aws

Panduan Developer

Amazon Kinesis Data Streams

Amazon Kinesis Data Streams: Panduan Developer

Table of Contents

Apa itu Amazon Kinesis Data Streams?

Anda dapat menggunakan Amazon Kinesis Data Streams untuk mengumpulkan dan memproses aliran besar catatan data secara real time. Anda dapat membuat aplikasi pemrosesan data, yang dikenal sebagai aplikasi Kinesis Data Streams. Aplikasi Kinesis Data Streams yang khas membaca data dari aliran data sebagai catatan data. Aplikasi ini dapat menggunakan Perpustakaan Klien Kinesis, dan dapat berjalan di instans Amazon EC2. Anda dapat mengirim catatan yang diproses ke dasbor, menggunakannya untuk menghasilkan peringatan, mengubah strategi harga dan periklanan secara dinamis, atau mengirim data ke berbagai layanan lainnya. AWS Untuk informasi tentang fitur dan harga Kinesis Data Streams, lihat Amazon Kinesis Data Streams.

Kinesis Data Streams adalah bagian dari platform data streaming Kinesis, bersama dengan Firehose, Kinesis Video Streams, dan Managed Service untuk Apache Flink.

Untuk informasi selengkapnya tentang solusi AWS big data, lihat Big Data di AWS. Untuk informasi selengkapnya tentang solusi data AWS streaming, lihat Apa itu Data Streaming? .

Topik

- Apa yang Dapat Dilakukan dengan Kinesis Data Streams?
- Manfaat Menggunakan Kinesis Data Streams
- Layanan Terkait

Apa yang Dapat Dilakukan dengan Kinesis Data Streams?

Anda dapat menggunakan Kinesis Data Streams untuk pengambilan dan agregasi data yang cepat dan berkelanjutan. Jenis data yang digunakan dapat mencakup data log infrastruktur TI, log aplikasi, media sosial, umpan data pasar, dan data clickstream web. Karena waktu respons untuk asupan dan pemrosesan data secara real time, pemrosesan biasanya ringan.

Berikut ini adalah skenario umum untuk menggunakan Kinesis Data Streams:

Asupan dan pemrosesan log dan umpan data yang dipercepat

> Anda dapat meminta produsen mendorong data langsung ke aliran. Misalnya, sistem push dan log aplikasi dan mereka tersedia untuk diproses dalam hitungan detik. Ini mencegah data log hilang jika front end atau server aplikasi gagal. Kinesis Data Streams menyediakan

asupan umpan data yang dipercepat karena Anda tidak mengumpulkan data di server sebelum mengirimkannya untuk masuk.

Metrik dan pelaporan waktu nyata

Anda dapat menggunakan data yang dikumpulkan ke Kinesis Data Streams untuk analisis dan pelaporan data sederhana secara real time. Misalnya, aplikasi pemrosesan data Anda dapat bekerja pada metrik dan pelaporan untuk log sistem dan aplikasi saat data sedang streaming, daripada menunggu untuk menerima kumpulan data.

Analisis data waktu nyata

Ini menggabungkan kekuatan pemrosesan paralel dengan nilai data waktu nyata. Misalnya, proses clickstream situs web secara real time, dan kemudian analisis keterlibatan kegunaan situs menggunakan beberapa aplikasi Kinesis Data Streams berbeda yang berjalan secara paralel.

Pemrosesan aliran yang kompleks

Anda dapat membuat Grafik Asiklik Terarah (DAG) dari aplikasi Kinesis Data Streams dan aliran data. Ini biasanya melibatkan menempatkan data dari beberapa aplikasi Kinesis Data Streams ke aliran lain untuk pemrosesan hilir oleh aplikasi Kinesis Data Streams yang berbeda.

Manfaat Menggunakan Kinesis Data Streams

Meskipun Anda dapat menggunakan Kinesis Data Streams untuk memecahkan berbagai masalah data streaming, penggunaan umum adalah agregasi data real-time diikuti dengan memuat data agregat ke gudang data atau cluster map-reduce.

Data dimasukkan ke dalam aliran data Kinesis, yang menjamin daya tahan dan elastisitas. Penundaan antara waktu rekaman dimasukkan ke dalam aliran dan waktu dapat diambil (put-to-get penundaan) biasanya kurang dari 1 detik. Dengan kata lain, aplikasi Kinesis Data Streams dapat mulai mengkonsumsi data dari aliran segera setelah data ditambahkan. Aspek layanan terkelola dari Kinesis Data Streams membebaskan Anda dari beban operasional dalam membuat dan menjalankan pipeline intake data. Anda dapat membuat aplikasi tipe pengurangan peta streaming. Elastisitas Kinesis Data Streams memungkinkan Anda untuk menskalakan aliran ke atas atau ke bawah, sehingga Anda tidak pernah kehilangan catatan data sebelum habis masa berlakunya.

Beberapa aplikasi Kinesis Data Streams dapat mengkonsumsi data dari aliran, sehingga beberapa tindakan, seperti pengarsipan dan pemrosesan, dapat berlangsung secara bersamaan dan independen. Misalnya, dua aplikasi dapat membaca data dari aliran yang sama. Aplikasi pertama

menghitung agregat yang sedang berjalan dan memperbarui tabel Amazon DynamoDB, dan aplikasi kedua mengompres dan mengarsipkan data ke penyimpanan data seperti Amazon Simple Storage Service (Amazon S3). Tabel DynamoDB dengan agregat berjalan kemudian dibaca oleh dasbor untuk laporan. up-to-the-minute

Perpustakaan Klien Kinesis memungkinkan konsumsi data yang toleran terhadap kesalahan dari aliran dan menyediakan dukungan penskalaan untuk aplikasi Kinesis Data Streams.

Layanan Terkait

Untuk informasi tentang penggunaan klaster EMR Amazon untuk membaca dan memproses aliran data Kinesis secara langsung, lihat Konektor Kinesis.

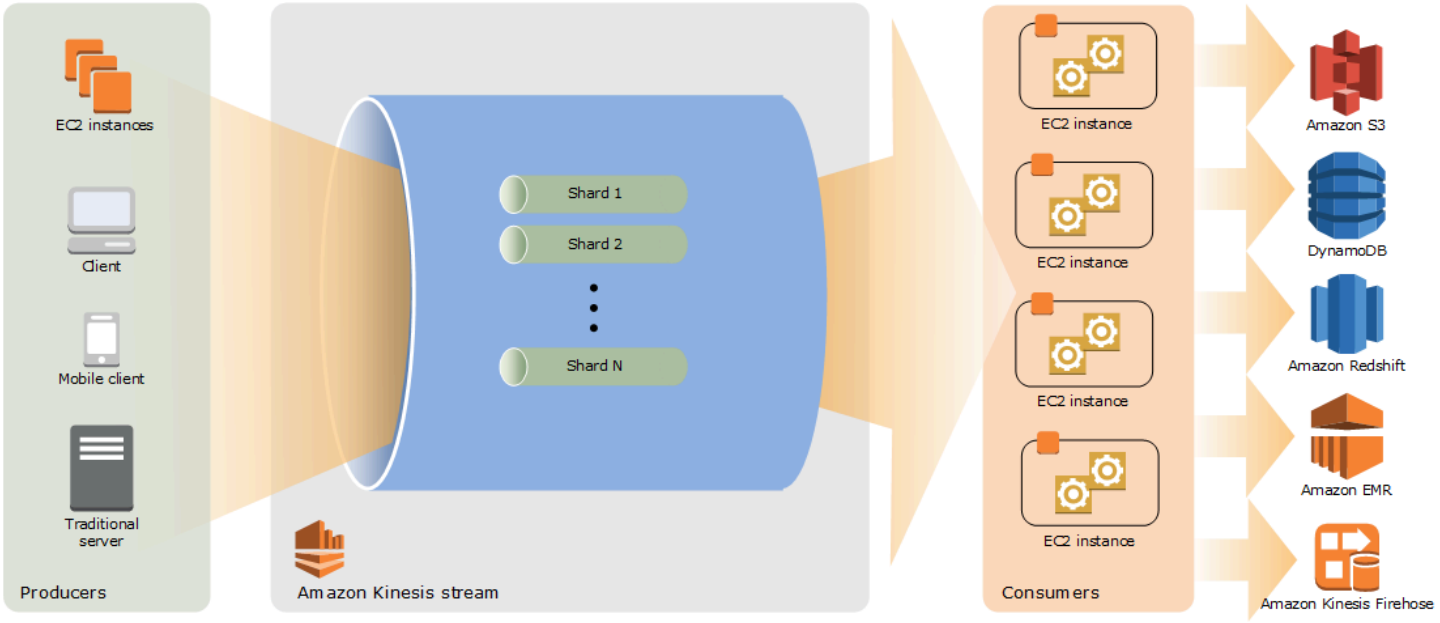
Terminologi dan Konsep Aliran Data Kinesis Amazon

Saat Anda memulai Amazon Kinesis Data Streams, Anda bisa mendapatkan keuntungan dari memahami arsitektur dan terminologinya.

