processor.ShardRecordProcessor;

   // import
    com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessorFactory;
   import software.amazon.kinesis.processor.ShardRecordProcessorFactory;

   // public class TestRecordProcessorFactory implements IRecordProcessorFactory {
   public class TestRecordProcessorFactory implements ShardRecordProcessorFactory {
   ```

2. Modifica la firma di ritorno per `createProcessor`.

   ```
   // public IRecordProcessor createProcessor() {
   public ShardRecordProcessor shardRecordProcessor() {
   ```

Di seguito è riportato un esempio di fabbrica di elaboratore di record in 2.0:

```
package com.amazonaws.kcl;
```

```
import software.amazon.kinesis.processor.ShardRecordProcessor;
import software.amazon.kinesis.processor.ShardRecordProcessorFactory;

public class TestRecordProcessorFactory implements ShardRecordProcessorFactory {
    @Override
    public ShardRecordProcessor shardRecordProcessor() {
        return new TestRecordProcessor();
    }
}
```

Migrare il lavoratore

Nella versione 2.0 della KCL, una nuova classe, denominata `Scheduler`, sostituisce la classe `Worker`. Di seguito è illustrato un esempio di un worker di KCL 1.x.

```
final KinesisClientLibConfiguration config = new KinesisClientLibConfiguration(...)
final IRecordProcessorFactory recordProcessorFactory = new RecordProcessorFactory();
final Worker worker = new Worker.Builder()
    .recordProcessorFactory(recordProcessorFactory)
    .config(config)
    .build();
```

Per migrare il lavoratore

1. Modifica la dichiarazione `import` per la classe `Worker` nelle dichiarazioni di importazione delle classi `Scheduler` e `ConfigsBuilder`.

   ```
   // import com.amazonaws.services.kinesis.clientlibrary.lib.worker.Worker;
   import software.amazon.kinesis.coordinator.Scheduler;
   import software.amazon.kinesis.common.ConfigsBuilder;
   ```

2. Crea `ConfigsBuilder` e `Scheduler` come mostrato nell'esempio seguente.

   Si consiglia di utilizzare `KinesisClientUtil` per creare `KinesisAsyncClient` e configurare `maxConcurrency` in `KinesisAsyncClient`.

   > **Important**
   >
   > Il client Amazon Kinesis potrebbe presentare un aumento significativo della latenza, a meno che non venga configurato `KinesisAsyncClient` per avere una

`maxConcurrency` sufficientemente alta per consentire tutti i canoni più ulteriori utilizzi di `KinesisAsyncClient`.

```
import java.util.UUID;

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;
import software.amazon.awssdk.services.cloudwatch.CloudWatchAsyncClient;
import software.amazon.awssdk.services.kinesis.KinesisAsyncClient;
import software.amazon.kinesis.common.ConfigsBuilder;
import software.amazon.kinesis.common.KinesisClientUtil;
import software.amazon.kinesis.coordinator.Scheduler;

...

Region region = Region.AP_NORTHEAST_2;
KinesisAsyncClient kinesisClient =
 KinesisClientUtil.createKinesisAsyncClient(KinesisAsyncClient.builder().region(region));
DynamoDbAsyncClient dynamoClient =
 DynamoDbAsyncClient.builder().region(region).build();
CloudWatchAsyncClient cloudWatchClient =
 CloudWatchAsyncClient.builder().region(region).build();

ConfigsBuilder configsBuilder = new ConfigsBuilder(streamName, applicationName,
 kinesisClient, dynamoClient, cloudWatchClient, UUID.randomUUID().toString(), new
 SampleRecordProcessorFactory());

Scheduler scheduler = new Scheduler(
    configsBuilder.checkpointConfig(),
    configsBuilder.coordinatorConfig(),
    configsBuilder.leaseManagementConfig(),
    configsBuilder.lifecycleConfig(),
    configsBuilder.metricsConfig(),
    configsBuilder.processorConfig(),
    configsBuilder.retrievalConfig()
    );
```

Configurazione del client Amazon Kinesis

Con il rilascio 2.0 della Kinesis Client Library, la configurazione del client si è spostata da una singola classe di configurazione (`KinesisClientLibConfiguration`) a sei classi di configurazione. La tabella seguente descrive la migrazione.

Campi di configurazione e relative nuove classi

Campo originale	Nuova classe di configurazione	Descrizione
`applicationName`	`ConfigsBuilder`	Il nome per l'applicazione della KCL. Utilizzato come predefinito per `tableName` e `consumerName` .
`tableName`	`ConfigsBuilder`	Consente di ignorare il nome della tabella utilizzato per la tabella di lease di Amazon DynamoDB.
`streamName`	`ConfigsBuilder`	Il nome del flusso dal quale l'applicazione elabora i record.
`kinesisEndpoint`	`ConfigsBuilder`	Questa opzione è stata eliminata. Configurazione client, vedere la sezione Rimozioni.
`dynamoDBEndpoint`	`ConfigsBuilder`	Questa opzione è stata eliminata. Configurazione client, vedere la sezione Rimozioni.
`initialPositionInStreamExtended`	`RetrievalConfig`	La posizione nello shard da cui il KCL inizia a recuperare i record, a partire dall'esecuzione iniziare all'applicazione.
`kinesisCredentialsProvider`	`ConfigsBuilder`	Questa opzione è stata eliminata. Configurazione client, vedere la sezione Rimozioni.
`dynamoDBCredentialsProvider`	`ConfigsBuilder`	Questa opzione è stata eliminata. Configurazione client, vedere la sezione Rimozioni.

Campo originale	Nuova classe di configurazione	Descrizione
`cloudWatchCredentialsProvider`	`ConfigsBuilder`	Questa opzione è stata eliminata. Configurazione client, vedere la sezione Rimozioni.
`failoverTimeMillis`	LeaseManagementConfig	Il numero di millisecondi che devono passare prima di poter considerare un proprietario di lease come fallito.
`workerIdentifier`	`ConfigsBuilder`	Un identificatore univoco che rappresenta la creazione dell'elaboratore di applicazione. Deve essere univoco.
`shardSyncIntervalMillis`	LeaseManagementConfig	Il periodo di tempo tra le chiamate di sincronizzazione dello shard.
`maxRecords`	`PollingConfig`	Consente di impostare il numero massimo di record restituiti da Kinesis.
`idleTimeBetweenReadsInMillis`	`CoordinatorConfig`	Questa opzione è stata eliminata. Vedi rimozione tempo di inattività.
`callProcessRecordsEvenForEmptyRecordList`	`ProcessorConfig`	Quando impostato, l'elaboratore di record viene chiamato anche quando nessun record è stato fornito da .
`parentShardPollIntervalMillis`	`CoordinatorConfig`	Con quale frequenza un elaboratore di record deve eseguire il polling per vedere se il shard padre è stata completato.
`cleanupLeasesUponShardCompletion`	`LeaseManagementConfig`	Quando impostati, i lease vengono rimossi non appena i lease figlio hanno iniziato l'elaborazione.

Campo originale	Nuova classe di configurazione	Descrizione
`ignoreUnexpectedChildShards`	`LeaseManagementConfig`	Quando impostato, i shard figlio che hanno un shard aperto vengono ignorati. Questo è principalmente per DynamoDB Streams.
`kinesisClientConfig`	`ConfigsBuilder`	Questa opzione è stata eliminata. Configurazione client, vedere la sezione Rimozioni.
`dynamoDBClientConfig`	`ConfigsBuilder`	Questa opzione è stata eliminata. Configurazione client, vedere la sezione Rimozioni.
`cloudWatchClientConfig`	`ConfigsBuilder`	Questa opzione è stata eliminata. Configurazione client, vedere la sezione Rimozioni.
`taskBackoffTimeMillis`	`LifecycleConfig`	Il tempo di attesa per riprovare operazioni non riuscite.
`metricsBufferTimeMillis`	`MetricsConfig`	Controlla la pubblicazione CloudWatch delle metriche.
`metricsMaxQueueSize`	`MetricsConfig`	Controlla la pubblicazione CloudWatch delle metriche.
`metricsLevel`	`MetricsConfig`	Controlla la pubblicazione CloudWatch delle metriche.
`metricsEnabledDimensions`	`MetricsConfig`	Controlla la pubblicazione CloudWatch delle metriche.
`validateSequenceNumberBeforeCheckpointing`	`CheckpointConfig`	Questa opzione è stata eliminata. Vedi la convalida del numero di sequenza del checkpoint.

Campo originale	Nuova classe di configurazione	Descrizione
`regionName`	`ConfigsBuilder`	Questa opzione è stata eliminata. Vedi la rimozione della configurazione client.
`maxLeasesForWorker`	`LeaseManagementConfig`	Il numero massimo di lease che una singola istanza dell'applicazione deve accettare.
`maxLeasesToStealAtOneTime`	`LeaseManagementConfig`	Il numero massimo di lease che un'applicazione deve tentare di intercettare simultaneamente.
`initialLeaseTableReadCapacity`	`LeaseManagementConfig`	La IOPs lettura DynamoDB utilizzata se la Kinesis Client Library deve creare una nuova tabella di lease DynamoDB.
`initialLeaseTableWriteCapacity`	`LeaseManagementConfig`	La IOPs lettura DynamoDB utilizzata se la Kinesis Client Library deve creare una nuova tabella di lease DynamoDB.
`initialPositionInStreamExtended`	LeaseManagementConfig	La posizione iniziale nel flusso nella quale l'applicazione dovrebbe iniziare. Questo viene utilizzato soltanto durante la creazione del lease iniziale.
`skipShardSyncAtWorkerInitializationIfLeasesExist`	`CoordinatorConfig`	Disabilita la sincronizzazione dei dati shard se la tabella di lease contiene lease esistenti. DA KinesisEco FARE: -438
`shardPrioritization`	`CoordinatorConfig`	Quale prioritizzazione shard utilizzare.
`shutdownGraceMillis`	N/D	Questa opzione è stata eliminata. Vedi Traslochi MultiLang .

Campo originale	Nuova classe di configurazione	Descrizione
`timeoutInSeconds`	N/D	Questa opzione è stata eliminata. Vedi MultiLang Rimozioni.
`retryGetRecordsInSeconds`	`PollingConfig`	Configura il ritardo tra i GetRecords tentativi di errore.
`maxGetRecordsThreadPool`	`PollingConfig`	La dimensione del pool di thread utilizzato per. GetRecords
`maxLeaseRenewalThreads`	`LeaseManagementConfig`	Controlla le dimensioni del pool di thread di rinnovo del lease. Quanto maggiori sono i lease che può richiedere l'applicazione, tanto più grande deve essere questo pool.
`recordsFetcherFactory`	`PollingConfig`	Consente di sostituire la factory utilizzata per creare fetcher che recuperano dai flussi.
`logWarningForTaskAfterMillis`	`LifecycleConfig`	Quanto tempo bisogna attendere prima che venga registrato un avviso se un'attività non è stata completata.
`listShardsBackoffTimeInMillis`	`RetrievalConfig`	Il numero di millisecondi di attesa tra le chiamate in `ListShards` quando si verificano errori.
`maxListShardsRetryAttempts`	`RetrievalConfig`	Il numero massimo di volte che `ListShards` effettua nuovi tentativi prima di desistere.

Rimozione dei tempi di inattività

Nella versione 1.x di &KCL;, `idleTimeBetweenReadsInMillis` corrispondeva a due quantità:

- La quantità di tempo tra controlli dell'attività. È ora possibile configurare questo periodo tra attività impostando `CoordinatorConfig#shardConsumerDispatchPollIntervalMillis`.
- La quantità di tempo di sospensione quando non è stato restituito alcun record da . Nella versione 2.0, nel fan-out ottimizzato, i record vengono inviati da chi li ha recuperati. L'attività sul consumo di shard avviene solo quando arriva una richiesta di push.

Rimozioni della configurazione del client

Nella versione 2.0, la KCL non crea più client. Spetta all'utente fornire un client valido. Con questa modifica, tutti i parametri di configurazione che controllavano la creazione del client sono stati rimossi. Se hai bisogno di questi parametri, puoi impostarli sui client prima di fornire i client a `ConfigsBuilder`.

Campo rimosso	Configurazione equivalente
`kinesisEndpoint`	Configura SDK `KinesisAsyncClient` con l'endpoint preferito: `KinesisAsyncClient.builder().endpointOverride(URI.create("https://<kinesis endpoint>")).build()` .
`dynamoDBEndpoint`	Configura SDK `DynamoDbAsyncClient` con l'endpoint preferito: `DynamoDbAsyncClient.builder().endpointOverride(URI.create("https://<dynamodb endpoint>")).build()` .
`kinesisClientConfig`	Configura SDK `KinesisAsyncClient` con la configurazione necessaria: `KinesisAsyncClient.builder().overrideConfiguration(<your configuration>).build()` .
`dynamoDBClientConfig`	Configura SDK `DynamoDbAsyncClient` con la configurazione necessaria: `DynamoDbAsyncClient.builder().overrideConfiguration(<your configuration>).build()` .
`cloudWatchClientConfig`	Configura SDK `CloudWatchAsyncClient` con la configurazione necessaria: `CloudWatchAsyncClient.builder().overrideConfiguration(<your configuration>).build()` .

Campo rimosso	Configurazione equivalente
`regionName`	Configura SDK con la regione preferita. Questo è uguale per tutti i client SDK. Ad esempio, `KinesisAsyncClient.builder().region(Region.US_WEST_2).build()` .

Sviluppa i consumatori con AWS SDK per Java

Puoi sviluppare consumatori personalizzati utilizzando Amazon Kinesis APIs Data Streams. Questa sezione descrive l'utilizzo di Kinesis APIs Data AWS SDK per Java Streams con.

> **Important**
>
> Il metodo consigliato per lo sviluppo di consumer del flusso di dati Kinesis personalizzati con condivisione integrale consiste nell'utilizzare la Kinesis Client Library (KCL). KCL ti aiuta a consumare ed elaborare i dati da un flusso di dati Kinesis occupandoti di molte delle attività complesse associate al calcolo distribuito. Per ulteriori informazioni, consulta Sviluppa i consumatori con KCL in Java.

Argomenti

- Sviluppa consumatori con produttività condivisa con AWS SDK per Java
- Sviluppa una maggiore fidelizzazione dei consumatori con il AWS SDK per Java
- Interagisci con i dati utilizzando lo Schema Registry AWS Glue

Sviluppa consumatori con produttività condivisa con AWS SDK per Java

Uno dei metodi per sviluppare utenti Kinesis Data Streams personalizzati con condivisione integrale consiste nell'utilizzare Amazon APIs Kinesis Data Streams con. AWS SDK per Java Questa sezione descrive l'utilizzo di Kinesis APIs Data AWS SDK per Java Streams con. È possibile chiamare Kinesis APIs Data Streams utilizzando altri linguaggi di programmazione diversi. Per ulteriori informazioni su tutto ciò che è disponibile AWS SDKs, consulta Inizia a sviluppare con Amazon Web Services.

Il codice di esempio Java in questa sezione mostra come eseguire le operazioni di base dell'API Kinesis Data Streams ed è suddiviso logicamente per tipo di operazione. Questi esempi non

rappresentano il codice pronto per la produzione. Non verificano la presenza di eccezioni o di account per tutte le possibili considerazioni di sicurezza o di prestazione.

Argomenti

- Ottieni dati da un flusso
- Usa gli iteratori shard
- Usa GetRecords
- Adattarsi a un reshard

Ottieni dati da un flusso

I Kinesis APIs Data Streams `getShardIterator` includono i metodi e che `getRecords` è possibile richiamare per recuperare i record da un flusso di dati. Si tratta di un modello pull, dove il codice disegna i record di dati direttamente dalle partizioni del flusso di dati.

> **Important**
>
> È consigliabile utilizzare il supporto del processore di record fornito dalla KCL per recuperare i record dati flussi di dati. Si tratta di un modello push, dove è possibile implementare il codice che elabora i dati. La KCL recupera i record di dati dal flusso di dati e li fornisce al tuo codice di applicazione. Inoltre, la KCL fornisce le funzionalità di failover, ripristino e bilanciamento del carico. Per ulteriori informazioni, consulta Sviluppo di consumer personalizzati con velocità di trasmissione effettiva condivisa tramite KCL.

Tuttavia, in alcuni casi potresti preferire utilizzare Kinesis APIs Data Streams. Ad esempio, per implementare strumenti personalizzati per il monitoraggio o il debug dei tuoi flussi di dati.

> **Important**
>
> Il flusso di dati Kinesis supporta le modifiche al periodo di conservazione dei record di dati del tuo flusso di dati. Per ulteriori informazioni, consulta Modifica il periodo di conservazione dei dati.

Usa gli iteratori shard

Puoi recuperare i record dal flusso per shard. Per ogni shard e per ogni batch di record recuperati da quello shard, è necessario ottenere un iteratore shard. L'iteratore shard viene utilizzato nell'oggetto `getRecordsRequest` per specificare lo shard da cui devono essere recuperati i dati. Il tipo associato all'iteratore shard determina il punto nello shard da cui devono essere recuperati i record (vedi più avanti in questa sezione per ulteriori dettagli). Prima di poter lavorare con lo shard iterator, è necessario recuperare lo shard. Per ulteriori informazioni, consulta Elenca i frammenti.

Ottieni l'iteratore shard iniziale utilizzando il metodo `getShardIterator`. Ottieni iteratori di shard per ulteriori batch di record utilizzando il metodo `getNextShardIterator` dell'oggetto `getRecordsResult` restituito dal metodo `getRecords`. Un iteratore shard è valido per 5 minuti. Se utilizzi un iteratore shard mentre è valido, puoi ottenerne uno nuovo. Ogni iteratore shard rimane valido per 5 minuti, anche dopo il suo utilizzo.

Per ottenere l'iteratore shard iniziale, crea un'istanza `GetShardIteratorRequest` e passala al metodo `getShardIterator`. Per configurare la richiesta, specifica il flusso e l'ID dello shard. Per informazioni su come ottenere gli stream del tuo account, consulta. AWS Visualizzazione dell'elenco di flussi Per informazioni su come ottenere gli shard in un flusso, consulta Elenca i frammenti.

```
String shardIterator;
GetShardIteratorRequest getShardIteratorRequest = new GetShardIteratorRequest();
getShardIteratorRequest.setStreamName(myStreamName);
getShardIteratorRequest.setShardId(shard.getShardId());
getShardIteratorRequest.setShardIteratorType("TRIM_HORIZON");

GetShardIteratorResult getShardIteratorResult =
 client.getShardIterator(getShardIteratorRequest);
shardIterator = getShardIteratorResult.getShardIterator();
```

Questo codice di esempio specifica `TRIM_HORIZON` come il tipo di iteratore che si utilizza per ottenere l'iteratore shard iniziale. Questo tipo di iterazione significa che i record devono essere restituiti a partire dal primo record aggiunto alla partizione anziché a partire dal record aggiunto più di recente, noto anche come estremità. I possibili tipi di iteratore sono i seguenti:

- `AT_SEQUENCE_NUMBER`
- `AFTER_SEQUENCE_NUMBER`
- `AT_TIMESTAMP`
- `TRIM_HORIZON`

- `LATEST`

Per ulteriori informazioni, consulta ShardIteratorType.

Per alcuni tipi di iteratore è necessario specificare un numero di sequenza in aggiunta al tipo, ad esempio:

```
getShardIteratorRequest.setShardIteratorType("AT_SEQUENCE_NUMBER");
getShardIteratorRequest.setStartingSequenceNumber(specialSequenceNumber);
```

Dopo aver ottenuto un record utilizzando `getRecords`, è possibile ottenere il numero di sequenza per il record chiamando il metodo `getSequenceNumber` del record.

```
record.getSequenceNumber()
```

Inoltre, il codice che aggiunge record per il flusso di dati è in grado di ottenere il numero di sequenza per un record aggiunto chiamando `getSequenceNumber` sul risultato di `putRecord`.

```
lastSequenceNumber = putRecordResult.getSequenceNumber();
```

È possibile utilizzare i numeri di sequenza per garantire che l'ordine dei record sia rigorosamente ascendente. Per ulteriori informazioni, consulta il codice di esempio in PutRecord Esempio.

Usa GetRecords

Dopo aver ottenuto l'iteratore shard, crea un'istanza di un oggetto `GetRecordsRequest`. Specifica l'iteratore per la richiesta utilizzando il metodo `setShardIterator`.

Facoltativamente, puoi anche impostare il numero di record da recuperare utilizzando il metodo `setLimit`. Il numero di record restituiti da `getRecords` è sempre pari o inferiori a questo limite. Se non si specifica questo limite, `getRecords` restituisce 10 MB di record recuperati. Il seguente codice di esempio imposta questo limite a 25 record.

Se non viene restituito alcun record, ciò significa che non sono attualmente disponibili record di dati da questo shard al numero di sequenza a cui fa riferimento l'iteratore shard. In questo caso la tua applicazione deve attendere per un periodo di tempo appropriato per le origini dati per il flusso. Di seguito, cerca di ottenere di nuovo dati dallo shard utilizzando l'iteratore shard restituito dalla chiamata precedente a `getRecords`.

Passa la `getRecordsRequest` al metodo `getRecords` e acquisisci il valore restituito come un oggetto `getRecordsResult`. Per ottenere i record di dati, chiama il metodo `getRecords` nell'oggetto `getRecordsResult`.

```
GetRecordsRequest getRecordsRequest = new GetRecordsRequest();
getRecordsRequest.setShardIterator(shardIterator);
getRecordsRequest.setLimit(25);

GetRecordsResult getRecordsResult = client.getRecords(getRecordsRequest);
List<Record> records = getRecordsResult.getRecords();
```

Per prepararti a un'altra chiamata a `getRecords`, ottieni l'iteratore shard successivo da `getRecordsResult`.

```
shardIterator = getRecordsResult.getNextShardIterator();
```

Per ottenere migliori risultati, sospendi l'attività per almeno 1 secondo (1.000 millisecondi) tra le chiamate a `getRecords` per evitare di superare il limite di frequenza di `getRecords`.

```
try {
  Thread.sleep(1000);
}
catch (InterruptedException e) {}
```

In genere, si deve effettuare la chiamata `getRecords` in un loop, anche quando si esegue il recupero di un singolo record in uno scenario di test. Una singola chiamata a `getRecords` può restituire un elenco di record vuoto, anche quando lo shard contiene più record in numeri di sequenza successivi. In questo caso, il `NextShardIterator` restituito con l'elenco vuoto fa riferimento a un numero di sequenza successivo nello shard e, in conclusione, le chiamate successive a `getRecords` restituiscono i record. L'esempio seguente dimostra l'uso di un loop.

Esempio: getRecords

Il seguente codice di esempio riflette i suggerimenti in merito a `getRecords` in questa sezione, tra cui l'esecuzione di chiamate in un loop.

```
// Continuously read data records from a shard
List<Record> records;

while (true) {

  // Create a new getRecordsRequest with an existing shardIterator
  // Set the maximum records to return to 25

  GetRecordsRequest getRecordsRequest = new GetRecordsRequest();
  getRecordsRequest.setShardIterator(shardIterator);
  getRecordsRequest.setLimit(25);

  GetRecordsResult result = client.getRecords(getRecordsRequest);

  // Put the result into record list. The result can be empty.
  records = result.getRecords();

  try {
    Thread.sleep(1000);
  }
  catch (InterruptedException exception) {
    throw new RuntimeException(exception);
  }

  shardIterator = result.getNextShardIterator();
}
```

Se si sta utilizzando la Kinesis Client Library, potrebbe effettuare più chiamate prima di restituire i dati. Questo comportamento è pianificato e non indica un problema con la KCL o con i tuoi dati.

Adattarsi a un reshard

Se `getRecordsResult.getNextShardIterator` restituisce `null`, indica che si è verificata una divisione o un'unione della partizione che ha interessato questa partizione. Questa partizione si trova ora nello stato `CLOSED` e hai letto tutti i record di dati disponibili da questa partizione.

In questo scenario, è possibile utilizzare `getRecordsResult.childShards` per conoscere le nuove partizioni secondarie della partizione in fase di elaborazione che sono state create dalla divisione o dall'unione. Per ulteriori informazioni, consulta ChildShard.

Nel caso di un frazionamento, i due nuovi shard hanno entrambi un `parentShardId` identico all'ID dello shard che si stava elaborando in precedenza. Il valore di `adjacentParentShardId` per entrambi gli shard è `null`.

Nel caso di una fusione, il singolo nuovo shard creato mediante la fusione ha un `parentShardId` identico all'ID di uno degli shard principali e un `adjacentParentShardId` identico all'ID dell'altro shard principale. La tua applicazione ha già letto tutti i dati provenienti da uno di questi shard. Questo è lo shard per cui `getRecordsResult.getNextShardIterator` ha restituito `null`. Se l'ordine dei dati è importante per la tua applicazione, assicurati che quest'ultima legga anche tutti i dati dall'altro shard principale prima di leggere qualsiasi nuovo dato dallo shard secondario creato dalla fusione.

Se sta utilizzando più processori per recuperare dati dal flusso (ad esempio, un processore per shard) e si verifica una frammentazione o una fusione di shard, devi aumentare o diminuire il numero di processori per adattarti alla variazione nel numero di shard.

Per ulteriori informazioni sul resharding, tra cui una discussione in merito agli stati degli shard - ad esempio `CLOSED` - consulta Condividi nuovamente uno stream.

Sviluppa una maggiore fidelizzazione dei consumatori con il AWS SDK per Java

Il fan-out avanzato è una funzionalità del flusso di dati Amazon Kinesis che consente ai consumer di ricevere dati da un flusso di dati con velocità di trasmissione effettiva dedicata fino a 2 MiB di dati al secondo per partizione. Un'applicazione consumer che utilizza il fan-out avanzato non è in competizione con altre applicazioni che ricevono dati dal flusso. Per ulteriori informazioni, consulta Sviluppa consumatori con fan-out migliorati con un throughput dedicato.

È possibile utilizzare le operazioni API per creare un'applicazione consumer che utilizza il fan-out avanzato nel flusso di dati Kinesis.

Registrazione di un'applicazione consumer con il fan-out avanzato mediante l'API del flusso di dati Kinesis

1. Chiama RegisterStreamConsumerper registrare la tua candidatura come consumatore che utilizza un fan-out avanzato. Il flusso di dati Kinesis genera un nome della risorsa Amazon (ARN) per il consumer e lo restituisce nella risposta.
2. Per iniziare ad ascoltare uno shard specifico, trasmetti l'ARN del consumatore in una chiamata a. SubscribeToShard Kinesis Data Streams inizia quindi a inviare all'utente i record da quello shard, sotto forma di eventi SubscribeToShardEventdi tipo su una connessione HTTP/2. La connessione rimane aperta per un massimo di 5 minuti. Chiama di SubscribeToShardnuovo se desideri continuare a ricevere i record dallo shard dopo il completamento `future` normale o eccezionale della chiamata. SubscribeToShard

   > **Note**
   >
   > L'API `SubscribeToShard` restituisce anche l'elenco delle partizioni secondarie della partizione corrente quando viene raggiunta la fine della partizione corrente.

3. Per annullare la registrazione di un consumatore che utilizza il fan-out avanzato, chiama. DeregisterStreamConsumer

Il seguente codice è un esempio di come è possibile sottoscrivere l'applicazione consumer a uno shard, rinnovare la sottoscrizione periodicamente e gestire gli eventi.

```
    import software.amazon.awssdk.services.kinesis.KinesisAsyncClient;
    import software.amazon.awssdk.services.kinesis.model.ShardIteratorType;
    import software.amazon.awssdk.services.kinesis.model.SubscribeToShardEvent;
    import software.amazon.awssdk.services.kinesis.model.SubscribeToShardRequest;
    import
 software.amazon.awssdk.services.kinesis.model.SubscribeToShardResponseHandler;

    import java.util.concurrent.CompletableFuture;

    /**
     * See https://github.com/awsdocs/aws-doc-sdk-examples/blob/master/javav2/
example_code/kinesis/src/main/java/com/example/kinesis/KinesisStreamEx.java
     * for complete code and more examples.
     */
    public class SubscribeToShardSimpleImpl {
```

```
        private static final String CONSUMER_ARN = "arn:aws:kinesis:us-
east-1:123456789123:stream/foobar/consumer/test-consumer:1525898737";
        private static final String SHARD_ID = "shardId-000000000000";

        public static void main(String[] args) {

            KinesisAsyncClient client = KinesisAsyncClient.create();

            SubscribeToShardRequest request = SubscribeToShardRequest.builder()
                    .consumerARN(CONSUMER_ARN)
                    .shardId(SHARD_ID)
                    .startingPosition(s -> s.type(ShardIteratorType.LATEST)).build();

            // Call SubscribeToShard iteratively to renew the subscription
 periodically.
            while(true) {
                // Wait for the CompletableFuture to complete normally or
 exceptionally.
                callSubscribeToShardWithVisitor(client, request).join();
            }

            // Close the connection before exiting.
            // client.close();
        }


        /**
         * Subscribes to the stream of events by implementing the
 SubscribeToShardResponseHandler.Visitor interface.
         */
        private static CompletableFuture<Void>
 callSubscribeToShardWithVisitor(KinesisAsyncClient client, SubscribeToShardRequest
 request) {
            SubscribeToShardResponseHandler.Visitor visitor = new
 SubscribeToShardResponseHandler.Visitor() {
                @Override
                public void visit(SubscribeToShardEvent event) {
                    System.out.println("Received subscribe to shard event " + event);
                }
            };
            SubscribeToShardResponseHandler responseHandler =
 SubscribeToShardResponseHandler
                    .builder()
```

```
                    .onError(t -> System.err.println("Error during stream - " +
t.getMessage()))
                    .subscriber(visitor)
                    .build();
            return client.subscribeToShard(request, responseHandler);
        }
    }
```

Se `event.ContinuationSequenceNumber` restituisce `null`, indica che si è verificata una divisione o un'unione della partizione che ha interessato questa partizione. Questa partizione si trova ora nello stato `CLOSED` e hai letto tutti i record di dati disponibili da questa partizione. In questo scenario, è possibile utilizzare `event.childShards` per conoscere le nuove partizioni secondarie della partizione in fase di elaborazione che sono state create dalla divisione o dall'unione. Per ulteriori informazioni, consulta ChildShard.

Interagisci con i dati utilizzando lo Schema Registry AWS Glue

Puoi integrare i tuoi flussi di dati Kinesis con lo Schema Registry. AWS Glue Lo AWS Glue Schema Registry ti consente di scoprire, controllare ed evolvere centralmente gli schemi, garantendo al contempo che i dati prodotti siano convalidati continuamente da uno schema registrato. Uno schema definisce la struttura e il formato di un registro di dati. Uno schema è una specifica con versioni per la pubblicazione, il consumo o l'archiviazione dei dati in modo affidabile. Lo AWS Glue Schema Registry consente di migliorare la qualità e la governance end-to-end dei dati all'interno delle applicazioni di streaming. Per ulteriori informazioni, consulta Registro degli schemi di AWS Glue. Uno dei modi per configurare questa integrazione è tramite l'API `GetRecords` Kinesis Data Streams disponibile AWS in Java SDK.

Per istruzioni dettagliate su come configurare l'integrazione di Kinesis Data Streams con Schema Registry `GetRecords` utilizzando Kinesis Data Streams, consulta la sezione «Interazione con i dati utilizzando APIs Kinesis Data Streams» in Caso d'uso: integrazione di Amazon Kinesis Data APIs Streams con il registro dello schema Glue. AWS

Sviluppa i consumatori utilizzando AWS Lambda

È possibile utilizzare una AWS Lambda funzione per elaborare i record in un flusso di dati. AWS Lambda è un servizio di elaborazione che consente di eseguire codice senza effettuare il provisioning o la gestione di server. Esegue il codice solo quando è necessario e si dimensiona automaticamente, da poche richieste al giorno a migliaia al secondo. I costi saranno calcolati in base al tempo di elaborazione effettivo. Quando il codice non è in esecuzione non viene addebitato alcun costo.

Con AWS Lambda, puoi eseguire codice praticamente per qualsiasi tipo di applicazione o servizio di backend, il tutto senza alcuna amministrazione. Esegue il codice in un'infrastruttura di calcolo ad alta disponibilità ed esegue tutte le operazioni di amministrazione delle risorse di calcolo, come gestione dei server e dei sistemi operativi, effettuazione del provisioning e scalabilità automatica della capacità, monitoraggio e registrazione del codice. Per ulteriori informazioni, consulta Utilizzo AWS Lambda con Amazon Kinesis.

Per informazioni sulla risoluzione dei problemi, consulta Perché il trigger del flusso di dati Kinesis non è in grado di richiamare la mia funzione Lambda?

Sviluppa i consumatori utilizzando Amazon Managed Service per Apache Flink

Puoi utilizzare un servizio gestito da Amazon per Apache Flink per elaborare e analizzare i dati in un flusso Kinesis utilizzando SQL, Java o Scala. Le applicazioni del servizio gestito per Apache Flink possono arricchire i dati utilizzando origini di riferimento, aggregare dati nel tempo o utilizzare il machine learning per trovare anomalie nei dati. Quindi puoi scrivere i risultati dell'analisi su un altro flusso Kinesis, un flusso di distribuzione Firehose o una funzione Lambda. Per ulteriori informazioni, consulta la Guida per gli sviluppatori del servizio gestito per Apache Flink per applicazioni SQL o la Guida per gli sviluppatori del servizio gestito per Apache Flink per applicazioni Flink.

Sviluppa i consumatori utilizzando Amazon Data Firehose

È possibile utilizzare un Firehose per leggere ed elaborare i record da un flusso Kinesis. Firehose è un servizio completamente gestito per la distribuzione di dati di streaming in tempo reale a destinazioni come Amazon S3, Amazon Redshift, Amazon OpenSearch Service e Splunk. Firehose supporta anche qualsiasi endpoint HTTP personalizzato o endpoint HTTP di proprietà di fornitori di servizi terzi supportati, tra cui Datadog, MongoDB e New Relic. È inoltre possibile configurare Firehose per trasformare i record di dati e convertire il formato dei record prima di consegnarli a destinazione. Per ulteriori informazioni, vedere Writing to Firehose Using Kinesis Data Streams.

Leggi i dati da Kinesis Data AWS Streams utilizzando altri servizi

I seguenti AWS servizi possono integrarsi direttamente con Amazon Kinesis Data Streams per leggere i dati dai flussi di dati Kinesis. Consulta le informazioni relative a ciascun servizio che ti interessa e consulta i riferimenti forniti.

Argomenti

- Leggi i dati da Kinesis Data Streams utilizzando Amazon EMR
- Leggi i dati da Kinesis Data Streams EventBridge utilizzando Amazon Pipes
- Leggi i dati da Kinesis Data Streams utilizzando AWS Glue
- Leggi i dati da Kinesis Data Streams utilizzando Amazon Redshift

Leggi i dati da Kinesis Data Streams utilizzando Amazon EMR

I cluster Amazon EMR possono leggere ed elaborare direttamente gli stream Kinesis, utilizzando strumenti familiari nell'ecosistema Hadoop come Hive, Pig, l'API Hadoop Streaming e Cascading. MapReduce Puoi anche unire i dati in tempo reale di Kinesis Data Streams con i dati esistenti su Amazon S3, Amazon DynamoDB e HDFS in un cluster in esecuzione. I dati possono essere caricati direttamente da Amazon EMR su Amazon S3 o DynamoDB per le attività di post-elaborazione.

Per ulteriori informazioni, consulta la sezione Amazon Kinesis nella Guida alle versioni di Amazon EMR.

Leggi i dati da Kinesis Data Streams EventBridge utilizzando Amazon Pipes

Amazon EventBridge Pipes supporta i flussi di dati Kinesis come fonte. Amazon EventBridge Pipes ti aiuta a creare point-to-point integrazioni tra produttori di eventi e consumatori con passaggi opzionali di trasformazione, filtro e arricchimento. È possibile utilizzare EventBridge Pipes per ricevere i record in un flusso di dati Kinesis e, facoltativamente, filtrare o migliorare questi record prima di inviarli a una delle destinazioni disponibili per l'elaborazione, incluso Kinesis Data Streams.

Per ulteriori informazioni, consulta Amazon Kinesis stream come sorgente nella Amazon EventBridge Release Guide.

Leggi i dati da Kinesis Data Streams utilizzando AWS Glue

Utilizzando AWS Glue lo streaming ETL, puoi creare processi di estrazione, trasformazione e caricamento (ETL) in streaming che vengono eseguiti in modo continuo e utilizzano dati da Amazon Kinesis Data Streams. I processi puliscono e trasformano i dati, quindi caricano i risultati in data lake Amazon S3 o datastore JDBC.

Per ulteriori informazioni, consulta Streaming dei processi ETL in AWS Glue nella Guida alle versioni di AWS Glue .

Leggi i dati da Kinesis Data Streams utilizzando Amazon Redshift

Amazon Redshift supporta l'importazione dati in streaming del flusso di dati Amazon Kinesis. La funzionalità di importazione dei dati in streaming di Amazon Redshift fornisce l'importazione a bassa latenza e ad alta velocità dei dati in streaming da Flusso di dati Amazon Kinesis in una vista materializzata di Amazon Redshift. L'ingestione dello streaming di Amazon Redshift elimina la necessità di effettuare lo staging dei dati in Amazon S3 prima di importarli in Amazon Redshift.

Per ulteriori informazioni, consulta la pagina Importazione in streaming nella Guida per le versioni di Amazon Redshift.

Leggi da Kinesis Data Streams utilizzando integrazioni di terze parti

Puoi leggere i dati dai flussi di dati di Amazon Kinesis Data Streams utilizzando una delle seguenti opzioni di terze parti che si integrano con Kinesis Data Streams. Seleziona l'opzione su cui desideri saperne di più e trova risorse e link alla documentazione pertinente.

Argomenti

- Apache Flink
- Piattaforma Adobe Experience
- Apache Druid
- Apache Spark
- Databricks
- Piattaforma Kafka Confluent
- Kinesumer
- Talend

Apache Flink

Apache Flink è un diffuso framework open source e motore di elaborazione distribuito per calcoli con stato su flussi di dati illimitati e limitati. Per ulteriori informazioni sull'utilizzo del flusso di dati Kinesis con Apache Flink, consulta Connettore del flusso di dati Amazon Kinesis.

Piattaforma Adobe Experience

La piattaforma Adobe Experience consente alle organizzazioni di centralizzare e standardizzare i dati dei clienti da qualsiasi sistema. Quindi applica data science e machine learning per migliorare notevolmente la progettazione e la distribuzione di esperienze ricche e personalizzate. Per ulteriori informazioni sull'utilizzo dei flussi di dati Kinesis utilizzando Adobe Experience Platform, consulta Amazon Kinesis connector.

Apache Druid

Druid è un database di analisi in tempo reale ad alte prestazioni che fornisce query in meno di un secondo su streaming e dati in batch su larga scala e sotto carico. Per ulteriori informazioni sull'acquisizione di flussi di dati Kinesis con Apache Druid, consulta Amazon Kinesis ingestion.

Apache Spark

Apache Spark è un motore di analisi unificato per l'elaborazione di dati su larga scala. Fornisce contenuti di alto livello APIs in Java, Scala, Python e R e un motore ottimizzato che supporta grafici di esecuzione generali. Puoi utilizzare Apache Spark per creare applicazioni di elaborazione di flussi che utilizzano i dati nei tuoi flussi di dati Kinesis.

Per utilizzare i flussi di dati Kinesis utilizzando Apache Spark Structured Streaming, usa il connettore Amazon Kinesis Data Streams. Questo connettore supporta il consumo con Enhanced Fan-Out, che fornisce all'applicazione una velocità di lettura dedicata fino a 2 MB di dati al secondo per shard. Per ulteriori informazioni, vedere Sviluppo di consumatori personalizzati con throughput dedicato (Enhanced Fan-Out).

Per utilizzare i flussi di dati Kinesis utilizzando Spark Streaming, consulta Spark Streaming + Kinesis Integration.

Databricks

Databricks è una piattaforma basata su cloud che fornisce un ambiente collaborativo per l'ingegneria dei dati, il data science e il machine learning. Per ulteriori informazioni sull'utilizzo dei flussi di dati Kinesis con Databricks, consulta Connect to Amazon Kinesis.

Piattaforma Kafka Confluent

La piattaforma Confluent si basa su Kafka e offre caratteristiche e funzionalità aggiuntive che aiutano le aziende a creare e gestire pipeline di dati e applicazioni di streaming in tempo reale. Per ulteriori

informazioni sull'utilizzo dei flussi di dati Kinesis utilizzando la piattaforma Confluent, consulta Amazon Kinesis Source Connector for Confluent Platform.

Kinesumer

Kinesumer è un client Go che implementa un client distribuito per gruppi di consumatori lato client per i flussi di dati Kinesis. Per ulteriori informazioni, consulta il repository GitHub Kinesumer.

Talend

Talend è un software di integrazione e gestione dei dati che consente agli utenti di raccogliere, trasformare e connettere dati da varie origini in modo scalabile ed efficiente. Per ulteriori informazioni sull'utilizzo dei flussi di dati Kinesis con Talend, consulta Connect talend a un flusso Amazon Kinesis.

Risolvi i problemi dei consumatori di Kinesis Data Streams

I seguenti argomenti offrono soluzioni a problemi comuni con i consumatori di Amazon Kinesis Data Streams:

- Errore di compilazione con il costruttore LeaseManagementConfig
- Alcuni record Kinesis Data Streams vengono ignorati quando si utilizza la Kinesis Client Library.
- I record che appartengono allo stesso frammento vengono elaborati da diversi processori di record contemporaneamente
- L'applicazione consumer sta leggendo a una velocità inferiore al previsto
- GetRecords restituisce un array di record vuoto anche quando ci sono dati nel flusso
- Lo shard iterator scade in modo imprevisto
- L'elaborazione dei record dei consumatori è in ritardo
- Errore di autorizzazione della chiave KMS non autorizzata
- DynamoDbException: il percorso del documento fornito nell'espressione di aggiornamento non è valido per l'aggiornamento
- Risolvi altri problemi comuni per i consumatori

Errore di compilazione con il costruttore LeaseManagementConfig

Durante l'aggiornamento alla versione 3.x o successiva di Kinesis Client Library (KCL), è possibile che si verifichi un errore di compilazione relativo al costruttore. `LeaseManagementConfig` Se stai

creando direttamente un `LeaseManagementConfig` oggetto per impostare le configurazioni anziché utilizzarlo `ConfigsBuilder` nelle versioni KCL 3.x o successive, potresti visualizzare il seguente messaggio di errore durante la compilazione del codice dell'applicazione KCL.