Topik

Arsitektur Tingkat Tinggi Kinesis Data Streams

Diagram berikut menggambarkan arsitektur tingkat tinggi Kinesis Data Streams. Para produsen terus mendorong data ke Kinesis Data Streams, dan konsumen memproses data secara real time. Konsumen (seperti aplikasi khusus yang berjalan di Amazon EC2 atau aliran pengiriman Amazon Data Firehose) dapat menyimpan hasilnya menggunakan AWS layanan seperti Amazon DynamoDB, Amazon Redshift, atau Amazon S3.



Terminologi Aliran Data Kinesis

Aliran Data Kinesis

Aliran data Kinesis adalah sekumpulan pecahan. Setiap pecahan memiliki urutan catatan data. Setiap catatan data memiliki nomor urut yang ditetapkan oleh Kinesis Data Streams.

Rekam Data

Catatan data adalah unit data yang disimpan dalam aliran data Kinesis. Catatan data terdiri dari nomor urut, kunci partisi, dan gumpalan data, yang merupakan urutan byte yang tidak dapat diubah. Kinesis Data Streams tidak memeriksa, menafsirkan, atau mengubah data dalam gumpalan dengan cara apa pun. Gumpalan data bisa mencapai 1 MB.

Mode Kapasitas

Mode kapasitas aliran data menentukan bagaimana kapasitas dikelola dan bagaimana Anda dikenakan biaya untuk penggunaan aliran data Anda. Saat ini, di Kinesis Data Streams, Anda dapat memilih antara mode sesuai permintaan dan mode yang disediakan untuk aliran data Anda. Untuk informasi selengkapnya, lihat Memilih Mode Kapasitas Aliran Data.

Dengan mode on-demand, Kinesis Data Streams secara otomatis mengelola pecahan untuk menyediakan throughput yang diperlukan. Anda hanya dikenakan biaya untuk throughput aktual yang Anda gunakan dan Kinesis Data Streams secara otomatis mengakomodasi kebutuhan throughput beban kerja Anda saat mereka naik atau turun. Untuk informasi selengkapnya, lihat Mode Sesuai Permintaan.

Dengan mode yang disediakan, Anda harus menentukan jumlah pecahan untuk aliran data. Kapasitas total aliran data adalah jumlah dari kapasitas pecahannya. Anda dapat menambah atau mengurangi jumlah pecahan dalam aliran data sesuai kebutuhan dan Anda dikenakan biaya untuk jumlah pecahan dengan tarif per jam. Untuk informasi selengkapnya, lihat Mode yang Disediakan.

Periode Retensi

Periode retensi adalah lamanya waktu rekaman data dapat diakses setelah ditambahkan ke aliran. Periode retensi stream diatur ke default 24 jam setelah pembuatan. Anda dapat meningkatkan periode retensi hingga 8760 jam (365 hari) menggunakan IncreaseStreamRetentionPeriodoperasi, dan mengurangi periode retensi hingga minimal 24 jam menggunakan DecreaseStreamRetentionPeriodoperasi. Biaya tambahan berlaku untuk streaming

dengan periode retensi yang ditetapkan lebih dari 24 jam. Untuk informasi selengkapnya, lihat Harga Amazon Kinesis Data Streams.

Produser

Produsen memasukkan catatan ke Amazon Kinesis Data Streams. Misalnya, server web yang mengirim data log ke aliran adalah produser.

Konsumen

Konsumen mendapatkan catatan dari Amazon Kinesis Data Streams dan memprosesnya. Konsumen ini dikenal sebagaiAplikasi Amazon Kinesis Data Streams.

Aplikasi Amazon Kinesis Data Streams

Aplikasi Amazon Kinesis Data Streams adalah konsumen aliran yang biasanya berjalan pada armada instans EC2.

Ada dua jenis konsumen yang dapat Anda kembangkan: konsumen fan-out bersama dan konsumen fan-out yang ditingkatkan. Untuk mempelajari tentang perbedaan di antara mereka, dan untuk melihat bagaimana Anda dapat membuat setiap jenis konsumen, lihatMembaca Data dari Amazon Kinesis Data Streams.

Output dari aplikasi Kinesis Data Streams dapat menjadi masukan untuk aliran lain, memungkinkan Anda untuk membuat topologi kompleks yang memproses data secara real time. Aplikasi juga dapat mengirim data ke berbagai AWS layanan lain. Mungkin ada beberapa aplikasi untuk satu aliran, dan setiap aplikasi dapat mengkonsumsi data dari aliran secara independen dan bersamaan.

Shard

Shard adalah urutan catatan data yang diidentifikasi secara unik dalam aliran. Aliran terdiri dari satu atau lebih pecahan, yang masing-masing menyediakan unit kapasitas tetap. Setiap pecahan dapat mendukung hingga 5 transaksi per detik untuk pembacaan, hingga total kecepatan baca data maksimum 2 MB per detik dan hingga 1.000 catatan per detik untuk penulisan, hingga total kecepatan penulisan data maksimum 1 MB per detik (termasuk kunci partisi). Kapasitas data aliran Anda adalah fungsi dari jumlah pecahan yang Anda tentukan untuk aliran. Kapasitas total aliran adalah jumlah dari kapasitas pecahannya.

Jika kecepatan data Anda meningkat, Anda dapat menambah atau mengurangi jumlah pecahan yang dialokasikan ke aliran Anda. Untuk informasi selengkapnya, lihat Membagikan Ulang Aliran.

Kunci partisi

Kunci partisi digunakan untuk mengelompokkan data dengan pecahan dalam aliran. Kinesis Data Streams memisahkan catatan data milik aliran menjadi beberapa pecahan. Ini menggunakan kunci partisi yang terkait dengan setiap catatan data untuk menentukan pecahan mana yang dimiliki oleh catatan data tertentu. Tombol partisi adalah string Unicode, dengan batas panjang maksimum 256 karakter untuk setiap tombol. Fungsi hash MD5 digunakan untuk memetakan kunci partisi ke nilai integer 128-bit dan untuk memetakan catatan data terkait ke pecahan menggunakan rentang kunci hash dari pecahan. Ketika sebuah aplikasi menempatkan data ke dalam aliran, itu harus menentukan kunci partisi.

Nomor Urutan

Setiap catatan data memiliki nomor urut yang unik per kunci partisi dalam pecahannya.
Kinesis Data Streams menetapkan nomor urut setelah Anda menulis ke aliran dengan atau. `client.putRecords` `client.putRecord` Nomor urutan untuk kunci partisi yang sama umumnya meningkat dari waktu ke waktu. Semakin lama periode waktu antara permintaan tulis, semakin besar nomor urut menjadi.