```
Cannot resolve constructor 'LeaseManagementConfig(String, DynamoDbAsyncClient,
 KinesisAsyncClient, String)'
```

KCL con versioni 3.x o successive richiede l'aggiunta di un altro parametro, ApplicationName (type: String), dopo il parametro TableName.

- Prima: leaseManagementConfig = new LeaseManagementConfig (tableName, DBClient dynamo, kinesisClient, streamName, workerIdentifier)
- Dopo: leaseManagementConfig = new LeaseManagementConfig (tablename, applicationName, dynamo, kinesisClient, streamNameDBClient, workerIdentifier)

Invece di creare direttamente un LeaseManagementConfig oggetto, consigliamo di utilizzarlo per impostare le configurazioni in KCL 3.x e versioni successive. `ConfigsBuilder` `ConfigsBuilder`offre un modo più flessibile e gestibile per configurare l'applicazione KCL.

Di seguito è riportato un esempio di utilizzo `ConfigsBuilder` per impostare le configurazioni KCL.

```
ConfigsBuilder configsBuilder = new ConfigsBuilder(
    streamName,
    applicationName,
    kinesisClient,
    dynamoClient,
    cloudWatchClient,
    UUID.randomUUID().toString(),
    new SampleRecordProcessorFactory()
);

Scheduler scheduler = new Scheduler(
    configsBuilder.checkpointConfig(),
    configsBuilder.coordinatorConfig(),
    configsBuilder.leaseManagementConfig()
    .failoverTimeMillis(60000), // this is an example
    configsBuilder.lifecycleConfig(),
    configsBuilder.metricsConfig(),
    configsBuilder.processorConfig(),
    configsBuilder.retrievalConfig()
```

```
);
```

Alcuni record Kinesis Data Streams vengono ignorati quando si utilizza la Kinesis Client Library.

La causa più comune di record ignorati è un'eccezione non gestita generata da `processRecords`. La Kinesis Client Library (KCL) si basa sul codice `processRecords` per gestire eventuali eccezioni generate dall'elaborazione dei record di dati. Qualsiasi eccezione generata da `processRecords` viene assorbita dalla KCL. Per evitare infiniti tentativi su un errore ricorrente, la KCL non invierà nuovamente batch di record elaborati al momento dell'eccezione. La KCL quindi chiama `processRecords` per i successivi batch di record di dati senza riavviare il processore del record. Ciò avviene nelle applicazioni dei consumatori che osservano i record ignorati. Per impedire che i record vengano ignorati, è necessario gestire tutte le eccezioni all'interno di `processRecords` in modo appropriato.

I record che appartengono allo stesso frammento vengono elaborati da diversi processori di record contemporaneamente

Per qualsiasi applicazione Kinesis Client Library (KCL) in esecuzione, una partizione ha un solo proprietario. Tuttavia, più processori di record possono elaborare temporaneamente lo stesso shard. Se un'istanza di lavoro perde la connettività di rete, KCL presume che il lavoratore irraggiungibile non stia più elaborando i record dopo la scadenza del tempo di failover e ordina ad altre istanze di lavoro di subentrare. Per un breve periodo, i nuovi processori record e i processori record dei lavoratori irraggiungibili possono elaborare i dati dello stesso shard.

Imposta un tempo di failover appropriato per la tua applicazione. Per le applicazioni a bassa latenza, l'impostazione predefinita di 10 secondi può rappresentare il tempo massimo che si desidera attendere. Tuttavia, nel caso in cui si prevedano problemi di connettività, ad esempio chiamate in aree geografiche dove la connettività potrebbe andare perduta con maggiore frequenza, questo numero può risultare troppo basso.

L'applicazione deve anticipare e gestire questo scenario, soprattutto perché la connettività di rete viene in genere ripristinata per il lavoratore precedentemente non raggiungibile. Se un record ha un processore shard assunto da un altro processore del record, è necessario gestire i seguenti due casi per eseguire un arresto regolare:

1. Una volta completata la chiamata corrente a`processRecords`, KCL richiama il metodo di spegnimento sul registratore indicando il motivo dello spegnimento «ZOMBIE». Si prevede che i tuoi processori record puliscano tutte le risorse in modo appropriato e quindi escano.
2. Quando provi a fare il checkpoint da un worker "zombie", la KCL genera `ShutdownException`. Dopo aver ricevuto questa eccezione, il tuo codice dovrebbe uscire in modo pulito dal metodo corrente.

Per ulteriori informazioni, consulta Gestisci i record duplicati.

L'applicazione consumer sta leggendo a una velocità inferiore al previsto

I motivi più comuni per cui il rendimento di lettura è più lento del previsto sono i seguenti:

1. Molteplici applicazioni consumatori hanno letture totali che superano i limiti per shard. Per ulteriori informazioni, consulta Quote e limiti. In questo caso, è possibile aumentare il numero di partizioni nel flusso di dati Kinesis.
2. Il limite che specifica il numero massimo di GetRecords per chiamata potrebbe essere stato configurato con un valore basso. Se utilizzi KCL, potresti aver configurato il lavoratore con un valore basso per la proprietà `maxRecords`. In generale, consigliamo di utilizzare i valori predefiniti del sistema per questa proprietà.
3. La logica all'interno della chiamata `processRecords` potrebbe richiedere più tempo del previsto per una serie di possibili motivi; la logica potrebbe essere intenso traffico della CPU, blocco I/O o collo di bottiglia sulla sincronizzazione. Per verificare se ciò è vero, il test esegue processori di record vuoti e confronta il rendimento di lettura. Per informazioni su come gestire i dati in entrata, consulta Usa il resharding, lo scaling e l'elaborazione parallela per modificare il numero di shard.

Se disponi di una sola applicazione consumer, puoi sempre leggere almeno due volte più velocemente della velocità di inserimento. Questo perché puoi scrivere fino a 1.000 record al secondo per scritture, con una velocità massima totale di scrittura dei dati pari a 1 MB al secondo (comprese le chiavi di partizioni). Ogni partizione può supportare fino a 5 transazioni al secondo, con una velocità di lettura totale massima di 2 MB al secondo. Considera che ogni lettura (chiamata GetRecords) ottiene un batch di record. La dimensione dei dati restituiti da GetRecords varia a seconda dell'utilizzo dello shard. La dimensione massima di dati che GetRecords può restituire è 10 MB. Se una chiamata restituisce tale limite, le chiamate successive effettuate entro i successivi 5 secondi generano un`ProvisionedThroughputExceededException`.

GetRecords restituisce un array di record vuoto anche quando ci sono dati nel flusso

Consumare o ottenere record è un modello di pull. Ci si aspetta che gli sviluppatori effettuino chiamate GetRecordsa ciclo continuo senza back-off. Ogni chiamata a GetRecords inoltre restituisce un valore `ShardIterator` che deve essere usato nella prossima iterazione del ciclo.

L'operazione GetRecords non si blocca. Al contrario, ritorna immediatamente; con uno dei record di dati pertinenti o con un elemento `Records` vuoto. Un elemento `Records` vuoto viene restituito a due condizioni:

1. Non ci sono più dati al momento nello shard.
2. Non ci sono dati vicino alla parte dello shard puntato dal `ShardIterator`.

Quest'ultima condizione è lieve, ma è un compromesso di progettazione necessario per evitare tempi di ricerca illimitati (latenza) per il recupero dei record. Pertanto, l'applicazione che consuma il flusso dovrebbe eseguire il ciclo e chiamare GetRecords nella gestione dei record vuoti normalmente.

In uno scenario di produzione, l'unica volta in cui il ciclo continuo deve essere chiuso è quando il valore `NextShardIterator` è `NULL`. Quando `NextShardIterator` è `NULL`, significa che l'attuale shard è stato chiuso e il valore `ShardIterator` punterebbe altrimenti oltre l'ultimo record. Se l'applicazione che consuma non chiama mai SplitShard oMergeShards, lo shard rimane aperto e le chiamate a GetRecords non restituiranno mai un valore `NextShardIterator` che è `NULL`.

Se utilizzi la Kinesis Client Library (KCL), il modello di consumo precedente viene riassunto automaticamente. Questo include la gestione automatica di un set di shard che cambia in modo dinamico. Con la KCL, lo sviluppatore fornisce solo la logica per elaborare i record in entrata. Questo è possibile perché la libreria genera continue chiamate ai GetRecords per te.

Lo shard iterator scade in modo imprevisto

Un nuovo iteratore shard viene restituito da ogni richiesta GetRecords (come `NextShardIterator`), che utilizzerai nella prossima richiesta GetRecords (come `ShardIterator`). Di solito, questo iteratore shard non scade prima dell'uso. Tuttavia, è possibile che gli iteratori shard scadano perché non hai chiamato GetRecords per più di 5 minuti o perché hai eseguito un riavvio della tua applicazione consumer.

Se lo shard iterator scade immediatamente prima di poterlo utilizzare, ciò potrebbe indicare che la tabella DynamoDB utilizzata da Kinesis non ha una capacità sufficiente per archiviare i dati del lease. Questa situazione è più probabile se disponi di un numero elevato di shard. Per risolvere il problema, aumenta la capacità di scrittura assegnata alla tabella shard. Per ulteriori informazioni, consulta Utilizza una tabella di leasing per tenere traccia degli shard elaborati dall'applicazione consumer KCL.

L'elaborazione dei record dei consumatori è in ritardo

Per la maggior parte dei casi d'uso, le applicazioni leggono i dati più recenti dal flusso. In alcune circostanze, le letture dei consumatori possono rimanere indietro, il che potrebbe non essere desiderato. Dopo aver identificato il punto in cui i consumatori stanno leggendo, guarda i motivi più comuni per cui i consumatori restano indietro.

Inizia con il parametro `GetRecords.IteratorAgeMilliseconds`, che monitora la posizione di lettura in tutti gli shard e i consumatori nel flusso. Nota che se l'età di un iteratore supera il 50% del periodo di conservazione (per impostazione predefinita 24 ore, configurabile fino a 7 giorni), sussiste il rischio di perdita di dati a causa della scadenza del record. Una rapida soluzione di ripiego consiste nell'aumentare il periodo di conservazione. Ciò interrompe la perdita di dati importanti man mano che si risolve il problema. Per ulteriori informazioni, consulta Monitoraggio del servizio flusso di dati Amazon Kinesis con Amazon CloudWatch. Successivamente, identifica il ritardo con cui la tua applicazione consumer sta leggendo ogni shard utilizzando una CloudWatch metrica personalizzata emessa dalla Kinesis Client Library (KCL),. `MillisBehindLatest` Per ulteriori informazioni, consulta Monitoraggio della Kinesis Client Library con Amazon CloudWatch.

Ecco i motivi più comuni per cui i consumatori possono rimanere indietro:

- Grandi aumenti improvvisi di `GetRecords.IteratorAgeMilliseconds` o `MillisBehindLatest` in genere indicano un problema transitorio, come i guasti delle operazioni API su un'applicazione downstream. Esamina questi aumenti improvvisi se una delle metriche mostra costantemente questo comportamento.
- Un aumento graduale di questi parametri indica che un consumer non è al passo con lo streaming perché non sta elaborando i record abbastanza velocemente. Le cause principali più comuni di questo comportamento sono risorse fisiche insufficienti o logica di elaborazione dei record che non è stata ridimensionata con un aumento del rendimento del flusso. Puoi verificare questo comportamento esaminando le altre CloudWatch metriche personalizzate emesse da KCL associate all'`processTask`operazione, tra cui`RecordProcessor.processRecords.Time`, e. `Success` `RecordsProcessed`

- Se ottieni un aumento del parametro `processRecords.Time` correlato con l'aumento del rendimento, devi analizzare la logica di elaborazione dei record per identificare il motivo per cui non si ridimensiona con il rendimento aumentato.
- Se ottieni un aumento per i valori `processRecords.Time` che non sono correlati con un maggiore rendimento, controlla se stai effettuando chiamate di blocco nel percorso critico, che sono spesso causa di rallentamenti nell'elaborazione dei record. Un approccio alternativo è aumentare il parallelismo aumentando il numero di shard. Infine, verifica di disporre di una quantità adeguata di risorse fisiche (memoria, utilizzo della CPU, tra le altre) sui nodi di elaborazione sottostanti durante i picchi di domanda.

Errore di autorizzazione della chiave KMS non autorizzata

Questo errore si verifica quando un'applicazione consumer legge da un flusso crittografato senza autorizzazioni sulla chiave. AWS KMS Per assegnare le autorizzazioni a un'applicazione per accedere a una chiave KMS, consulta Utilizzo delle policy della chiave in AWS KMS e Utilizzo delle policy IAM con AWS KMS.

DynamoDbException: il percorso del documento fornito nell'espressione di aggiornamento non è valido per l'aggiornamento

Quando si utilizza KCL 3.x con AWS SDK per Java le versioni da 2.27.19 a 2.27.23, è possibile riscontrare la seguente eccezione DynamoDB:

«software.amazon.awssdk.services.dynamodb.model. DynamoDbException: Il percorso del documento fornito nell'espressione di aggiornamento non è valido per l'aggiornamento (Service:, Status Code: 400, Request ID: xxx)» DynamoDb

Questo errore si verifica a causa di un problema noto AWS SDK per Java che riguarda la tabella di metadati DynamoDB gestita da KCL 3.x. Il problema è stato introdotto nella versione 2.27.19 e riguarda tutte le versioni fino alla 2.27.23. Il problema è stato risolto nella versione 2.27.24. AWS SDK per Java Per prestazioni e stabilità ottimali, consigliamo l'aggiornamento alla versione 2.28.0 o successiva.

Risolvi altri problemi comuni per i consumatori

- Perché il trigger del flusso di dati Kinesis non è in grado di richiamare la mia funzione Lambda?

- Come posso rilevare e risolvere le ReadProvisionedThroughputExceeded eccezioni in Kinesis Data Streams?
- Perché riscontro problemi di latenza elevata con il flusso di dati Kinesis?
- Perché il mio flusso di dati Kinesis restituisce un errore interno del server 500?
- Come posso risolvere un'applicazione KCL bloccata per il flusso di dati Kinesis?
- Posso usare diverse applicazioni Amazon Kinesis Client Library con la stessa tabella Amazon DynamoDB?

Ottimizza i consumatori di Amazon Kinesis Data Streams

Puoi ottimizzare ulteriormente il tuo consumatore Amazon Kinesis Data Streams in base al comportamento specifico che vedi.

Consulta i seguenti argomenti per identificare le soluzioni.

Argomenti

- Migliora l'elaborazione a bassa latenza
- Elabora dati serializzati utilizzando AWS Lambda Amazon Kinesis Producer Library
- Usa il resharding, lo scaling e l'elaborazione parallela per modificare il numero di shard
- Gestisci i record duplicati
- Gestisci l'avvio, lo spegnimento e la limitazione

Migliora l'elaborazione a bassa latenza

Il ritardo di propagazione è definito come la end-to-end latenza dal momento in cui un record viene scritto nello stream fino a quando non viene letto da un'applicazione consumer. Il ritardo varia in base a una serie di fattori, ma è interessato principalmente dall'intervallo di polling delle applicazioni di consumo.

Per la maggior parte delle applicazioni, ti consigliamo di eseguire il polling di ogni shard una volta al secondo per applicazione. In questo modo è possibile avere più applicazioni di consumo che elaborano un flusso simultaneamente senza considerare Flusso di dati Amazon Kinesis, senza raggiungere i limiti di 5 chiamate `GetRecords` al secondo. Inoltre, l'elaborazione di batch di dati di dimensioni maggiori tende a essere più efficiente nel ridurre la rete e altre latenze downstream nella tua applicazione.

I valori predefiniti della KCL sono impostati per seguire la best practice di eseguire il polling ogni secondo. Questi valori predefiniti comportano ritardi di propagazione medi che generalmente sono inferiori a 1 secondo.

I record del flusso di dati Kinesis sono disponibili per essere letti subito dopo essere stati scritti. Ci sono alcuni casi d'uso in cui è necessario sfruttare questo vantaggio e in cui è richiesto il consumo dei dati dal flusso non appena disponibili. È possibile ridurre in modo significativo il ritardo di propagazione sostituendo le impostazioni di default della KCL più di frequente, come mostrato nei seguenti esempi.

Codice di configurazione della KCL in Java:

```
kinesisClientLibConfiguration = new
        KinesisClientLibConfiguration(applicationName,
        streamName,
        credentialsProvider,

 workerId).withInitialPositionInStream(initialPositionInStream).withIdleTimeBetweenReadsInMilli
```

Impostazione di file delle proprietà per la KCL in Python e Ruby:

```
idleTimeBetweenReadsInMillis = 250
```

> **Note**
>
> Dato che il flusso di dati Kinesis ha un limite di 5 chiamate `GetRecords` al secondo, per partizione, l'impostazione della proprietà `idleTimeBetweenReadsInMillis` con un valore inferiore ai 200 millisecondi può comportare che la tua applicazione osservi l'eccezione `ProvisionedThroughputExceededException`. Un numero eccessivo di queste eccezioni può comportare backoff esponenziali e, pertanto, può causare latenze impreviste e significative nell'elaborazione. Se si imposta questa proprietà in modo tale che il valore sia pari o appena superiore a 200 millisecondi e da avere più di un'applicazione di elaborazione, noterai un throttling simile.

Elabora dati serializzati utilizzando AWS Lambda Amazon Kinesis Producer Library

La Amazon Kinesis Producer Library (KPL) aggrega piccoli record formattati dall'utente in record più grandi fino a 1 MB per utilizzare meglio il throughput di Amazon Kinesis Data Streams. Sebbene KCL per Java supporti la disaggregazione di questi record, è necessario utilizzare un modulo speciale per disaggregare i record quando si utilizza come consumatore dei propri stream. AWS Lambda Puoi ottenere il codice di progetto e le istruzioni necessarie da GitHub Amazon Kinesis Producer Library Deaggregation Modules for Lambda. AWS I componenti di questo progetto ti danno la possibilità di elaborare dati serializzati KPL all'interno AWS Lambda, in Java, Node.js e Python. Questi componenti possono anche essere utilizzati come parte di un'applicazione KCL multilingue.

Usa il resharding, lo scaling e l'elaborazione parallela per modificare il numero di shard

Il resharding ti consente di aumentare o diminuire il numero di shard in un flusso in modo da adattarsi alle variazioni di velocità nella circolazione di dati nel flusso. In genere il resharding viene eseguito da un'applicazione amministrativa che monitora i parametri di gestione di dati negli shard. Sebbene la KCL stessa non avvii le operazioni di ripartizionamento, è progettata per adattarsi alle variazioni nel numero di partizioni causate dal ripartizionamento.

Come indicato in Utilizza una tabella di leasing per tenere traccia degli shard elaborati dall'applicazione consumer KCL, la KCL monitora le partizioni nel flusso utilizzando una tabella di Amazon DynamoDB. Quando nuove partizioni vengono create come risultato del ripartizionamento, la KCL rileva le nuove partizioni e completa le nuove righe nella tabella. I lavoratori trovano automaticamente i nuovi shard e creano processori per gestire i dati provenienti da questi shard. Anche la KCL distribuisce le partizioni nel flusso tra tutti i worker e i processori di record disponibili.

La KCL garantisce che i dati esistenti nelle partizioni prima del ripartizionamento vengano elaborati per primi. Dopo che i dati sono stati elaborati, i dati dal nuovo shard vengono inviati a processori di record. In questo modo, la KCL conserva l'ordine in cui i record di dati sono stati aggiunti al flusso per una determinata chiave di partizione.

Esempio: ripartizione, scalabilità ed elaborazione parallela

L'esempio seguente spiega come la KCL ti aiuta a gestire il dimensionamento e il ripartizionamento:

- Ad esempio, se l'applicazione è in esecuzione su un' EC2 istanza e sta elaborando un flusso di dati Kinesis con quattro shard. Questa istanza dispone di un worker della KCL e di quattro processori

di record (un processore di record per ogni partizione). Questi quattro processori di record sono eseguiti in parallelo all'interno dello stesso processo.

- Di seguito, se si dimensiona l'applicazione per utilizzare un'altra istanza, si avranno due istanze per l'elaborazione di un flusso con quattro shard. Quando il worker della KCL si avvia nella seconda istanza, esegue un bilanciamento del carico con la prima istanza in modo che ogni istanza sia in grado di elaborare due partizioni.
- Se si decide di frazionare i quattro shard in cinque shard. La KCL coordina nuovamente l'elaborazione su più istanze; un'istanza elabora tre partizioni, mentre l'altra istanza elabora due partizioni. Un coordinamento simile si verifica quando si fondono shard.

Di solito, quando si utilizza la KCL, è necessario accertarsi che il numero di istanze non superi il numero di partizioni (tranne nel caso in cui si desideri rimanere in attesa di errori) Ogni partizione viene elaborata da esattamente un worker della KCL e ha esattamente un processore di record corrispondente, pertanto non sono mai necessarie istanze multiple per elaborare una partizione. Tuttavia, un lavoratore è in grado di elaborare qualsiasi numero di shard, pertanto non è un problema se il numero di shard supera il numero di istanze.

Per aumentare l'elaborazione nella tua applicazione, è consigliabile testare una combinazione dei seguenti approcci:

- Aumentare la dimensione dell'istanza (dato che tutti i processori di record sono eseguiti in parallelo all'interno di un processo)
- Aumentare il numero di istanze fino al numero massimo di shard aperti (dato che gli shard possono essere elaborati in modo indipendente)
- Aumentare il numero di shard (aumentando di conseguenza il livello di parallelismo)

Si noti che è possibile utilizzare il dimensionamento automatico per dimensionare automaticamente le istanze in base ai parametri appropriati. Per ulteriori informazioni, consulta la Amazon EC2 Auto Scaling User Guide.

Quando il resharding aumenta il numero di shard nello stream, il corrispondente aumento del numero di record processor aumenta il carico sulle EC2 istanze che li ospitano. Se le istanze fanno parte di un gruppo Auto Scaling e il carico aumenta a sufficienza, il gruppo Auto Scaling aggiunge più istanze per gestire l'aumento del carico. È necessario configurare le istanze per avviare la tua applicazione del flusso di dati Amazon Kinesis all'avvio, in modo che i worker e i processori di record aggiuntivi diventino immediatamente attivi nella nuova istanza.

Per ulteriori informazioni sul resharding, consulta Condividi nuovamente uno stream.

Gestisci i record duplicati

Ci sono due motivi principali per cui i record possono essere distribuiti più di una volta alla tua applicazione del flusso di dati Amazon Kinesis: i nuovi tentativi del producer e i nuovi tentativi del consumer. La tua applicazione deve prevedere e gestire in modo appropriato l'elaborazione di singoli record più volte.

Il produttore riprova

Considera un producer in cui si verifica un timeout della rete dopo una chiamata a `PutRecord`, ma prima di poter ricevere una conferma da Flusso di dati Amazon Kinesis . Il producer non può essere certo se il record è stato distribuito a . Considerando che ogni record è importante per l'applicazione, il producer sarebbe stato scritto per riprovare la chiamata con gli stessi dati. Se entrambe le chiamate a `PutRecord` sugli stessi dati sono state inviate correttamente al flusso di dati Kinesis saranno presenti due record del flusso di dati Kinesis. Anche se i due record dispongono di dati identici, hanno anche numeri di sequenza univoci. Le applicazioni che richiedono garanzie rigorose devono incorporare una chiave primaria nel record per rimuovere i duplicati più avanti nel corso dell'elaborazione. Si noti che il numero di duplicati dovuti a nuovi tentativi del producer è solitamente basso rispetto al numero di duplicati dovuti a nuovi tentativi del consumer.

> **Note**
>
> Se utilizzi l' AWS SDK`PutRecord`, scopri il comportamento di SDK Retry nella guida per l'utente di AWS SDKs and Tools.

Ritentativi dei consumatori

I nuovi tentativi del consumer (applicazione di elaborazione dei dati) si verificano quando si riavviano i processori di record. I processori di record per lo stesso shard vengono riavviati nei seguenti casi:

1. Il lavoratore viene terminato in modo inaspettato
2. Le istanze del lavoratore vengono aggiunte o rimosse
3. Gli shard sono fusi o frazionati
4. L'applicazione viene distribuita

In tutti questi casi, la mappatura shards-to-worker-to -record-processor viene continuamente aggiornata per l'elaborazione del bilanciamento del carico. I processori di shard migrati ad altre istanze riavviano l'elaborazione di record dall'ultimo checkpoint. Ciò comporta un'elaborazione di record duplicati, come illustrato nell'esempio seguente. Per ulteriori informazioni sul bilanciamento del carico, consulta Usa il resharding, lo scaling e l'elaborazione parallela per modificare il numero di shard.

Esempio: un cliente riprova con conseguente riconsegna dei record

In questo esempio, si dispone di un'applicazione che legge i record da un flusso in modo continuo, aggrega i record in un file locale e carica il file in Amazon S3. Per semplicità, supponiamo di avere solo uno shard e un lavoratore che elabora lo shard. Considera il seguente esempio di sequenza di eventi, supponendo che l'ultimo checkpoint si è verificato al numero di record 10.000:

1. Un lavoratore legge il batch successivo di record dallo shard, i record da 10.001 a 20.000.
2. Di seguito, il lavoratore trasferisce il batch di record al processore di record associato.
3. Il processore di record aggrega i dati, crea un file Amazon S3 e carica correttamente il file in Amazon S3.
4. Il lavoratore viene terminato in modo inaspettato prima che possa verificarsi un nuovo checkpoint.
5. La applicazione, il lavoratore e il processore di record si riavviano.
6. A partire da questo momento, il lavoratore comincia a leggere dall'ultimo checkpoint eseguito correttamente, in questo caso 10.001.

Pertanto, i record 10.001-20.000 vengono consumati più di una volta.

Essere resilienti ai tentativi dei consumatori

Anche se è possibile che i record siano elaborati più di una volta, l'applicazione potrebbe voler presentare gli effetti collaterali come se i record fossero stati elaborati solo una volta (elaborazione idempotente). Le soluzioni a questo problema variano in complessità e accuratezza. Se la destinazione dei dati finali è in grado di gestire correttamente i duplicati, ti consigliamo di affidarti alla destinazione finale per ottenere l'elaborazione idempotente. Ad esempio, con Opensearch è possibile utilizzare una combinazione di controllo delle versioni e univoche IDs per evitare elaborazioni duplicate.

Nell'applicazione di esempio nella sezione precedente, l'applicazione legge in modo continuo i record da un flusso, aggrega i record in un file locale e carica il file in Amazon S3. Come indicato, i record da

10.001 a 20.000 vengono consumati più di una volta e ciò comporta la presenza di più file Amazon S3 con gli stessi dati. Un modo per mitigare i duplicati in questo esempio è di assicurarsi che nella fase 3 sia utilizzato il seguente schema:

1. Il processore di record utilizza un numero fisso di record per file Amazon S3, ad esempio 5.000.
2. Il nome di file utilizza questo schema: prefisso di Amazon S3, ID della partizione e `First-Sequence-Num`. In questo caso, potrebbe essere qualcosa di simile a `sample-shard000001-10001`.
3. Una volta caricato il file Amazon S3, crea un checkpoint specificando `Last-Sequence-Num`. In questo caso, il checkpoint verrebbe eseguito al numero di record 15.000.

Con questo schema, anche se i record vengono elaborati più di una volta, il file Amazon S3 risultante ha lo stesso nome e gli stessi dati. I nuovi tentativi hanno come risultato esclusivamente la scrittura degli stessi dati nello stesso file più di una volta.

Nel caso di un'operazione di reshard, il numero di record rimasti nello shard potrebbe essere inferiore al numero fisso desiderato. In questo caso, il tuo metodo `shutdown()` deve scaricare il file ad Amazon S3 ed eseguire il checkpoint nell'ultimo numero di sequenza. Il suddetto schema è compatibile anche con le operazioni di reshard.

Gestisci l'avvio, lo spegnimento e la limitazione

Di seguito sono elencate alcune considerazioni aggiuntive da integrare nella progettazione della tua applicazione del flusso di dati Amazon Kinesis.

Argomenti

- Avvia produttori e consumatori di dati
- Chiudi un'applicazione Amazon Kinesis Data Streams
- Limitazione della lettura

Avvia produttori e consumatori di dati

Per impostazione predefinita, la KCL inizia a leggere i record dall'estremità del flusso, che è il record aggiunto più di recente. In questa configurazione, se un'applicazione produttrice di dati aggiunge record al flusso prima che i processori di record riceventi siano in esecuzione, i record non vengono letti dai processori di record dopo l'avvio.

Per modificare il comportamento dei processori di record in modo che leggano sempre i dati dall'inizio del flusso, imposta il seguente valore nel file proprietà per la tua applicazione del flusso di dati Amazon Kinesis:

```
initialPositionInStream = TRIM_HORIZON
```

Per impostazione predefinita, Flusso di dati Amazon Kinesis archivia tutti i dati per 24 ore. Supporta inoltre la conservazione prolungata fino a 7 giorni e la conservazione a lungo termine fino a 365 giorni. Questo intervallo di tempo viene chiamato il periodo di conservazione. Se si imposta la posizione iniziale in `TRIM_HORIZON` il processore di record verrà avviato con i dati meno recenti nel flusso, secondo quanto definito dal periodo di conservazione. Anche con l'impostazione `TRIM_HORIZON`, qualora un processore di record venisse avviato dopo che è trascorso un intervallo di tempo maggiore rispetto al periodo di conversazione alcuni record nel flusso non sarebbero più disponibili. Per questo motivo, dovreste sempre fare in modo che le applicazioni consumer leggano lo stream e utilizzino la CloudWatch metrica `GetRecords.IteratorAgeMilliseconds` per monitorare che le applicazioni stiano tenendo il passo con i dati in entrata.

In alcuni scenari, perdere i primi record nel flusso potrebbe essere una buona opzione per i processori di record. Ad esempio, potreste eseguire alcuni record iniziali attraverso lo stream per verificare che lo stream funzioni end-to-end come previsto. Dopo avere eseguito questa verifica iniziale, avvieresti i tuoi lavoratori e cominceresti a integrare i dati di produzione nel flusso.

Per ulteriori informazioni sull'impostazione `TRIM_HORIZON`, consulta Usa gli iteratori shard.

Chiudi un'applicazione Amazon Kinesis Data Streams

Quando l'applicazione Amazon Kinesis Data Streams ha completato l'attività prevista, è necessario chiuderla EC2 chiudendo le istanze su cui è in esecuzione. Puoi terminare le istanze utilizzando la AWS Management Console o la AWS CLI.

Dopo aver eseguito l'arresto della tua applicazione del flusso di dati Amazon Kinesis, è necessario eliminare la tabella Amazon DynamoDB che la KCL ha utilizzato per monitorare lo stato dell'applicazione.

Limitazione della lettura

Il throughput di un flusso viene assegnato a livello di shard. Ogni partizione ha una velocità di trasmissione effettiva di lettura fino a 5 transazioni al secondo, fino a una velocità di lettura totale massima di 2 MB al secondo. Se un'applicazione (o un gruppo di applicazioni che operano nello

stesso flusso) prova a ottenere i dati da una partizione una velocità superiore, il flusso di dati Kinesis applica la limitazione delle operazioni Get corrispondenti.

In un'applicazione del flusso di dati Amazon Kinesis, se un processore di record sta elaborando dati più rapidamente rispetto al limite, ad esempio nel caso di un failover, si verifica una limitazione. Dato che KCL gestisce le interazioni tra l'applicazione e il flusso di dati Kinesis, si verificano eccezioni di limitazione nel codice della KCL anziché nel codice dell'applicazione. Tuttavia, dato che la KCL registra queste eccezioni, è possibile vederle nei log.

Se ritieni che la tua applicazione sia sottoposta a throttling in modo costante, è consigliabile considerare un aumento del numero di shard per il flusso.

Monitoraggio dei flussi di dati Kinesis

È possibile monitorare i flussi di dati nel flusso di dati Amazon Kinesis utilizzando le seguenti funzionalità:

- CloudWatch parametri: il flusso di dati Kinesis invia parametri CloudWatch Amazon personalizzati con monitoraggio dettagliato per ogni flusso.
- Agente Kinesis: l'agente Kinesis pubblica CloudWatch parametri personalizzati che aiutano a valutare se l'agente funziona come previsto.
- Registrazione API: il flusso di dati Kinesis utilizza AWS CloudTrail per registrare le chiamate alle API e archiviare i dati in un bucket Amazon S3.
- Kinesis Client Library: la Kinesis Client Library (KCL) fornisce parametri per partizione, worker e applicazione KCL.
- Kinesis Producer Library: la Amazon Kinesis Kinesis fornisce parametri per partizione, worker e applicazione KPL.

Per ulteriori informazioni sui problemi di monitoraggio, domande e risoluzione dei problemi più comuni, consulta:

- Quali parametri devo utilizzare per monitorare e risolvere i problemi del flusso di dati Kinesis?
- Perché il IteratorAgeMilliseconds valore di Kinesis Data Streams continua ad aumentare?

Monitoraggio del servizio flusso di dati Amazon Kinesis con Amazon CloudWatch

Il flusso di dati Amazon Kinesis si integra con Amazon Kinesis in modo da poter raccogliere CloudWatch, visualizzare e analizzare parametri per i flussi CloudWatch di dati Kinesis. Ad esempio, per tenere traccia dell'utilizzo degli shard, è possibile monitorare i parametri `IncomingBytes` e `OutgoingBytes` e confrontarli con il numero di shard nel flusso.

I parametri di flusso e i parametri configurati vengono raccolti automaticamente e inviati ogni minuto. CloudWatch I parametri sono conservati per due settimane; dopo tale periodo, i dati vengono eliminati.

La tabella seguente descrive un monitoraggio di base a livello di flusso e un monitoraggio avanzato a livello di partizione per i flussi di dati Kinesis.

Tipo	Descrizione
Monitoraggio di base (a livello di flusso)	I dati a livello di flusso sono inviati automatic amente ogni minuto, senza alcun costo aggiuntivo.
Monitoraggio avanzato (a livello di shard)	I dati a livello di shard sono inviati ogni minuto, con un costo aggiuntivo. Per ottenere questo livello di dati, è necessario impostarlo specifica mente per il flusso utilizzando l'operazione EnableEnhancedMonitoring. Per ulteriori informazioni sui prezzi, consulta la pagina CloudWatch del prodotto Amazon.

Parametri e dimensioni del flusso di dati Amazon Kinesis

Il flusso di dati Kinesis invia i CloudWatch parametri a due livelli: il livello del flusso e, facoltativamente, il livello della partizione. Le metriche a livello di stream sono destinate ai casi d'uso di monitoraggio più comuni in condizioni normali. I parametri a livello di shard sono per attività di monitoraggio specifiche, solitamente correlate alla risoluzione dei problemi, e sono abilitati utilizzando l'operazione EnableEnhancedMonitoring.

Per una spiegazione delle statistiche raccolte dalle CloudWatch metriche, consulta CloudWatch Statistics in the Amazon CloudWatch User Guide.

Argomenti

- Parametri di base a livello di flusso
- Parametri a livello di partizione migliorati
- Parametri del flusso di dati Amazon Kinesis
- Parametri del flusso di dati Amazon Kinesis consigliati

Parametri di base a livello di flusso

Il namespace `AWS/Kinesis` include i parametri a livello di flusso descritti di seguito.

Il flusso di dati Kinesis invia questi parametri a livello di flusso a intervalli di un minuto CloudWatch . Tali parametri sono sempre disponibili.

Parametro	Descrizione
`GetRecords.Bytes`	Il numero di byte recuperati dal flusso Kinesis misurati durante il periodo di tempo specificato. Le statistic he Minimo, Massimo e Media rappresentano i byte in un'unica operazione `GetRecords` per il flusso nel periodo di tempo specificato. Nome parametro a livello di shard: `OutgoingBytes` . Dimensioni: StreamName Statistiche: minimo, massimo, media, somma, esempi Unità: byte
`GetRecords.IteratorAge`	Questa metrica non è più in uso. Utilizza `GetRecord s.IteratorAgeMilliseconds` .
`GetRecords.Iterato rAgeMilliseconds`	L'età dell'ultimo record in tutte le chiamate `GetRecord s` effettuate su un flusso Kinesis, misurate durante il periodo di tempo specificato. L'età è la differenza tra l'ora corrente e il momento in cui l'ultimo record della chiamata `GetRecords` è stato scritto nel flusso. Le statistiche Minimo e Massimo possono essere utilizzate per tenere traccia dell'attività delle applicazioni consumer di Kinesis. Il valore zero indica che i record in fase di lettura sono totalmente assorbiti dal flusso. Nome parametro a livello di shard: `IteratorA geMilliseconds` . Dimensioni: StreamName

Parametro	Descrizione
	Statistiche: minimo, massimo, media, esempi Unità: millisecondi
`GetRecords.Latency`	Il tempo necessario per operazione `GetRecords` , misurato durante il periodo specificato. Dimensioni: StreamName Statistiche: minimo, massimo, media Unità: millisecondi
`GetRecords.Records`	Il numero di record recuperati dallo shard, misurati durante il periodo di tempo specificato. Le statistiche Minimo, Massimo e Media rappresentano i record in un'unica operazione `GetRecords` per il flusso nel periodo di tempo specificato. Nome parametro a livello di shard: `OutgoingRecords` . Dimensioni: StreamName Statistiche: minimo, massimo, media, somma, esempi Unità: numero
`GetRecords.Success`	Il numero di operazioni `GetRecords` riuscite per flusso, misurate durante il periodo di tempo specificato. Dimensioni: StreamName Statistiche: media, somma, esempi Unità: numero

Parametro	Descrizione
`IncomingBytes`	Il numero di byte correttamente inseriti nel flusso Kinesis durante il periodo di tempo specificato. Questo parametro include i byte delle operazioni `PutRecord` e `PutRecords` . Le statistiche Minimo, Massimo e Media rappresentano i byte in un'unica operazione put per il flusso nel periodo di tempo specificato. Nome parametro a livello di shard: `IncomingBytes` . Dimensioni: StreamName Statistiche: minimo, massimo, media, somma, esempi Unità: byte
`IncomingRecords`	Il numero di record inseriti correttamente nel flusso Kinesis durante il periodo di tempo specificato. Questo parametro include il numero di record delle operazioni `PutRecord` e `PutRecords` . Le statistiche Minimo, Massimo e Media rappresentano i record in un'unica operazione put per il flusso nel periodo di tempo specifica to. Nome parametro a livello di shard: `IncomingRecords` . Dimensioni: StreamName Statistiche: minimo, massimo, media, somma, esempi Unità: numero

Parametro	Descrizione
PutRecord.Bytes	Il numero di byte inseriti nel flusso Kinesis mediante l'operazione PutRecord durante il periodo di tempo specificato. Dimensioni: StreamName Statistiche: minimo, massimo, media, somma, esempi Unità: byte
PutRecord.Latency	Il tempo necessario per operazione PutRecord , misurato durante il periodo specificato. Dimensioni: StreamName Statistiche: minimo, massimo, media Unità: millisecondi
PutRecord.Success	Il numero di operazioni PutRecord riuscite per flusso Kinesis misurate durante il periodo di tempo specificato. La media riflette la percentuale di scritture riuscite in un flusso. Dimensioni: StreamName Statistiche: media, somma, esempi Unità: numero
PutRecords.Bytes	Il numero di byte inseriti nel flusso Kinesis mediante l'operazione PutRecords durante il periodo di tempo specificato. Dimensioni: StreamName Statistiche: minimo, massimo, media, somma, esempi Unità: byte

Parametro	Descrizione
PutRecords.Latency	Il tempo necessario per operazione PutRecords , misurato durante il periodo specificato. Dimensioni: StreamName Statistiche: minimo, massimo, media Unità: millisecondi
PutRecords.Records	Questo parametro è obsoleto. Utilizza PutRecords.SuccessfulRecords . Dimensioni: StreamName Statistiche: minimo, massimo, media, somma, esempi Unità: numero
PutRecords.Success	Il numero di operazioni PutRecords con almeno un record riuscito, per flusso Kinesis, misurate durante il periodo di tempo specificato. Dimensioni: StreamName Statistiche: media, somma, esempi Unità: numero
PutRecords.TotalRecords	Il numero di record inviati in un'operazione PutRecords per il flusso di dati Kinesis misurata nel periodo di tempo specificato. Dimensioni: StreamName Statistiche: minimo, massimo, media, somma, esempi Unità: numero

Parametro	Descrizione
PutRecords.SuccessfulRecords	Il numero di record riusciti in un'operazione PutRecords per flusso di dati Kinesis, misurati durante il periodo di tempo specificato. Dimensioni: StreamName Statistiche: minimo, massimo, media, somma, esempi Unità: numero
PutRecords.FailedRecords	Il numero di record rifiutati a causa di errori interni in un'operazione PutRecords per il flusso di dati Kinesis, misurato nel periodo di tempo specificato. È probabile che si verifichino errori interni occasionali e che sia necessario riprovare. Dimensioni: StreamName Statistiche: minimo, massimo, media, somma, esempi Unità: numero
PutRecords.ThrottledRecords	Il numero di record rifiutati a causa della limitazione in un'operazione PutRecords per il flusso di dati Kinesis misurata nel periodo di tempo specificato. Dimensioni: StreamName Statistiche: minimo, massimo, media, somma, esempi Unità: numero

Parametro	Descrizione
`ReadProvisionedThroughputExceeded`	Il numero di chiamate `GetRecords` sottoposte a throttling per il flusso durante il periodo di tempo specificato. La statistica Media è quella più utilizzata per questo parametro. Quando il valore della statistica Minimo è 1, tutti i record sono stati sottoposti a throttling per il flusso durante il periodo di tempo specificato. Quando il valore della statistica Massimo è 0 (zero), nessun record è stato sottoposto a throttling per il flusso durante il periodo di tempo specificato. Nome parametro a livello di shard: `ReadProvisionedThroughputExceeded` . Dimensioni: StreamName Statistiche: minimo, massimo, media, somma, esempi Unità: numero
`SubscribeToShard.RateExceeded`	Questo parametro viene emesso quando un tentativo di nuovo abbonamento non va a buon fine perché è attivo un abbonamento dello stesso consumer oppure hai superato il numero di chiamate al secondo consentite per questa operazione. Dimensioni: StreamName, ConsumerName
`SubscribeToShard.Success`	Questa metrica registra se la SubscribeToShard sottoscrizione è stata stabilita con successo. L'abbonamento non dura più di 5 minuti. Pertanto, questa metrica viene emessa almeno una volta ogni 5 minuti. Dimensioni:, StreamName ConsumerName

Parametro	Descrizione
`SubscribeToShardEvent.Bytes`	Il numero di byte ricevuti dallo shard, misurati durante il periodo di tempo specificato. Le statistiche Minimo, Massimo e Media rappresentano i byte pubblicati in un singolo evento per il periodo di tempo specificato. Nome parametro a livello di shard: `OutgoingBytes` . Dimensioni: StreamName, ConsumerName Statistiche: minimo, massimo, media, somma, esempi Unità: byte
`SubscribeToShardEvent.MillisBehindLatest`	Il numero di millisecondi in cui i record letti provengono dall'estremità del flusso, a indicare il ritardo rispetto all'ora corrente del consumatore. Dimensioni:, StreamName ConsumerName Statistiche: minimo, massimo, media, esempi Unità: millisecondi
`SubscribeToShardEvent.Records`	Il numero di record ricevuti dallo shard, misurati durante il periodo di tempo specificato. Le statistiche Minimo, Massimo e Media rappresentano i record pubblicati in un singolo evento per il periodo di tempo specificato. Nome parametro a livello di shard: `OutgoingRecords` . Dimensioni: StreamName, ConsumerName Statistiche: minimo, massimo, media, somma, esempi Unità: numero

Parametro	Descrizione
`SubscribeToShardEvent.Success`	Questo parametro viene emesso ogni volta che un evento viene pubblicato correttamente. Viene emesso solo in presenza di un abbonamento attivo. Dimensioni: StreamName, ConsumerName Statistiche: minimo, massimo, media, somma, esempi Unità: numero
`WriteProvisionedThroughputExceeded`	Il numero di record rifiutati a causa del throttling per il flusso durante il periodo di tempo specificato. Questo parametro include il throttling delle operazioni `PutRecord` e `PutRecords`. La statistica Media è quella più utilizzata per questo parametro. Quando il valore della statistica Minimo non è 0 (zero), i record sono stati sottoposti a throttling per il flusso durante il periodo di tempo specificato. Quando il valore della statistica Massimo è 0 (zero), nessun record è stato sottoposto a throttling per il flusso durante il periodo di tempo specificato. Nome parametro a livello di shard: `WriteProvisionedThroughputExceeded`. Dimensioni: StreamName Statistiche: minimo, massimo, media, somma, esempi Unità: numero

Parametri a livello di partizione migliorati

Il namespace `AWS/Kinesis` include i parametri a livello di shard descritti di seguito.

Kinesis invia i parametri a livello di partizione a intervalli di un minuto. CloudWatch Ogni dimensione di parametro crea una CloudWatch metrica ed effettua circa 43.200 chiamate `PutMetricData` API al mese. Per impostazione predefinita, tali parametri non sono attivati. I parametri avanzati emessi da Kinesis comportano un addebito. Per ulteriori informazioni, consulta la pagina CloudWatch dei prezzi di Amazon alla voce Amazon CloudWatch Custom Metrics. I costi sono forniti per shard per parametro per mese.

Parametro	Descrizione
`IncomingBytes`	Il numero di byte inseriti nello shard durante il periodo di tempo specificato. Questo parametro include i byte delle operazioni `PutRecord` e `PutRecords` . Le statistic he Minimo, Massimo e Media rappresentano i byte in un'unica operazione put per lo shard nel periodo di tempo specificato. Nome parametro a livello di flusso: `IncomingBytes` . Dimensioni: StreamName, ShardId Statistiche: minimo, massimo, media, somma, esempi Unità: byte
`IncomingRecords`	Il numero di record inseriti nello shard durante il periodo di tempo specificato. Questo parametro include il numero di record delle operazioni `PutRecord` e `PutRecords` . Le statistiche Minimo, Massimo e Media rappresentano i record in un'unica operazione put per lo shard nel periodo di tempo specificato. Nome parametro a livello di flusso: `IncomingRecords` . Dimensioni: StreamName, ShardId Statistiche: minimo, massimo, media, somma, esempi Unità: numero

Parametro	Descrizione
IteratorAgeMilliseconds	L'età dell'ultimo record in tutte le chiamate GetRecords effettuate su uno shard, misurata durante il periodo di tempo specificato. L'età è la differenza tra l'ora corrente e il momento in cui l'ultimo record della chiamata GetRecords è stato scritto nel flusso. Le statistiche Minimo e Massimo possono essere utilizzate per tenere traccia dell'attività delle applicazioni consumer di Kinesis. Il valore 0 (zero) indica che i record in fase di lettura sono totalmente assorbiti dal flusso. Nome parametro a livello di flusso: GetRecords.IteratorAgeMilliseconds . Dimensioni: StreamName, ShardId Statistiche: minimo, massimo, media, esempi Unità: millisecondi
OutgoingBytes	Il numero di byte recuperati dallo shard, misurati durante il periodo di tempo specificato. Le statistiche Minimo, Massimo e Media rappresentano i byte restituiti in un'unica operazione GetRecords o pubblicati in un singolo evento SubscribeToShard per lo shard nel periodo di tempo specificato. Nome parametro a livello di flusso: GetRecords.Bytes . Dimensioni: StreamName, ShardId Statistiche: minimo, massimo, media, somma, esempi Unità: byte

Parametro	Descrizione
`OutgoingRecords`	Il numero di record recuperati dallo shard, misurati durante il periodo di tempo specificato. Le statistiche Minimo, Massimo e Media rappresentano i record restituiti in un'unica operazione `GetRecords` o pubblicati in un singolo evento `SubscribeToShard` per lo shard nel periodo di tempo specificato. Nome parametro a livello di flusso: `GetRecords.Records` . Dimensioni: StreamName, ShardId Statistiche: minimo, massimo, media, somma, esempi Unità: numero

Parametro	Descrizione
`ReadProvisionedThroughputExceeded`	Il numero di chiamate `GetRecords` sottoposte a throttling per lo shard durante il periodo di tempo specificato. Questo numero di eccezioni copre tutte le dimensioni dei seguenti limiti: 5 letture per shard per secondo o 2 MB per secondo per shard. La statistica Media è quella più utilizzata per questo parametro. Quando il valore della statistica Minimo è 1, tutti i record sono stati sottoposti a throttling per lo shard durante il periodo di tempo specificato. Quando il valore della statistica Massimo è 0 (zero), nessun record è stato sottoposto a throttling per lo shard durante il periodo di tempo specificato. Nome parametro a livello di flusso: `ReadProvisionedThroughputExceeded` . Dimensioni: StreamName, ShardId Statistiche: minimo, massimo, media, somma, esempi Unità: numero

Parametro	Descrizione
`WriteProvisionedThroughputExceeded`	Il numero di record rifiutati a causa del throttling dello shard durante il periodo di tempo specificato. Questo parametro include il throttling delle operazioni `PutRecord` e `PutRecords` e copre tutte le dimensioni dei seguenti limiti: 1.000 record per secondo per shard o 1 MB per secondo per shard. La statistica Media è quella più utilizzata per questo parametro. Quando il valore della statistica Minimo non è 0 (zero), i record sono stati sottoposti a throttling per lo shard durante il periodo di tempo specificato. Quando il valore della statistica Massimo è 0 (zero), nessun record è stato sottoposto a throttling per lo shard durante il periodo di tempo specificato. Nome parametro a livello di flusso: `WriteProvisionedThroughputExceeded` . Dimensioni: StreamName, ShardId Statistiche: minimo, massimo, media, somma, esempi Unità: numero

Parametri del flusso di dati Amazon Kinesis

Dimensione	Descrizione
`StreamName`	Il nome del flusso Kinesis. Tutte le statistiche disponibili vengono filtrate per `StreamName` .

Parametri del flusso di dati Amazon Kinesis consigliati

Diversi parametri del flusso di dati Amazon Kinesis potrebbero essere di particolare interesse per i clienti del flusso di dati Kinesis. Il seguente elenco fornisce i parametri consigliati e i loro utilizzi.

Parametro	Note per l'utilizzo
`GetRecords.IteratorAgeMilliseconds`	Monitora la posizione di lettura di tutti gli shard e i consumatori nel flusso. Se l'età di un iteratore supera il 50% del periodo di conservazione (per impostazione predefinita, 24 ore, configurabile fino a 7 giorni), c'è il rischio di una perdita di dati a causa della scadenza del record. È consigliabile utilizzare CloudWatch allarmi per la statistica Massimo, in modo che possano avvisarti in anticipo che questa perdita rappresenta un rischio. Per uno scenario di esempio che utilizza questo parametro, consulta L'elaborazione dei record dei consumatori è in ritardo.
`ReadProvisionedThroughputExceeded`	Quando l'elaborazione di record nel lato del consumer è in ritardo, talvolta è difficile sapere dov'è il collo di bottiglia. Utilizza questo parametro per determinare se le operazioni di lettura vengono sottoposte a throttling a causa del fatto che superano i tuoi limiti di throughput di lettura. La statistica Media è quella più utilizzata per questo parametro.
`WriteProvisionedThroughputExceeded`	Ha lo stesso scopo del parametro `ReadProvisionedThroughputExceeded` ma per il lato del producer (put) del flusso. La statistica Media è quella più utilizzata per questo parametro.
`PutRecord.Success`, `PutRecords.Success`	Ti consigliamo di utilizzare CloudWatch allarmi per la statistica Media per indicare errori dei record verso il flusso. Scegli uno o entrambi i tipi di put, in base a ciò che utilizza il tuo producer. Se si utilizza la Amazon Kinesis Producer Library (KPL), utilizzare. `PutRecords.Success`
`GetRecords.Success`	Ti consigliamo di utilizzare CloudWatch allarmi per la statistica Media per indicare errori dei record a partire dal flusso.

Accesso ai CloudWatch parametri di Amazon per il flusso di dati Kinesis

Puoi monitorare i parametri per il flusso di dati Kinesis utilizzando CloudWatch la console, la riga di comando o l'API. CloudWatch Le procedure seguenti mostrano come accedere ai parametri utilizzando questi diversi metodi.

Accesso ai parametri tramite la console CloudWatch

1. Apri la CloudWatch console all'indirizzo https://console.aws.amazon.com/cloudwatch/.
2. Nella barra di navigazione, scegli una regione.
3. Nel riquadro di navigazione, seleziona Parametri.
4. Nel riquadro CloudWatch Metriche per categoria, scegli Kinesis Metrics.
5. Fai clic sulla riga pertinente per visualizzare le statistiche relative all'oggetto specificato e. MetricNameStreamName

   Nota: la maggior parte dei nomi delle statistiche delle console corrisponde ai nomi delle CloudWatch metriche corrispondenti elencati in precedenza, ad eccezione di Read Throughput e Write Throughput. Queste statistiche vengono calcolate in intervalli di 5 minuti: Velocità di trasmissione effettiva di scrittura monitora la **`IncomingBytes`** CloudWatch parametri mentre Velocità di trasmissione effettiva di lettura monitora il flusso di trasmissione effettiva di lettura. `GetRecords.Bytes`
6. (Facoltativo) Nel riquadro del grafico, seleziona una statistica e un periodo di tempo, quindi crea un CloudWatch allarme utilizzando queste impostazioni.

Per accedere alle metriche utilizzando AWS CLI

Utilizzate le metriche e i comandi dell'elenco. get-metric-statistics

Accesso ai parametri tramite la CLI CloudWatch

Utilizza i comandi mon-list-metrics e mon-get-stats.

Accesso ai parametri tramite l'API CloudWatch

Utilizzare le operazioni ListMetrics e GetMetricStatistics.

Monitoraggio dell'integrità dell'agente del flusso di dati Kinesis con Amazon CloudWatch

L'agente pubblica CloudWatch parametri personalizzati con uno spazio dei nomi di. AWS KinesisAgent Questi parametri consentono di valutare se l'agente sta inviando dati a Flussi di dati Kinesis come specificato, se è integro e se sta consumando la quantità di risorse di CPU e di memoria appropriate nel producer di dati. I parametri come il numero di record e di byte inviati sono utili per comprendere la velocità a cui l'agente sta inviando dati allo flusso. Quando questi parametri si trovano al di sotto delle soglie previste in alcune percentuali o passano a zero, potrebbe trattarsi di problemi di configurazione, di errori di rete o di problemi correlati allo stato dell'agente. I parametri come il consumo di CPU e memoria di host e i contatori di errore dell'agente indicano l'utilizzo delle risorse da parte del producer e forniscono informazioni utili in merito a possibili errori di configurazione o di host. Infine, l'agente registra anche le eccezioni di servizio per aiutare a verificare i problemi dell'agente. Queste metriche sono riportate nella regione specificata nell'impostazione di configurazione dell'agente. `cloudwatch.endpoint` CloudWatch I parametri pubblicati da più agenti Kinesis vengono aggregati o combinati. Per ulteriori informazioni sulla configurazione dell'agente, consulta Specificare le impostazioni di configurazione dell'agente.

Monitora con CloudWatch

L'agente del flusso di dati Kinesis invia i parametri seguenti a. CloudWatch

Parametro	Descrizione
`BytesSent`	Il numero di byte inviati al flusso di dati Kinesis durante il periodo di tempo specificato. Unità: byte
`RecordSendAttempts`	Il numero di record tentati (sia per la prima volta che come nuovo tentativo) in una chiamata a `PutRecords` durante il periodo di tempo specificato. Unità: numero
`RecordSendErrors`	Il numero di record che hanno restituito uno stato di errore in una chiamata a `PutRecords` , inclusi i nuovi tentativi, durante il periodo di tempo specificato.

Parametro	Descrizione
	Unità: numero
`ServiceErrors`	Il numero di chiamate a `PutRecords` che hanno causato un errore di servizio (diverso da un errore di throttling) durante il periodo di tempo specificato. Unità: numero

Registra le chiamate API di flusso di dati Amazon Kinesis tramite AWS CloudTrail

Il flusso di dati Amazon Kinesis è AWS CloudTrail integrato con, un servizio che offre un record delle operazioni eseguite da un utente, un ruolo AWS o un servizio nel flusso di dati Kinesis. CloudTrail acquisisce tutte le chiamate API per il flusso di dati Kinesis come eventi. Le chiamate acquisite includono le chiamate dalla console Kinesis Data Firehose e le chiamate di codice alle operazioni API di Kinesis Data Firehose. Se viene creato un trail, è possibile abilitare la distribuzione continua di CloudTrail eventi in un bucket Amazon S3, inclusi gli eventi per Kinesis Data Streams. Se invece non configuri un trail, è comunque possibile visualizzare gli eventi più recenti nella CloudTrail console di in Event history (Cronologia eventi). Le informazioni raccolte da CloudTrail, consentono di determinare la richiesta effettuata a Kinesis Data Streams, l'indirizzo IP da cui è partita la richiesta, l'autore della richiesta, il momento in cui è stata eseguita e altri dettagli.

Per ulteriori informazioni CloudTrail, incluso come configurarlo e abilitarlo, consulta la Guida per l'AWS CloudTrail utente di.

Informazioni sul flusso di dati Kinesis in CloudTrail

CloudTrail è abilitato sull' AWS account al momento della sua creazione. Quando si verifica un'attività evento supportata in Kinesis Data Streams, tale attività viene registrata CloudTrail in un evento AWS insieme ad altri eventi di servizio di in Cronologia eventi. È possibile visualizzare, cercare e scaricare gli eventi recenti nell' AWS account. Per ulteriori informazioni, consulta Visualizzazione di eventi mediante la cronologia eventi di CloudTrail.

Per una registrazione continua degli eventi nell' AWS account, inclusi gli eventi relativi al flusso di dati Kinesis, crea un trail. Un trail consente di CloudTrail distribuire i file di log in un bucket Amazon S3. Per impostazione predefinita, quando si crea un trail nella console, il trail sarà valido in tutte le

AWS Regioni. Il trail registra gli eventi di tutte le Regioni nella AWS partizione e distribuisce i file di log nel bucket Amazon S3 specificato. Inoltre, puoi configurare altri AWS servizi per analizzare con maggiore dettaglio e usare i dati evento raccolti nei CloudTrail log. Per ulteriori informazioni, consulta gli argomenti seguenti:

- Panoramica della creazione di un trail
- CloudTrail Servizi e integrazioni supportati
- Configurazione delle notifiche Amazon SNS per CloudTrail
- Ricezione di file di CloudTrail log da più regioni e Ricezione di file di CloudTrail log da più account

Il flusso di dati Kinesis supporta la registrazione delle operazioni seguenti come CloudTrail eventi nei file di log:

- AddTagsToStream
- CreateStream
- DecreaseStreamRetentionPeriod
- DeleteStream
- DeregisterStreamConsumer
- DescribeStream
- DescribeStreamConsumer
- DisableEnhancedMonitoring
- EnableEnhancedMonitoring
- GetRecords
- GetShardIterator
- IncreaseStreamRetentionPeriod
- ListStreamConsumers
- ListStreams
- ListTagsForStream
- MergeShards
- PutRecord
- PutRecords
- RegisterStreamConsumer

- RemoveTagsFromStream
- SplitShard
- StartStreamEncryption
- StopStreamEncryption
- SubscribeToShard
- UpdateShardCount
- UpdateStreamMode

Ogni evento o voce di log contiene informazioni sull'utente che ha generato la richiesta. Le informazioni di identità consentono di determinare quanto segue:

- Se la richiesta è stata effettuata con le credenziali utente root o AWS Identity and Access Management (IAM).
- Se la richiesta è stata effettuata con le credenziali di sicurezza temporanee per un ruolo o un utente federato.
- Se la richiesta è stata effettuata da un altro AWS servizio.

Per ulteriori informazioni, consulta Elemento CloudTrail userIdentity.

Esempio: voci dei file di log del flusso di dati Kinesis

Un trail è una configurazione che consente la distribuzione di eventi come i file di log in un bucket Amazon S3 specificato dall'utente. CloudTrail i file di log possono contenere una o più voci di log. Un evento rappresenta una singola richiesta da qualsiasi sorgente e include informazioni sull'azione richiesta, la data e l'ora dell'operazione, i parametri della richiesta e così via. I file di log di CloudTrail non sono una traccia di stack ordinata delle chiamate API pubbliche, pertanto non vengono visualizzati in un ordine specifico.

L'esempio seguente mostra una voce di CloudTrail registro che illustra le `MergeShards` azioni `CreateStream` `DescribeStreamListStreams`,`DeleteStream`,`SplitShard`, e.

```
{
    "Records": [
        {
            "eventVersion": "1.01",
            "userIdentity": {
                "type": "IAMUser",
```

```
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::012345678910:user/Alice",
                "accountId": "012345678910",
                "accessKeyId": "EXAMPLE_KEY_ID",
                "userName": "Alice"
            },
            "eventTime": "2014-04-19T00:16:31Z",
            "eventSource": "kinesis.amazonaws.com",
            "eventName": "CreateStream",
            "awsRegion": "us-east-1",
            "sourceIPAddress": "127.0.0.1",
            "userAgent": "aws-sdk-java/unknown-version Linux/x.xx",
            "requestParameters": {
                "shardCount": 1,
                "streamName": "GoodStream"
            },
            "responseElements": null,
            "requestID": "db6c59f8-c757-11e3-bc3b-57923b443c1c",
            "eventID": "b7acfcd0-6ca9-4ee1-a3d7-c4e8d420d99b"
        },
        {
            "eventVersion": "1.01",
            "userIdentity": {
                "type": "IAMUser",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::012345678910:user/Alice",
                "accountId": "012345678910",
                "accessKeyId": "EXAMPLE_KEY_ID",
                "userName": "Alice"
            },
            "eventTime": "2014-04-19T00:17:06Z",
            "eventSource": "kinesis.amazonaws.com",
            "eventName": "DescribeStream",
            "awsRegion": "us-east-1",
            "sourceIPAddress": "127.0.0.1",
            "userAgent": "aws-sdk-java/unknown-version Linux/x.xx",
            "requestParameters": {
                "streamName": "GoodStream"
            },
            "responseElements": null,
            "requestID": "f0944d86-c757-11e3-b4ae-25654b1d3136",
            "eventID": "0b2f1396-88af-4561-b16f-398f8eaea596"
        },
        {
```

```
            "eventVersion": "1.01",
            "userIdentity": {
                "type": "IAMUser",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::012345678910:user/Alice",
                "accountId": "012345678910",
                "accessKeyId": "EXAMPLE_KEY_ID",
                "userName": "Alice"
            },
            "eventTime": "2014-04-19T00:15:02Z",
            "eventSource": "kinesis.amazonaws.com",
            "eventName": "ListStreams",
            "awsRegion": "us-east-1",
            "sourceIPAddress": "127.0.0.1",
            "userAgent": "aws-sdk-java/unknown-version Linux/x.xx",
            "requestParameters": {
                "limit": 10
            },
            "responseElements": null,
            "requestID": "a68541ca-c757-11e3-901b-cbcfe5b3677a",
            "eventID": "22a5fb8f-4e61-4bee-a8ad-3b72046b4c4d"
        },
        {
            "eventVersion": "1.01",
            "userIdentity": {
                "type": "IAMUser",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::012345678910:user/Alice",
                "accountId": "012345678910",
                "accessKeyId": "EXAMPLE_KEY_ID",
                "userName": "Alice"
            },
            "eventTime": "2014-04-19T00:17:07Z",
            "eventSource": "kinesis.amazonaws.com",
            "eventName": "DeleteStream",
            "awsRegion": "us-east-1",
            "sourceIPAddress": "127.0.0.1",
            "userAgent": "aws-sdk-java/unknown-version Linux/x.xx",
            "requestParameters": {
                "streamName": "GoodStream"
            },
            "responseElements": null,
            "requestID": "f10cd97c-c757-11e3-901b-cbcfe5b3677a",
            "eventID": "607e7217-311a-4a08-a904-ec02944596dd"
```

```
        },
        {
            "eventVersion": "1.01",
            "userIdentity": {
                "type": "IAMUser",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::012345678910:user/Alice",
                "accountId": "012345678910",
                "accessKeyId": "EXAMPLE_KEY_ID",
                "userName": "Alice"
            },
            "eventTime": "2014-04-19T00:15:03Z",
            "eventSource": "kinesis.amazonaws.com",
            "eventName": "SplitShard",
            "awsRegion": "us-east-1",
            "sourceIPAddress": "127.0.0.1",
            "userAgent": "aws-sdk-java/unknown-version Linux/x.xx",
            "requestParameters": {
                "shardToSplit": "shardId-000000000000",
                "streamName": "GoodStream",
                "newStartingHashKey": "11111111"
            },
            "responseElements": null,
            "requestID": "a6e6e9cd-c757-11e3-901b-cbcfe5b3677a",
            "eventID": "dcd2126f-c8d2-4186-b32a-192dd48d7e33"
        },
        {
            "eventVersion": "1.01",
            "userIdentity": {
                "type": "IAMUser",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::012345678910:user/Alice",
                "accountId": "012345678910",
                "accessKeyId": "EXAMPLE_KEY_ID",
                "userName": "Alice"
            },
            "eventTime": "2014-04-19T00:16:56Z",
            "eventSource": "kinesis.amazonaws.com",
            "eventName": "MergeShards",
            "awsRegion": "us-east-1",
            "sourceIPAddress": "127.0.0.1",
            "userAgent": "aws-sdk-java/unknown-version Linux/x.xx",
            "requestParameters": {
                "streamName": "GoodStream",
```

```
                "adjacentShardToMerge": "shardId-000000000002",
                "shardToMerge": "shardId-000000000001"
            },
            "responseElements": null,
            "requestID": "e9f9c8eb-c757-11e3-bf1d-6948db3cd570",
            "eventID": "77cf0d06-ce90-42da-9576-71986fec411f"
        }
    ]
}
```

Monitoraggio della Kinesis Client Library con Amazon CloudWatch

La Kinesis Client Library (KCL) per il flusso di dati Amazon Kinesis pubblica parametri Amazon Amazon CloudWatch Amazon personalizzati per tuo conto, utilizzando il nome dell'applicazione KCL come spazio dei nomi. Puoi visualizzare queste metriche accedendo alla console e scegliendo Custom Metrics. CloudWatch Per ulteriori informazioni sui parametri personalizzati, consulta Publish Custom Metrics nella Amazon CloudWatch User Guide.

C'è un costo nominale per i parametri caricati CloudWatch da KCL specificamente, si applicano i costi per i parametri CloudWatch personalizzati di Amazon e per le richieste API di CloudWatch Amazon. Per ulteriori informazioni, consulta la pagina CloudWatch dei prezzi di Amazon.

Argomenti

- Parametri e spazio dei nomi
- Parametri e dimensioni
- Configurazione metrica
- Elenco parametri

Parametri e spazio dei nomi

Lo spazio dei nomi utilizzato per caricare i parametri è il nome dell'applicazione che è necessario specificare quando si lancia la KCL.

Parametri e dimensioni

Sono disponibili due opzioni per controllare quali parametri vengono caricati su CloudWatch:

Livelli di parametri

A ogni parametro viene assegnato un livello individuale. Quando si imposta un livello di segnalazione dei parametri, i parametri con un livello individuale al di sotto del livello di segnalazione non vengono inviati a. CloudWatch I livelli sono: `NONE`, `SUMMARY` e `DETAILED`. L'impostazione predefinita è, `DETAILED` ovvero tutte le metriche vengono inviate a CloudWatch. Un livello di reporting di `NONE` significa che nessun parametro viene inviato. Per informazioni su quali livelli sono assegnati a quali parametri, consulta Elenco parametri.

Dimensioni abilitate

Ogni parametro KCL ha dimensioni associate a cui vengono inviate anche. CloudWatch In KCL 2.x, se KCL è configurato per elaborare un singolo flusso di dati, tutte le dimensioni dei parametri (`Operation`, `ShardId` e `WorkerIdentifier`) sono abilitate per impostazione predefinita. Inoltre, in KCL 2.x, se KCL è configurato per elaborare un singolo flusso di dati, la dimensione `Operation` non può essere disabilitata. In KCL 2.x, se KCL è configurato per elaborare più flussi di dati, tutte le dimensioni dei parametri (, `Operation` `ShardIdStreamId`, e`WorkerIdentifier`) sono abilitate per impostazione predefinita. Inoltre, in KCL 2.x, se KCL è configurato per elaborare più flussi di dati, le dimensioni `Operation` e le `StreamId` dimensioni non possono essere disabilitate. `StreamId`la dimensione è disponibile solo per le metriche per-shard.

In KCL 1.x, solo le dimensioni `Operation` e `ShardId` sono abilitate per impostazione predefinita, mentre la dimensione `WorkerIdentifier` è disabilitata. In KCL 1.x, la dimensione `Operation` non può essere disabilitata.

Per informazioni sulle dimensioni dei CloudWatch parametri, consulta Dimensioni nell'argomento CloudWatch Concetti di Amazon Concetti nella Guida per l' CloudWatch utente di Amazon Amazon.

Quando la `WorkerIdentifier` dimensione è abilitata, se un valore diverso viene utilizzato per la proprietà ID worker ogni volta che un determinato worker KCL viene riavviato, nuovi set di parametri con nuovi valori per la `WorkerIdentifier` dimensione. CloudWatch Se necessiti che il valore della dimensione `WorkerIdentifier` sia lo stesso ogni volta che viene riavviato un determinato worker KCL, devi specificare esplicitamente lo stesso valore di ID worker durante l'inizializzazione per ogni worker. Si noti che il valore di ID worker per ogni worker attivo della KCL deve essere univoco per tutti i worker KCL.

Configurazione metrica

I livelli dei parametri e le dimensioni abilitate possono essere configurati utilizzando l' KinesisClientLibConfiguration istanza, che viene inviata al worker quando si lancia l'applicazione KCL. Nel MultiLangDaemon caso, è possibile specificare `metricsEnabledDimensions` le proprietà `metricsLevel` and nel file .properties utilizzato per avviare l'applicazione KCL. MultiLangDaemon

Ai livelli di parametri può essere assegnato uno dei tre valori seguenti: NONE, SUMMARY o DETAILED. I valori delle dimensioni abilitate devono essere stringhe separate da virgole con l'elenco delle dimensioni consentite per i parametri. CloudWatch Le dimensioni utilizzate dall'applicazione KCL sono `Operation`, `ShardId` e `WorkerIdentifier`.

Elenco parametri

Le seguenti tabelle riportano i parametri della KCL raggruppati in base all'ambito e all'operazione.

Argomenti

- Per-KCL-application metriche
- Parametri per lavoratore
- Parametri per partizione

Per-KCL-application metriche

Questi parametri vengono aggregati in tutti i worker KCL nell'ambito dell'applicazione, come definito dallo spazio dei nomi Amazon. CloudWatch

Argomenti

- LeaseAssignmentManager
- InitializeTask
- ShutdownTask
- ShardSyncTask
- BlockOnParentTask
- PeriodicShardSyncManager
- MultistreamTracker

LeaseAssignmentManager

L'`LeaseAssignmentManager`operazione è responsabile dell'assegnazione dei contratti di locazione ai lavoratori e del riequilibrio dei contratti di locazione tra i lavoratori per ottenere un utilizzo uniforme delle risorse dei lavoratori. La logica di questa operazione include la lettura dei metadati relativi al leasing dalla tabella dei leasing e delle metriche dalla tabella delle metriche dei lavoratori e l'esecuzione delle assegnazioni del leasing.

Parametro	Descrizione
LeaseAndWorkerMetricsLoad.Ora	Tempo impiegato per caricare tutti i leasing e le metriche dei lavoratori nel Lease Assignment Manager (LAM), il nuovo algoritmo di assegnazione del leasing e di bilanciamento del carico introdotto in KCL 3.x. Livello parametro: Detailed Unità: millisecondi
TotalLeases	Numero totale di leasing per l'attuale applicazione KCL. Livello parametro: Summary Unità: numero
NumWorkers	Numero totale di lavoratori nell'attuale applicazione KCL. Livello parametro: Summary Unità: numero
AssignExpiredOrUnassignedLeases.Ora	Tempo necessario per eseguire l'assegnazione in memoria dei leasing scaduti. Livello parametro: Detailed Unità: millisecondi
LeaseSpillover	Numero di leasing che non sono stati assegnati perché è stato raggiunto il limite del numero massimo di leasing o della produttività massima per lavoratore.

Parametro	Descrizione
	Livello parametro: Summary Unità: numero
BalanceWorkerVariance.Ora	È ora di eseguire il bilanciamento in memoria dei contratti di locazione tra i lavoratori. Livello parametro: Detailed Unità: millisecondi
NumOfLeasesReassignment	Numero totale di riassegnazioni di leasing effettuate nell'attuale iterazione di riassegnazione. Livello parametro: Summary Unità: numero
FailedAssignmentCount	Numero di errori nelle AssignLease chiamate alla tabella di lease di DynamoDB. Livello parametro: Detailed Unità: numero
ParallelyAssignLeases.Ora	È ora di inserire nuove assegnazioni nella tabella di lease di DynamoDB. Livello parametro: Detailed Unità: millisecondi
ParallelyAssignLeases.Successo	Numero di nuovi incarichi completati con successo. Livello parametro: Detailed Unità: numero

Parametro	Descrizione
TotalStaleWorkerMetricsEntry	Numero totale di voci delle metriche dei lavoratori che devono essere ripulite. Livello parametro: Detailed Unità: numero
StaleWorkerMetricsCleanup.Ora	È ora di eseguire l'eliminazione delle voci delle metriche dei lavoratori dalla tabella delle metriche dei lavoratori DynamoDB. Livello parametro: Detailed Unità: millisecondi
Orario	Tempo impiegato dall'operazione. `LeaseAssignmentManager` Livello parametro: Summary Unità: millisecondi
Riuscito	Numero di volte in cui l'operazione `LeaseAssignmentManager` viene completata correttamente. Livello parametro: Summary Unità: numero
ForceLeaderRelease	Indica che il responsabile dell'assegnazione del leasing ha fallito per 3 volte consecutive e che il lavoratore dirigente sta rilasciando la leadership. Livello parametro: Summary Unità: numero

Parametro	Descrizione
NumWorkersWithInvalidEntry	Numero di voci relative alle metriche dei lavoratori considerate non valide. Livello parametro: Summary Unità: numero
NumWorkersWithFailingWorkerMetric	Numero di voci delle metriche dei lavoratori che hanno -1 (che rappresenta il valore della metrica del lavoratore non è disponibile) come valore per le metriche dei lavoratori. Livello parametro: Summary Unità: numero
LeaseDeserializationFailureCount	Voce di leasing dalla tabella dei leasing che non è stata deserializzata. Livello parametro: Summary Unità: numero

InitializeTask

L'operazione `InitializeTask` è responsabile dell'inizializzazione del processore di record per l'applicazione KCL. La logica per questa operazione include l'acquisizione di un iteratore di partizione da Flussi di dati Kinesis e l'inizializzazione del processore di record.

Parametro	Descrizione
KinesisDataFetcher.getIterator.Successo	Numero di operazioni `GetShardIterator` riuscite per applicazione KCL. Livello parametro: Detailed Unità: numero
KinesisDataFetcher.getIterator.Time	Tempo impiegato per l'operazione `GetShardIterator` per l'applicazione KCL in questione.

Parametro	Descrizione
	Livello parametro: Detailed Unità: millisecondi
RecordProcessor.inizializza. Ora	Tempo impiegato dal metodo di inizializzazione del processore di record. Livello parametro: Summary Unità: millisecondi
Riuscito	Numero di inizializzazioni corrette del processore di record. Livello parametro: Summary Unità: numero
Orario	Tempo impiegato dal worker KCL per l'inizializzazione del processore di record. Livello parametro: Summary Unità: millisecondi

ShutdownTask

L'operazione `ShutdownTask` avvia la sequenza di arresto per l'elaborazione di shard. Questa situazione può verificarsi perché uno shard viene frazionato o fuso, oppure quando il lease dello shard viene perso dal lavoratore. In entrambi i casi, viene invocata la funzione `shutdown()` del processore di record. Nuovi shard vengono individuati anche nel caso in cui uno shard sia stato frazionato o fuso; ciò comporta la creazione di uno o due nuovi shard.

Parametro	Descrizione
CreateLease.Successo	Numero di volte in cui le nuove partizioni secondarie vengono aggiunte correttamente nella tabella DynamoDB dell'applicazione KCL dopo l'arresto della partizione principale. Livello parametro: Detailed

Parametro	Descrizione
	Unità: numero
CreateLease.Ora	Tempo impiegato per l'aggiunta delle informazioni relative alla nuova partizione secondaria nella tabella DynamoDB dell'applicazione KCL. Livello parametro: Detailed Unità: millisecondi
UpdateLease.Successo	Numero di checkpoint finali corretti durante l'arresto del processore di record. Livello parametro: Detailed Unità: numero
UpdateLease.Ora	Tempo impiegato dall'operazione di checkpoint durante l'arresto del processore di record. Livello parametro: Detailed Unità: millisecondi
RecordProcessor.Spegnimento. Ora	Tempo impiegato dal metodo di arresto del processore di record. Livello parametro: Summary Unità: millisecondi
Riuscito	Numero di attività di arresto corrette. Livello parametro: Summary Unità: numero
Orario	Tempo impiegato dal worker KCL per l'attività di arresto. Livello parametro: Summary Unità: millisecondi

ShardSyncTask

L'operazione `ShardSyncTask` rileva le modifiche alle informazioni relative alla partizione per il flusso di dati Kinesis, pertanto le nuove partizioni possono essere elaborate dall'applicazione KCL.

Parametro	Descrizione
CreateLease.Successo	Numero di tentativi riusciti di aggiunta di nuove informazioni relative alla partizione nella tabella DynamoDB dell'applicazione KCL. Livello parametro: Detailed Unità: numero
CreateLease.Ora	Tempo impiegato per l'aggiunta delle informazioni relative alla nuova partizione nella tabella DynamoDB dell'applicazione KCL. Livello parametro: Detailed Unità: millisecondi
Riuscito	Numero di operazioni di sincronizzazione dello shard corrette. Livello parametro: Summary Unità: numero
Orario	Tempo impiegato per l'operazione di sincronizzazione dello shard. Livello parametro: Summary Unità: millisecondi

BlockOnParentTask

Se lo shard viene frazionato o fuso con altri shard, vengono creati nuovi shard secondari. L'operazione `BlockOnParentTask` garantisce che l'elaborazione dei record per le nuove partizioni non inizi fino a quando le partizioni principali non siano state completamente elaborate dalla KCL.

Parametro	Descrizione
Riuscito	Numero di controlli corretti per il completamento dello shard principale. Livello parametro: Summary Unità: numero
Orario	Tempo impiegato per il completamento degli shard principali. Livello parametro: Summary Unità: millisecondi

PeriodicShardSyncManager

`PeriodicShardSyncManager` è responsabile dell'esame dei flussi di dati elaborati dall'applicazione consumer KCL, dell'identificazione dei flussi di dati con leasing parziali e della loro trasmissione per la sincronizzazione.

I seguenti parametri sono disponibili quando KCL è configurato per elaborare un singolo flusso di dati (quindi il valore di NumStreamsToSync e NumStreamsWithPartialLeases è impostato su 1) e anche quando KCL è configurato per elaborare più flussi di dati.

Parametro	Descrizione
NumStreamsToSync	Il numero di flussi di dati (per AWS account) elaborati dall'applicazione consumer che contiene lease parziali e che devono essere trasferiti per la sincronizzazione. Livello parametro: Summary Unità: numero
NumStreamsWithPartialLeases	Il numero di flussi di dati (per AWS account) elaborati dall'applicazione consumer che contengono lease parziali. Livello parametro: Summary

Parametro	Descrizione
	Unità: numero
Riuscito	Il numero di volte in cui `PeriodicShardSyncManager` è riuscito a identificare con successo i lease parziali nei flussi di dati elaborati dall'applicazione consumer. Livello parametro: Summary Unità: numero
Orario	Il tempo (in millisecondi) che l'applicazione consumer `PeriodicShardSyncManager` impiega per esaminare i flussi di dati elaborati dall'applicazione consumer, al fine di determinare quali flussi di dati richiedono la sincronizzazione delle partizioni. Livello parametro: Summary Unità: millisecondi

MultistreamTracker

L'interfaccia `MultistreamTracker` consente di creare applicazioni consumer KCL in grado di elaborare più flussi di dati contemporaneamente.

Parametro	Descrizione
DeletedStreams.Conta	Il numero di flussi di dati eliminati in questo periodo di tempo. Livello parametro: Summary Unità: numero
ActiveStreams.Conta	Il numero di flussi di dati attivi in fase di elaborazione. Livello parametro: Summary Unità: numero

Parametro	Descrizione
StreamsPendingDeletion.Conta	Il numero di flussi di dati in attesa di eliminazione in base a `FormerStreamsLeasesDeletionStrategy` . Livello parametro: Summary Unità: numero

Parametri per lavoratore

Questi parametri vengono aggregati in tutti i processori di record che consumano dati da un flusso di dati Kinesis, come ad esempio, un'istanza Amazon. EC2

Argomenti

- WorkerMetricStatsReporter
- LeaseDiscovery
- RenewAllLeases
- TakeLeases

WorkerMetricStatsReporter

L'`WorkerMetricStatReporter`operazione è responsabile della pubblicazione periodica delle metriche del lavoratore corrente nella tabella delle metriche del lavoratore. Queste metriche vengono utilizzate dall'`LeaseAssignmentManager`operazione per eseguire assegnazioni di leasing.

Parametro	Descrizione
InMemoryMetricStatsReporterFailure	Numero di errori nell'acquisizione del valore della metrica del lavoratore in memoria, a causa del fallimento di alcune metriche del lavoratore. Livello parametro: Summary Unità: numero
WorkerMetricStatsReporter.Ora	Tempo impiegato dall'`WorkerMetricsStats` operazione. Livello parametro: Summary

Parametro	Descrizione
	Unità: millisecondi
WorkerMetricStatsReporter.Successo	Numero di volte in cui l'operazione `WorkerMetricsStats` viene completata correttamente. Livello parametro: Summary Unità: numero

LeaseDiscovery

L'`LeaseDiscovery`operazione è responsabile dell'identificazione dei nuovi contratti di locazione assegnati all'attuale lavoratore dall'`LeaseAssignmentManager`operazione. La logica di questa operazione prevede l'identificazione dei leasing assegnati al lavoratore corrente mediante la lettura dell'indice secondario globale della tabella dei leasing.

Parametro	Descrizione
ListLeaseKeysForWorker.Ora	È ora di richiamare l'indice secondario globale nella tabella delle locazioni e ottenere l'assegnazione delle chiavi di leasing al lavoratore corrente. Livello parametro: Detailed Unità: millisecondi
FetchNewLeases.Ora	È ora di recuperare tutti i nuovi lease dalla tabella di lease. Livello parametro: Detailed Unità: millisecondi
NewLeasesDiscovered	Numero totale di nuovi contratti di locazione assegnati ai lavoratori. Livello parametro: Detailed Unità: numero
Orario	Tempo impiegato dall'`LeaseDiscovery` operazione.

Parametro	Descrizione
	Livello parametro: Summary Unità: millisecondi
Riuscito	Numero di volte in cui l'operazione `LeaseDiscovery` viene completata correttamente. Livello parametro: Summary Unità: numero
OwnerMismatch	Numero di mancate corrispondenze tra il proprietario e la risposta del GSI e la lettura coerente della tabella di leasing. Livello parametro: Detailed Unità: numero

RenewAllLeases

L'operazione `RenewAllLeases` rinnova periodicamente i lease di shard di proprietà di una determinata istanza di lavoratore.

Parametro	Descrizione
RenewLease.Successo	Numero di rinnovi corretti di lease da parte del lavoratore. Livello parametro: Detailed Unità: numero
RenewLease.Ora	Tempo impiegato dall'operazione di rinnovo del lease. Livello parametro: Detailed Unità: millisecondi
CurrentLeases	Numero di lease di shard di proprietà del lavoratore dopo il rinnovo di tutti i lease.

Parametro	Descrizione
	Livello parametro: Summary Unità: numero
LostLeases	Numero di lease di shard perduti a seguito di un tentativo di rinnovo di tutti i lease di proprietà del lavoratore. Livello parametro: Summary Unità: numero
Riuscito	Numero di volte in cui l'operazione di rinnovo del contratto di locazione ha avuto successo per il lavoratore. Livello parametro: Summary Unità: numero
Orario	Tempo impiegato per rinnovare tutti i lease per il lavoratore. Livello parametro: Summary Unità: millisecondi

TakeLeases

L'operazione `TakeLeases` bilancia l'elaborazione di record tra tutti i worker KCL. Se il worker KCL corrente ha un numero di lease di partizione inferiore rispetto al necessario, utilizza i lease di partizione di un altro worker che è sovraccarico.

Parametro	Descrizione
ListLeases.Successo	Numero di volte in cui tutti i lease di partizione sono stati recuperati correttamente dalla tabella DynamoDB dell'applicazione KCL. Livello parametro: Detailed Unità: numero

Parametro	Descrizione
ListLeases.Ora	Tempo impiegato per recuperare tutti i lease di partizione dalla tabella DynamoDB dell'applicazione KCL. Livello parametro: Detailed Unità: millisecondi
TakeLease.Successo	Numero di volte in cui il worker ha utilizzato correttamente lease di partizione di altri worker KCL. Livello parametro: Detailed Unità: numero
TakeLease.Ora	Tempo impiegato per aggiornare la tabella di lease con i lease di altri lavoratori utilizzati dal lavoratore. Livello parametro: Detailed Unità: millisecondi
NumWorkers	Numero totale di lavoratori, come identificato da un determinato lavorator e. Livello parametro: Summary Unità: numero
NeededLeases	Numero di lease di shard che l'attuale lavoratore necessita per un carico equilibrato di elaborazione di shard. Livello parametro: Detailed Unità: numero
LeasesToTake	Numero di lease di altri lavoratori che il lavoratore cercherà di utilizzare. Livello parametro: Detailed Unità: numero

Parametro	Descrizione
TakenLeases	Numero di lease di altri lavoratori utilizzati correttamente dal lavoratore. Livello parametro: Summary Unità: numero
TotalLeases	Numero totale di partizioni che l'applicazione KCL sta elaborando. Livello parametro: Detailed Unità: numero
ExpiredLeases	Numero totale di shard che non vengono elaborati da nessun lavoratore, come identificato dal lavoratore specifico. Livello parametro: Summary Unità: numero
Riuscito	Numero di volte in cui l'operazione `TakeLeases` viene completata correttamente. Livello parametro: Summary Unità: numero
Orario	Tempo impiegato dall'operazione `TakeLeases` per un lavoratore. Livello parametro: Summary Unità: millisecondi

Parametri per partizione

Questi parametri si aggregano in un singolo processore di record.

ProcessTask

L'operazione `ProcessTask` chiama GetRecords con la posizione attuale dell'iteratore per recuperare record dal flusso e richiama la funzione `processRecords` del processore di record.

Parametro	Descrizione
KinesisDataFetcher .getRecords.Success	Numero di operazioni `GetRecords` riuscite per partizione del flusso di dati Kinesis. Livello parametro: Detailed Unità: numero
KinesisDataFetcher .getRecords.Time	Tempo impiegato per l'operazione `GetRecords` per la partizione del flusso di dati Kinesis Livello parametro: Detailed Unità: millisecondi
UpdateLease.Succes so	Numero di checkpoint corretti eseguiti dal processore di record per un determinato shard. Livello parametro: Detailed Unità: numero
UpdateLease.Ora	Tempo impiegato per ogni operazione di checkpoint per un determinato shard. Livello parametro: Detailed Unità: millisecondi
DataBytesProcessed	Dimensione totale dei record elaborati in byte in ogni invocazione di `ProcessTask` . Livello parametro: Summary Unità: byte
RecordsProcessed	Numero di record elaborati in ogni invocazione di `ProcessTask` . Livello parametro: Summary

Parametro	Descrizione
	Unità: numero
ExpiredIterator	Numero di messaggi ExpiredIteratorException ricevuti durante la chiamata`GetRecords` . Livello parametro: Summary Unità: numero
MillisBehindLatest	Ritardo dell'attuale iteratore rispetto all'ultimo record (estremità) nello shard. Questo valore è inferiore o pari alla differenza temporale tra l'ultimo record in una risposta e l'orario attuale. Si tratta di un indicator e più accurato della distanza di un frammento dalla punta rispetto al confronto dei timestamp del record dell'ultima risposta. Questo valore si applica all'ultimo gruppo di record, non alla media di tutti i timestamp di ogni record. Livello parametro: Summary Unità: millisecondi
RecordProcessor.processRecords.time	Tempo impiegato dal metodo `processRecords` del processore di record. Livello parametro: Summary Unità: millisecondi
Riuscito	Numero di operazioni di attività di processo corrette. Livello parametro: Summary Unità: numero
Orario	Tempo impiegato per l'operazione di attività di processo. Livello parametro: Summary Unità: millisecondi

Monitoraggio della Kinesis Client Library con Amazon CloudWatch

La Amazon Kinesis Kinesis pubblica parametri Amazon Kinesis personalizzati per tuo conto. CloudWatch Puoi visualizzare questi parametri accedendo alla console e scegliendo Custom Metrics. CloudWatch Per ulteriori informazioni sui parametri personalizzati, consulta Publish Custom Metrics nella Amazon CloudWatch User Guide.

C'è un costo nominale per i parametri caricati CloudWatch dalla KCL specificamente, si applicano i costi per i parametri CloudWatch personalizzati di Amazon e per le richieste API di CloudWatch Amazon. Per ulteriori informazioni, consulta la pagina CloudWatch dei prezzi di Amazon. Per la raccolta di parametri locali non verrà addebitato alcun costo di CloudWatch.

Argomenti

- Metriche, dimensioni e namespace
- Livello metrico e granularità
- Accesso locale e CloudWatch caricamento su Amazon
- Elenco parametri

Metriche, dimensioni e namespace

È possibile specificare un nome di applicazione al momento dell'avvio della KPL, che viene poi utilizzato come parte di uno spazio dei nomi durante il caricamento dei parametri. Si tratta di un'opzione facoltativa; se il nome di un'applicazione non è impostato la KPL fornisce un valore predefinito.

È anche possibile configurare la KPL per aggiungere dimensioni aggiuntive arbitrarie ai parametri. Questa opzione è utile se desideri dati di alta precisione nelle tue metriche. CloudWatch Ad esempio, è possibile aggiungere il nome host come una dimensione, che consente di identificare le distribuzioni di carico irregolari nel tuo parco. Tutte le impostazioni di configurazione della KPL sono immutabili, perciò non è possibile modificare queste dimensioni aggiuntive dopo l'inizializzazione dell'istanza KPL.

Livello metrico e granularità

Sono disponibili due opzioni per controllare il numero di parametri caricati su CloudWatch:

Livelli di parametri

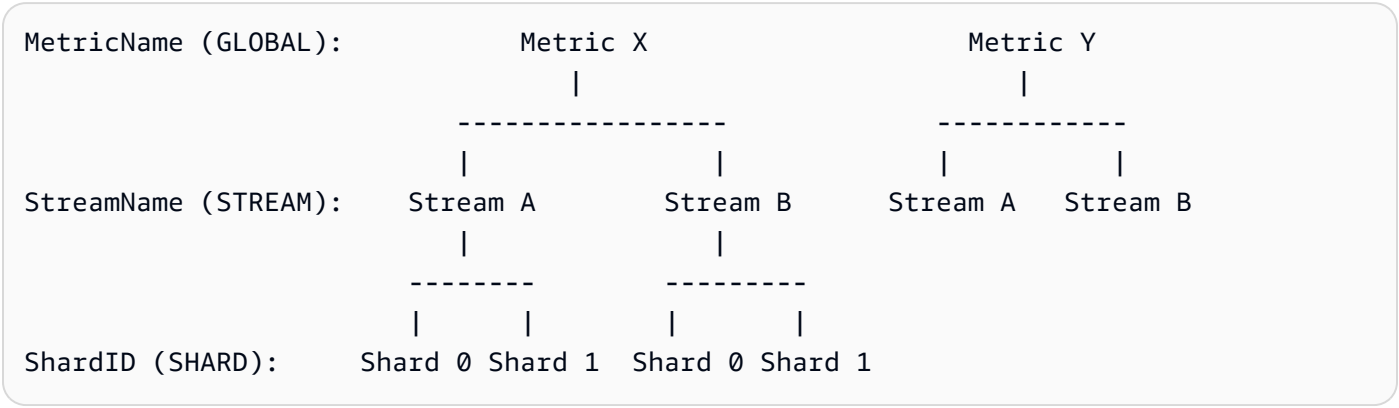
Si tratta di una calibrazione approssimativa dell'importanza di un parametro. A ogni parametro viene assegnato un livello. Quando si imposta un livello, le metriche con livelli al di sotto di tale livello non vengono inviate CloudWatch. I livelli sono `NONE`, `SUMMARY` e `DETAILED`. L'impostazione predefinita è `DETAILED`; cioè, tutti i parametri. `NONE` significa nessun parametro, per cui nessun parametro viene assegnato a quel livello.

Granularità

Ciò consente di controllare se lo stesso parametro è emesso a ulteriori livelli di granularità. I livelli sono `GLOBAL`, `STREAM` e `SHARD`. L'impostazione predefinita è `SHARD`, che contiene la maggior parte dei parametri granulari.

Quando `SHARD` viene scelto, i parametri vengono emessi con il nome di flusso e l'ID dello shard come dimensioni. Inoltre, lo stesso parametro è emesso anche con solo la dimensione del nome di flusso e il parametro senza il nome di flusso. Ciò significa che, per un determinato parametro, due flussi con due partizioni produrranno ciascuno sette CloudWatch parametri: uno per ogni partizione, uno per ogni flusso e uno generale. Tutti questi parametri descrivono le stesse statistiche, ma a livelli differenti di granularità. Per un'illustrazione, si veda il seguente diagramma.

I livelli differenti di granularità formano una gerarchia e tutti i parametri nel sistema formano alberi radicati nei nomi parametri:



Non tutti i parametri sono disponibili a livello di shard; alcuni sono disponibili a livello di flusso o sono di natura globale. Questi parametri non vengono prodotti a livello di shard, anche se hai abilitato i parametri a livello di shard (`Metric Y` nel diagramma precedente).

Quando si specifica una dimensione aggiuntiva, è necessario fornire valori per`tuple:<DimensionName, DimensionValue, Granularity>`. La granularità viene

utilizzata per determinare dove viene inserita la dimensione personalizzata nella gerarchia: `GLOBAL` significa che la dimensione aggiuntiva viene inserita dopo il nome parametro, `STREAM` significa che viene inserita dopo il nome del flusso e `SHARD` significa che viene inserita dopo l'ID dello shard. Se sono indicate più dimensioni aggiuntive per livello di granularità, queste dimensioni sono inserite nell'ordine determinato.

Accesso locale e CloudWatch caricamento su Amazon

I parametri per l'istanza di KPL corrente sono disponibili in locale in tempo reale; puoi inviare una query alla KPL in qualsiasi momento per ottenerli. La KPL calcola in locale la somma, la media, il minimo, il massimo e il conteggio di ogni parametro, ad esempio. CloudWatch

È possibile ottenere statistiche cumulative dall'inizio del programma fino al momento attuale o tramite una finestra continua per gli ultimi N secondi, dove N è un numero intero compreso tra 1 e 60.

Tutte i parametri sono disponibili per essere caricati su. CloudWatch Ciò è particolarmente utile per aggregare i data tra più host, per il monitoraggio e per la creazione di allarmi. Questa funzionalità non è disponibile in locale.

Come descritto in precedenza, è possibile selezionare quali parametri caricare con le impostazioni di livello di parametro e granularità. I parametri che non vengono caricati sono disponibili in locale.

Il caricamento di singoli punti di dati è insostenibile in quando potrebbe produrre milioni di caricamenti al secondo, se il traffico è elevato. Per questo motivo, il KPL aggrega parametri in locale in bucket di 1 minuto e carica un oggetto di statistiche una volta al minuto, per ogni CloudWatch parametro abilitato.

Elenco parametri

Parametro	Descrizione
`UserRecordsReceived`	Conteggio del numero di record utenti logici ricevuti dal core KPL per operazioni put. Non disponibile a livello di shard. Livello parametro: Detailed Unità: numero

Parametro	Descrizione
`UserRecordsPending`	Campione periodico del numero di record di utenti attualmente in sospeso. Un record è in sospeso se è attualmente memorizzato nel buffer e in attesa di essere inviato oppure inviato e in elaborazione al servizio di backend. Non disponibile a livello di shard. La KPL fornisce un metodo dedicato per recuperare questo parametro a livello globale e consentire ai clienti di gestire la loro velocità di put. Livello parametro: Detailed Unità: numero
`UserRecordsPut`	Conteggio del numero di record di utenti logici inseriti correttamente. La KPL restituisce zero per i record non riusciti. In questo modo, la media offre la percentuale dei record inseriti correttamente, il conteggio offre il numero totale di tentativi e la differenza tra il conteggio e la somma offre il conteggio dei record non inseriti correttamente. Livello parametro: Summary Unità: numero
`UserRecordsDataPut`	Byte inseriti correttamente nei record di utenti logici. Livello parametro: Detailed Unità: byte

Parametro	Descrizione
KinesisRecordsPut	Conteggio del numero di record del flusso di dati Kinesis inseriti correttamente (ogni record del flusso di dati Kinesis può contenere più record di utente). La KPL restituisce zero per i record non riusciti. In questo modo, la media offre la percentuale dei record inseriti correttamente, il conteggio offre il numero totale di tentativi e la differenza tra il conteggio e la somma offre il conteggio dei record non inseriti correttamente. Livello parametro: Summary Unità: numero
KinesisRecordsDataPut	Byte nei record del flusso di dati Kinesis. Livello parametro: Detailed Unità: byte
ErrorsByCode	Conteggio di ogni tipo di codice di errore. Ciò introduce un'ulteriore dimensione di ErrorCode , in aggiunta alle dimensioni normali come StreamName e ShardId. Non tutti gli errori possono essere rintracciati in uno shard. Gli errori che non possono essere rintracciati vengono emessi esclusivamente a livello di flusso o a livello globale. Questo parametro acquisisce informazioni su elementi come il throttling, le variazioni nella mappatura degli shard, gli errori interni, l'indisponibilità del servizio, i tempi di attesa e così via. Gli errori dell'API del flusso di dati Kinesis vengono conteggiati una volta per ogni record del flusso di dati Kinesis. Più record utente all'interno di un record di Flussi di dati Kinesis non generano conteggi multipli. Livello parametro: Summary Unità: numero

Parametro	Descrizione
`AllErrors`	Questo parametro viene attivato dagli stessi errori di Errors by Code, ma non distingue tra i tipi. È utile come monitoraggio generale della percentuale di errori, senza richiedere una somma manuale dei conteggi di tutti i diversi tipi di errori. Livello parametro: Summary Unità: numero
`RetriesPerRecord`	Numero di nuovi tentativi eseguiti per record di utente. Viene emesso uno zero per i record che risultano corretti in un solo tentativo. I dati vengono emessi nel momento in cui termina il record di un utente (se termina correttamente, non è possibile effettuare un nuovo tentativo). Se il valore record time-to-live è elevato, questa metrica potrebbe subire un notevole ritardo. Livello parametro: Detailed Unità: numero
`BufferingTime`	Il tempo tra l'arrivo di un record utente alla KPL e la sua uscita per il back-end. Queste informazioni sono nuovamente trasmesse all'utente per ogni record, ma sono disponibili anche sotto forma di statistica aggregata. Livello parametro: Summary Unità: millisecondi
`Request Time`	Tempo necessario per eseguire `PutRecordsRequests` . Livello parametro: Detailed Unità: millisecondi

Parametro	Descrizione
`User Records per Kinesis Record`	Il numero di record utente logici aggregati in un singolo record di Flussi di dati Kinesis. Livello parametro: Detailed Unità: numero
`Amazon Kinesis Records per PutRecord sRequest`	Il numero di record del flusso di dati Kinesis aggregati in un singolo `PutRecordsRequest` . Non disponibile a livello di shard. Livello parametro: Detailed Unità: numero
`User Records per PutRecord sRequest`	Numero totale di record di utenti contenuti all'interno di una `PutRecord sRequest` . È approssimativamente equivalente al prodotto dei due parametri precedenti. Non disponibile a livello di shard. Livello parametro: Detailed Unità: numero

Sicurezza in Flusso di dati Amazon Kinesis

Per, la sicurezza del cloud ha la massima priorità. AWS In quanto AWS cliente, puoi trarre vantaggio da un'architettura di data center e di rete progettata per soddisfare i requisiti delle aziende più esigenti a livello di sicurezza.

La sicurezza è una responsabilità condivisa tra AWS e l'utente. Il modello di responsabilità condivisa descrive questo come sicurezza del cloud e sicurezza nel cloud:

- Sicurezza del cloud: AWS è responsabile della protezione dell'infrastruttura che esegue AWS i servizi nel AWS cloud. AWS fornisce, inoltre, servizi che puoi utilizzare in modo sicuro. L'efficacia della nostra sicurezza è regolarmente testata e verificata da revisori di terze parti come parte dei programmi di conformitàAWS. Per ulteriori informazioni sui programmi di conformità che si applicano al flusso di dati Kinesis, consulta Servizi AWS coperti dal programma di conformità.
- Sicurezza nel cloud: la tua responsabilità è determinata dal AWS servizio che viene utilizzato. L'utente è anche responsabile per altri fattori, tra cui la riservatezza dei dati, i requisiti dell'azienda e leggi e normative applicabili.

Questa documentazione consente di comprendere come applicare il modello di responsabilità condivisa quando si usa . I seguenti argomenti illustrano come configurare per soddisfare gli obiettivi di sicurezza e conformità. È inoltre illustrato come utilizzare altri AWS servizi che consentono di monitorare e proteggere le risorse del flusso di dati Kinesis.

Argomenti

- Protezione dei dati in Flusso di dati Amazon Kinesis
- Controllo dell'accesso alle risorse del flusso di dati Amazon Kinesis tramite IAM
- Convalida della conformità per Flusso di dati Amazon Kinesis
- Resilienza in Flusso di dati Amazon Kinesis
- Sicurezza dell'infrastruttura nel flusso di dati Kinesis
- Best practice di sicurezza per il flusso di dati Kinesis

Protezione dei dati in Flusso di dati Amazon Kinesis

La crittografia lato server con le chiavi di AWS Key Management Service (AWS KMS) consente di rispettare in modo più agevole i rigidi requisiti di gestione dei dati tramite la crittografia dei dati a riposo in Flusso di dati Amazon Kinesis.

> **Note**
>
> Se si richiedono moduli crittografici convalidati FIPS 140-2 quando si accede ad AWS tramite un'interfaccia a riga di comando o un'API, utilizzare un endpoint FIPS. Per ulteriori informazioni sugli endpoint FIPS disponibili, consulta il Federal Information Processing Standard (FIPS) 140-2.

Argomenti

- Cos'è la crittografia lato server per Kinesis Streams?
- Costi, regioni e considerazioni sulle prestazioni
- Quali sono i primi passi per iniziare a utilizzare la crittografia lato server?
- Crea e utilizza chiavi KMS generate dall'utente
- Autorizzazioni per utilizzare le chiavi KMS generate dall'utente
- Verifica e risolvi i problemi relativi alle autorizzazioni delle chiavi KMS
- Utilizzo di flusso di dati Amazon Kinesis con endpoint VPC di interfaccia

Cos'è la crittografia lato server per Kinesis Streams?

La crittografia lato server è una funzionalità del flusso di dati Amazon Kinesis che crittografa automaticamente i dati prima che diventino a riposo utilizzando AWS KMS una chiave master del cliente (CMK) da te specificata. I dati vengono crittografati prima di essere scritti sul livello di storage del flusso di e vengono decrittografati dopo essere stati recuperati dallo storage. Di conseguenza, i dati vengono sempre crittografati mentre sono inattivi all'interno del servizio del flusso di dati Kinesis. In questo modo sarà possibile soddisfare severi requisiti normativi e migliorare la sicurezza dei dati.

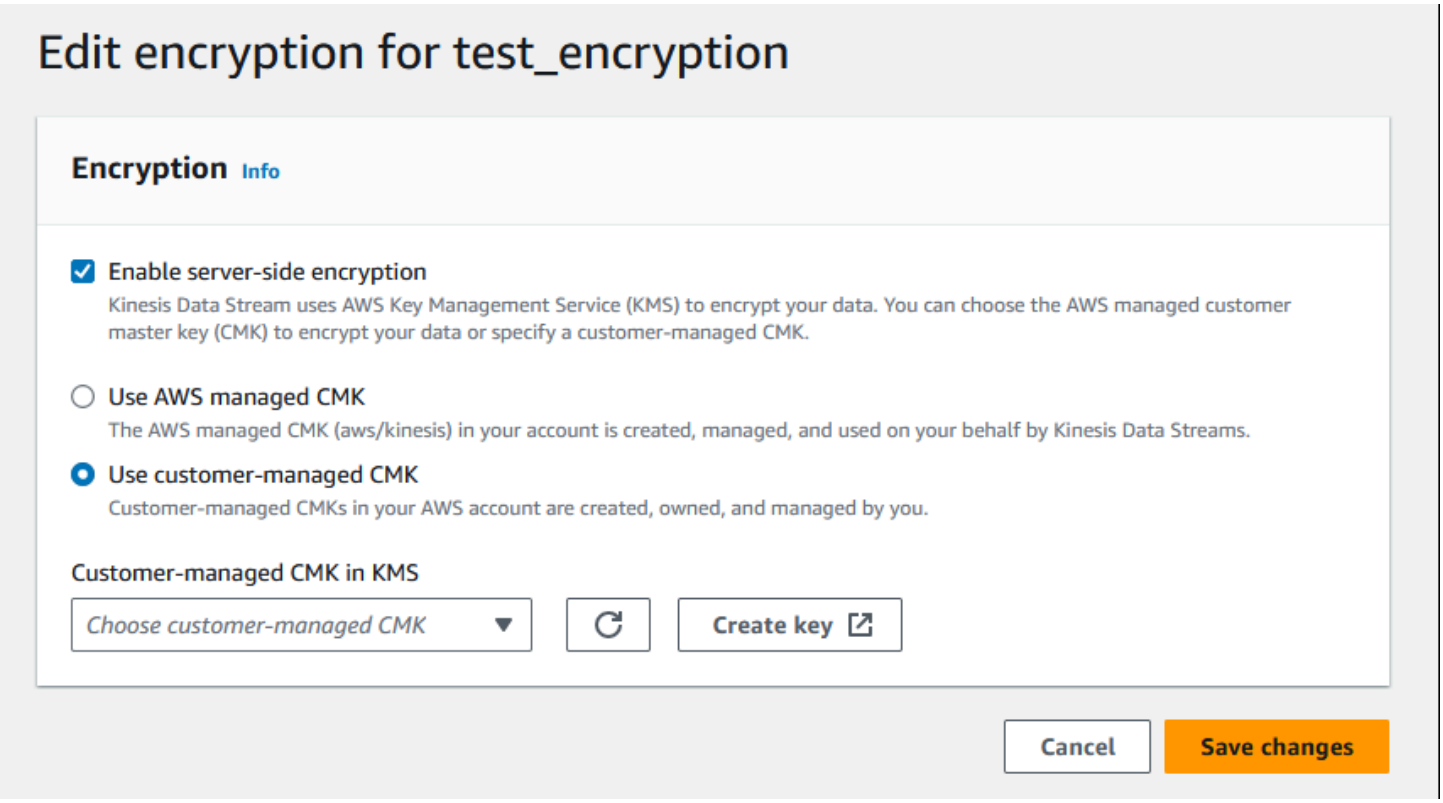
Grazie alla crittografia lato server, produttori e consumatori del flusso non devono gestire le chiavi master o le operazioni di crittografia. I dati vengono crittografati automaticamente quando entrano ed escono dal servizio Kinesis Data Streams, quindi i dati archiviati vengono crittografati. AWS KMS fornisce tutte le chiavi master utilizzate dalla funzionalità di crittografia lato server. AWS

KMS semplifica l'utilizzo di una CMK per Kinesis gestita AWS da, una CMK AWS KMS specificata dall'utente o una chiave master importata nel servizio. AWS KMS

> **Note**
>
> La crittografia lato server codifica i dati in entrata solo dopo l'attivazione della crittografia. I dati preesistenti in un flusso non crittografato non vengono crittografati dopo l'attivazione della crittografia lato server.

Quando si crittografano i flussi di dati e si condivide l'accesso ad altri principali, è necessario concedere l'autorizzazione sia nella policy della chiave per la AWS KMS chiave che nelle policy IAM nell'account esterno. Per ulteriori informazioni, consulta Autorizzazione per gli utenti in altri account a utilizzare una chiave KMS.

Se hai abilitato la crittografia lato server per un flusso di dati con chiave KMS AWS gestita da e desideri condividere l'accesso tramite una policy delle risorse, devi passare all'utilizzo della chiave gestita dal cliente (CMK), come illustrato di seguito:



Inoltre, devi consentire alle entità principali di condivisione di accedere alla CMK utilizzando le funzionalità di condivisione tra account di KMS. Ricorda di apportare la modifica anche alle policy IAM

per le entità principali di condivisione. Per ulteriori informazioni, consulta Autorizzazione per gli utenti in altri account a utilizzare una chiave KMS.

Costi, regioni e considerazioni sulle prestazioni

Quando si applica la crittografia lato server, si è soggetti all'utilizzo delle AWS KMS API e ai costi delle chiavi. A differenza delle chiavi master KMS personalizzate, la chiave master del cliente (CMK) `(Default) aws/kinesis` è disponibile gratuitamente. Tuttavia, occorre comunque corrispondere un pagamento per i costi di utilizzo dell'API affrontati da Flusso di dati Amazon Kinesis per tuo conto.

I costi di utilizzo dell'API vengono applicati per ogni CMK, incluse quelle personalizzate. Il flusso di dati Kinesis chiama AWS KMS ogni cinque minuti circa mentre ruota la chiave dati. In un mese di 30 giorni, il costo totale di chiamate AWS KMS API avviate da un flusso Kinesis dovrebbe pochi dollari. Questo costo varia in base al numero di credenziali utente che utilizzi sui produttori di dati e sui consumatori poiché ogni credenziale dell'utente richiede una chiamata API esclusiva a. AWS KMS Quando utilizzi un ruolo IAM per l'autenticazione, ogni chiamata di ruolo presunta ha come risultato credenziali utente univoche. Per risparmiare sui costi di KMS, puoi memorizzare nella cache le credenziali utente restituite dalla chiamata di ruolo presunta.

Di seguito vengono descritti i costi per risorsa:

Chiavi

- La CMK per Kinesis gestita AWS da (alias `aws/kinesis` =) è gratuita.
- Le chiavi di KMS generate dall'utente sono soggette ai costi delle chiavi di KMS. Per ulteriori informazioni, consulta Prezzi di AWS Key Management Service.

I costi di utilizzo dell'API vengono applicati per ogni CMK, incluse quelle personalizzate. Il flusso di dati Kinesis chiama KMS ogni cinque minuti circa mentre ruota la chiave dati. In un mese di 30 giorni, il costo totale di chiamate API KMS avviate da un flusso di dati Kinesis dovrebbe pochi dollari. Questo costo varia in base al numero di credenziali utente che utilizzi sui produttori di dati e sui consumatori poiché ogni credenziale dell'utente richiede una chiamata API esclusiva a. AWS Quando utilizzi il ruolo IAM per l'autenticazione, ognuno di essi assume-role-call genererà credenziali utente uniche e potresti voler memorizzare nella cache le credenziali utente restituite da assume-role-call per risparmiare sui costi KMS.

Utilizzo dell'API KMS

Per ogni flusso crittografato, quando si legge da TIP e si utilizza una singola chiave di accesso utente/account IAM tra lettori e scrittori, il servizio Kinesis chiama il servizio AWS KMS circa 12 volte ogni 5 minuti. La mancata lettura dal TIP potrebbe comportare un aumento delle chiamate al AWS KMS servizio. Le richieste API per generare nuove chiavi di crittografia dei dati sono soggette a costi di AWS KMS utilizzo. Per ulteriori informazioni, consulta la sezione relativa ai Prezzi di AWS Key Management Service: Utilizzo.

Disponibilità della crittografia lato server per regione

Attualmente, la crittografia lato server dei flussi Kinesis è disponibile in tutte le Regioni supportate dal flusso di dati Kinesis, incluse AWS GovCloud (Stati Uniti-Ovest) e le regioni Cina. Per ulteriori informazioni sulle regioni supportate per Kinesis Data https://docs.aws.amazon.com/general/latest/gr/akStreams, consulta .html.

Considerazioni sulle prestazioni

A causa del sovraccarico del servizio di applicazione della crittografia, l'applicazione della crittografia lato server aumenta la latenza tipica di `PutRecord`, `PutRecords` e `GetRecords` di meno di 100µs.

Quali sono i primi passi per iniziare a utilizzare la crittografia lato server?

Il modo più semplice per iniziare a utilizzare la crittografia lato server è utilizzare la AWS Management Console e la chiave di servizio KMS di Amazon Kinesis. `aws/kinesis`

La procedura seguente mostra come abilitare la crittografia lato server per un flusso .

Abilitazione della crittografia lato server per un flusso Kinesis

1. Accedi a AWS Management Console e apri la console Flusso di dati Amazon Kinesis.
2. Crea o seleziona un flusso Kinesis nella AWS Management Console.
3. Selezionare la scheda Details (Dettagli).
4. In Server-side encryption (Crittografia lato server), scegliere Edit (Modifica).
5. Se non si desidera utilizzare una chiave master KMS generata dall'utente, accertarsi che la chiave master KMS (Default) aws/kinesis (Predefinito aws/kinesis) sia selezionata. Questa è la chiave master KMS generata dal servizio . Selezionare Enabled (Abilitato), quindi Save (Salva).

> **Note**
>
> La chiave master del servizio Kinesis predefinito è gratuita, tuttavia, le chiamate API effettuate da Kinesis al AWS KMS servizio sono soggette ai costi di utilizzo KMS.

6. Le transizioni del flusso tramite uno stato in attesa. Una volta che il flusso ritorna allo stato attivo con crittografia attivata, tutti i dati in entrata scritti nel flusso vengono crittografati utilizzando la chiave master KMS selezionata.
7. Per disabilitare la crittografia lato server, scegli Disabilitata in Crittografia lato server in, quindi scegli Salva. AWS Management Console

Crea e utilizza chiavi KMS generate dall'utente

In questa sezione viene descritto come creare e utilizzare le chiavi KMS proprie piuttosto che utilizzare la chiave master amministrata da Amazon Kinesis.

Crea chiavi KMS generate dall'utente

Per istruzioni sulla creazione di chiavi personalizzate, consulta Creating Keys nella AWS Key Management Service Developer Guide. Dopo aver creato le chiavi per l'account, il servizio del flusso di dati Kinesis restituisce tali chiavi nell'elenco Chiave master KMS.

Usa chiavi KMS generate dall'utente

Dopo aver applicato le autorizzazioni corrette a consumer, producer e amministratori, puoi utilizzare le chiavi KMS personalizzate nel tuo AWS account o in un altro account. AWS Tutte le chiavi master di KMS del tuo account vengono visualizzate nell'elenco KMS Master Key (Chiave master di KMS) all'interno di AWS Management Console.

Per utilizzare le chiavi master KMS personalizzate che si trovano in un altro account, hai bisogno delle autorizzazioni per quelle chiavi. Inoltre, devi specificare l'ARN della chiave master KMS nella casella di input ARN in AWS Management Console.

Autorizzazioni per utilizzare le chiavi KMS generate dall'utente

Prima di utilizzare la crittografia lato server con una chiave KMS generata dall'utente, è necessario configurare le policy AWS KMS chiave per consentire la crittografia dei flussi e la crittografia e la

decrittografia dei record di flusso. Per esempi e ulteriori informazioni sulle AWS KMS autorizzazioni di, consulta la sezione relativa alle Autorizzazioni dell'API AWS KMS: Documentazione di riferimento per operazioni e risorse.

> **Note**
>
> L'utilizzo della chiave del servizio predefinita per la crittografia non richiede l'applicazione delle autorizzazioni &IAM; personalizzate.

Prima di utilizzare le chiavi master KMS generate dall'utente, assicurati che i producer e i consumer del flusso Kinesis (principali IAM) siano utenti nella policy della chiave master KMS. In caso contrario, le operazioni di lettura e scrittura dai flussi non riusciranno, causando perdite di dati, ritardi delle elaborazioni o blocchi delle applicazioni. Puoi gestire le autorizzazioni relative alle chiavi KMS tramite le policy IAM. Per ulteriori informazioni, consulta Utilizzo di policy IAM con AWS KMS.

Autorizzazioni di produttore di esempio

I producer del flusso Kinesis devono disporre dell'autorizzazione `kms:GenerateDataKey`.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
        "Effect": "Allow",
        "Action": [
            "kms:GenerateDataKey"
        ],
        "Resource": "arn:aws:kms:us-
west-2:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab"
    },
    {
        "Effect": "Allow",
        "Action": [
            "kinesis:PutRecord",
            "kinesis:PutRecords"
        ],
        "Resource": "arn:aws:kinesis:*:123456789012:MyStream"
    }
  ]
}
```