> **Note**
>
> Nomor urutan tidak dapat digunakan sebagai indeks untuk kumpulan data dalam aliran yang sama. Untuk memisahkan kumpulan data secara logis, gunakan kunci partisi atau buat aliran terpisah untuk setiap kumpulan data.

Perpustakaan Klien Kinesis

Perpustakaan Klien Kinesis dikompilasi ke dalam aplikasi Anda untuk mengaktifkan konsumsi data yang toleran terhadap kesalahan dari aliran. Perpustakaan Klien Kinesis memastikan bahwa untuk setiap pecahan ada prosesor rekaman yang menjalankan dan memproses pecahan itu. Perpustakaan juga menyederhanakan membaca data dari aliran. Perpustakaan Klien Kinesis menggunakan tabel Amazon DynamoDB untuk menyimpan data kontrol. Ini menciptakan satu tabel per aplikasi yang memproses data.

Ada dua versi utama dari Perpustakaan Klien Kinesis. Yang mana yang Anda gunakan tergantung pada jenis konsumen yang ingin Anda buat. Untuk informasi selengkapnya, lihat Membaca Data dari Amazon Kinesis Data Streams.

Nama Aplikasi

Nama aplikasi Amazon Kinesis Data Streams mengidentifikasi aplikasi. Setiap aplikasi Anda harus memiliki nama unik yang dicakup ke AWS akun dan Wilayah yang digunakan oleh aplikasi. Nama ini digunakan sebagai nama untuk tabel kontrol di Amazon DynamoDB dan namespace untuk metrik Amazon. CloudWatch

Enkripsi Sisi-Server

Amazon Kinesis Data Streams dapat secara otomatis mengenkripsi data sensitif saat produsen memasukkannya ke dalam aliran. Kinesis Data AWS KMSStreams menggunakan kunci master untuk enkripsi. Untuk informasi selengkapnya, lihat Perlindungan Data di Amazon Kinesis Data Streams.

> **Note**
>
> Untuk membaca dari atau menulis ke aliran terenkripsi, produsen dan aplikasi konsumen harus memiliki izin untuk mengakses kunci master. Untuk informasi tentang pemberian izin kepada aplikasi produsen dan konsumen, lihat. the section called “Izin untuk Menggunakan Kunci Master KMS Buatan Pengguna”

> **Note**
>
> Menggunakan enkripsi sisi server menimbulkan biaya AWS Key Management Service ().AWS KMSUntuk informasi selengkapnya, lihat Harga Layanan ManajemenAWS Utama.

Kuota dan Batas

Amazon Kinesis Data Streams memiliki kuota dan batas aliran dan pecahan berikut.

Kuota	Mode sesuai permintaan	Mode yang disediakan
Jumlah aliran data	Tidak ada kuota atas jumlah stream dalam akun AndaAWS. Secara default, Anda dapat membuat hingga 50 aliran data dengan mode kapasitas sesuai permintaan. Jika Anda memerlukan peningkatan kuota ini, silakan naikkan tiket dukungan.	Tidak ada kuota atas jumlah aliran dengan mode yang disediakan dalam akun.
Jumlah serpihan	Tidak ada batas atas. Jumlah pecahan tergantung pada jumlah data yang dicerna dan tingkat throughput yang Anda butuhkan. Kinesis Data Streams secara otomatis menskalakan jumlah pecahan sebagai respons terhadap perubahan volume data dan lalu lintas.	Tidak ada batas atas. Kuota pecahan default adalah 500 pecahan per AWS akun untuk AWS Wilayah berikut: AS Timur (Virginia N.), AS Barat (Oregon), dan Eropa (Irlandia). Untuk semua wilayah lain, kuota shard default adalah 200 pecahan per akun. AWS Untuk meminta peningkat an kuota shards-per-data streaming, lihat Meminta Peningkatan Kuota.
Throughput aliran data	Secara default, aliran data baru yang dibuat dengan mode kapasitas sesuai permintaan memiliki 4 MB/s tulis dan 8 MB/s throughpu t baca. Ketika lalu lintas meningkat, aliran data	Tidak ada batas atas. Throughput maksimum tergantung pada jumlah pecahan yang disediakan untuk aliran. Setiap pecahan dapat mendukung hingga 1 MB/detik atau 1.000 rekam/

Kuota	Mode sesuai permintaan	Mode yang disediakan
	dengan mode kapasitas sesuai permintaan skala hingga 200 MB/s tulis dan throughput baca 400 MB/s. Jika Anda memerlukan peningkatan hingga 2 Gb/s tulis dan kapasitas baca 4 Gb/s, kirimkan tiket dukungan	detik throughput tulis atau hingga 2 MB/detik atau 2.000 rekam/detik throughput baca. Jika Anda membutuhkan lebih banyak kapasitas konsumsi, Anda dapat dengan mudah meningkatkan jumlah pecahan dalam aliran menggunakan AWS Management Console atau API. UpdateShardCount
Ukuran payload data	Ukuran maksimum payload data rekaman sebelumnya `base64-encoding` hingga 1 MB.	
`GetRecords` ukuran transaksi	GetRecordsdapat mengambil hingga 10 MB data per panggilan dari satu pecahan, dan hingga 10.000 catatan per panggilan. Setiap panggilan `GetRecords` dihitung sebagai satu transaksi baca. Setiap pecahan dapat mendukung hingga lima transaksi baca per detik. Setiap transaksi baca dapat menyediakan hingga 10.000 catatan dengan kuota atas 10 MB per transaksi.	
Tingkat baca data per pecahan	Setiap pecahan dapat mendukung hingga total kecepatan baca data maksimum 2 MB per detik via GetRecords. Jika panggilan untuk `GetRecords` mengembalikan 10 MB, panggilan berikutnya yang dilakukan dalam 5 detik berikutnya akan memberikan pengecualian.	
Jumlah konsumen terdaftar per aliran data	Anda dapat membuat hingga 20 konsumen terdaftar (Enhanced Fan-out Limit) untuk setiap aliran data.	
Beralih antara mode yang disediakan dan sesuai permintaan	Untuk setiap aliran data di AWS akun Anda, Anda dapat beralih antara mode kapasitas sesuai permintaan dan yang disediakan dua kali dalam 24 jam.	

Batas API

Seperti kebanyakan AWS API, operasi API Kinesis Data Streams dibatasi kecepatan. Batas berikut berlaku per AWS akun per wilayah. Untuk informasi selengkapnya tentang API Kinesis Data Streams, lihat Referensi API Amazon Kinesis.

Batas API Pesawat Kontrol KDS

Bagian berikut menjelaskan batasan untuk API bidang kontrol KDS. API bidang kontrol KDS memungkinkan Anda membuat dan mengelola aliran data Anda. Batas ini berlaku per AWS akun per wilayah.

Batas API Pesawat Kontrol

API	Batas panggilan API	Per Akun/Aliran	Deskripsi
AddTagsToStream	5 transaksi per detik (TPS)	Per Aliran	50 tag per aliran data
CreateStream	5 TPS	Per Akun	Tidak ada kuota atas jumlah stream yang dapat Anda miliki di akun. Anda menerima `LimitExceededException` ketika membuat `CreateStream` permintaan ketika Anda mencoba untuk melakukan salah satu dari berikut: • Memiliki lebih dari lima aliran di `CREATING` negara bagian kapan saja. • Buat lebih banyak pecahan daripada

API	Batas panggilan API	Per Akun/Aliran	Deskripsi
			yang diizinkan untuk akun Anda.
DecreaseStreamRetentionPeriod	5 TPS	Per Aliran	Nilai minimum periode retensi aliran data adalah 24 jam.
DeleteResourcePolicy	5 TPS	Per Akun	Jika Anda memerlukan peningkatan batas ini, naikkan tiket Support.
DeleteStream	5 TPS	Per Akun	
DeregisterStreamConsumer	5 TPS	Per Aliran	
DescribeLimits	1 TPS	Per Akun	
DescribeStream	10 TPS	Per Akun	
DescribeStreamConsumer	20 TPS	Per Aliran	
DescribeStreamSummary	20 TPS	Per Akun	
DisableEnhancedMonitoring	5 TPS	Per Aliran	
EnableEnhancedMonitoring	5 TPS	Per Aliran	

API	Batas panggilan API	Per Akun/Aliran	Deskripsi
GetResourcePolicy	5 TPS	Per Akun	Jika Anda memerlukan peningkatan batas ini, naikkan tiket Support.
IncreaseStreamRetentionPeriod	5 TPS	Per Aliran	Nilai maksimum periode retensi aliran adalah 8760 jam (365 hari).
ListShards	1000 TPS	Per Aliran	
ListStreamConsumers	5 TPS	Per Aliran	
ListStreams	5 TPS	Per Akun	
ListTagsForStream	5 TPS	Per Aliran	
MergeShards	5 TPS	Per Aliran	Hanya berlaku untuk yang disediakan.
PutResourcePolicy	5 TPS	Per Akun	Jika Anda memerlukan peningkatan batas ini, naikkan tiket Support.

API	Batas panggilan API	Per Akun/Aliran	Deskripsi
RegisterStreamConsumer	5 TPS	Per Aliran	Anda dapat mendaftarkan hingga 20 konsumen per aliran data. Konsumen tertentu hanya dapat didaftarkan dengan satu aliran data pada satu waktu. Hanya 5 konsumen yang dapat dibuat secara bersamaan. Dengan kata lain, Anda tidak dapat memiliki lebih dari 5 konsumen dalam satu `CREATING` status pada saat yang bersamaan. Mendaftarkan konsumen ke-6 sementara ada 5 di a `CREATING`
RemoveTagsFromStream	5 TPS	Per Aliran	
SplitShard	5 TPS	Per Aliran	Hanya berlaku untuk yang disediakan
StartStreamEncryption		Per Aliran	Anda dapat berhasil menerapkan kunci AWS KMS baru untuk enkripsi sisi server 25 kali dalam periode 24 jam bergulir.

API	Batas panggilan API	Per Akun/Aliran	Deskripsi
StopStreamEncryption		Per Aliran	Anda dapat berhasil menonaktifkan enkripsi sisi server 25 kali dalam periode 24 jam bergulir.
UpdateShardCount		Per Aliran	Hanya berlaku untuk yang disediakan. Batas default pada jumlah pecahan adalah 10.000. Ada batasan tambahan pada API ini. Untuk informasi lebih lanjut, lihat UpdateShardCount.
UpdateStreamMode		Per aliran	Untuk setiap aliran data di AWS akun Anda, Anda dapat beralih antara mode kapasitas sesuai permintaan dan yang disediakan dua kali dalam 24 jam.

Batas API Pesawat Data KDS

Bagian berikut menjelaskan batasan untuk API bidang data KDS. API bidang data KDS memungkinkan Anda menggunakan aliran data untuk mengumpulkan dan memproses catatan data secara real time. Batasan ini berlaku per pecahan dalam aliran data Anda.

Batas API Data Plane

API	Batas panggilan API	Batas muatan	Detail tambahan
GetRecords	5 TPS	Jumlah maksimum catatan yang dapat dikembalikan per panggilan adalah 10.000. Ukuran maksimum data yang `GetRecords` dapat dikembalikan adalah 10 MB.	Jika panggilan mengembalikan jumlah data ini, panggilan berikutnya dilakukan dalam 5 detik berikutnya a`ProvisionedThroughputExceededException`. Jika throughput yang disediakan tidak mencukupi di stream, panggilan berikutnya dilakukan dalam 1 detik berikutnya a akan melempar. `ProvisionedThroughputExceededException`
GetShardIterator	5 TPS		Sebuah iterator shard kedaluwarsa 5 menit setelah dikembalikan ke pemohon. Jika GetShardIterator permintaan dibuat terlalu sering, Anda menerima a ProvisionedThroughputExceededException.

API	Batas panggilan API	Batas muatan	Detail tambahan
PutRecord	1000 TPS	Setiap pecahan dapat mendukung penulisan hingga 1.000 catatan per detik, hingga total penulisan data maksimum 1 MB per detik.	
PutRecords		Setiap PutRecords permintaan dapat mendukung hingga 500 catatan. Setiap catatan dalam permintaan dapat sebesar 1 MB, hingga batas 5 MB untuk seluruh permintaan, termasuk kunci partisi. Setiap pecahan dapat mendukung penulisan hingga 1.000 catatan per detik, hingga total penulisan data maksimum 1 MB per detik.	

API	Batas panggilan API	Batas muatan	Detail tambahan
SubscribeToShard	Anda dapat melakukan satu panggilan Subscribe ToShard per detik per konsumen terdaftar per pecahan.		Jika Anda menelepon SubscribeToShard lagi dengan ConsumerArn yang sama ShardId dan dalam waktu 5 detik setelah panggilan berhasil, Anda akan mendapatkan file. ResourceInUseException

Meningkatkan Kuota

Anda dapat menggunakan Service Quotas untuk meminta kenaikan kuota, jika kuota dapat disesuaikan. Beberapa permintaan diselesaikan secara otomatis, sementara yang lain dikirimkan ke AWS Support. Anda dapat melacak status permintaan peningkatan kuota yang dikirimkan ke AWS Support. Permintaan untuk meningkatkan service quotas tidak menerima prioritas dukungan. Jika Anda memiliki permintaan mendesak, hubungi AWS Support. Untuk informasi selengkapnya, lihat Apa itu Service Quotas?

Untuk meminta peningkatan kuota layanan, ikuti prosedur yang diuraikan dalam Meminta Peningkatan Kuota.

Menyiapkan Amazon Kinesis Data Streams

Sebelum Anda menggunakan Amazon Kinesis Data Streams untuk pertama kalinya, selesaikan tugas berikut ini.

Tugas

- Mendaftar AWS
- Unduh Pustaka dan Alat
- Mengonfigurasi Lingkungan Pengembangan Anda

Mendaftar AWS

Saat Anda mendaftar ke Amazon Web Services (AWS),AWS akun Anda secara otomatis terdaftar untuk semua layanan diAWS, termasuk Kinesis Data Streams. Anda hanya dikenakan biaya untuk layanan yang Anda gunakan.

Jika Anda sudah memiliki akun AWS, lewati ke tugas berikutnya. Jika Anda belum memiliki akun AWS, gunakan prosedur berikut untuk membuatnya.

Untuk mendaftar akun AWS

1. Buka https://portal.aws.amazon.com/billing/signup.
2. Ikuti petunjuk online.

   Bagian dari prosedur pendaftaran melibatkan menerima panggilan telepon dan memasukkan kode verifikasi di keypad telepon.

   Ketika Anda mendaftar untukAkun AWS, sebuah Pengguna root akun AWSdibuat. Pengguna root memiliki akses ke semuaLayanan AWS dan sumber daya di akun. Sebagai praktik terbaik keamanan, tetapkan akses administratif ke pengguna administratif, dan gunakan hanya pengguna root untuk melakukan tugas yang memerlukan akses pengguna root.

Unduh Pustaka dan Alat

Pustaka dan alat berikut ini akan membantu Anda bekerja dengan Kinesis Data Streams:

- Referensi API Amazon Kinesis adalah rangkaian operasi dasar yang didukung Kinesis Data Streams. Untuk informasi selengkapnya tentang melakukan operasi dasar menggunakan kode Java, lihat yang berikut ini:
  - Mengembangkan Produsen Menggunakan API Amazon Kinesis Data Streams denganAWS SDK for Java
  - Mengembangkan Konsumen Kustom dengan Throughput Bersama MenggunakanAWS SDK for Java
  - Membuat dan Mengelola Streaming
- AWSSDK untuk Go, Java, JavaScript, .NET, Node.js, PHP, Python, dan Ruby menyertakan dukungan Kinesis Data Streams dan sampel. Jika versi AndaAWS SDK for Java tidak mencakup sampel untuk Kinesis Data Streams, Anda juga dapat mengunduhnya dari GitHub.
- Kinesis Client Library (KCL) menyediakan model easy-to-use pemrograman untuk memproses data. KCL dapat membantu Anda memulai dengan cepat dengan Kinesis Data Streams di Jawa, .NET, Python, dan Ruby. Node.js Untuk informasi selengkapnya, lihat Membaca Data dari Aliran.
- AWS Command Line InterfaceMendukung Kinesis Data Streams. AWS CLI memungkinkan Anda mengontrol beberapa layanan AWS dari baris perintah dan mengotomatiskan layanan melalui skrip.