Autorizzazioni di consumo di esempio

I consumer del flusso Kinesis devono disporre dell'autorizzazione `kms:Decrypt`.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
        "Effect": "Allow",
        "Action": [
            "kms:Decrypt"
        ],
        "Resource": "arn:aws:kms:us-
west-2:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab"
    },
    {
        "Effect": "Allow",
        "Action": [
            "kinesis:GetRecords",
            "kinesis:DescribeStream"
        ],
        "Resource": "arn:aws:kinesis:*:123456789012:MyStream"
    }
  ]
}
```

Servizio gestito da Amazon per Apache Flink e AWS Lambda utilizzano i ruoli per utilizzare i flussi Kinesis. Assicurati di aggiungere le autorizzazioni `kms:Decrypt` per i ruoli utilizzati da tali consumatori.

Autorizzazioni di amministratore del flusso

Gli amministratori del flusso Kinesis devono avere l'autorizzazione per chiamare `kms:List*` e `kms:DescribeKey*`.

Verifica e risolvi i problemi relativi alle autorizzazioni delle chiavi KMS

Dopo aver abilitato la crittografia su un flusso Kinesis, ti consigliamo di monitorare il successo delle tue `putRecord` `getRecords` chiamate e delle chiamate utilizzando i seguenti parametri Amazon CloudWatch : `putRecords`

- `PutRecord.Success`

- `PutRecords.Success`
- `GetRecords.Success`

Per ulteriori informazioni, consulta Monitoraggio dei flussi di dati Kinesis

Utilizzo di flusso di dati Amazon Kinesis con endpoint VPC di interfaccia

È possibile usare un endpoint VPC di interfaccia per evitare che il traffico tra Amazon VPC e il flusso di dati Kinesis lasci la rete Amazon. Gli endpoint VPC dell'interfaccia non richiedono un gateway Internet, un dispositivo NAT, una connessione VPN o una connessione. AWS Direct Connect Gli endpoint VPC di interfaccia si basano su AWS PrivateLink, una AWS tecnologia che permette la comunicazione privata tra AWS servizi che utilizzano un'interfaccia di rete elastica con private nel IPs tuo Amazon VPC. Per ulteriori informazioni, consulta Amazon Virtual Private Cloud and Interface VPC Endpoints ().AWS PrivateLink

Argomenti

- Utilizzo di endpoint VPC di interfaccia per il flusso di dati Kinesis
- Controllo dell'accesso agli endpoint VPC per il flusso di dati Kinesis
- Disponibilità di policy di endpoint VPC per il flusso di dati Kinesis

Utilizzo di endpoint VPC di interfaccia per il flusso di dati Kinesis

Per iniziare, non è necessario cambiare le impostazioni per i flussi, i produttori o i consumatori. Creare un endpoint VPC di interfaccia per il flusso di dati Kinesis per avviare il flusso di traffico da e verso le risorse di Amazon VPC attraverso l'endpoint VPC dell'interfaccia. Gli endpoint VPC con interfaccia FIPS sono disponibili per le regioni degli Stati Uniti. Per ulteriori informazioni, consulta Creazione di un endpoint di interfaccia.

Amazon Kinesis Producer Library (KPL) e Kinesis Consumer Library (KCL) chiamano i servizi quali AWS Amazon e Amazon CloudWatch DynamoDB utilizzando endpoint pubblici o endpoint VPC dell'interfaccia privati, a seconda di cosa è in uso. Ad esempio se l'applicazione della KCL è in esecuzione su un VPC con un endpoint VPC dell'interfaccia DynamoDB abilitato, le chiamate fra DynamoDB e l'applicazione della KCL passano attraverso l'endpoint VPC dell'interfaccia.

Controllo dell'accesso agli endpoint VPC per il flusso di dati Kinesis

Le policy degli endpoint VPC consentono di controllare l'accesso allegando una policy a un endpoint VPC o utilizzando campi aggiuntivi in una policy allegata a un utente, gruppo o ruolo IAM per limitare l'accesso solo tramite l'endpoint VPC specificato. Utilizza queste policy per limitare l'accesso a flussi specifici a un endpoint VPC specifico quando le usi insieme alle policy IAM per concedere l'accesso solo alle azioni del flusso di dati Kinesis attraverso l'endpoint VPC specificato.

Di seguito sono riportati esempi di policy di endpoint per l'accesso ai flussi di dati Kinesis.

- Esempio di policy VPC: accesso in sola lettura: questa policy di esempio può essere collegata a un endpoint VPC. Per ulteriori informazioni, consulta Controllo dell'accesso alle risorse VPC di Amazon. Limita le operazioni solo all'elenco e alla descrizione di un flusso di dati Kinesis attraverso un endpoint VPC a cui è collegato.

```
{
  "Statement": [
    {
      "Sid": "ReadOnly",
      "Principal": "*",
      "Action": [
        "kinesis:List*",
        "kinesis:Describe*"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

- Esempio di policy VPC: limitare l'accesso a un determinato flusso di dati Kinesis – Questa policy di esempio può essere collegata a un endpoint VPC. Limita l'accesso a un flusso di dati specifico tramite l'endpoint VPC a cui è collegato.

```
{
  "Statement": [
    {
      "Sid": "AccessToSpecificDataStream",
      "Principal": "*",
      "Action": "kinesis:*",
```

```
      "Effect": "Allow",
      "Resource": "arn:aws:kinesis:us-east-1:123456789012:stream/MyStream"
    }
  ]
}
```

- Esempio di policy IAM: limita l'accesso a un flusso specifico solo da un endpoint VPC specifico: questa policy di esempio può essere associata a un utente, ruolo o gruppo IAM. Limita l'accesso a un flusso di dati Kinesis specificato solo da un endpoint VPC specificato.

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "AccessFromSpecificEndpoint",
         "Action": "kinesis:*",
         "Effect": "Deny",
         "Resource": "arn:aws:kinesis:us-east-1:123456789012:stream/MyStream",
         "Condition": { "StringNotEquals" : { "aws:sourceVpce": "vpce-11aa22bb" } }
      }
   ]
}
```

Disponibilità di policy di endpoint VPC per il flusso di dati Kinesis

Gli endpoint VPC dell'interfaccia del flusso di dati Kinesis con le policy sono supportati nelle seguenti Regioni:

- Europa (Parigi)
- Europa (Irlanda)
- Stati Uniti orientali (Virginia settentrionale)
- Europa (Stoccolma)
- Stati Uniti orientali (Ohio)
- Europa (Francoforte)
- Sud America (San Paolo)
- Europa (Londra)

- Asia Pacifico (Tokyo)
- Stati Uniti occidentali (California settentrionale)
- Asia Pacifico (Singapore)
- Asia Pacifico (Sydney)
- Cina (Pechino)
- Cina (Ningxia)
- Asia Pacifico (Hong Kong)
- Medio Oriente (Bahrein)
- Medio Oriente (Emirati Arabi Uniti)
- Europa (Milano)
- Africa (Città del Capo)
- Asia Pacifico (Mumbai)
- Asia Pacifico (Seoul)
- Canada (Centrale)
- Stati Uniti occidentali (Oregon) tranne usw2-az4
- AWS GovCloud (Stati Uniti orientali)
- AWS GovCloud (Stati Uniti occidentali)
- Asia Pacifico (Osaka-Locale)
- Europa (Zurigo)
- Asia Pacific (Hyderabad)

Controllo dell'accesso alle risorse del flusso di dati Amazon Kinesis tramite IAM

AWS Identity and Access Management (IAM) ti consente di fare quanto segue:

- Creare utenti e gruppi all'interno AWS dell'account
- Assegnare credenziali di sicurezza univoche a ciascun utente all'interno dell'account AWS
- Controllare le autorizzazioni di ciascun utente per eseguire attività utilizzando le risorse AWS
- Consentire agli utenti di altri AWS account di condividere AWS le tue risorse

- Creare ruoli per l' AWS account e definire gli utenti o i servizi che possono assumerli
- Utilizzare le identità esistenti nella tua impresa per concedere autorizzazioni per eseguire attività utilizzando risorse AWS

Utilizzando IAM con il flusso di dati Kinesis, puoi controllare se gli utenti all'interno dell'organizzazione sono in grado di eseguire un'attività utilizzando specifiche operazioni API del flusso di dati Kinesis e se possono utilizzare risorse AWS specifiche.

Se stai sviluppando un'applicazione utilizzando Kinesis Client Library (KCL), la policy dovrà includere le autorizzazioni per Amazon DynamoDB e Amazon CloudWatch; la KCL utilizza DynamoDB per tenere traccia delle informazioni sullo stato per l'applicazione e per inviare parametri di KCL per tuo conto. CloudWatch CloudWatch Per ulteriori informazioni sull'KCL, consulta Sviluppa i consumatori di KCL 1.x.

Per ulteriori informazioni su IAM, consulta:

- AWS Identity and Access Management (IAM)
- Nozioni di base
- Guida per l'utente di IAM

Per ulteriori informazioni sulla gestione dell'accesso ad Amazon DynamoDB, consulta Utilizzo di IAM per il controllo degli accessi alle risorse Amazon DynamoDB nella Guida per gli sviluppatori di Amazon DynamoDB.

Per ulteriori informazioni su IAM e Amazon CloudWatch, consulta Controlling User Access to Your AWS Account nella Amazon CloudWatch User Guide.

Indice

- Sintassi delle policy
- Operazioni per il flusso di dati Kinesis
- Nomi della risorsa Amazon (ARNs) per il flusso di dati Kinesis
- Esempi di policy per il flusso di dati Kinesis
- Condivisione del flusso di dati con un altro account
- Configura una AWS Lambda funzione per la lettura da Kinesis Data Streams in un altro account
- Condivisione dell'accesso utilizzando policy basate su risorse

Sintassi delle policy

Una policy IAM è un documento JSON costituito da una o più dichiarazioni. Ogni dichiarazione è strutturata come segue:

```
{
  "Statement":[{
    "Effect":"effect",
    "Action":"action",
    "Resource":"arn",
    "Condition":{
      "condition":{
        "key":"value"
        }
      }
    }
  ]
}
```

Una dichiarazione è costituita da diversi elementi:

- Effect (Effetto): l'elemento effect può essere `Allow` o `Deny`. Per impostazione predefinita, gli utenti IAM non dispongono dell'autorizzazione per l'utilizzo di risorse e operazioni API, pertanto tutte le richieste vengono rifiutate. Un permesso esplicito sostituisce l'impostazione predefinita. Un rifiuto esplicito sovrascrive tutti i consensi.
- Action (Operazione): l'elemento action corrisponde all'operazione API specifica per la quale si concede o si nega l'autorizzazione.
- Resource (Risorsa): la risorsa che viene modificata dall'operazione. Per specificare una risorsa nella dichiarazione, devi utilizzare il relativo Amazon Resource Name (ARN).
- Condition: le condizioni sono facoltative. Possono essere utilizzate per controllare quando sarà in vigore una policy.

Quando crei e gestisci policy &IAM;, puoi utilizzare gli strumenti Generatore di policy IAM e Simulatore di policy IAM.

Operazioni per il flusso di dati Kinesis

In una dichiarazione di policy IAM, è possibile specificare qualsiasi operazione API per qualsiasi servizio che supporta IAM. Per il flusso di dati Kinesis, utilizza il seguente prefisso con il nome

dell'operazione API: `kinesis:`. For example: `kinesis:CreateStream`, `kinesis:ListStreams` e `kinesis:DescribeStreamSummary`.

Per specificare più azioni in una sola istruzione, separa ciascuna di esse con una virgola come mostrato di seguito:

```
"Action": ["kinesis:action1", "kinesis:action2"]
```

Puoi anche specificare più operazioni tramite caratteri jolly. Ad esempio, puoi specificare tutte le operazioni il cui nome inizia con la parola "Get" come segue:

```
"Action": "kinesis:Get*"
```

Per specificare tutte le operazioni del flusso di dati Kinesis, utilizza il carattere jolly (*) come mostrato di seguito:

```
"Action": "kinesis:*"
```

Per l'elenco completo delle operazioni API del flusso di dati Kinesis, consulta la Documentazione di riferimento delle API di Amazon Kinesis.

Nomi della risorsa Amazon (ARNs) per il flusso di dati Kinesis

Ogni dichiarazione di policy IAM si applica alle risorse specificate utilizzando le relative ARNs.

Usa il seguente formato di risorsa ARN per il flusso di dati Kinesis:

```
arn:aws:kinesis:region:account-id:stream/stream-name
```

Per esempio:

```
"Resource": arn:aws:kinesis:*:111122223333:stream/my-stream
```

Esempi di policy per il flusso di dati Kinesis

Le policy di esempio seguenti dimostrano in che modo puoi controllare l'accesso degli utenti al flusso di dati Kinesis.

Example 1: Allow users to get data from a stream

Example

Questa policy consente a un utente o a un gruppo di eseguire le operazioni `DescribeStreamSummary`, `GetShardIterator` e `GetRecords` nel flusso specificato e `ListStreams` su qualsiasi flusso. Questa policy può essere applicata a utenti che devono essere in grado di ottenere dati da un determinato flusso.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kinesis:Get*",
                "kinesis:DescribeStreamSummary"
            ],
            "Resource": [
                "arn:aws:kinesis:us-east-1:111122223333:stream/stream1"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "kinesis:ListStreams"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

Example 2: Allow users to add data to any stream in the account

Example

Questa policy consente a un utente o un gruppo di utilizzare l'operazione `PutRecord` con uno qualsiasi dei flussi dell'account. Questa policy potrebbe essere applicata agli utenti che devono essere in grado di aggiungere record di dati a tutti i flussi in un account.

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kinesis:PutRecord"
            ],
            "Resource": [
                "arn:aws:kinesis:us-east-1:111122223333:stream/*"
            ]
        }
    ]
}
```

Example 3: Allow any Kinesis Data Streams action on a specific stream

Example

Questa policy consente a un utente o a un gruppo di utilizzare qualsiasi operazione del flusso di dati Kinesis sul flusso specificato. Questa policy potrebbe essere applicata agli utenti che devono avere il controllo amministrativo su un determinato flusso.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": [
                "arn:aws:kinesis:us-east-1:111122223333:stream/stream1"
            ]
        }
    ]
}
```

Example 4: Allow any Kinesis Data Streams action on any stream

Example

Questa policy consente a un utente o a un gruppo di utilizzare qualsiasi operazione del flusso di dati Kinesis su qualsiasi flusso in un account. Poiché questa policy concede l'accesso completo a tutti i flussi, devi limitarla ai soli amministratori.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": [
                "arn:aws:kinesis:*:111122223333:stream/*"
            ]
        }
    ]
}
```

Condivisione del flusso di dati con un altro account

> **Note**
>
> La Kinesis Producer Library attualmente non supporta la specificazione di un ARN di flusso durante la scrittura su un flusso di dati. Utilizza l' AWS SDK se desideri scrivere su un flusso di dati tra account.

Collega una policy basata sulle risorse al tuo flusso di dati per garantire l'accesso a un altro account, utente o ruolo IAM. Le policy basate sulle risorse sono documenti di policy JSON che colleghi a una risorsa come un flusso di dati. Queste policy concedono all'entità principale specificata l'autorizzazione per eseguire operazioni specifiche sulla risorsa e definiscono le condizioni in cui ciò si applica. Una policy può avere più dichiarazioni. È necessario specificare un principale in una policy basata sulle risorse. Le entità possono includere account, ruoli, utenti federati o AWS servizi. È possibile configurare le policy nella console, nell'API o nell'SDK di Kinesis Data Streams.

Ricorda che la condivisione dell'accesso ai consumatori registrati come Fan-out avanzato richiede una policy sia sull'ARN del flusso di dati sia sull'ARN del consumatore.

Abilitazione dell'accesso multi-account

Per consentire l'accesso multi-account, puoi specificare un intero account o entità IAM in un altro account come principale in una policy basata sulle risorse. L'aggiunta di un principale multi-account a una policy basata sulle risorse rappresenta solo una parte della relazione di trust. Quando l'entità

principale e la risorsa sono in AWS account separati, è necessario usare anche una policy basata su identità per concedere all'identità principale l'accesso alla risorsa. Tuttavia, se una policy basata su risorse concede l'accesso a un principale nello stesso account, non sono richieste ulteriori policy basate su identità.

Per ulteriori informazioni sull'utilizzo delle policy basate sulle risorse per l'accesso multi-account, consulta Accesso alle risorse multi-account in IAM.

Gli amministratori dei flussi di dati possono utilizzare AWS Identity and Access Management le policy per specificare chi ha accesso a cosa. In altre parole, quale principale può eseguire operazioni su quali risorse, e in quali condizioni. L'elemento `Action`di una policy JSON descrive le operazioni che è possibile utilizzare per consentire o negare l'accesso a un criterio. Le operazioni di policy hanno spesso lo stesso nome dell'operazione AWS API.

Operazioni di Kinesis Data Streams che possono essere condivise:

Azione	Livello di accesso
DescribeStreamConsumer	Consumer
DescribeStreamSummary	Flusso di dati
GetRecords	Flusso di dati
GetShardIterator	Flusso di dati
ListShards	Flusso di dati
PutRecord	Flusso di dati
PutRecords	Flusso di dati
SubscribeToShard	Consumer

Di seguito sono riportati alcuni esempi dell'utilizzo delle policy basate sulle risorse per concedere l'accesso multi-account al flusso di dati o al consumatore registrato.

Per eseguire un'operazione su più account, è necessario specificare l'ARN del flusso per l'accesso al flusso di dati e l'ARN consumatore per l'accesso dei consumatori registrati.

Esempi di policy basate sulle risorse per il flusso di dati Kinesis

La condivisione di un consumatore registrato implica sia una policy sul flusso di dati che una policy per i consumatori dovuta alle operazioni necessarie.

> **Note**
>
> Di seguito alcuni esempi dei valori validi per `Principal`:
>
> - `{"AWS": "123456789012"}`
> - Utente IAM – `{"AWS": "arn:aws:iam::123456789012:user/user-name"}`
> - Ruolo IAM – `{"AWS":["arn:aws:iam::123456789012:role/role-name"]}`
> - Principali multipli (può essere una combinazione di account, utente, ruolo) – `{"AWS": ["123456789012", "123456789013", "arn:aws:iam::123456789012:user/user-name"]}`

Example 1: Write access to the data stream

Example

```
{
    "Version": "2012-10-17",
    "Id": "__default_write_policy_ID",
    "Statement": [
        {
            "Sid": "writestatement",
            "Effect": "Allow",
            "Principal": {
                "AWS": "Account12345"
            },
            "Action": [
                "kinesis:DescribeStreamSummary",
                "kinesis:ListShards",
                "kinesis:PutRecord",
                "kinesis:PutRecords"
            ],
            "Resource": "arn:aws:kinesis:us-east-2:123456789012:stream/
datastreamABC"
        }
    ]
```

```
}
```

Example 2: Read access to the data stream

Example

```
{
    "Version": "2012-10-17",
    "Id": "__default_sharedthroughput_read_policy_ID",
    "Statement": [
        {
            "Sid": "sharedthroughputreadstatement",
            "Effect": "Allow",
            "Principal": {
                "AWS": "Account12345"
            },
            "Action": [
                "kinesis:DescribeStreamSummary",
                "kinesis:ListShards",
                "kinesis:GetRecords",
                "kinesis:GetShardIterator"
            ],
            "Resource": "arn:aws:kinesis:us-east-2:123456789012:stream/
datastreamABC"
        }
    ]
}
```

Example 3: Share enhanced fan-out read access to a registered consumer

Example

Dichiarazione sulla policy del flusso di dati:

```
{
    "Version": "2012-10-17",
    "Id": "__default_sharedthroughput_read_policy_ID",
    "Statement": [
        {
            "Sid": "consumerreadstatement",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::Account12345:role/role-name"
            },
```

```
            "Action": [
                "kinesis:DescribeStreamSummary",
                "kinesis:ListShards"
            ],
            "Resource": "arn:aws:kinesis:us-east-2:123456789012:stream/
datastreamABC"
        }
    ]
}
```

Dichiarazione della policy del consumatore:

```
{
    "Version": "2012-10-17",
    "Id": "__default_efo_read_policy_ID",
    "Statement": [
        {
            "Sid": "eforeadstatement",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::Account12345:role/role-name"
            },
            "Action": [
                "kinesis:DescribeStreamConsumer",
                "kinesis:SubscribeToShard"
            ],
            "Resource": "arn:aws:kinesis:us-east-2:123456789012:stream/
datastreamABC/consumer/consumerDEF:1674696300"
        }
    ]
}
```

Wildcard (*) non è supportato per operazioni o campi principali, per mantenere il principio del privilegio minimo.