Mengonfigurasi Lingkungan Pengembangan Anda

Untuk menggunakan KCL, pastikan bahwa lingkungan pengembangan Java Anda memenuhi persyaratan berikut:

- Java 1.7 (Java SE 7 JDK) atau yang lebih baru. Anda dapat mengunduh perangkat lunak Java terbaru dari Java SE Downloads di situs Oracle.
- Paket Apache Commons (Kode, Klien HTTP, dan Logging)
- prosesor Jackson JSON

Perhatikan bahwa AWS SDK for Javatermasuk Apache Commons dan Jackson di folder pihak ketiga. Namun, SDK for Java bekerja dengan Java 1.6, sedangkan Kinesis Client Library membutuhkan Java 1.7.

Memulai dengan Amazon Kinesis Data Streams

Informasi di bagian ini membantu Anda mulai menggunakan Amazon Kinesis Data Streams. Jika Anda baru mengenal Kinesis Data Streams, mulailah dengan membiasakan diri dengan konsep dan terminologi yang disajikan. Terminologi dan Konsep Aliran Data Kinesis Amazon

Bagian ini menunjukkan kepada Anda cara melakukan operasi dasar Amazon Kinesis Data Streams menggunakan. AWS Command Line Interface Anda akan mempelajari prinsip-prinsip aliran data Kinesis Data Streams yang mendasar dan langkah-langkah yang diperlukan untuk menempatkan dan mendapatkan data dari aliran data Kinesis.

Topik

- Menginstal dan MengonfigurasiAWS CLI
- Melakukan Pengoperasian Kinesis Data Stream Dasar MenggunakanAWS CLI

Untuk akses CLI, Anda memerlukan ID kunci akses dan kunci akses rahasia. Gunakan kredensi sementara alih-alih kunci akses jangka panjang bila memungkinkan. Kredensi sementara mencakup ID kunci akses, kunci akses rahasia, dan token keamanan yang menunjukkan kapan kredensialnya kedaluwarsa. Untuk informasi selengkapnya, lihat Menggunakan kredensi sementara dengan AWS sumber daya di Panduan Pengguna IAM.

Anda dapat menemukan petunjuk pengaturan step-by-step IAM dan kunci keamanan terperinci di Buat Pengguna IAM.

Pada bagian ini, perintah spesifik yang dibahas diberikan secara verbatim, kecuali di mana nilai spesifik harus berbeda untuk setiap run. Selain itu, contohnya menggunakan wilayah AS Barat (Oregon), tetapi langkah-langkah di bagian ini berfungsi di salah satu wilayah tempat Kinesis Data Streams didukung.

Menginstal dan MengonfigurasiAWS CLI

Instal AWS CLI

Untuk langkah-langkah rinci tentang cara menginstalAWS CLIuntuk Windows dan untuk sistem operasi Linux, OS X, dan Unix, lihatMenginstalAWSCLI.

Gunakan perintah berikut untuk mencantumkan opsi dan layanan yang tersedia:

```
aws help
```

Anda akan menggunakan layanan Kinesis Data Streams, sehingga Anda dapat meninjauAWS CLIsubcommand yang terkait dengan Kinesis Data Streams menggunakan perintah berikut:

```
aws kinesis help
```

Perintah ini menghasilkan output yang mencakup perintah Kinesis Data Streams yang tersedia:

```
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
```

Daftar perintah ini sesuai dengan Kinesis Data Streams API yang didokumentasikan dalamReferensi API Layanan Amazon Kinesis. Misalnya,`create-stream`perintah sesuai dengan`CreateStream`Tindakan API.

ParameterAWS CLIsekarang berhasil diinstal, tetapi tidak dikonfigurasi. Hal ini ditunjukkan di bagian selanjutnya.

Konfigurasi AWS CLI

Untuk penggunaan umum,`aws configure`Perintah adalah cara tercepat untuk mengaturAWS CLIinstalasi. Untuk informasi selengkapnya, lihatMengonfigurasiAWSCLI.

Melakukan Pengoperasian Kinesis Data Stream Dasar MenggunakanAWS CLI

Bagian ini menjelaskan penggunaan dasar aliran Kinesis dari baris perintah menggunakanAWS CLI. Pastikan Anda sudah familiar dengan konsep yang dibahasTerminologi dan Konsep Aliran Data Kinesis Amazon.

> **Note**
>
> Setelah membuat stream, akun Anda dikenakan biaya nominal untuk penggunaan Kinesis Data Streams karena Kinesis Data Streams tidak memenuhi syarat untukAWStingkat gratis. Setelah Anda selesai dengan tutorial ini, hapusAWSsumber daya untuk berhenti menimbulkan biaya. Untuk informasi selengkapnya, lihat Langkah 4: Pembersihan.

Topik

Langkah 1: Membuat Pengaliran

Langkah pertama Anda adalah membuat stream dan memverifikasi bahwa itu berhasil dibuat. Gunakan perintah berikut untuk membuat aliran bernama “Foo”:

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

Selanjutnya, terbitkan perintah berikut untuk memeriksa kemajuan pembuatan stream:

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

Anda harus mendapatkan output yang mirip dengan contoh berikut:

```
{
    "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
    }
}
```

Dalam contoh ini, aliran memiliki status MENCIPTAKAN, yang berarti tidak cukup siap untuk digunakan. Periksa lagi dalam beberapa saat, dan Anda akan melihat output yang mirip dengan contoh berikut:

```
{
    "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
    }
}
```

Ada informasi dalam output ini yang tidak perlu Anda khawatirkan untuk tutorial ini. Hal utama untuk saat ini adalah`"StreamStatus": "ACTIVE"`, yang memberitahu Anda bahwa aliran siap untuk digunakan, dan informasi pada pecahan tunggal yang Anda minta. Anda juga dapat memverifikasi keberadaan aliran baru Anda dengan menggunakan`list-streams`perintah, seperti yang ditunjukkan di sini:

```
aws kinesis list-streams
```

Output:

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

Langkah 2: Catatan

Sekarang setelah Anda memiliki aliran aktif, Anda siap untuk memasukkan beberapa data. Untuk tutorial ini, Anda akan menggunakan perintah yang paling sederhana,`put-record`, yang menempatkan catatan data tunggal yang berisi teks “testdata” ke dalam sungai:

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

Perintah ini, jika berhasil, akan menghasilkan output yang mirip dengan contoh berikut:

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

Selamat, Anda baru saja menambahkan data ke aliran! Selanjutnya Anda akan melihat bagaimana untuk mendapatkan data dari sungai.