Gestione programmatica della policy per il flusso di dati

Al di fuori della AWS Management Console, Kinesis Data Streams dispone di tre API per la gestione delle policy relative al flusso di dati:

- PutResourcePolicy

- GetResourcePolicy
- DeleteResourcePolicy

Utilizza `PutResourePolicy` per collegare o sovrascrivere una policy del flusso di dati o del consumatore. Utilizza `GetResourcePolicy` per controllare e visualizzare una policy del flusso di dati o del consumatore specificato. Utilizza `DeleteResourcePolicy` per eliminare una policy del flusso di dati o del consumatore specificato.

Limiti delle policy

Le policy delle risorse di Kinesis Data Streams prevedono le seguenti restrizioni:

- I Wildcard (*) non sono supportati per evitare che sia concesso un accesso più ampio attraverso le policy delle risorse direttamente collegate a un flusso di dati o a un consumatore registrato. Inoltre, controllare attentamente le seguenti policy per verificare che non concedano un ampio accesso:
  - Policy basate sull'identità collegate ai AWS principali associati (ad esempio, ruoli IAM)
  - Policy basate sulle risorse collegate alle AWS risorse associate di (ad esempio, AWS Key Management Service le chiavi KMS)
- AWS I principali del servizio non sono supportati per evitare potenziali «confusi».
- I principali federati non sono supportati.
- IDs Gli utenti canonici non sono supportati.
- La dimensione della policy non può superare i 20 KB.

Condivisione dell'accesso ai dati crittografati

Se hai abilitato la crittografia lato server per un flusso di dati con chiave KMS AWS gestita da e desideri condividere l'accesso tramite una policy delle risorse, devi passare all'utilizzo della chiave gestita dal cliente (CMK). Per ulteriori informazioni, consulta Cos'è la crittografia lato server per Kinesis Streams?. Inoltre, devi consentire alle entità principali di condivisione di accedere alla CMK utilizzando le funzionalità di condivisione tra account di KMS. Ricorda di apportare la modifica anche alle policy IAM per le entità principali di condivisione. Per ulteriori informazioni, consulta Autorizzazione per gli utenti in altri account a utilizzare una chiave KMS.

Configura una AWS Lambda funzione per la lettura da Kinesis Data Streams in un altro account

Per vedere un esempio di come configurare una funzione Lambda per la lettura da Kinesis Data Streams in un altro account, consulta Condivisione dell'accesso con funzioni su più account AWS Lambda.

Condivisione dell'accesso utilizzando policy basate su risorse

> **Note**
>
> Aggiornare una policy esistente basata sulle risorse significa sostituire quella già esistente, quindi assicurati di includere tutte le informazioni necessarie nella nuova policy.

Condivisione dell'accesso con funzioni su più account AWS Lambda

Operatore Lambda

1. Vai alla console IAM per creare un ruolo IAM che verrà utilizzato come ruolo di esecuzione Lambda per la tua AWS Lambda funzione. Aggiungere la policy IAM policy `AWSLambdaKinesisExecutionRole` gestita con le autorizzazioni di invocazione Kinesis Data Streams e Lambda richieste. Questa policy concede anche l'accesso a tutte le potenziali risorse di Kinesis Data Streams a cui potresti avere accesso.
2. Nella AWS Lambda console, crea una AWS Lambda funzione per elaborare i record in un flusso di dati Kinesis Data Streams e, durante la configurazione del ruolo di esecuzione, scegli il ruolo creato nel passaggio precedente.
3. Fornisci il ruolo di esecuzione al proprietario della risorsa Kinesis Data Streams per la configurazione della policy delle risorse.
4. Completa la configurazione della funzione Lambda.

Proprietario della risorsa di Kinesis Data Streams

1. Ottieni il ruolo di esecuzione Lambda multi-account che richiamerà la funzione Lambda.
2. Nella console di Amazon Kinesis Data Streams, scegli il flusso di dati. Scegli la scheda Condivisione del flusso di dati e poi il pulsante Crea una policy di condivisione per avviare l'editor visivo delle policy. Per condividere un consumatore registrato all'interno di un flusso di

dati, scegli il consumatore e poi seleziona Crea una policy di condivisione. Puoi anche scrivere direttamente la policy JSON.

3. Specifica il ruolo di esecuzione Lambda multi-account come principale e le operazioni Kinesis Data Streams esatte a cui condividi l'accesso. Ricorda di includere l'operazione `kinesis:DescribeStream`. Per ulteriori informazioni su esempi di policy delle risorse per Kinesis Data Streams, consulta Esempi di policy basate sulle risorse per il flusso di dati Kinesis.
4. Scegli Crea policy o usa PutResourcePolicyper allegare la policy alla tua risorsa.

Condivisione dell'accesso con i consumatori KCL su più account

- Se utilizzi KCL 1.x, assicurati di usare la versione KCL 1.15.0 o successive.
- Se utilizzi KCL 2.x, assicurati di usare la versione KCL 2.5.3 o successive.

Operatore KCL

1. Fornisci al proprietario della risorsa il tuo utente IAM o il ruolo IAM che eseguirà l'applicazione KCL.
2. Chiedi al proprietario della risorsa il flusso di dati o l'ARN consumatore.
3. Ricorda di specificare l'ARN del flusso fornito come parte della configurazione KCL.
   - Per KCL 1.x: utilizza il KinesisClientLibConfigurationcostruttore e fornisci l'ARN del flusso.
   - Per KCL 2.x: puoi fornire solo l'ARN dello stream o StreamTrackerla Kinesis Client Library. ConfigsBuilder Per StreamTracker, fornisci l'ARN del flusso e l'Epoch creazione dalla DynamoDB Lease Table generata dalla libreria. Se vuoi leggere da un consumatore registrato condiviso come Fan-out migliorato, utilizza StreamTracker e fornisci anche l'ARN consumatore.

Proprietario della risorsa di Kinesis Data Streams

1. Ottieni l'utente IAM o il ruolo IAM multi-account che eseguirà l'applicazione KCL.
2. Nella console di Amazon Kinesis Data Streams, scegli il flusso di dati. Scegli la scheda Condivisione del flusso di dati e poi il pulsante Crea una policy di condivisione per avviare l'editor visivo delle policy. Per condividere un consumatore registrato all'interno di un flusso di dati, scegli il consumatore e poi seleziona Crea una policy di condivisione. Puoi anche scrivere direttamente la policy JSON.

3. Specifica l'utente IAM o il ruolo IAM dell'applicazione KCL su più account come le operazioni Kinesis Data Streams principali ed esatte a cui condividi l'accesso. Per ulteriori informazioni su esempi di policy delle risorse per Kinesis Data Streams, consulta Esempi di policy basate sulle risorse per il flusso di dati Kinesis.
4. Scegli Crea policy o usa PutResourcePolicyper allegare la policy alla tua risorsa.

Condivisione dell'accesso ai dati crittografati

Se hai abilitato la crittografia lato server per un flusso di dati con chiave KMS AWS gestita da e desideri condividere l'accesso tramite una policy delle risorse, devi passare all'utilizzo della chiave gestita dal cliente (CMK). Per ulteriori informazioni, consulta Cos'è la crittografia lato server per Kinesis Streams?. Inoltre, devi consentire alle entità principali di condivisione di accedere alla CMK utilizzando le funzionalità di condivisione tra account di KMS. Ricorda di apportare la modifica anche alle policy IAM per le entità principali di condivisione. Per ulteriori informazioni, consulta Autorizzazione per gli utenti in altri account a utilizzare una chiave KMS.

Convalida della conformità per Flusso di dati Amazon Kinesis

Revisori di terze parti valutano la sicurezza e la conformità di Flusso di dati Amazon Kinesis come parte di più programmi di conformità di. AWS Questi includono SOC, PCI, FedRAMP, HIPAA e altri.

Per un elenco di AWS servizi che rientrano nell'ambito di programmi di conformità specifici, consulta AWS Servizi coperti dal programma di compliance. Per informazioni generali, consulta Programmi di conformità di AWS.

Puoi scaricare i report di audit di terze parti utilizzando AWS Artifact. Per ulteriori informazioni, consulta Download dei report in AWS Artifact.

La tua responsabilità di conformità durante l'utilizzo di è determinata dalla riservatezza dei dati, dagli obiettivi dell'azienda e dalle leggi e normative applicabili. Se l'utilizzo del flusso di dati Kinesis è soggetto a conformità con standard quali HIPAA, PCI o FedRAMP, fornisce risorse utili AWS :

- Guide Quick Start per la sicurezza e conformità: queste guide all'implementazione illustrano considerazioni relative all'architettura e forniscono fasi per l'implementazione di ambienti di base incentrati sulla sicurezza e sulla conformità su. AWS
- Whitepaper sull'architettura per compliance e sicurezza HIPAA: questo whitepaper descrive in che modo le aziende possono utilizzare per creare applicazioni conformi ai requisiti HIPAA. AWS

- AWS Risorse per la conformità: questa raccolta di cartelle di lavoro e guide potrebbe essere utile per il settore e la posizione.
- AWS Config: questo AWS servizio valuta il livello di conformità delle configurazioni delle risorse con best practice interne, linee guida e regolamenti industriali.
- AWS Security Hub: questo AWS servizio fornisce una visione completa dello stato di sicurezza all'interno di AWS che consente di verificare la conformità con standard industriali di sicurezza e best practice.

Resilienza in Flusso di dati Amazon Kinesis

L'infrastruttura AWS globale di è basata su AWS regioni e zone di disponibilità. AWS Le regioni forniscono più zone di disponibilità fisicamente separate e isolate che sono connesse tramite reti altamente ridondanti, a bassa latenza e throughput elevato. Con le zone di disponibilità, è possibile progettare e gestire applicazioni e database che eseguono il failover automatico tra zone di disponibilità senza interruzioni. Le zone di disponibilità sono più disponibili, tolleranti ai guasti e scalabili rispetto alle infrastrutture tradizionali a data center singolo o multiplo.

Per ulteriori informazioni sulle AWS regioni e sulle zone di disponibilità, consulta Infrastruttura AWS globale di.

Oltre all'infrastruttura AWS globale, il flusso di dati Kinesis offre numerose funzionalità per supportare la resilienza dei dati e le esigenze di backup.

Ripristino di emergenza in Flusso di dati Amazon Kinesis

È possibile che si verifichino errori ai seguenti livelli quando si utilizza un'applicazione flusso di dati Amazon Kinesis per elaborare i dati da un flusso:

- Errore di un processore di record
- Errore di un lavoratore o errore dell'istanza dell'applicazione che ha istanziato il lavoratore
- Un' EC2 istanza che ospita una o più istanze dell'applicazione potrebbe fallire

Errore del processore di registrazione

L'operatore richiama i metodi del processore di registrazione utilizzando le attività Java ExecutorService. Se si verifica un errore di un'attività, il lavoratore mantiene il controllo dello shard

che il processore di record stava elaborando. Il lavoratore avvia una nuova attività del processore di record per elaborare il suddetto shard. Per ulteriori informazioni, consulta Limitazione della lettura.

Errore del lavoratore o dell'applicazione

In caso di errore di un worker o di un'istanza di flusso di dati Amazon Kinesis, è necessario rilevare e gestire la situazione. Ad esempio, se il metodo `Worker.run` genera un'eccezione, è necessario identificarla e gestirla.

In caso di errore dell'applicazione stessa, è necessario rilevarlo e riavviare l'applicazione. Quando l'applicazione si avvia, avvia un'istanza di un nuovo lavoratore, che a sua volta avvia un'istanza di nuovi processori di record ai quali vengono automaticamente assegnati shard da elaborare. Questi potrebbero essere gli stessi shard che questi processori di record stavano elaborando prima dell'errore o shard che sono nuovi per questi processori.

In una situazione in cui il lavoratore o l'applicazione si guasta, l'errore non viene rilevato e vi sono altre istanze dell'applicazione in esecuzione su altre EC2 istanze, i lavoratori di queste altre istanze gestiscono l'errore. Creano processori di record aggiuntivi per elaborare gli shard che non sono più elaborati dal lavoro che ha prodotto l'errore. Il carico su queste altre EC2 istanze aumenta di conseguenza.

Lo scenario descritto qui presuppone che, anche in caso di errore del worker o dell'applicazione, l' EC2 istanza di hosting è ancora in esecuzione e pertanto non viene riavviata da un gruppo con dimensionamento automatico.

Errore dell' EC2 istanza Amazon

È consigliabile eseguire EC2 le istanze per la tua applicazione in un gruppo con dimensionamento automatico. In questo modo, in caso di errore di una delle EC2 istanze, il gruppo con dimensionamento automatico avvia automaticamente una nuova istanza in sostituzione. È necessario configurare le istanze per avviare l'applicazione Flusso di dati Amazon Kinesis all'avvio.

Sicurezza dell'infrastruttura nel flusso di dati Kinesis

In qualità di servizio gestito, Flusso di dati Amazon Kinesis è protetto AWS dalle procedure di sicurezza di rete globale descritte nel whitepaper Amazon Web Services: Overview of Security Processes (Amazon Web Services: Panoramica dei processi di sicurezza).

Utilizza le chiamate API AWS pubblicate di per accedere al flusso di dati Kinesis tramite la rete. I client devono supportare Transport Layer Security (TLS) 1.2 o versioni successive. I client devono,

inoltre, supportare le suite di cifratura con PFS (Perfect Forward Secrecy), ad esempio Ephemeral Diffie-Hellman (DHE) o Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). La maggior parte dei sistemi moderni, come Java 7 e versioni successive, supporta tali modalità.

Inoltre, le richieste devono essere firmate utilizzando un ID chiave di accesso e una chiave di accesso segreta associata a un principale IAM. O puoi utilizzare AWS Security Token Service (AWS STS) per generare credenziali di sicurezza temporanee per sottoscrivere le richieste.

Best practice di sicurezza per il flusso di dati Kinesis

Flusso di dati Amazon Kinesis fornisce una serie di funzionalità di sicurezza che occorre valutare durante lo sviluppo e l'implementazione delle policy di sicurezza. Le seguenti best practice sono linee guida generali e non rappresentano una soluzione di sicurezza completa. Poiché queste best practice potrebbero non essere appropriate o sufficienti per l'ambiente, gestiscile come considerazioni utili anziché prescrizioni.

Implementazione dell'accesso con privilegi minimi

Attribuite le autorizzazioni, puoi decidere chi ottiene tali autorizzazioni e verso quali risorse del flusso di dati Kinesis. È possibile abilitare operazioni specifiche che si desidera consentire su tali risorse. Pertanto è necessario concedere solo le autorizzazioni necessarie per eseguire un'attività. L'implementazione dell'accesso con privilegi minimi è fondamentale per ridurre i rischi di sicurezza e l'impatto risultante da errori o intenzioni dannose.

Uso di ruoli IAM

Le applicazioni producer e client devono avere credenziali valide per accedere ai flussi di dati . Non devi archiviare AWS credenziali direttamente in un'applicazione client o in un bucket Amazon S3. Si tratta di credenziali a lungo termine che non vengono automaticamente ruotate e potrebbero avere un impatto aziendale significativo se vengono compromesse.

Al contrario, è consigliabile utilizzare un ruolo IAM per gestire le credenziali temporanee per le applicazioni producer e client per accedere ai flussi di dati Kinesis. Quando utilizzi un ruolo, non devi necessariamente usare credenziali a lungo termine (ad esempio, nome utente e password o chiavi di accesso) per accedere ad altre risorse.

Per ulteriori informazioni, consulta gli argomenti seguenti nella Guida per l'utente IAM:

- Ruoli IAM

- Scenari comuni per ruoli: utenti, applicazioni e servizi

Implementazione della crittografia lato server in risorse dipendenti

I dati a riposo e i dati in transito possono essere crittografati nel flusso di dati Kinesis. Per ulteriori informazioni, consulta Protezione dei dati in Flusso di dati Amazon Kinesis.

CloudTrail Da utilizzare per monitorare le chiamate API

Il flusso di dati Kinesis è AWS CloudTrail integrato con, un servizio che offre un record delle operazioni eseguite da un utente, un ruolo AWS o un servizio nel flusso di dati Kinesis.

Le informazioni raccolte da consentono CloudTrail di determinare la richiesta effettuata a Kinesis Data Streams, l'indirizzo IP da cui è partita la richiesta, l'autore della richiesta, il momento in cui è stata eseguita e altri dettagli.

Per ulteriori informazioni, consulta the section called “Registra le chiamate API di flusso di dati Amazon Kinesis tramite AWS CloudTrail”.

Utilizzo del servizio con un AWS SDK

AWS I software Development Kit (SDKs) sono disponibili per molti linguaggi di programmazione popolari. Ogni SDK fornisce un'API, esempi di codice, e documentazione che facilitano agli sviluppatori la creazione di applicazioni nel loro linguaggio preferito.

Documentazione sugli SDK	Esempi di codice
AWS SDK per C++	AWS SDK per C++ esempi di codice
AWS CLI	AWS CLI esempi di codice
AWS SDK per Go	AWS SDK per Go esempi di codice
AWS SDK per Java	AWS SDK per Java esempi di codice
AWS SDK per JavaScript	AWS SDK per JavaScript esempi di codice
AWS SDK per Kotlin	AWS SDK per Kotlin esempi di codice
AWS SDK per .NET	AWS SDK per .NET esempi di codice
AWS SDK per PHP	AWS SDK per PHP esempi di codice
AWS Strumenti per PowerShell	Strumenti per esempi di PowerShell codice
AWS SDK per Python (Boto3)	AWS SDK per Python (Boto3) esempi di codice
AWS SDK per Ruby	AWS SDK per Ruby esempi di codice
AWS SDK for Rust	AWS SDK for Rust esempi di codice
SDK AWS per SAP ABAP	SDK AWS per SAP ABAP esempi di codice
SDK AWS per Swift	SDK AWS per Swift esempi di codice

Esempio di disponibilità

Non riesci a trovare quello che ti serve? Richiedi un esempio di codice utilizzando il link Provide feedback (Fornisci un feedback) nella parte inferiore di questa pagina.

Esempi di codice per l'utilizzo di Kinesis AWS SDKs

I seguenti esempi di codice mostrano come utilizzare Kinesis con un kit di sviluppo AWS software (SDK).

Le nozioni di base sono esempi di codice che mostrano come eseguire le operazioni essenziali all'interno di un servizio.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Utilizzo del servizio con un AWS SDK Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Esempi di codice

- Esempi di base per l'utilizzo di Kinesis AWS SDKs
  - Scopri le basi di Kinesis con un SDK AWS
  - Azioni per l'utilizzo di Kinesis AWS SDKs
    - Utilizzo AddTagsToStream con un AWS SDK o una CLI
    - Utilizzo CreateStream con un AWS SDK o una CLI
    - Utilizzo DeleteStream con un AWS SDK o una CLI
    - Utilizzo DeregisterStreamConsumer con un AWS SDK o una CLI
    - Utilizzo DescribeStream con un AWS SDK o una CLI
    - Utilizzo GetRecords con un AWS SDK o una CLI
    - Utilizzare GetShardIterator con una CLI
    - Utilizzare ListStreamConsumers con un SDK AWS
    - Utilizzo ListStreams con un AWS SDK o una CLI
    - Utilizzo ListTagsForStream con un AWS SDK o una CLI
    - Utilizzo PutRecord con un AWS SDK o una CLI
    - Utilizzo PutRecords con un AWS SDK o una CLI
    - Utilizzo RegisterStreamConsumer con un AWS SDK o una CLI
- Esempi serverless per Kinesis

- Richiamare una funzione Lambda da un trigger Kinesis
- Segnalazione di errori di elementi batch per funzioni Lambda con un trigger Kinesis

Esempi di base per l'utilizzo di Kinesis AWS SDKs

I seguenti esempi di codice mostrano come utilizzare le nozioni di base di Amazon AWS SDKs Kinesis con.

Esempi

- Scopri le basi di Kinesis con un SDK AWS
- Azioni per l'utilizzo di Kinesis AWS SDKs
  - Utilizzo AddTagsToStream con un AWS SDK o una CLI
  - Utilizzo CreateStream con un AWS SDK o una CLI
  - Utilizzo DeleteStream con un AWS SDK o una CLI
  - Utilizzo DeregisterStreamConsumer con un AWS SDK o una CLI
  - Utilizzo DescribeStream con un AWS SDK o una CLI
  - Utilizzo GetRecords con un AWS SDK o una CLI
  - Utilizzare GetShardIterator con una CLI
  - Utilizzare ListStreamConsumers con un SDK AWS
  - Utilizzo ListStreams con un AWS SDK o una CLI
  - Utilizzo ListTagsForStream con un AWS SDK o una CLI
  - Utilizzo PutRecord con un AWS SDK o una CLI
  - Utilizzo PutRecords con un AWS SDK o una CLI
  - Utilizzo RegisterStreamConsumer con un AWS SDK o una CLI

Scopri le basi di Kinesis con un SDK AWS

L'esempio di codice seguente mostra come:

- Crea uno stream e inserisci un record.
- Crea un iteratore shard.
- Leggi il record, quindi ripulisci le risorse.

SAP ABAP

SDK per SAP ABAP

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
   DATA lo_stream_describe_result TYPE REF TO /aws1/cl_knsdescrstreamoutput.
   DATA lo_stream_description TYPE REF TO /aws1/cl_knsstreamdescription.
   DATA lo_sharditerator TYPE REF TO /aws1/cl_knsgetsharditerator01.
   DATA lo_record_result TYPE REF TO /aws1/cl_knsputrecordoutput.

   "Create stream."
   TRY.
       lo_kns->createstream(
           iv_streamname = iv_stream_name
           iv_shardcount = iv_shard_count ).
       MESSAGE 'Stream created.' TYPE 'I'.
     CATCH /aws1/cx_knsinvalidargumentex.
       MESSAGE 'The specified argument was not valid.' TYPE 'E'.
     CATCH /aws1/cx_knslimitexceededex.
       MESSAGE 'The request processing has failed because of a limit exceeded
exception.' TYPE 'E'.
     CATCH /aws1/cx_knsresourceinuseex.
       MESSAGE 'The request processing has failed because the resource is in
use.' TYPE 'E'.
   ENDTRY.

   "Wait for stream to becomes active."
   lo_stream_describe_result = lo_kns->describestream( iv_streamname =
iv_stream_name ).
   lo_stream_description = lo_stream_describe_result->get_streamdescription( ).
   WHILE lo_stream_description->get_streamstatus( ) <> 'ACTIVE'.
     IF sy-index = 30.
       EXIT.               "maximum 5 minutes"
     ENDIF.
     WAIT UP TO 10 SECONDS.
```

```
      lo_stream_describe_result = lo_kns->describestream( iv_streamname =
 iv_stream_name ).
      lo_stream_description = lo_stream_describe_result-
>get_streamdescription( ).
    ENDWHILE.

    "Create record."
    TRY.
        lo_record_result = lo_kns->putrecord(
            iv_streamname = iv_stream_name
            iv_data       = iv_data
            iv_partitionkey = iv_partition_key ).
        MESSAGE 'Record created.' TYPE 'I'.
      CATCH /aws1/cx_knsinvalidargumentex.
        MESSAGE 'The specified argument was not valid.' TYPE 'E'.
      CATCH /aws1/cx_knskmsaccessdeniedex.
        MESSAGE 'You do not have permission to perform this AWS KMS action.' TYPE
 'E'.
      CATCH /aws1/cx_knskmsdisabledex.
        MESSAGE 'KMS key used is disabled.' TYPE 'E'.
      CATCH /aws1/cx_knskmsinvalidstateex.
        MESSAGE 'KMS key used is in an invalid state. ' TYPE 'E'.
      CATCH /aws1/cx_knskmsnotfoundex.
        MESSAGE 'KMS key used is not found.' TYPE 'E'.
      CATCH /aws1/cx_knskmsoptinrequired.
        MESSAGE 'KMS key option is required.' TYPE 'E'.
      CATCH /aws1/cx_knskmsthrottlingex.
        MESSAGE 'The rate of requests to AWS KMS is exceeding the request
 quotas.' TYPE 'E'.
      CATCH /aws1/cx_knsprovthruputexcdex.
        MESSAGE 'The request rate for the stream is too high, or the requested
 data is too large for the available throughput.' TYPE 'E'.
      CATCH /aws1/cx_knsresourcenotfoundex.
        MESSAGE 'Resource being accessed is not found.' TYPE 'E'.
    ENDTRY.

    "Create a shard iterator in order to read the record."
    TRY.
        lo_sharditerator = lo_kns->getsharditerator(
          iv_shardid = lo_record_result->get_shardid( )
          iv_sharditeratortype = iv_sharditeratortype
          iv_streamname = iv_stream_name ).
        MESSAGE 'Shard iterator created.' TYPE 'I'.
      CATCH /aws1/cx_knsinvalidargumentex.
```

```
        MESSAGE 'The specified argument was not valid.' TYPE 'E'.
      CATCH /aws1/cx_knsprovthruputexcdex.
        MESSAGE 'The request rate for the stream is too high, or the requested
data is too large for the available throughput.' TYPE 'E'.
      CATCH /aws1/cx_sgmresourcenotfound.
        MESSAGE 'Resource being accessed is not found.' TYPE 'E'.
    ENDTRY.

    "Read the record."
    TRY.
        oo_result = lo_kns->getrecords(                    " oo_result is
returned for testing purposes. "
            iv_sharditerator   = lo_sharditerator->get_sharditerator( ) ).
        MESSAGE 'Shard iterator created.' TYPE 'I'.
      CATCH /aws1/cx_knsexpirediteratorex.
        MESSAGE 'Iterator expired.' TYPE 'E'.
      CATCH /aws1/cx_knsinvalidargumentex.
        MESSAGE 'The specified argument was not valid.' TYPE 'E'.
      CATCH /aws1/cx_knskmsaccessdeniedex.
        MESSAGE 'You do not have permission to perform this AWS KMS action.' TYPE
'E'.
      CATCH /aws1/cx_knskmsdisabledex.
        MESSAGE 'KMS key used is disabled.' TYPE 'E'.
      CATCH /aws1/cx_knskmsinvalidstateex.
        MESSAGE 'KMS key used is in an invalid state. ' TYPE 'E'.
      CATCH /aws1/cx_knskmsnotfoundex.
        MESSAGE 'KMS key used is not found.' TYPE 'E'.
      CATCH /aws1/cx_knskmsoptinrequired.
        MESSAGE 'KMS key option is required.' TYPE 'E'.
      CATCH /aws1/cx_knskmsthrottlingex.
        MESSAGE 'The rate of requests to AWS KMS is exceeding the request
quotas.' TYPE 'E'.
      CATCH /aws1/cx_knsprovthruputexcdex.
        MESSAGE 'The request rate for the stream is too high, or the requested
data is too large for the available throughput.' TYPE 'E'.
      CATCH /aws1/cx_knsresourcenotfoundex.
        MESSAGE 'Resource being accessed is not found.' TYPE 'E'.
    ENDTRY.

    "Delete stream."
    TRY.
        lo_kns->deletestream(
            iv_streamname = iv_stream_name ).
        MESSAGE 'Stream deleted.' TYPE 'I'.
```

```
     CATCH /aws1/cx_knslimitexceededex.
       MESSAGE 'The request processing has failed because of a limit exceeded
exception.' TYPE 'E'.
     CATCH /aws1/cx_knsresourceinuseex.
       MESSAGE 'The request processing has failed because the resource is in
use.' TYPE 'E'.
   ENDTRY.
```

- Per informazioni dettagliate sulle API, consulta i seguenti argomenti nella Documentazione di riferimento delle API SDK AWS per SAP ABAP.
  - CreateStream
  - DeleteStream
  - GetRecords
  - GetShardIterator
  - PutRecord

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUtilizzo del servizio con un AWS SDK. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Azioni per l'utilizzo di Kinesis AWS SDKs

I seguenti esempi di codice mostrano come eseguire singole azioni Kinesis con. AWS SDKs Ogni esempio include un collegamento a GitHub, dove puoi trovare le istruzioni per la configurazione e l'esecuzione del codice.

Gli esempi seguenti includono solo le operazioni più comunemente utilizzate. Per un elenco completo, consulta Amazon Kinesis API Reference.

Esempi

- Utilizzo AddTagsToStream con un AWS SDK o una CLI
- Utilizzo CreateStream con un AWS SDK o una CLI
- Utilizzo DeleteStream con un AWS SDK o una CLI
- Utilizzo DeregisterStreamConsumer con un AWS SDK o una CLI
- Utilizzo DescribeStream con un AWS SDK o una CLI
- Utilizzo GetRecords con un AWS SDK o una CLI

- Utilizzare GetShardIterator con una CLI
- Utilizzare ListStreamConsumers con un SDK AWS
- Utilizzo ListStreams con un AWS SDK o una CLI
- Utilizzo ListTagsForStream con un AWS SDK o una CLI
- Utilizzo PutRecord con un AWS SDK o una CLI
- Utilizzo PutRecords con un AWS SDK o una CLI
- Utilizzo RegisterStreamConsumer con un AWS SDK o una CLI

Utilizzo **`AddTagsToStream`** con un AWS SDK o una CLI

Gli esempi di codice seguenti mostrano come utilizzare `AddTagsToStream`.

.NET

SDK per .NET

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.Kinesis;
using Amazon.Kinesis.Model;

/// <summary>
/// This example shows how to apply key/value pairs to an Amazon Kinesis
/// stream.
/// </summary>
public class TagStream
{
    public static async Task Main()
    {
        IAmazonKinesis client = new AmazonKinesisClient();

        string streamName = "AmazonKinesisStream";
```

```
            var tags = new Dictionary<string, string>
            {
                { "Project", "Sample Kinesis Project" },
                { "Application", "Sample Kinesis App" },
            };

            var success = await ApplyTagsToStreamAsync(client, streamName, tags);

            if (success)
            {
                Console.WriteLine($"Taggs successfully added to {streamName}.");
            }
            else
            {
                Console.WriteLine("Tags were not added to the stream.");
            }
        }

        /// <summary>
        /// Applies the set of tags to the named Kinesis stream.
        /// </summary>
        /// <param name="client">The initialized Kinesis client.</param>
        /// <param name="streamName">The name of the Kinesis stream to which
        /// the tags will be attached.</param>
        /// <param name="tags">A sictionary containing key/value pairs which
        /// will be used to create the Kinesis tags.</param>
        /// <returns>A Boolean value which represents the success or failure
        /// of AddTagsToStreamAsync.</returns>
        public static async Task<bool> ApplyTagsToStreamAsync(
            IAmazonKinesis client,
            string streamName,
            Dictionary<string, string> tags)
        {
            var request = new AddTagsToStreamRequest
            {
                StreamName = streamName,
                Tags = tags,
            };

            var response = await client.AddTagsToStreamAsync(request);

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
    }
```

- Per i dettagli sull'API, consulta la AddTagsToStreamsezione AWS SDK per .NET API Reference.

CLI

AWS CLI

Per aggiungere tag a un flusso di dati

L'`add-tags-to-stream`esempio seguente assegna un tag con la chiave `samplekey` e il valore `example` allo stream specificato.

```
aws kinesis add-tags-to-stream \
    --stream-name samplestream \
    --tags samplekey=example
```

Questo comando non produce alcun output.

Per ulteriori informazioni, consulta Tagging Your Streams nella Amazon Kinesis Data Streams Developer Guide.

- Per i dettagli sull'API, consulta AddTagsToStreamCommand Reference.AWS CLI

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUtilizzo del servizio con un AWS SDK. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`CreateStream`** con un AWS SDK o una CLI

Gli esempi di codice seguenti mostrano come utilizzare `CreateStream`.

Gli esempi di operazioni sono estratti di codice da programmi più grandi e devono essere eseguiti nel contesto. È possibile visualizzare questa operazione nel contesto nel seguente esempio di codice:

- Informazioni di base

.NET

SDK per .NET

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
using System;
using System.Threading.Tasks;
using Amazon.Kinesis;
using Amazon.Kinesis.Model;

/// <summary>
/// This example shows how to create a new Amazon Kinesis stream.
/// </summary>
public class CreateStream
{
    public static async Task Main()
    {
        IAmazonKinesis client = new AmazonKinesisClient();

        string streamName = "AmazonKinesisStream";
        int shardCount = 1;

        var success = await CreateNewStreamAsync(client, streamName,
shardCount);
        if (success)
        {
            Console.WriteLine($"The stream, {streamName} successfully
created.");
        }
    }

    /// <summary>
    /// Creates a new Kinesis stream.
    /// </summary>
    /// <param name="client">An initialized Kinesis client.</param>
    /// <param name="streamName">The name for the new stream.</param>
    /// <param name="shardCount">The number of shards the new stream will
```

```
        /// use. The throughput of the stream is a function of the number of
        /// shards; more shards are required for greater provisioned
        /// throughput.</param>
        /// <returns>A Boolean value indicating whether the stream was created.</
returns>
        public static async Task<bool> CreateNewStreamAsync(IAmazonKinesis
 client, string streamName, int shardCount)
        {
            var request = new CreateStreamRequest
            {
                StreamName = streamName,
                ShardCount = shardCount,
            };

            var response = await client.CreateStreamAsync(request);

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
    }
```

- Per i dettagli sull'API, consulta la CreateStreamsezione AWS SDK per .NET API Reference.

CLI

AWS CLI

Per creare un flusso di dati

L'`create-stream`esempio seguente crea un flusso di dati denominato samplestream con 3 shard.

```
aws kinesis create-stream \
    --stream-name samplestream \
    --shard-count 3
```

Questo comando non produce alcun output.

Per ulteriori informazioni, consulta Creating a Stream nella Amazon Kinesis Data Streams Developer Guide.

- Per i dettagli sull'API, consulta AWS CLI Command CreateStreamReference.

Java

SDK per Java 2.x

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.kinesis.KinesisClient;
import software.amazon.awssdk.services.kinesis.model.CreateStreamRequest;
import software.amazon.awssdk.services.kinesis.model.KinesisException;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class CreateDataStream {
    public static void main(String[] args) {

        final String usage = """

                Usage:
                    <streamName>

                Where:
                    streamName - The Amazon Kinesis data stream (for example,
 StockTradeStream).
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }
```

```
        String streamName = args[0];
        Region region = Region.US_EAST_1;
        KinesisClient kinesisClient = KinesisClient.builder()
                .region(region)
                .build();
        createStream(kinesisClient, streamName);
        System.out.println("Done");
        kinesisClient.close();
    }

    public static void createStream(KinesisClient kinesisClient, String
 streamName) {
        try {
            CreateStreamRequest streamReq = CreateStreamRequest.builder()
                    .streamName(streamName)
                    .shardCount(1)
                    .build();

            kinesisClient.createStream(streamReq);

        } catch (KinesisException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- Per i dettagli sull'API, consulta la CreateStreamsezione AWS SDK for Java 2.x API Reference.

PowerShell

Strumenti per PowerShell

Esempio 1: crea un nuovo stream. Per impostazione predefinita, questo cmdlet non restituisce alcun output, pertanto viene aggiunto lo PassThru switch - per restituire il valore fornito al StreamName parametro - per un uso successivo.

```
$streamName = New-KINStream -StreamName "mystream" -ShardCount 1 -PassThru
```

- Per i dettagli sull'API, vedere CreateStreamin AWS Strumenti per PowerShell Cmdlet Reference.

Python

SDK per Python (Boto3)

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
class KinesisStream:
    """Encapsulates a Kinesis stream."""

    def __init__(self, kinesis_client):
        """
        :param kinesis_client: A Boto3 Kinesis client.
        """
        self.kinesis_client = kinesis_client
        self.name = None
        self.details = None
        self.stream_exists_waiter = kinesis_client.get_waiter("stream_exists")


    def create(self, name, wait_until_exists=True):
        """
        Creates a stream.

        :param name: The name of the stream.
        :param wait_until_exists: When True, waits until the service reports that
                                  the stream exists, then queries for its
 metadata.
        """
        try:
            self.kinesis_client.create_stream(StreamName=name, ShardCount=1)
            self.name = name
            logger.info("Created stream %s.", name)
            if wait_until_exists:
                logger.info("Waiting until exists.")
```

```
                self.stream_exists_waiter.wait(StreamName=name)
                self.describe(name)
        except ClientError:
            logger.exception("Couldn't create stream %s.", name)
            raise
```

- Per i dettagli sull'API, consulta CreateStream AWSSDK for Python (Boto3) API Reference.