Langkah 3: Dapatkan Rekam

GetShardIterator

Sebelum Anda bisa mendapatkan data dari aliran yang Anda butuhkan untuk mendapatkan iterator shard untuk beling yang Anda minati. Sebuah iterator shard mewakili posisi aliran dan beling dari mana konsumen (`get-record`perintah dalam hal ini) akan membaca. Anda akan menggunakan`get-shard-iterator`perintah, sebagai berikut:

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

Ingat bahwa`aws kinesis`perintah memiliki Kinesis Data Streams API di belakang mereka, jadi jika Anda ingin tahu tentang salah satu parameter yang ditampilkan, Anda dapat membaca tentang mereka di`GetShardIterator`Topik referensi API Eksekusi yang berhasil akan menghasilkan output yang mirip dengan contoh berikut (gulir horizontal untuk melihat seluruh output):

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

String panjang karakter yang tampaknya acak adalah iterator beling (milik Anda akan berbeda). Anda akan perlu untuk copy/paste shard iterator ke perintah get, ditampilkan berikutnya. Iterator Shard memiliki masa pakai 300 detik yang valid, yang seharusnya cukup waktu bagi Anda untuk menyalin/menempelkan iterator shard ke perintah berikutnya. Perhatikan bahwa Anda harus menghapus baris baru dari iterator shard Anda sebelum menempelkan ke perintah berikutnya. Jika Anda mendapatkan pesan kesalahan bahwa iterator shard tidak lagi valid, cukup jalankan`get-shard-iterator`perintah lagi.

GetRecords

Parameter`get-records`perintah mendapat data dari sungai, dan memutuskan untuk panggilan ke`GetRecords`di API Kinesis Data Streams. The shard iterator menentukan posisi di beling dari mana Anda ingin mulai membaca catatan data secara berurutan. Jika tidak ada catatan yang tersedia di bagian pecahan yang iterator menunjuk ke,`GetRecords`mengembalikan daftar kosong. Perhatikan

bahwa mungkin diperlukan beberapa panggilan untuk sampai ke bagian dari pecahan yang berisi catatan.

Pada contoh berikut`get-records`perintah (gulir horizontal untuk melihat seluruh perintah):

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

Jika Anda menjalankan tutorial ini dari prosesor perintah Unix-type seperti bash, Anda dapat mengotomatisasi akuisisi iterator shard menggunakan perintah bersarang, seperti ini (gulir horizontal untuk melihat seluruh perintah):

```
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
```

Jika Anda menjalankan tutorial ini dari sistem yang mendukungPowerShell, Anda dapat mengotomatisasi akuisisi iterator shard menggunakan perintah seperti ini (gulir horizontal untuk melihat seluruh perintah):

```
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])
```

Hasil sukses dari`get-records`perintah akan meminta catatan dari aliran Anda untuk shard yang Anda tentukan ketika Anda memperoleh iterator shard, seperti dalam contoh berikut (gulir horizontal untuk melihat seluruh output):

```
{
  "Records":[ {
    "Data":"dGVzdGRhdGE=",
    "PartitionKey":"123”,
    "ApproximateArrivalTimestamp": 1.441215410867E9,
    "SequenceNumber":"49544985256907370027570885864065577703022652638596431874"
  } ],
  "MillisBehindLatest":24000,

 "NextShardIterator":"AAAAAAAAAAEDOW3ugseWPE4503kqN1yN1UaodY8unE0sYslMUmC6lX9hlig5+t4RtZM0/
```

```
tALfiI4QGjunVgJvQsjxjh2aLyxaAaPr
+LaoENQ7eVs4EdYXgKyThTZGPcca2fVXYJWL3yafv9dsDwsYVedI66dbMZFC8rPMWc797zxQkv4pSKvPOZvrUIudb8UkH3V
}
```

Perhatikan bahwa`get-records`dijelaskan di atas sebagaipermintaan, yang berarti Anda mungkin menerima nol atau lebih catatan bahkan jika ada catatan dalam aliran Anda, dan catatan apa pun yang dikembalikan mungkin tidak mewakili semua catatan yang saat ini ada di stream Anda. Ini sangat normal, dan kode produksi hanya akan memilih aliran untuk catatan pada interval yang sesuai (kecepatan pemungutan suara ini akan bervariasi tergantung pada persyaratan desain aplikasi spesifik Anda).

Hal pertama yang mungkin Anda perhatikan tentang catatan Anda di bagian tutorial ini adalah bahwa datanya tampak sampah -; ini bukan teks yang jelas`testdata`Kami mengirim. Hal ini disebabkan oleh jalannya`put-record`menggunakan Base64 encoding untuk memungkinkan Anda untuk mengirim data biner. Namun, dukungan Kinesis Data Streams diAWS CLItidak menyediakan Base64decodingkarena Base64 decoding ke konten biner mentah dicetak ke stdout dapat menyebabkan perilaku yang tidak diinginkan dan potensi masalah keamanan pada platform dan terminal tertentu. Jika Anda menggunakan decoder Base64 (misalnya,https://www.base64decode.org/) untuk secara manual decoded`GVzdGRhdGE=`Anda akan melihat bahwa itu adalah, pada kenyataannya,`testdata`. Ini cukup demi tutorial ini karena, dalam prakteknya,AWS CLIjarang digunakan untuk mengkonsumsi data, tetapi lebih sering untuk memantau keadaan sungai dan mendapatkan informasi, seperti yang ditunjukkan sebelumnya (`describe-stream`dan`list-streams`). Tutorial masa depan akan menunjukkan kepada Anda bagaimana membangun aplikasi konsumen berkualitas produksi menggunakan Kinesis Client Library (KCL), di mana Base64 diurus untuk Anda. Untuk informasi lebih lanjut tentang KCL, lihatMengembangkan Konsumen Kustom dengan Throughput Bersama Menggunakan KCL.

Hal ini tidak selalu terjadi`get-records`akan mengembalikan semua catatan di stream/shard yang ditentukan. Ketika itu terjadi, gunakan`NextShardIterator`dari hasil terakhir untuk mendapatkan set berikutnya catatan. Jadi jika lebih banyak data yang dimasukkan ke dalam aliran (situasi normal dalam aplikasi produksi), Anda dapat menyimpan polling untuk data menggunakan`get-records`setiap kali. Namun, jika Anda tidak menelepon`get-records`menggunakan iterator shard berikutnya dalam seumur hidup iterator pecahan 300 kedua, Anda akan mendapatkan pesan kesalahan, dan Anda akan perlu menggunakan`get-shard-iterator`perintah untuk mendapatkan iterator beling segar.

Juga disediakan dalam output ini`MillisBehindLatest`, yang merupakan jumlah milidetikGetRecordsRespon operasi adalah dari ujung aliran, menunjukkan seberapa jauh di

belakang waktu konsumen saat ini. Nilai nol menunjukkan pemrosesan catatan sedang dilakukan, dan tidak ada catatan baru untuk diproses saat ini. Dalam kasus tutorial ini, Anda mungkin melihat angka yang cukup besar jika Anda telah meluangkan waktu untuk membaca bersama saat Anda pergi. Itu bukan masalah, secara default, catatan data tetap dalam aliran selama 24 jam menunggu Anda untuk mengambilnya. Kerangka waktu ini disebut periode retensi dan dapat dikonfigurasi hingga 365 hari.

Perhatikan bahwa suksesget-recordsakan selalu memilikiNextShardIteratorbahkan jika tidak ada lagi catatan saat ini di sungai. Ini adalah model pemungutan suara yang mengasumsikan produser berpotensi menempatkan lebih banyak catatan ke dalam aliran pada waktu tertentu. Meskipun Anda dapat menulis rutinitas pemungutan suara Anda sendiri, jika Anda menggunakan KCL yang disebutkan sebelumnya untuk mengembangkan aplikasi konsumen, polling ini diurus untuk Anda.

Jika Anda meneleponget-recordssampai tidak ada lagi catatan dalam aliran dan beling Anda menarik dari, Anda akan melihat output dengan catatan kosong mirip dengan contoh berikut (gulir horizontal untuk melihat seluruh output):

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

Langkah 4: Pembersihan

Akhirnya, Anda akan ingin menghapus stream Anda untuk membebaskan sumber daya dan menghindari biaya yang tidak diinginkan ke akun Anda, seperti yang disebutkan sebelumnya. Lakukan ini dalam praktek setiap saat Anda telah membuat aliran dan tidak akan menggunakannya karena biaya bertambah per aliran apakah Anda menempatkan dan mendapatkan data dengan itu atau tidak. Perintah bersih-bersih sederhana:

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

Hasil sukses tidak ada output, jadi Anda mungkin ingin menggunakandescribe-streamuntuk memeriksa kemajuan penghapusan:

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

Jika Anda menjalankan perintah ini segera setelah perintah delete, Anda mungkin akan melihat bagian output yang mirip dengan contoh berikut:

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

Setelah aliran dihapus sepenuhnya,`describe-stream`akan menghasilkan kesalahan “tidak ditemukan”:

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

Contoh Tutorial untuk Amazon Kinesis Data Streams

Contoh tutorial di bagian ini dirancang untuk lebih membantu Anda dalam memahami konsep dan fungsionalitas Amazon Kinesis Data Streams.