Rust

SDK per Rust

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
async fn make_stream(client: &Client, stream: &str) -> Result<(), Error> {
    client
        .create_stream()
        .stream_name(stream)
        .shard_count(4)
        .send()
        .await?;

    println!("Created stream");

    Ok(())
}
```

- Per i dettagli sulle API, consulta la CreateStreamguida di riferimento all'API AWS SDK for Rust.

SAP ABAP

SDK per SAP ABAP

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
    TRY.
        lo_kns->createstream(
            iv_streamname = iv_stream_name
            iv_shardcount = iv_shard_count ).
        MESSAGE 'Stream created.' TYPE 'I'.
      CATCH /aws1/cx_knsinvalidargumentex.
        MESSAGE 'The specified argument was not valid.' TYPE 'E'.
      CATCH /aws1/cx_knslimitexceededex.
        MESSAGE 'The request processing has failed because of a limit exceed
exception.' TYPE 'E'.
      CATCH /aws1/cx_knsresourceinuseex.
        MESSAGE 'The request processing has failed because the resource is in
use.' TYPE 'E'.
    ENDTRY.
```

- Per i dettagli sulle API, CreateStreamconsulta AWS SDK for SAP ABAP API reference.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Utilizzo del servizio con un AWS SDK Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`DeleteStream`** con un AWS SDK o una CLI

Gli esempi di codice seguenti mostrano come utilizzare `DeleteStream`.

Gli esempi di operazioni sono estratti di codice da programmi più grandi e devono essere eseguiti nel contesto. È possibile visualizzare questa operazione nel contesto nel seguente esempio di codice:

- Informazioni di base

.NET

SDK per .NET

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
using System;
using System.Threading.Tasks;
using Amazon.Kinesis;
using Amazon.Kinesis.Model;

/// <summary>
/// Shows how to delete an Amazon Kinesis stream.
/// </summary>
public class DeleteStream
{
    public static async Task Main()
    {
        IAmazonKinesis client = new AmazonKinesisClient();
        string streamName = "AmazonKinesisStream";

        var success = await DeleteStreamAsync(client, streamName);

        if (success)
        {
            Console.WriteLine($"Stream, {streamName} successfully deleted.");
        }
        else
        {
            Console.WriteLine("Stream not deleted.");
        }
    }

    /// <summary>
    /// Deletes a Kinesis stream.
    /// </summary>
    /// <param name="client">An initialized Kinesis client object.</param>
    /// <param name="streamName">The name of the string to delete.</param>
```

```
        /// <returns>A Boolean value representing the success of the operation.</
returns>
        public static async Task<bool> DeleteStreamAsync(IAmazonKinesis client,
 string streamName)
        {
            // If EnforceConsumerDeletion is true, any consumers
            // of this stream will also be deleted. If it is set
            // to false and this stream has any consumers, the
            // call will fail with a ResourceInUseException.
            var request = new DeleteStreamRequest
            {
                StreamName = streamName,
                EnforceConsumerDeletion = true,
            };

            var response = await client.DeleteStreamAsync(request);

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
    }
```

- Per i dettagli sull'API, consulta la DeleteStreamsezione AWS SDK per .NET API Reference.

CLI

AWS CLI

Per eliminare un flusso di dati

L'`delete-stream`esempio seguente elimina il flusso di dati specificato.

```
aws kinesis delete-stream \
    --stream-name samplestream
```

Questo comando non produce alcun output.

Per ulteriori informazioni, consulta Eliminazione di uno stream nella Amazon Kinesis Data Streams Developer Guide.

- Per i dettagli sull'API, consulta Command DeleteStreamReference AWS CLI .

Java

SDK per Java 2.x

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.kinesis.KinesisClient;
import software.amazon.awssdk.services.kinesis.model.DeleteStreamRequest;
import software.amazon.awssdk.services.kinesis.model.KinesisException;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DeleteDataStream {

    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <streamName>

                Where:
                    streamName - The Amazon Kinesis data stream (for example,
 StockTradeStream)
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }
```

```
        String streamName = args[0];
        Region region = Region.US_EAST_1;
        KinesisClient kinesisClient = KinesisClient.builder()
                .region(region)
                .build();

        deleteStream(kinesisClient, streamName);
        kinesisClient.close();
        System.out.println("Done");
    }

    public static void deleteStream(KinesisClient kinesisClient, String
 streamName) {
        try {
            DeleteStreamRequest delStream = DeleteStreamRequest.builder()
                    .streamName(streamName)
                    .build();

            kinesisClient.deleteStream(delStream);

        } catch (KinesisException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- Per i dettagli sull'API, consulta la DeleteStreamsezione AWS SDK for Java 2.x API Reference.

PowerShell

Strumenti per PowerShell

Esempio 1: elimina lo stream specificato. Viene richiesta una conferma prima dell'esecuzione del comando. Per sopprimere la richiesta di conferma, utilizzare l'interruttore -Force.

```
Remove-KINStream -StreamName "mystream"
```

- Per i dettagli sull'API, vedere DeleteStreamin AWS Strumenti per PowerShell Cmdlet Reference.

Python

SDK per Python (Boto3)

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
class KinesisStream:
    """Encapsulates a Kinesis stream."""

    def __init__(self, kinesis_client):
        """
        :param kinesis_client: A Boto3 Kinesis client.
        """
        self.kinesis_client = kinesis_client
        self.name = None
        self.details = None
        self.stream_exists_waiter = kinesis_client.get_waiter("stream_exists")


    def delete(self):
        """
        Deletes a stream.
        """
        try:
            self.kinesis_client.delete_stream(StreamName=self.name)
            self._clear()
            logger.info("Deleted stream %s.", self.name)
        except ClientError:
            logger.exception("Couldn't delete stream %s.", self.name)
            raise
```

- Per i dettagli sull'API, consulta DeleteStream AWSSDK for Python (Boto3) API Reference.

Rust

SDK per Rust

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
async fn remove_stream(client: &Client, stream: &str) -> Result<(), Error> {
    client.delete_stream().stream_name(stream).send().await?;

    println!("Deleted stream.");

    Ok(())
}
```

- Per i dettagli sulle API, consulta la DeleteStreamguida di riferimento all'API AWS SDK for Rust.

SAP ABAP

SDK per SAP ABAP

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
    TRY.
        lo_kns->deletestream(
            iv_streamname = iv_stream_name ).
        MESSAGE 'Stream deleted.' TYPE 'I'.
      CATCH /aws1/cx_knslimitexceededex.
        MESSAGE 'The request processing has failed because of a limit exceed
 exception.' TYPE 'E'.
      CATCH /aws1/cx_knsresourceinuseex.
```

```
        MESSAGE 'The request processing has failed because the resource is in
 use.' TYPE 'E'.
    ENDTRY.
```

- Per i dettagli sulle API, DeleteStreamconsulta AWS SDK for SAP ABAP API reference.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Utilizzo del servizio con un AWS SDK Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`DeregisterStreamConsumer`** con un AWS SDK o una CLI

Gli esempi di codice seguenti mostrano come utilizzare `DeregisterStreamConsumer`.

.NET

SDK per .NET

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
    using System;
    using System.Threading.Tasks;
    using Amazon.Kinesis;
    using Amazon.Kinesis.Model;

    /// <summary>
    /// Shows how to deregister a consumer from an Amazon Kinesis stream.
    /// </summary>
    public class DeregisterConsumer
    {
        public static async Task Main(string[] args)
        {
            IAmazonKinesis client = new AmazonKinesisClient();

            string streamARN = "arn:aws:kinesis:us-west-2:000000000000:stream/
AmazonKinesisStream";
```

```
            string consumerName = "CONSUMER_NAME";
            string consumerARN = "arn:aws:kinesis:us-west-2:000000000000:stream/
AmazonKinesisStream/consumer/CONSUMER_NAME:000000000000";

            var success = await DeregisterConsumerAsync(client, streamARN,
 consumerARN, consumerName);

            if (success)
            {
                Console.WriteLine($"{consumerName} successfully deregistered.");
            }
            else
            {
                Console.WriteLine($"{consumerName} was not successfully
 deregistered.");
            }
        }

        /// <summary>
        /// Deregisters a consumer from a Kinesis stream.
        /// </summary>
        /// <param name="client">An initialized Kinesis client object.</param>
        /// <param name="streamARN">The ARN of a Kinesis stream.</param>
        /// <param name="consumerARN">The ARN of the consumer.</param>
        /// <param name="consumerName">The name of the consumer.</param>
        /// <returns>A Boolean value representing the success of the operation.</
returns>
        public static async Task<bool> DeregisterConsumerAsync(
            IAmazonKinesis client,
            string streamARN,
            string consumerARN,
            string consumerName)
        {
            var request = new DeregisterStreamConsumerRequest
            {
                StreamARN = streamARN,
                ConsumerARN = consumerARN,
                ConsumerName = consumerName,
            };

            var response = await client.DeregisterStreamConsumerAsync(request);

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

```
    }
```

- Per i dettagli sull'API, consulta la DeregisterStreamConsumersezione AWS SDK per .NET API Reference.

CLI

AWS CLI

Per annullare la registrazione di un consumatore con flusso di dati

L'`deregister-stream-consumer`esempio seguente annulla la registrazione del consumatore specificato dal flusso di dati specificato.

```
aws kinesis deregister-stream-consumer \
    --stream-arn arn:aws:kinesis:us-west-2:123456789012:stream/samplestream \
    --consumer-name KinesisConsumerApplication
```

Questo comando non produce alcun output.

Per ulteriori informazioni, consulta Developing Consumer with Enhanced Fan-Out Using the Kinesis Data Streams API nella Amazon Kinesis Data Streams Developer Guide.

- Per i dettagli sull'API, consulta Command Reference. DeregisterStreamConsumerAWS CLI

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUtilizzo del servizio con un AWS SDK. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`DescribeStream`** con un AWS SDK o una CLI

Gli esempi di codice seguenti mostrano come utilizzare `DescribeStream`.

CLI

AWS CLI

Per descrivere un flusso di dati

L'`describe-stream`esempio seguente restituisce i dettagli del flusso di dati specificato.

```
aws kinesis describe-stream \
    --stream-name samplestream
```

Output:

```
{
    "StreamDescription": {
        "Shards": [
            {
                "ShardId": "shardId-000000000000",
                "HashKeyRange": {
                    "StartingHashKey": "0",
                    "EndingHashKey": "113427455640312821154458202477256070484"
                },
                "SequenceNumberRange": {
                    "StartingSequenceNumber":
 "496008716829570364423650249261910734372510605801286533 14"
                }
            },
            {
                "ShardId": "shardId-000000000001",
                "HashKeyRange": {
                    "StartingHashKey": "113427455640312821154458202477256070485",
                    "EndingHashKey": "226854911280625642308916404954512140969"
                },
                "SequenceNumberRange": {
                    "StartingSequenceNumber":
 "496008716829793371875635555493326091555237089416346337 46"
                }
            },
            {
                "ShardId": "shardId-000000000002",
                "HashKeyRange": {
                    "StartingHashKey": "226854911280625642308916404954512140970",
                    "EndingHashKey": "340282366920938463463374607431768211455"
                },
                "SequenceNumberRange": {
                    "StartingSequenceNumber":
 "496008716830016379327620861724741448737963573031406141 78"
                }
            }
```

```
        ],
        "StreamARN": "arn:aws:kinesis:us-west-2:123456789012:stream/
samplestream",
        "StreamName": "samplestream",
        "StreamStatus": "ACTIVE",
        "RetentionPeriodHours": 24,
        "EnhancedMonitoring": [
            {
                "ShardLevelMetrics": []
            }
        ],
        "EncryptionType": "NONE",
        "KeyId": null,
        "StreamCreationTimestamp": 1572297168.0
    }
}
```

Per ulteriori informazioni, consulta Creating and Managing Streams nella Amazon Kinesis Data Streams Developer Guide.

- Per i dettagli sull'API, consulta Command DescribeStreamReference AWS CLI .

PowerShell

Strumenti per PowerShell

Esempio 1: restituisce i dettagli dello stream specificato.

```
Get-KINStream -StreamName "mystream"
```

Output:

```
HasMoreShards        : False
RetentionPeriodHours : 24
Shards               : {}
StreamARN            : arn:aws:kinesis:us-west-2:123456789012:stream/mystream
StreamName           : mystream
StreamStatus         : ACTIVE
```

- Per i dettagli sull'API, vedere DescribeStreamin AWS Strumenti per PowerShell Cmdlet Reference.

Python

SDK per Python (Boto3)

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
class KinesisStream:
    """Encapsulates a Kinesis stream."""

    def __init__(self, kinesis_client):
        """
        :param kinesis_client: A Boto3 Kinesis client.
        """
        self.kinesis_client = kinesis_client
        self.name = None
        self.details = None
        self.stream_exists_waiter = kinesis_client.get_waiter("stream_exists")


    def describe(self, name):
        """
        Gets metadata about a stream.

        :param name: The name of the stream.
        :return: Metadata about the stream.
        """
        try:
            response = self.kinesis_client.describe_stream(StreamName=name)
            self.name = name
            self.details = response["StreamDescription"]
            logger.info("Got stream %s.", name)
        except ClientError:
            logger.exception("Couldn't get %s.", name)
            raise
        else:
            return self.details
```

- Per i dettagli sull'API, consulta DescribeStream AWSSDK for Python (Boto3) API Reference.

Rust

SDK per Rust

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
async fn show_stream(client: &Client, stream: &str) -> Result<(), Error> {
    let resp = client.describe_stream().stream_name(stream).send().await?;

    let desc = resp.stream_description.unwrap();

    println!("Stream description:");
    println!("  Name:              {}:", desc.stream_name());
    println!("  Status:            {:?}", desc.stream_status());
    println!("  Open shards:       {:?}", desc.shards.len());
    println!("  Retention (hours): {}", desc.retention_period_hours());
    println!("  Encryption:        {:?}", desc.encryption_type.unwrap());

    Ok(())
}
```

- Per i dettagli sulle API, consulta la DescribeStreamguida di riferimento all'API AWS SDK for Rust.

SAP ABAP

SDK per SAP ABAP

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
    TRY.
        oo_result = lo_kns->describestream(
            iv_streamname = iv_stream_name ).
        DATA(lt_stream_description) = oo_result->get_streamdescription( ).
        MESSAGE 'Streams retrieved.' TYPE 'I'.
      CATCH /aws1/cx_knslimitexceededex.
        MESSAGE 'The request processing has failed because of a limit exceed
exception.' TYPE 'E'.
      CATCH /aws1/cx_knsresourcenotfoundex.
        MESSAGE 'Resource being accessed is not found.' TYPE 'E'.
    ENDTRY.
```

- Per i dettagli sulle API, DescribeStreamconsulta AWS SDK for SAP ABAP API reference.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Utilizzo del servizio con un AWS SDK Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`GetRecords`** con un AWS SDK o una CLI

Gli esempi di codice seguenti mostrano come utilizzare `GetRecords`.

Gli esempi di operazioni sono estratti di codice da programmi più grandi e devono essere eseguiti nel contesto. È possibile visualizzare questa operazione nel contesto nel seguente esempio di codice:

- Informazioni di base

CLI

AWS CLI

Per ottenere record da uno shard

L'`get-records`esempio seguente ottiene i record di dati dallo shard di un flusso di dati Kinesis utilizzando l'iteratore shard specificato.

```
aws kinesis get-records \
    --shard-iterator AAAAAAAAAAF7/0mWD7IuHj1yGv/
TKuNgx2ukD5xipCY4cy4gU96orWwZwcSXh3K9tAmGYeOZyLZrvzzeOFVf9iN99hUPw/w/
```

```
b0YWYeehfNvnf1DYt5XpDJghLKr3DzgznkTmMymDP3R+3wRKeuEw6/kdxY2yKJH0veaiekaVc4N2VwK/
GvaGP2Hh9Fg7N++q0Adg6fIDQPt4p8RpavDbk+A4sL9SWGE1
```

Output:

```
{
    "Records": [],
    "MillisBehindLatest": 80742000
}
```

Per ulteriori informazioni, consulta Developing Consumers Using the Kinesis Data Streams API AWS with the SDK for Java nella Amazon Kinesis Data Streams Developer Guide.

- Per i dettagli sull'API, consulta Command Reference. GetRecordsAWS CLI

Java

SDK per Java 2.x

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.kinesis.KinesisClient;
import software.amazon.awssdk.services.kinesis.model.DescribeStreamResponse;
import software.amazon.awssdk.services.kinesis.model.DescribeStreamRequest;
import software.amazon.awssdk.services.kinesis.model.Shard;
import software.amazon.awssdk.services.kinesis.model.GetShardIteratorRequest;
import software.amazon.awssdk.services.kinesis.model.GetShardIteratorResponse;
import software.amazon.awssdk.services.kinesis.model.Record;
import software.amazon.awssdk.services.kinesis.model.GetRecordsRequest;
import software.amazon.awssdk.services.kinesis.model.GetRecordsResponse;
import java.util.ArrayList;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
```

```
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class GetRecords {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <streamName>

                Where:
                    streamName - The Amazon Kinesis data stream to read from (for
 example, StockTradeStream).
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String streamName = args[0];
        Region region = Region.US_EAST_1;
        KinesisClient kinesisClient = KinesisClient.builder()
                .region(region)
                .build();

        getStockTrades(kinesisClient, streamName);
        kinesisClient.close();
    }

    public static void getStockTrades(KinesisClient kinesisClient, String
 streamName) {
        String shardIterator;
        String lastShardId = null;
        DescribeStreamRequest describeStreamRequest =
 DescribeStreamRequest.builder()
                .streamName(streamName)
                .build();

        List<Shard> shards = new ArrayList<>();
```

```
        DescribeStreamResponse streamRes;
        do {
            streamRes = kinesisClient.describeStream(describeStreamRequest);
            shards.addAll(streamRes.streamDescription().shards());

            if (shards.size() > 0) {
                lastShardId = shards.get(shards.size() - 1).shardId();
            }
        } while (streamRes.streamDescription().hasMoreShards());

        GetShardIteratorRequest itReq = GetShardIteratorRequest.builder()
                .streamName(streamName)
                .shardIteratorType("TRIM_HORIZON")
                .shardId(lastShardId)
                .build();

        GetShardIteratorResponse shardIteratorResult =
 kinesisClient.getShardIterator(itReq);
        shardIterator = shardIteratorResult.shardIterator();

        // Continuously read data records from shard.
        List<Record> records;

        // Create new GetRecordsRequest with existing shardIterator.
        // Set maximum records to return to 1000.
        GetRecordsRequest recordsRequest = GetRecordsRequest.builder()
                .shardIterator(shardIterator)
                .limit(1000)
                .build();

        GetRecordsResponse result = kinesisClient.getRecords(recordsRequest);

        // Put result into record list. Result may be empty.
        records = result.records();

        // Print records
        for (Record record : records) {
            SdkBytes byteBuffer = record.data();
            System.out.printf("Seq No: %s - %s%n", record.sequenceNumber(), new
 String(byteBuffer.asByteArray()));
        }
    }
}
```

- Per i dettagli sull'API, consulta la GetRecordssezione AWS SDK for Java 2.x API Reference.

PowerShell

Strumenti per PowerShell

Esempio 1: Questo esempio mostra come restituire ed estrarre dati da una serie di uno o più record. L'iteratore supplierd to Get-KINRecord determina la posizione iniziale dei record da restituire, che in questo esempio vengono acquisiti in una variabile, $records. È quindi possibile accedere a ogni singolo record indicizzando la raccolta $records. Supponendo che i dati nel record siano testo con codifica UTF-8, il comando finale mostra come estrarre i dati dall'oggetto e MemoryStream restituirli come testo alla console.

```
$records
$records = Get-KINRecord -ShardIterator "AAAAAAAAAAGIc....9VnbiRNaP"
```

Output:

```
MillisBehindLatest NextShardIterator            Records
------------------ -----------------            -------
0                  AAAAAAAAAAERNIq...uDn11HuUs  {Key1, Key2}
```

```
$records.Records[0]
```

Output:

```
ApproximateArrivalTimestamp Data                    PartitionKey SequenceNumber
--------------------------- ----                    ------------ --------------
3/7/2016 5:14:33 PM         System.IO.MemoryStream Key1
 4955986459776...931586
```

```
[Text.Encoding]::UTF8.GetString($records.Records[0].Data.ToArray())
```

Output:

```
test data from string
```

- Per i dettagli sull'API, vedere GetRecordsin Cmdlet Reference.AWS Strumenti per PowerShell

Python

SDK per Python (Boto3)

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
class KinesisStream:
    """Encapsulates a Kinesis stream."""

    def __init__(self, kinesis_client):
        """
        :param kinesis_client: A Boto3 Kinesis client.
        """
        self.kinesis_client = kinesis_client
        self.name = None
        self.details = None
        self.stream_exists_waiter = kinesis_client.get_waiter("stream_exists")


    def get_records(self, max_records):
        """
        Gets records from the stream. This function is a generator that first
 gets
        a shard iterator for the stream, then uses the shard iterator to get
 records
        in batches from the stream. The shard iterator can be accessed through
 the
        'details' property, which is populated using the 'describe' function of
 this class.
        Each batch of records is yielded back to the caller until the specified
        maximum number of records has been retrieved.

        :param max_records: The maximum number of records to retrieve.
        :return: Yields the current batch of retrieved records.
```

```
        """
        try:
            response = self.kinesis_client.get_shard_iterator(
                StreamName=self.name,
                ShardId=self.details["Shards"][0]["ShardId"],
                ShardIteratorType="LATEST",
            )
            shard_iter = response["ShardIterator"]
            record_count = 0
            while record_count < max_records:
                response = self.kinesis_client.get_records(
                    ShardIterator=shard_iter, Limit=10
                )
                shard_iter = response["NextShardIterator"]
                records = response["Records"]
                logger.info("Got %s records.", len(records))
                record_count += len(records)
                yield records
        except ClientError:
            logger.exception("Couldn't get records from stream %s.", self.name)
            raise



    def describe(self, name):
        """
        Gets metadata about a stream.

        :param name: The name of the stream.
        :return: Metadata about the stream.
        """
        try:
            response = self.kinesis_client.describe_stream(StreamName=name)
            self.name = name
            self.details = response["StreamDescription"]
            logger.info("Got stream %s.", name)
        except ClientError:
            logger.exception("Couldn't get %s.", name)
            raise
        else:
            return self.details
```

- Per i dettagli sull'API, consulta GetRecords AWSSDK for Python (Boto3) API Reference.

SAP ABAP

SDK per SAP ABAP

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
   TRY.
       oo_result = lo_kns->getrecords(             " oo_result is returned for
testing purposes. "
           iv_sharditerator = iv_shard_iterator ).
       DATA(lt_records) = oo_result->get_records( ).
       MESSAGE 'Record retrieved.' TYPE 'I'.
     CATCH /aws1/cx_knsexpirediteratorex.
       MESSAGE 'Iterator expired.' TYPE 'E'.
     CATCH /aws1/cx_knsinvalidargumentex.
       MESSAGE 'The specified argument was not valid.' TYPE 'E'.
     CATCH /aws1/cx_knskmsaccessdeniedex.
       MESSAGE 'You do not have permission to perform this AWS KMS action.' TYPE
'E'.
     CATCH /aws1/cx_knskmsdisabledex.
       MESSAGE 'KMS key used is disabled.' TYPE 'E'.
     CATCH /aws1/cx_knskmsinvalidstateex.
       MESSAGE 'KMS key used is in an invalid state. ' TYPE 'E'.
     CATCH /aws1/cx_knskmsnotfoundex.
       MESSAGE 'KMS key used is not found.' TYPE 'E'.
     CATCH /aws1/cx_knskmsoptinrequired.
       MESSAGE 'KMS key option is required.' TYPE 'E'.
     CATCH /aws1/cx_knskmsthrottlingex.
       MESSAGE 'The rate of requests to AWS KMS is exceeding the request
quotas.' TYPE 'E'.
     CATCH /aws1/cx_knsprovthruputexcdex.
       MESSAGE 'The request rate for the stream is too high, or the requested
data is too large for the available throughput.' TYPE 'E'.
     CATCH /aws1/cx_knsresourcenotfoundex.
       MESSAGE 'Resource being accessed is not found.' TYPE 'E'.
   ENDTRY.
```

- Per i dettagli sulle API, GetRecordsconsulta AWS SDK for SAP ABAP API reference.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Utilizzo del servizio con un AWS SDK Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzare **`GetShardIterator`** con una CLI

Gli esempi di codice seguenti mostrano come utilizzare `GetShardIterator`.

Gli esempi di operazioni sono estratti di codice da programmi più grandi e devono essere eseguiti nel contesto. È possibile visualizzare questa operazione nel contesto nel seguente esempio di codice:

- Informazioni di base

CLI

AWS CLI

Per ottenere un iteratore shard

L'`get-shard-iterator`esempio seguente utilizza il tipo `AT_SEQUENCE_NUMBER` shard iterator e genera un iteratore shard per iniziare a leggere i record di dati esattamente dalla posizione indicata dal numero di sequenza specificato.

```
aws kinesis get-shard-iterator \
    --stream-name samplestream \
    --shard-id shardId-000000000001 \
    --shard-iterator-type LATEST
```

Output:

```
{
    "ShardIterator": "AAAAAAAAAAFEvJjIYI+3jw/4aqgH9FifJ+n48XWTh/
IFIsbILP6o5eDueD39NXNBfpZ10WL5K6ADXk8w+5H+Qhd9cFA9k268CPXCz/kebq1TGYI7Vy
+lUkA9BuN3xvATxMBGxRY3zYK05gqgvaIRn94O8SqeEqwhigwZxNWxID3Ej7YYYcxQi8Q/fIrCjGAy/
n2r5Z9G864YpWDfN9upNNQAR/iiOWKs"
```

```
}
```

Per ulteriori informazioni, consulta Developing Consumers Using the Kinesis Data Streams API AWS with the SDK for Java nella Amazon Kinesis Data Streams Developer Guide.

- Per i dettagli sull'API, consulta Command Reference. GetShardIteratorAWS CLI

PowerShell

Strumenti per PowerShell

Esempio 1: restituisce un iteratore di shard per lo shard e la posizione iniziale specificati. I dettagli degli identificatori di shard e dei numeri di sequenza possono essere ottenuti dall'output del Get-KINStream cmdlet, facendo riferimento all'insieme Shards dell'oggetto stream restituito. L'iteratore restituito può essere utilizzato con il Get-KINRecord cmdlet per estrarre i record di dati nello shard.

```
Get-KINShardIterator -StreamName "mystream" -ShardId "shardId-000000000000" -
ShardIteratorType AT_SEQUENCE_NUMBER -StartingSequenceNumber "495598645..."
```

Output:

```
AAAAAAAAAAGIc....9VnbiRNaP
```

- Per i dettagli sull'API, vedere GetShardIteratorin Cmdlet Reference.AWS Strumenti per PowerShell

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Utilizzo del servizio con un AWS SDK Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzare **`ListStreamConsumers`** con un SDK AWS

Il seguente esempio di codice mostra come utilizzare`ListStreamConsumers`.

.NET

SDK per .NET

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
    using System;
    using System.Collections.Generic;
    using System.Threading.Tasks;
    using Amazon.Kinesis;
    using Amazon.Kinesis.Model;

    /// <summary>
    /// List the consumers of an Amazon Kinesis stream.
    /// </summary>
    public class ListConsumers
    {
        public static async Task Main()
        {
            IAmazonKinesis client = new AmazonKinesisClient();

            string streamARN = "arn:aws:kinesis:us-east-2:000000000000:stream/
AmazonKinesisStream";
            int maxResults = 10;

            var consumers = await ListConsumersAsync(client, streamARN,
 maxResults);

            if (consumers.Count > 0)
            {
                consumers
                    .ForEach(c => Console.WriteLine($"Name: {c.ConsumerName} ARN:
 {c.ConsumerARN}"));
            }
            else
            {
                Console.WriteLine("No consumers found.");
            }
```

```
        }

        /// <summary>
        /// Retrieve a list of the consumers for a Kinesis stream.
        /// </summary>
        /// <param name="client">An initialized Kinesis client object.</param>
        /// <param name="streamARN">The ARN of the stream for which we want to
        /// retrieve a list of clients.</param>
        /// <param name="maxResults">The maximum number of results to return.</
param>
        /// <returns>A list of Consumer objects.</returns>
        public static async Task<List<Consumer>>
 ListConsumersAsync(IAmazonKinesis client, string streamARN, int maxResults)
        {
            var request = new ListStreamConsumersRequest
            {
                StreamARN = streamARN,
                MaxResults = maxResults,
            };

            var response = await client.ListStreamConsumersAsync(request);

            return response.Consumers;
        }
    }
```

- Per i dettagli sull'API, consulta la ListStreamConsumerssezione AWS SDK per .NET API Reference.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUtilizzo del servizio con un AWS SDK. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`ListStreams`** con un AWS SDK o una CLI

Gli esempi di codice seguenti mostrano come utilizzare `ListStreams`.

.NET

SDK per .NET

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
    using System;
    using System.Collections.Generic;
    using System.Threading.Tasks;
    using Amazon.Kinesis;
    using Amazon.Kinesis.Model;

    /// <summary>
    /// Retrieves and displays a list of existing Amazon Kinesis streams.
    /// </summary>
    public class ListStreams
    {
        public static async Task Main(string[] args)
        {
            IAmazonKinesis client = new AmazonKinesisClient();
            var response = await client.ListStreamsAsync(new
ListStreamsRequest());

            List<string> streamNames = response.StreamNames;

            if (streamNames.Count > 0)
            {
                streamNames
                    .ForEach(s => Console.WriteLine($"Stream name: {s}"));
            }
            else
            {
                Console.WriteLine("No streams were found.");
            }
        }
    }
```

- Per i dettagli sull'API, consulta la ListStreamssezione AWS SDK per .NET API Reference.

CLI

AWS CLI

Per elencare i flussi di dati

L'`list-streams`esempio seguente elenca tutti i flussi di dati attivi nell'account e nella regione correnti.

```
aws kinesis list-streams
```

Output:

```
{
    "StreamNames": [
        "samplestream",
        "samplestream1"
    ]
}
```

Per ulteriori informazioni, consulta Listing Streams nella Amazon Kinesis Data Streams Developer Guide.

- Per i dettagli sull'API, consulta Command ListStreamsReference AWS CLI .

Rust

SDK per Rust

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
async fn show_streams(client: &Client) -> Result<(), Error> {
    let resp = client.list_streams().send().await?;
```

```
    println!("Stream names:");

    let streams = resp.stream_names;
    for stream in &streams {
        println!("  {}", stream);
    }

    println!("Found {} stream(s)", streams.len());

    Ok(())
}
```

- Per i dettagli sulle API, consulta la ListStreamsguida di riferimento all'API AWS SDK for Rust.

SAP ABAP

SDK per SAP ABAP

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
    TRY.
        oo_result = lo_kns->liststreams(        " oo_result is returned for
testing purposes. "
            "Set Limit to specify that a maximum of streams should be returned."
            iv_limit = iv_limit ).
        DATA(lt_streams) = oo_result->get_streamnames( ).
        MESSAGE 'Streams listed.' TYPE 'I'.
      CATCH /aws1/cx_knslimitexceededex.
        MESSAGE 'The request processing has failed because of a limit exceed
exception.' TYPE 'E'.
    ENDTRY.
```

- Per i dettagli sulle API, ListStreamsconsulta AWS SDK for SAP ABAP API reference.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Utilizzo del servizio con un AWS SDK Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`ListTagsForStream`** con un AWS SDK o una CLI

Gli esempi di codice seguenti mostrano come utilizzare `ListTagsForStream`.

.NET

SDK per .NET

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.Kinesis;
using Amazon.Kinesis.Model;

/// <summary>
/// Shows how to list the tags that have been attached to an Amazon Kinesis
/// stream.
/// </summary>
public class ListTags
{
    public static async Task Main()
    {
        IAmazonKinesis client = new AmazonKinesisClient();
        string streamName = "AmazonKinesisStream";

        await ListTagsAsync(client, streamName);
    }

    /// <summary>
    /// List the tags attached to a Kinesis stream.
    /// </summary>
    /// <param name="client">An initialized Kinesis client object.</param>
```

```
        /// <param name="streamName">The name of the Kinesis stream for which you
        /// wish to display tags.</param>
        public static async Task ListTagsAsync(IAmazonKinesis client, string
streamName)
        {
            var request = new ListTagsForStreamRequest
            {
                StreamName = streamName,
                Limit = 10,
            };

            var response = await client.ListTagsForStreamAsync(request);
            DisplayTags(response.Tags);

            while (response.HasMoreTags)
            {
                request.ExclusiveStartTagKey = response.Tags[response.Tags.Count
- 1].Key;
                response = await client.ListTagsForStreamAsync(request);
            }
        }

        /// <summary>
        /// Displays the items in a list of Kinesis tags.
        /// </summary>
        /// <param name="tags">A list of the Tag objects to be displayed.</param>
        public static void DisplayTags(List<Tag> tags)
        {
            tags
                .ForEach(t => Console.WriteLine($"Key: {t.Key} Value:
{t.Value}"));
        }
    }
```

- Per i dettagli sull'API, consulta la ListTagsForStreamsezione AWS SDK per .NET API Reference.

CLI

AWS CLI

Per elencare i tag per un flusso di dati

L'`list-tags-for-stream`esempio seguente elenca i tag allegati al flusso di dati specificato.

```
aws kinesis list-tags-for-stream \
    --stream-name samplestream
```

Output:

```
{
    "Tags": [
        {
            "Key": "samplekey",
            "Value": "example"
        }
    ],
    "HasMoreTags": false
}
```

Per ulteriori informazioni, consulta Tagging Your Streams nella Amazon Kinesis Data Streams Developer Guide.

- Per i dettagli sull'API, consulta ListTagsForStreamCommand Reference.AWS CLI

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUtilizzo del servizio con un AWS SDK. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`PutRecord`** con un AWS SDK o una CLI

Gli esempi di codice seguenti mostrano come utilizzare `PutRecord`.

Gli esempi di operazioni sono estratti di codice da programmi più grandi e devono essere eseguiti nel contesto. È possibile visualizzare questa operazione nel contesto nel seguente esempio di codice:

- Informazioni di base

CLI

AWS CLI

Per scrivere un record in un flusso di dati

L'`put-record`esempio seguente scrive un singolo record di dati nel flusso di dati specificato utilizzando la chiave di partizione specificata.