Topik

- Tutorial: Memproses Data Stock Real-Time Menggunakan KPL dan KCL 2.x
- Tutorial: Memproses Data Stock Real-Time Menggunakan KPL dan KCL 1.x
- Tutorial: Menganalisis Data Stok Real-Time Menggunakan Managed Service untuk Apache Flink untuk Aplikasi Flink
- Tutorial: MenggunakanAWSLambda dengan Amazon Kinesis Data Streams
- Solusi Data Streaming AWS untuk Amazon Kinesis

Tutorial: Memproses Data Stock Real-Time Menggunakan KPL dan KCL 2.x

Skenario untuk tutorial ini melibatkan menelan perdagangan saham ke dalam aliran data dan menulis aplikasi Amazon Kinesis Data Streams sederhana yang melakukan perhitungan di stream. Anda akan belajar cara mengirim aliran catatan ke Kinesis Data Streams dan mengimplementasikan aplikasi yang mengkonsumsi dan memproses catatan dalam waktu hampir nyata.

> **Important**
>
> Setelah Anda membuat stream, akun Anda dikenakan biaya nominal untuk penggunaan Kinesis Data Streams karena Kinesis Data Streams tidak memenuhi syarat untuk Tingkat Gratis. AWS Setelah aplikasi konsumen dimulai, aplikasi ini juga dikenakan biaya nominal untuk penggunaan Amazon DynamoDB. Aplikasi konsumen menggunakan DynamoDB untuk melacak status pemrosesan. Setelah selesai dengan aplikasi ini, hapus AWS sumber daya Anda untuk menghentikan biaya. Untuk informasi selengkapnya, lihat Langkah 7: Finishing Up.

Kode tidak mengakses data pasar saham aktual, tetapi mensimulasikan aliran perdagangan saham. Ia melakukannya dengan menggunakan generator perdagangan saham acak yang memiliki titik awal data pasar riil untuk 25 saham teratas berdasarkan kapitalisasi pasar per Februari 2015. Jika

Anda memiliki akses ke aliran perdagangan saham secara real-time, Anda mungkin tertarik untuk mendapatkan statistik yang berguna dan tepat waktu dari aliran itu. Misalnya, Anda mungkin ingin melakukan analisis jendela geser di mana Anda menentukan stok paling populer yang dibeli dalam 5 menit terakhir. Atau Anda mungkin ingin pemberitahuan setiap kali ada pesanan jual yang terlalu besar (yaitu, ia memiliki terlalu banyak saham). Anda dapat memperluas kode dalam seri ini untuk menyediakan fungsionalitas tersebut.

Anda dapat mengerjakan langkah-langkah dalam tutorial ini di komputer desktop atau laptop Anda dan menjalankan kode produser dan konsumen pada mesin yang sama atau platform apa pun yang mendukung persyaratan yang ditentukan.

Contoh yang ditampilkan menggunakan wilayah AS Barat (Oregon), tetapi mereka bekerja pada salah satu AWSwilayah yang mendukung Kinesis Data Streams.

Tugas

Prasyarat

Anda harus memenuhi persyaratan berikut untuk menyelesaikan tutorial ini:

Akun Layanan Web Amazon

Sebelum memulai, pastikan Anda terbiasa dengan konsep yang dibahasTerminologi dan Konsep Aliran Data Kinesis Amazon, terutama dengan aliran, pecahan, produsen, dan konsumen. Hal ini juga membantu untuk telah menyelesaikan langkah-langkah dalam panduan berikut:Menginstal dan MengonfigurasiAWS CLI.

Anda harus memiliki AWS akun dan browser web untuk mengaksesAWS Management Console.

Untuk akses konsol, gunakan nama pengguna dan kata sandi IAM Anda untuk masuk ke halaman masuk AWS Management Consoledari IAM. Untuk informasi tentang kredensi AWS keamanan, termasuk akses terprogram dan alternatif untuk kredensi jangka panjang, lihat kredensi AWSkeamanan di Panduan Pengguna IAM. Untuk detail tentang masuk ke AndaAkun AWS, lihat Cara masuk ke AWS dalam Panduan AWS Sign-In Pengguna.

Untuk informasi selengkapnya tentang IAM dan petunjuk pengaturan kunci keamanan, lihat Membuat Pengguna IAM.

Persyaratan Perangkat Lunak Sistem

Sistem yang Anda gunakan untuk menjalankan aplikasi harus memiliki Java 7 atau lebih tinggi diinstal. Untuk mengunduh dan menginstal Java Development Kit (JDK) terbaru, buka situs instalasi Java SE Oracle.

Jika Anda memiliki IDE Java, seperti Eclipse, Anda dapat menggunakannya untuk membuka, mengedit, membangun, dan menjalankan kode sumber.

Anda memerlukan AWS SDK for Javaversi terbaru. Jika Anda menggunakan Eclipse sebagai IDE Anda, Anda dapat menginstal AWSToolkit untuk Eclipse sebagai gantinya.

Aplikasi konsumen memerlukan Kinesis Client Library (KCL) versi 2.2.9 atau lebih tinggi, yang dapat Anda peroleh dari GitHub https://github.com/awslabs/ /tree/master. amazon-kinesis-client

Langkah Selanjutnya

Langkah 1: Membuat Stream Data

Langkah 1: Membuat Stream Data

Pertama, Anda harus membuat aliran data yang akan Anda gunakan dalam langkah-langkah selanjutnya dari tutorial ini.

Untuk membuat aliran

1. Masuk ke AWS Management Console dan buka konsol Kinesis di https://console.aws.amazon.com/kinesis.
2. Pilih Data Streams (Aliran Data) di panel navigasi.
3. Di bilah navigasi, perluas pemilih wilayah dan pilih wilayah.

4. Pilih Buat aliran Kinesis.
5. Masukkan nama untuk aliran data Anda (misalnya,**`StockTradeStream`**).
6. Masukkan **1** jumlah pecahan, tetapi tinggalkan Perkirakan jumlah pecahan yang Anda perlukan untuk diciutkan.
7. Pilih Buat aliran Kinesis.

Pada halaman daftar aliran Kinesis, status streaming Anda muncul `CREATING` saat streaming sedang dibuat. Saat streaming siap digunakan, status akan berubah menjadi`ACTIVE`.

Jika Anda memilih nama stream Anda, di halaman yang muncul, tab Detail menampilkan ringkasan konfigurasi aliran data Anda. Bagian Monitoring menampilkan informasi pemantauan untuk aliran.

Langkah Selanjutnya

Langkah 2: Buat Kebijakan dan Pengguna IAM

Langkah 2: Buat Kebijakan dan Pengguna IAM

Praktik terbaik keamanan untuk AWS menentukan penggunaan izin berbutir halus untuk mengontrol akses ke sumber daya yang berbeda. AWS Identity and Access Management (IAM) memungkinkan Anda untuk mengelola pengguna dan izin pengguna di. AWS Kebijakan IAM secara eksplisit mencantumkan tindakan yang diizinkan dan sumber daya di mana tindakan tersebut berlaku.

Berikut ini adalah izin minimum yang umumnya diperlukan untuk produsen dan konsumen Kinesis Data Streams.