```
aws kinesis put-record \
    --stream-name samplestream \
    --data sampledatarecord \
    --partition-key samplepartitionkey
```

Output:

```
{
    "ShardId": "shardId-000000000009",
    "SequenceNumber": "49600902273357540915989931256901506243878407835297513618",
    "EncryptionType": "KMS"
}
```

Per ulteriori informazioni, consulta Developing Producers Using the Amazon Kinesis Data Streams API AWS with the SDK for Java nella Amazon Kinesis Data Streams Developer Guide.

- Per i dettagli sull'API, consulta Command Reference. PutRecordAWS CLI

Java

SDK per Java 2.x

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.kinesis.KinesisClient;
```

```
import software.amazon.awssdk.services.kinesis.model.PutRecordRequest;
import software.amazon.awssdk.services.kinesis.model.KinesisException;
import software.amazon.awssdk.services.kinesis.model.DescribeStreamRequest;
import software.amazon.awssdk.services.kinesis.model.DescribeStreamResponse;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class StockTradesWriter {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <streamName>

                Where:
                    streamName - The Amazon Kinesis data stream to which records
 are written (for example, StockTradeStream)
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String streamName = args[0];
        Region region = Region.US_EAST_1;
        KinesisClient kinesisClient = KinesisClient.builder()
                .region(region)
                .build();

        // Ensure that the Kinesis Stream is valid.
        validateStream(kinesisClient, streamName);
        setStockData(kinesisClient, streamName);
        kinesisClient.close();
    }
```

```
   public static void setStockData(KinesisClient kinesisClient, String
streamName) {
       try {
           // Repeatedly send stock trades with a 100 milliseconds wait in
between.
           StockTradeGenerator stockTradeGenerator = new StockTradeGenerator();

           // Put in 50 Records for this example.
           int index = 50;
           for (int x = 0; x < index; x++) {
               StockTrade trade = stockTradeGenerator.getRandomTrade();
               sendStockTrade(trade, kinesisClient, streamName);
               Thread.sleep(100);
           }

       } catch (KinesisException | InterruptedException e) {
           System.err.println(e.getMessage());
           System.exit(1);
       }
       System.out.println("Done");
   }

   private static void sendStockTrade(StockTrade trade, KinesisClient
kinesisClient,
           String streamName) {
       byte[] bytes = trade.toJsonAsBytes();

       // The bytes could be null if there is an issue with the JSON
serialization by
       // the Jackson JSON library.
       if (bytes == null) {
           System.out.println("Could not get JSON bytes for stock trade");
           return;
       }

       System.out.println("Putting trade: " + trade);
       PutRecordRequest request = PutRecordRequest.builder()
               .partitionKey(trade.getTickerSymbol()) // We use the ticker
symbol as the partition key, explained in
                                                      // the Supplemental
Information section below.
               .streamName(streamName)
               .data(SdkBytes.fromByteArray(bytes))
               .build();
```

```
        try {
            kinesisClient.putRecord(request);
        } catch (KinesisException e) {
            System.err.println(e.getMessage());
        }
    }

    private static void validateStream(KinesisClient kinesisClient, String
 streamName) {
        try {
            DescribeStreamRequest describeStreamRequest =
 DescribeStreamRequest.builder()
                    .streamName(streamName)
                    .build();

            DescribeStreamResponse describeStreamResponse =
 kinesisClient.describeStream(describeStreamRequest);

            if (!
describeStreamResponse.streamDescription().streamStatus().toString().equals("ACTIVE"))
 {
                System.err.println("Stream " + streamName + " is not active.
 Please wait a few moments and try again.");
                System.exit(1);
            }

        } catch (KinesisException e) {
            System.err.println("Error found while describing the stream " +
 streamName);
            System.err.println(e);
            System.exit(1);
        }
    }
}
```

- Per i dettagli sull'API, consulta la PutRecordsezione AWS SDK for Java 2.x API Reference.

PowerShell

Strumenti per PowerShell

Esempio 1: scrive un record contenente la stringa fornita al parametro -Text.

```
Write-KINRecord -Text "test data from string" -StreamName "mystream" -
PartitionKey "Key1"
```

Esempio 2: Scrive un record contenente i dati contenuti nel file specificato. Il file viene trattato come una sequenza di byte, quindi se contiene testo, deve essere scritto con la codifica necessaria prima di utilizzarlo con questo cmdlet.

```
Write-KINRecord -FilePath "C:\TestData.txt" -StreamName "mystream" -PartitionKey
 "Key2"
```

- Per i dettagli sull'API, vedere PutRecordin Cmdlet Reference.AWS Strumenti per PowerShell

Python

SDK per Python (Boto3)

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
class KinesisStream:
    """Encapsulates a Kinesis stream."""

    def __init__(self, kinesis_client):
        """
        :param kinesis_client: A Boto3 Kinesis client.
        """
        self.kinesis_client = kinesis_client
        self.name = None
        self.details = None
        self.stream_exists_waiter = kinesis_client.get_waiter("stream_exists")
```

```
    def put_record(self, data, partition_key):
        """
        Puts data into the stream. The data is formatted as JSON before it is
passed
        to the stream.

        :param data: The data to put in the stream.
        :param partition_key: The partition key to use for the data.
        :return: Metadata about the record, including its shard ID and sequence
number.
        """
        try:
            response = self.kinesis_client.put_record(
                StreamName=self.name, Data=json.dumps(data),
PartitionKey=partition_key
            )
            logger.info("Put record in stream %s.", self.name)
        except ClientError:
            logger.exception("Couldn't put record in stream %s.", self.name)
            raise
        else:
            return response
```

- Per i dettagli sull'API, consulta PutRecord AWSSDK for Python (Boto3) API Reference.

Rust

SDK per Rust

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
async fn add_record(client: &Client, stream: &str, key: &str, data: &str) ->
 Result<(), Error> {
    let blob = Blob::new(data);
```

```
    client
        .put_record()
        .data(blob)
        .partition_key(key)
        .stream_name(stream)
        .send()
        .await?;

    println!("Put data into stream.");

    Ok(())
}
```

- Per i dettagli sulle API, consulta la PutRecordguida di riferimento all'API AWS SDK for Rust.

SAP ABAP

SDK per SAP ABAP

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
    TRY.
        oo_result = lo_kns->putrecord(            " oo_result is returned for
testing purposes. "
            iv_streamname = iv_stream_name
            iv_data       = iv_data
            iv_partitionkey = iv_partition_key ).
        MESSAGE 'Record created.' TYPE 'I'.
      CATCH /aws1/cx_knsinvalidargumentex.
        MESSAGE 'The specified argument was not valid.' TYPE 'E'.
      CATCH /aws1/cx_knskmsaccessdeniedex.
        MESSAGE 'You do not have permission to perform this AWS KMS action.' TYPE
'E'.
      CATCH /aws1/cx_knskmsdisabledex.
        MESSAGE 'KMS key used is disabled.' TYPE 'E'.
      CATCH /aws1/cx_knskmsinvalidstateex.
```

```
        MESSAGE 'KMS key used is in an invalid state. ' TYPE 'E'.
      CATCH /aws1/cx_knskmsnotfoundex.
        MESSAGE 'KMS key used is not found.' TYPE 'E'.
      CATCH /aws1/cx_knskmsoptinrequired.
        MESSAGE 'KMS key option is required.' TYPE 'E'.
      CATCH /aws1/cx_knskmsthrottlingex.
        MESSAGE 'The rate of requests to AWS KMS is exceeding the request
quotas.' TYPE 'E'.
      CATCH /aws1/cx_knsprovthruputexcdex.
        MESSAGE 'The request rate for the stream is too high, or the requested
data is too large for the available throughput.' TYPE 'E'.
      CATCH /aws1/cx_knsresourcenotfoundex.
        MESSAGE 'Resource being accessed is not found.' TYPE 'E'.
    ENDTRY.
```

- Per i dettagli sulle API, PutRecordconsulta AWS SDK for SAP ABAP API reference.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Utilizzo del servizio con un AWS SDK Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`PutRecords`** con un AWS SDK o una CLI

Gli esempi di codice seguenti mostrano come utilizzare `PutRecords`.

CLI

AWS CLI

Per scrivere più record in un flusso di dati

L'`put-records`esempio seguente scrive un record di dati utilizzando la chiave di partizione specificata e un altro record di dati utilizzando una chiave di partizione diversa in una singola chiamata.

```
aws kinesis put-records \
    --stream-name samplestream \
    --
records Data=blob1,PartitionKey=partitionkey1 Data=blob2,PartitionKey=partitionkey2
```

Output:

```
{
    "FailedRecordCount": 0,
    "Records": [
        {
            "SequenceNumber":
 "49600883331171471519674795588238531498465399900093808706",
            "ShardId": "shardId-000000000004"
        },
        {
            "SequenceNumber":
 "49600902273357540915989931256902715169698037101720764562",
            "ShardId": "shardId-000000000009"
        }
    ],
    "EncryptionType": "KMS"
}
```

Per ulteriori informazioni, consulta Developing Producers Using the Amazon Kinesis Data Streams API AWS with the SDK for Java nella Amazon Kinesis Data Streams Developer Guide.

- Per i dettagli sull'API, consulta Command Reference. PutRecordsAWS CLI

JavaScript

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
import { PutRecordsCommand, KinesisClient } from "@aws-sdk/client-kinesis";

/**
 * Put multiple records into a Kinesis stream.
 * @param {{ streamArn: string }} config
 */
export const main = async ({ streamArn }) => {
```

```
  const client = new KinesisClient({});
  try {
    await client.send(
      new PutRecordsCommand({
        StreamARN: streamArn,
        Records: [
          {
            Data: new Uint8Array(),
            /**
             * Determines which shard in the stream the data record is assigned
 to.
             * Partition keys are Unicode strings with a maximum length limit of
 256
             * characters for each key. Amazon Kinesis Data Streams uses the
 partition
             * key as input to a hash function that maps the partition key and
             * associated data to a specific shard.
             */
            PartitionKey: "TEST_KEY",
          },
          {
            Data: new Uint8Array(),
            PartitionKey: "TEST_KEY",
          },
        ],
      }),
    );
  } catch (caught) {
    if (caught instanceof Error) {
      //
    } else {
      throw caught;
    }
  }
};

// Call function if run directly.
import { fileURLToPath } from "node:url";
import { parseArgs } from "node:util";

if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const options = {
    streamArn: {
      type: "string",
```

```
      description: "The ARN of the stream.",
    },
  };

  const { values } = parseArgs({ options });
  main(values);
}
```

- Per i dettagli sull'API, consulta la PutRecordssezione AWS SDK per JavaScript API Reference.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUtilizzo del servizio con un AWS SDK. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`RegisterStreamConsumer`** con un AWS SDK o una CLI

Gli esempi di codice seguenti mostrano come utilizzare `RegisterStreamConsumer`.

.NET

SDK per .NET

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
using System;
using System.Threading.Tasks;
using Amazon.Kinesis;
using Amazon.Kinesis.Model;

/// <summary>
/// This example shows how to register a consumer to an Amazon Kinesis
/// stream.
/// </summary>
public class RegisterConsumer
{
```

```
        public static async Task Main()
        {
            IAmazonKinesis client = new AmazonKinesisClient();
            string consumerName = "NEW_CONSUMER_NAME";
            string streamARN = "arn:aws:kinesis:us-east-2:000000000000:stream/
AmazonKinesisStream";

            var consumer = await RegisterConsumerAsync(client, consumerName,
 streamARN);

            if (consumer is not null)
            {
                Console.WriteLine($"{consumer.ConsumerName}");
            }
        }

        /// <summary>
        /// Registers the consumer to a Kinesis stream.
        /// </summary>
        /// <param name="client">The initialized Kinesis client object.</param>
        /// <param name="consumerName">A string representing the consumer.</
param>
        /// <param name="streamARN">The ARN of the stream.</param>
        /// <returns>A Consumer object that contains information about the
 consumer.</returns>
        public static async Task<Consumer> RegisterConsumerAsync(IAmazonKinesis
 client, string consumerName, string streamARN)
        {
            var request = new RegisterStreamConsumerRequest
            {
                ConsumerName = consumerName,
                StreamARN = streamARN,
            };

            var response = await client.RegisterStreamConsumerAsync(request);
            return response.Consumer;
        }
    }
```

- Per i dettagli sull'API, consulta la RegisterStreamConsumersezione AWS SDK per .NET API Reference.

CLI

AWS CLI

Per registrare un consumatore di flussi di dati

L'`register-stream-consumer`esempio seguente registra un consumatore chiamato `KinesisConsumerApplication` con il flusso di dati specificato.

```
aws kinesis register-stream-consumer \
    --stream-arn arn:aws:kinesis:us-west-2:012345678912:stream/samplestream \
    --consumer-name KinesisConsumerApplication
```

Output:

```
{
    "Consumer": {
        "ConsumerName": "KinesisConsumerApplication",
        "ConsumerARN": "arn:aws:kinesis:us-west-2: 123456789012:stream/
samplestream/consumer/KinesisConsumerApplication:1572383852",
        "ConsumerStatus": "CREATING",
        "ConsumerCreationTimestamp": 1572383852.0
    }
}
```

Per ulteriori informazioni, consulta Developing Consumer with Enhanced Fan-Out Using the Kinesis Data Streams API nella Amazon Kinesis Data Streams Developer Guide.

- Per i dettagli sull'API, consulta Command Reference. RegisterStreamConsumerAWS CLI

SAP ABAP

SDK per SAP ABAP

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
    TRY.
```

```
        oo_result = lo_kns->registerstreamconsumer(       " oo_result is returned
for testing purposes. "
            iv_streamarn = iv_stream_arn
            iv_consumername = iv_consumer_name ).
        MESSAGE 'Stream consumer registered.' TYPE 'I'.
      CATCH /aws1/cx_knsinvalidargumentex.
        MESSAGE 'The specified argument was not valid.' TYPE 'E'.
      CATCH /aws1/cx_sgmresourcelimitexcd.
        MESSAGE 'You have reached the limit on the number of resources.' TYPE
'E'.
      CATCH /aws1/cx_sgmresourceinuse.
        MESSAGE 'Resource being accessed is in use.' TYPE 'E'.
      CATCH /aws1/cx_sgmresourcenotfound.
        MESSAGE 'Resource being accessed is not found.' TYPE 'E'.
    ENDTRY.
```

- Per i dettagli sulle API, RegisterStreamConsumerconsulta AWS SDK for SAP ABAP API reference.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Utilizzo del servizio con un AWS SDK Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Esempi serverless per Kinesis

I seguenti esempi di codice mostrano come usare Kinesis con. AWS SDKs

Esempi

- Richiamare una funzione Lambda da un trigger Kinesis
- Segnalazione di errori di elementi batch per funzioni Lambda con un trigger Kinesis

Richiamare una funzione Lambda da un trigger Kinesis

I seguenti esempi di codice spiegano come implementare una funzione Lambda che riceve un evento attivato dalla ricezione di record da un flusso Kinesis. La funzione recupera il payload Kinesis, lo decodifica da Base64 e registra il contenuto del record.

.NET

SDK per .NET

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento Kinesis con Lambda tramite .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.KinesisEvents;
using AWS.Lambda.Powertools.Logging;

// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeri

namespace KinesisIntegrationSampleCode;

public class Function
{
    // Powertools Logger requires an environment variables against your function
    // POWERTOOLS_SERVICE_NAME
    [Logging(LogEvent = true)]
    public async Task FunctionHandler(KinesisEvent evnt, ILambdaContext context)
    {
        if (evnt.Records.Count == 0)
        {
            Logger.LogInformation("Empty Kinesis Event received");
            return;
        }

        foreach (var record in evnt.Records)
        {
            try
            {
```

```
                Logger.LogInformation($"Processed Event with EventId:
 {record.EventId}");
                string data = await GetRecordDataAsync(record.Kinesis, context);
                Logger.LogInformation($"Data: {data}");
                // TODO: Do interesting work based on the new data
            }
            catch (Exception ex)
            {
                Logger.LogError($"An error occurred {ex.Message}");
                throw;
            }
        }
        Logger.LogInformation($"Successfully processed {evnt.Records.Count}
 records.");
    }

    private async Task<string> GetRecordDataAsync(KinesisEvent.Record record,
 ILambdaContext context)
    {
        byte[] bytes = record.Data.ToArray();
        string data = Encoding.UTF8.GetString(bytes);
        await Task.CompletedTask; //Placeholder for actual async work
        return data;
    }
}
```

Go

SDK per Go V2

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento Kinesis con Lambda tramite Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main
```

```
import (
 "context"
 "log"

 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

func handler(ctx context.Context, kinesisEvent events.KinesisEvent) error {
 if len(kinesisEvent.Records) == 0 {
  log.Printf("empty Kinesis event received")
  return nil
 }

 for _, record := range kinesisEvent.Records {
  log.Printf("processed Kinesis event with EventId: %v", record.EventID)
  recordDataBytes := record.Kinesis.Data
  recordDataText := string(recordDataBytes)
  log.Printf("record data: %v", recordDataText)
  // TODO: Do interesting work based on the new data
 }
 log.Printf("successfully processed %v records", len(kinesisEvent.Records))
 return nil
}

func main() {
 lambda.Start(handler)
}
```

Java

SDK per Java 2.x

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento Kinesis con Lambda tramite Java.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.LambdaLogger;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.KinesisEvent;

public class Handler implements RequestHandler<KinesisEvent, Void> {
    @Override
    public Void handleRequest(final KinesisEvent event, final Context context) {
        LambdaLogger logger = context.getLogger();
        if (event.getRecords().isEmpty()) {
            logger.log("Empty Kinesis Event received");
            return null;
        }
        for (KinesisEvent.KinesisEventRecord record : event.getRecords()) {
            try {
                logger.log("Processed Event with EventId: "+record.getEventID());
                String data = new String(record.getKinesis().getData().array());
                logger.log("Data:"+ data);
                // TODO: Do interesting work based on the new data
            }
            catch (Exception ex) {
                logger.log("An error occurred:"+ex.getMessage());
                throw ex;
            }
        }
        logger.log("Successfully processed:"+event.getRecords().size()+"
 records");
        return null;
    }

}
```

JavaScript

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Consumo di un evento Kinesis con Lambda utilizzando. JavaScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const record of event.Records) {
    try {
      console.log(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      console.log(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      console.error(`An error occurred ${err}`);
      throw err;
    }
  }
  console.log(`Successfully processed ${event.Records.length} records.`);
};

async function getRecordDataAsync(payload) {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}
```

Consumo di un evento Kinesis con Lambda utilizzando. TypeScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
  KinesisStreamEvent,
```

```
  Context,
  KinesisStreamHandler,
  KinesisStreamRecordPayload,
} from "aws-lambda";
import { Buffer } from "buffer";
import { Logger } from "@aws-lambda-powertools/logger";

const logger = new Logger({
  logLevel: "INFO",
  serviceName: "kinesis-stream-handler-sample",
});

export const functionHandler: KinesisStreamHandler = async (
  event: KinesisStreamEvent,
  context: Context
): Promise<void> => {
  for (const record of event.Records) {
    try {
      logger.info(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      logger.info(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      logger.error(`An error occurred ${err}`);
      throw err;
    }
    logger.info(`Successfully processed ${event.Records.length} records.`);
  }
};

async function getRecordDataAsync(
  payload: KinesisStreamRecordPayload
): Promise<string> {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}
```

PHP

SDK per PHP

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento Kinesis con Lambda tramite PHP.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\Kinesis\KinesisEvent;
use Bref\Event\Kinesis\KinesisHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler extends KinesisHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
     * @throws \Bref\Event\InvalidLambdaEvent
     */
    public function handleKinesis(KinesisEvent $event, Context $context): void
    {
        $this->logger->info("Processing records");
        $records = $event->getRecords();
        foreach ($records as $record) {
            $data = $record->getData();
```

```
            $this->logger->info(json_encode($data));
            // TODO: Do interesting work based on the new data

            // Any exception thrown will be logged and the invocation will be
 marked as failed
        }
        $totalRecords = count($records);
        $this->logger->info("Successfully processed $totalRecords records");
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

Python

SDK per Python (Boto3)

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento Kinesis con Lambda tramite Python.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
import base64
def lambda_handler(event, context):

    for record in event['Records']:
        try:
            print(f"Processed Kinesis Event - EventID: {record['eventID']}")
            record_data = base64.b64decode(record['kinesis']
['data']).decode('utf-8')
            print(f"Record Data: {record_data}")
            # TODO: Do interesting work based on the new data
        except Exception as e:
            print(f"An error occurred {e}")
            raise e
```

```
    print(f"Successfully processed {len(event['Records'])} records.")
```

Ruby

SDK per Ruby

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento Kinesis con Lambda tramite Ruby.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
require 'aws-sdk'

def lambda_handler(event:, context:)
  event['Records'].each do |record|
    begin
      puts "Processed Kinesis Event - EventID: #{record['eventID']}"
      record_data = get_record_data_async(record['kinesis'])
      puts "Record Data: #{record_data}"
      # TODO: Do interesting work based on the new data
    rescue => err
      $stderr.puts "An error occurred #{err}"
      raise err
    end
  end
  puts "Successfully processed #{event['Records'].length} records."
end

def get_record_data_async(payload)
  data = Base64.decode64(payload['data']).force_encoding('UTF-8')
  # Placeholder for actual async work
  # You can use Ruby's asynchronous programming tools like async/await or fibers
 here.
  return data
end
```

Rust

SDK per Rust

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento Kinesis con Lambda tramite Rust.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::event::kinesis::KinesisEvent;
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

async fn function_handler(event: LambdaEvent<KinesisEvent>) -> Result<(), Error>
 {
    if event.payload.records.is_empty() {
        tracing::info!("No records found. Exiting.");
        return Ok(());
    }

    event.payload.records.iter().for_each(|record| {
        tracing::info!("EventId:
 {}",record.event_id.as_deref().unwrap_or_default());

        let record_data = std::str::from_utf8(&record.kinesis.data);

        match record_data {
            Ok(data) => {
                // log the record data
                tracing::info!("Data: {}", data);
            }
            Err(e) => {
                tracing::error!("Error: {}", e);
            }
        }
    });

    tracing::info!(
        "Successfully processed {} records",
```

```
        event.payload.records.len()
    );

    Ok(())
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        // disable printing the name of the module in every log line.
        .with_target(false)
        // disabling time is handy because CloudWatch will add the ingestion
 time.
        .without_time()
        .init();

    run(service_fn(function_handler)).await
}
```

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUtilizzo del servizio con un AWS SDK. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Segnalazione di errori di elementi batch per funzioni Lambda con un trigger Kinesis

I seguenti esempi di codice spiegano come implementare una risposta batch parziale per funzioni Lambda che ricevono eventi da un flusso Kinesis. La funzione riporta gli errori degli elementi batch nella risposta, segnalando a Lambda di riprovare tali messaggi in un secondo momento.

.NET

SDK per .NET

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione di errori di elementi batch di Kinesis con Lambda tramite .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text;
using System.Text.Json.Serialization;
using Amazon.Lambda.Core;
using Amazon.Lambda.KinesisEvents;
using AWS.Lambda.Powertools.Logging;

// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeri

namespace KinesisIntegration;

public class Function
{
    // Powertools Logger requires an environment variables against your function
    // POWERTOOLS_SERVICE_NAME
    [Logging(LogEvent = true)]
    public async Task<StreamsEventResponse> FunctionHandler(KinesisEvent evnt,
 ILambdaContext context)
    {
        if (evnt.Records.Count == 0)
        {
            Logger.LogInformation("Empty Kinesis Event received");
            return new StreamsEventResponse();
        }

        foreach (var record in evnt.Records)
        {
            try
            {
                Logger.LogInformation($"Processed Event with EventId:
 {record.EventId}");
                string data = await GetRecordDataAsync(record.Kinesis, context);
                Logger.LogInformation($"Data: {data}");
                // TODO: Do interesting work based on the new data
            }
            catch (Exception ex)
            {
                Logger.LogError($"An error occurred {ex.Message}");
```

```
                /* Since we are working with streams, we can return the failed
 item immediately.
                   Lambda will immediately begin to retry processing from this
 failed item onwards. */
                return new StreamsEventResponse
                {
                    BatchItemFailures = new
 List<StreamsEventResponse.BatchItemFailure>
                    {
                        new StreamsEventResponse.BatchItemFailure
 { ItemIdentifier = record.Kinesis.SequenceNumber }
                    }
                };
            }
        }
        Logger.LogInformation($"Successfully processed {evnt.Records.Count}
 records.");
        return new StreamsEventResponse();
    }

    private async Task<string> GetRecordDataAsync(KinesisEvent.Record record,
 ILambdaContext context)
    {
        byte[] bytes = record.Data.ToArray();
        string data = Encoding.UTF8.GetString(bytes);
        await Task.CompletedTask; //Placeholder for actual async work
        return data;
    }
}

public class StreamsEventResponse
{
    [JsonPropertyName("batchItemFailures")]
    public IList<BatchItemFailure> BatchItemFailures { get; set; }
    public class BatchItemFailure
    {
        [JsonPropertyName("itemIdentifier")]
        public string ItemIdentifier { get; set; }
    }
}
```

Go

SDK per Go V2

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione di errori di elementi batch di Kinesis con Lambda tramite Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "fmt"
 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

func handler(ctx context.Context, kinesisEvent events.KinesisEvent)
 (map[string]interface{}, error) {
 batchItemFailures := []map[string]interface{}{}

 for _, record := range kinesisEvent.Records {
  curRecordSequenceNumber := ""

  // Process your record
  if /* Your record processing condition here */ {
   curRecordSequenceNumber = record.Kinesis.SequenceNumber
  }

  // Add a condition to check if the record processing failed
  if curRecordSequenceNumber != "" {
   batchItemFailures = append(batchItemFailures, map[string]interface{}
{"itemIdentifier": curRecordSequenceNumber})
  }
 }

 kinesisBatchResponse := map[string]interface{}{
```

```
  "batchItemFailures": batchItemFailures,
 }
 return kinesisBatchResponse, nil
}

func main() {
 lambda.Start(handler)
}
```

Java

SDK per Java 2.x

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione di errori di elementi batch di Kinesis con Lambda tramite Java.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.KinesisEvent;
import com.amazonaws.services.lambda.runtime.events.StreamsEventResponse;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;

public class ProcessKinesisRecords implements RequestHandler<KinesisEvent,
 StreamsEventResponse> {

    @Override
    public StreamsEventResponse handleRequest(KinesisEvent input, Context
 context) {

        List<StreamsEventResponse.BatchItemFailure> batchItemFailures = new
 ArrayList<>();
```

```
        String curRecordSequenceNumber = "";

        for (KinesisEvent.KinesisEventRecord kinesisEventRecord :
 input.getRecords()) {
            try {
                //Process your record
                KinesisEvent.Record kinesisRecord =
 kinesisEventRecord.getKinesis();
                curRecordSequenceNumber = kinesisRecord.getSequenceNumber();

            } catch (Exception e) {
                /* Since we are working with streams, we can return the failed
 item immediately.
                   Lambda will immediately begin to retry processing from this
 failed item onwards. */
                batchItemFailures.add(new
 StreamsEventResponse.BatchItemFailure(curRecordSequenceNumber));
                return new StreamsEventResponse(batchItemFailures);
            }
        }

       return new StreamsEventResponse(batchItemFailures);
    }
}
```

JavaScript

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione di errori di elementi batch di Kinesis con Lambda tramite Javascript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const record of event.Records) {
```

```
    try {
      console.log(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      console.log(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      console.error(`An error occurred ${err}`);
      /* Since we are working with streams, we can return the failed item
 immediately.
           Lambda will immediately begin to retry processing from this failed
 item onwards. */
      return {
        batchItemFailures: [{ itemIdentifier: record.kinesis.sequenceNumber }],
      };
    }
  }
  console.log(`Successfully processed ${event.Records.length} records.`);
  return { batchItemFailures: [] };
};

async function getRecordDataAsync(payload) {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}
```

Segnalazione degli errori degli elementi batch di Kinesis utilizzando Lambda. TypeScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
  KinesisStreamEvent,
  Context,
  KinesisStreamHandler,
  KinesisStreamRecordPayload,
  KinesisStreamBatchResponse,
} from "aws-lambda";
import { Buffer } from "buffer";
import { Logger } from "@aws-lambda-powertools/logger";

const logger = new Logger({
  logLevel: "INFO",
```

```
  serviceName: "kinesis-stream-handler-sample",
});

export const functionHandler: KinesisStreamHandler = async (
  event: KinesisStreamEvent,
  context: Context
): Promise<KinesisStreamBatchResponse> => {
  for (const record of event.Records) {
    try {
      logger.info(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      logger.info(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      logger.error(`An error occurred ${err}`);
      /* Since we are working with streams, we can return the failed item
 immediately.
           Lambda will immediately begin to retry processing from this failed
 item onwards. */
      return {
        batchItemFailures: [{ itemIdentifier: record.kinesis.sequenceNumber }],
      };
    }
  }
  logger.info(`Successfully processed ${event.Records.length} records.`);
  return { batchItemFailures: [] };
};

async function getRecordDataAsync(
  payload: KinesisStreamRecordPayload
): Promise<string> {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}
```

PHP

SDK per PHP

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione di errori di elementi batch di Kinesis con Lambda tramite PHP.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\Kinesis\KinesisEvent;
use Bref\Event\Handler as StdHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler implements StdHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
     * @throws \Bref\Event\InvalidLambdaEvent
     */
    public function handle(mixed $event, Context $context): array
    {
        $kinesisEvent = new KinesisEvent($event);
        $this->logger->info("Processing records");
        $records = $kinesisEvent->getRecords();
```

```
        $failedRecords = [];
        foreach ($records as $record) {
            try {
                $data = $record->getData();
                $this->logger->info(json_encode($data));
                // TODO: Do interesting work based on the new data
            } catch (Exception $e) {
                $this->logger->error($e->getMessage());
                // failed processing the record
                $failedRecords[] = $record->getSequenceNumber();
            }
        }
        $totalRecords = count($records);
        $this->logger->info("Successfully processed $totalRecords records");

        // change format for the response
        $failures = array_map(
            fn(string $sequenceNumber) => ['itemIdentifier' => $sequenceNumber],
            $failedRecords
        );

        return [
            'batchItemFailures' => $failures
        ];
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

Python

SDK per Python (Boto3)

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione di errori di elementi batch di Kinesis con Lambda tramite Python.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def handler(event, context):
    records = event.get("Records")
    curRecordSequenceNumber = ""

    for record in records:
        try:
            # Process your record
            curRecordSequenceNumber = record["kinesis"]["sequenceNumber"]
        except Exception as e:
            # Return failed record's sequence number
            return {"batchItemFailures":[{"itemIdentifier":
 curRecordSequenceNumber}]}

    return {"batchItemFailures":[]}
```

Ruby

SDK per Ruby

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione di errori di elementi batch di Kinesis con Lambda tramite Ruby.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
require 'aws-sdk'

def lambda_handler(event:, context:)
  batch_item_failures = []

  event['Records'].each do |record|
    begin
      puts "Processed Kinesis Event - EventID: #{record['eventID']}"
      record_data = get_record_data_async(record['kinesis'])
```

```
      puts "Record Data: #{record_data}"
      # TODO: Do interesting work based on the new data
    rescue StandardError => err
      puts "An error occurred #{err}"
      # Since we are working with streams, we can return the failed item
 immediately.
      # Lambda will immediately begin to retry processing from this failed item
 onwards.
      return { batchItemFailures: [{ itemIdentifier: record['kinesis']
['sequenceNumber'] }] }
    end
  end

  puts "Successfully processed #{event['Records'].length} records."
  { batchItemFailures: batch_item_failures }
end

def get_record_data_async(payload)
  data = Base64.decode64(payload['data']).force_encoding('utf-8')
  # Placeholder for actual async work
  sleep(1)
  data
end
```

Rust

SDK per Rust

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione di errori di elementi batch di Kinesis con Lambda tramite Rust.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::{
    event::kinesis::KinesisEvent,
    kinesis::KinesisEventRecord,
    streams::{KinesisBatchItemFailure, KinesisEventResponse},
```

```
};
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

async fn function_handler(event: LambdaEvent<KinesisEvent>) ->
 Result<KinesisEventResponse, Error> {
    let mut response = KinesisEventResponse {
        batch_item_failures: vec![],
    };

    if event.payload.records.is_empty() {
        tracing::info!("No records found. Exiting.");
        return Ok(response);
    }

    for record in &event.payload.records {
        tracing::info!(
            "EventId: {}",
            record.event_id.as_deref().unwrap_or_default()
        );

        let record_processing_result = process_record(record);

        if record_processing_result.is_err() {
            response.batch_item_failures.push(KinesisBatchItemFailure {
                item_identifier: record.kinesis.sequence_number.clone(),
            });
            /* Since we are working with streams, we can return the failed item
 immediately.
            Lambda will immediately begin to retry processing from this failed
 item onwards. */
            return Ok(response);
        }
    }

    tracing::info!(
        "Successfully processed {} records",
        event.payload.records.len()
    );

    Ok(response)
}

fn process_record(record: &KinesisEventRecord) -> Result<(), Error> {
    let record_data = std::str::from_utf8(record.kinesis.data.as_slice());
```

```
    if let Some(err) = record_data.err() {
        tracing::error!("Error: {}", err);
        return Err(Error::from(err));
    }

    let record_data = record_data.unwrap_or_default();

    // do something interesting with the data
    tracing::info!("Data: {}", record_data);

    Ok(())
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        // disable printing the name of the module in every log line.
        .with_target(false)
        // disabling time is handy because CloudWatch will add the ingestion
 time.
        .without_time()
        .init();

    run(service_fn(function_handler)).await
}
```

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUtilizzo del servizio con un AWS SDK. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Cronologia dei documenti

La tabella seguente descrive le modifiche sostanziali apportate alla documentazione del flusso di dati Amazon Kinesis.

Modifica	Descrizione	Data della modifica
End-of-support date e politica del ciclo di vita delle versioni per KPL e KCL.	Sono state aggiunte informazioni sulle end-of-support date e sulla politica del ciclo di vita delle versioni per Amazon Kinesis Producer Library (KPL) e Amazon Kinesis Client Library (KCL). Per ulteriori informazioni, consultare Politica del ciclo di vita della versione KPL e Politica del ciclo di vita della versione KCL.	13 marzo 2025
È stato aggiunto il supporto per la condivisione di flussi di dati tra account.	Aggiunto Condivisione del flusso di dati con un altro account.	22 novembre 2023
È stato aggiunto il supporto per le modalità di capacità del flusso di dati on demand e assegnata.	Aggiunto Scegli la modalità di capacità del flusso di dati.	29 novembre 2021
Nuovi contenuti per la crittografia lato server.	Aggiunto Protezione dei dati in Flusso di dati Amazon Kinesis.	7 luglio 2017
Nuovi contenuti per metriche migliorate CloudWatch .	Aggiornato Monitoraggio dei flussi di dati Kinesis.	19 aprile 2016

Modifica	Descrizione	Data della modifica
Nuovi contenuti per l'agente Kinesis ottimizzato.	Aggiornato Scrivi su Amazon Kinesis Data Streams utilizzando Kinesis Agent.	11 aprile 2016
Nuovi contenuti per l'utilizzo di agenti Kinesis.	Aggiunto Scrivi su Amazon Kinesis Data Streams utilizzando Kinesis Agent.	2 ottobre 2015
Aggiorna contenuti KPL per il rilascio 0.10.0.	Aggiunto Sviluppa produttori utilizzando Amazon Kinesis Producer Library (KPL).	15 luglio 2015
Aggiorna l'argomento parametri KCL per i parametri configurabili.	Aggiunto Monitoraggio della Kinesis Client Library con Amazon CloudWatch.	9 luglio 2015
Contenuti riorganizzati.	Argomenti di contenuto molto più organizzati per una visualizzazione ad albero più concisa e raggruppamento più logico.	01 luglio 2015
Nuovo argomento KPL della guida per sviluppatori.	Aggiunto Sviluppa produttori utilizzando Amazon Kinesis Producer Library (KPL).	02 giugno 2015
Nuovo argomento parametri KCL.	Aggiunto Monitoraggio della Kinesis Client Library con Amazon CloudWatch.	19 maggio 2015
Supporto per KCL .NET	Aggiunto Sviluppa un utente di Kinesis Client Library in .NET.	1 maggio 2015
Supporto per KCL Node.js	Aggiunto Sviluppa un utente della Kinesis Client Library in Node.js.	26 marzo 2015
Supporto per KCL Ruby	Aggiungi link alla libreria KCL Ruby.	12 gennaio 2015

Modifica	Descrizione	Data della modifica
Nuova API PutRecords	Sono state aggiunte informazioni sulla nuova PutRecords API athe section called “Aggiungi più record con PutRecords”.	15 dicembre 2014
Supporto per il tagging	Aggiunto Etichetta le tue risorse Amazon Kinesis Data Streams.	11 settembre 2014
Nuova CloudWatch metrica	Aggiunto il parametro `GetRecords.IteratorAgeMilliseconds` a Parametri e dimensioni del flusso di dati Amazon Kinesis.	3 settembre 2014
Nuovo capitolo di monitoraggio	Aggiunto Monitoraggio dei flussi di dati Kinesis e Monitoraggio del servizio flusso di dati Amazon Kinesis con Amazon CloudWatch.	30 luglio 2014
Limite di shard predefinito	Quote e limiti è stato aggiornato: il limite di shard predefinito è stato alzato da 5 a 10.	25 febbraio 2014
Limite di shard predefinito	Quote e limiti è stato aggiornato: il limite di shard predefinito è stato alzato da 2 a 5.	28 gennaio 2014
Aggiornamenti versioni API	Aggiornamenti per la versione 2013-12-02 dell'API del flusso di dati Kinesis.	12 dicembre 2013
Rilascio iniziale	Versione iniziale della Guida per gli sviluppatori di Amazon Kinesis.	14 Novembre 2013

Le traduzioni sono generate tramite traduzione automatica. In caso di conflitto tra il contenuto di una traduzione e la versione originale in Inglese, quest'ultima prevarrà.