Produser

Tindakan	Resource	Tujuan
`DescribeStream` , `DescribeStreamSummary` , `DescribeStreamConsumer`	Aliran data kinesis	Sebelum mencoba membaca catatan, konsumen memeriks ada, apakah data tersebut aktif, dan apakah pecahan terse aliran data.
`SubscribeToShard` , `RegisterStreamConsumer`	Aliran data kinesis	Berlangganan dan mendaftarkan konsumen ke pecahan.

Tindakan	Resource	Tujuan
`PutRecord` , `PutRecords`	Aliran data kinesis	Menulis catatan ke Kinesis Data Streams.

Konsumen

Tindakan	Sumber Daya	Tujuan
`DescribeStream`	Aliran data kinesis	Sebelum mencoba membaca catatan, konsumen memeriks ada, apakah data tersebut aktif, dan apakah pecahan terse aliran data.
`GetRecords` , `GetShardIterator`	Aliran data kinesis	Membaca catatan dari pecahan.
`CreateTable` , `DescribeTable` , `GetItem`, `PutItem`, `Scan`, `UpdateItem`	Tabel Amazon DynamoDB	Jika konsumen dikembangkan menggunakan Kinesis Clien versi 1.x atau 2.x), dibutuhkan izin ke tabel DynamoDB untu pemrosesan aplikasi.
`DeleteItem`	Tabel Amazon DynamoDB	Untuk saat konsumen melakukan operasi split/merge pada Streams.
`PutMetricData`	CloudWatc hLog Amazon	KCL juga mengunggah metrik keCloudWatch, yang bergun aplikasi.

Untuk tutorial ini, Anda akan membuat kebijakan IAM tunggal yang memberikan semua izin di atas. Dalam produksi, Anda mungkin ingin membuat dua kebijakan, satu untuk produsen dan satu untuk konsumen.

Untuk membuat kebijakan IAM

1. Temukan Amazon Resource Name (ARN) untuk aliran data baru yang Anda buat pada langkah di atas. Anda dapat menemukan ARN ini terdaftar sebagai Streaming ARN di bagian atas tab Detail. Format ARN adalah sebagai berikut:

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

wilayah

Kode AWS wilayah; misalnya,`us-west-2`. Untuk informasi selengkapnya, lihat Konsep Wilayah dan Availability Zone.

akun

ID AWS akun, seperti yang ditunjukkan pada Pengaturan Akun.

nama

Nama aliran data yang Anda buat pada langkah di atas, yaitu`StockTradeStream`.

2. Tentukan ARN untuk tabel DynamoDB yang akan digunakan oleh konsumen (dan akan dibuat oleh instance konsumen pertama). Itu harus dalam format berikut:

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

Wilayah dan ID akun identik dengan nilai dalam ARN aliran data yang Anda gunakan untuk tutorial ini, tetapi namanya adalah nama tabel DynamoDB yang dibuat dan digunakan oleh aplikasi konsumen. KCL menggunakan nama aplikasi sebagai nama tabel. Pada langkah ini, gunakan `StockTradesProcessor` untuk nama tabel DynamoDB, karena itu adalah nama aplikasi yang digunakan dalam langkah-langkah selanjutnya dalam tutorial ini.

3. Di konsol IAM, di Kebijakan (https://console.aws.amazon.com/iam/home#policies), pilih Buat kebijakan. Jika ini adalah pertama kalinya Anda bekerja dengan kebijakan IAM, pilih Mulai, Buat Kebijakan.
4. Pilih Pilih di samping Pembuat Kebijakan.
5. Pilih Amazon Kinesis sebagai AWS layanan.
6. Pilih`DescribeStream`,`GetShardIterator`,`GetRecords`,`PutRecord`,, dan `PutRecords` sebagai tindakan yang diizinkan.
7. Masukkan ARN dari data stream yang Anda gunakan dalam tutorial ini.
8. Gunakan Tambahkan Pernyataan untuk masing-masing berikut ini:

AWSLayanan	Tindakan	ARN
Amazon DynamoDB	`CreateTable` , `DeleteItem` , `DescribeTable` , `GetItem`, `PutItem`, `Scan`, `UpdateItem`	ARN tabel DynamoDB yang Anda buat di Langkah 2 dari prosedur ini.
Amazon CloudWatch	`PutMetricData`	*

Tanda bintang (*) yang digunakan saat menentukan ARN tidak diperlukan. Dalam hal ini, itu karena tidak ada sumber daya khusus CloudWatch di mana `PutMetricData` tindakan dipanggil.

9. Pilih Langkah Selanjutnya.
10. Ubah Nama Kebijakan menjadi`StockTradeStreamPolicy`, tinjau kode, dan pilih Buat Kebijakan.

Dokumen kebijakan yang dihasilkan akan terlihat seperti ini:

```
{
  "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": [
        "*"
      ]
    }
  ]
}
```

Untuk membuat pengguna IAM

1. Buka konsol IAM di https://console.aws.amazon.com/iam/.
2. Pada halaman Pengguna, pilih Tambah pengguna.
3. Untuk Nama pengguna, ketik `StockTradeStreamUser`.
4. Untuk Jenis akses, pilih Akses terprogram, lalu pilih Berikutnya: Izin.

5. Pilih Lampirkan kebijakan yang sudah ada secara langsung.
6. Cari berdasarkan nama untuk kebijakan yang Anda buat dalam prosedur di atas (`StockTradeStreamPolicy`. Pilih kotak di sebelah kiri nama kebijakan, lalu pilih Berikutnya: Tinjauan.
7. Tinjau detail dan ringkasan, lalu pilih Buat pengguna.
8. Salin ID kunci akses, dan simpan secara pribadi. Di bawah Kunci akses rahasia, pilih Tampilkan, dan simpan kunci itu secara pribadi juga.
9. Tempel kunci akses dan rahasia ke file lokal di tempat aman yang hanya dapat Anda akses. Untuk aplikasi ini, buat file bernama `~/.aws/credentials` (dengan izin ketat). File harus dalam format berikut:

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

Untuk melampirkan kebijakan IAM ke pengguna

1. Di konsol IAM, buka Kebijakan dan pilih Tindakan Kebijakan.
2. Pilih `StockTradeStreamPolicy` dan Lampirkan.
3. Pilih `StockTradeStreamUser` dan Lampirkan Kebijakan.

Langkah Selanjutnya

Langkah 3: Unduh dan Bangun Kode

Langkah 3: Unduh dan Bangun Kode

Topik ini memberikan contoh kode implementasi untuk sampel perdagangan saham menelan ke dalam aliran data (produser) dan pemrosesan data ini (konsumen).

Untuk mengunduh dan membuat kode

1. Unduh kode sumber dari amazon-kinesis-learning GitHub repo https://github.com/aws-samples/ ke komputer Anda.
2. Buat proyek di IDE Anda dengan kode sumber, mengikuti struktur direktori yang disediakan.
3. Tambahkan pustaka berikut ke proyek:

- Perpustakaan Klien Amazon Kinesis (KCL)
- AWS SDK
- Apache HttpCore
- Apache HttpClient
- Apache Commons
- Apache Commons Logging
- Jambu biji (Perpustakaan Inti Google Untuk Java)
- Anotasi Jackson
- Jackson Inti
- Jackson Databind
- Jackson Format Data: CBOR
- Waktu Joda

4. Tergantung pada IDE Anda, proyek mungkin dibangun secara otomatis. Jika tidak, membangun proyek menggunakan langkah-langkah yang sesuai untuk IDE Anda.

Jika Anda menyelesaikan langkah-langkah ini dengan sukses, Anda sekarang siap untuk pindah ke bagian berikutnya,the section called “Langkah 4: Menerapkan Produser”.

Langkah Selanjutnya

Langkah 4: Menerapkan Produser

Tutorial ini menggunakan skenario dunia nyata dari pemantauan perdagangan pasar saham. Prinsip-prinsip berikut menjelaskan secara singkat bagaimana skenario ini memetakan ke produsen dan struktur kode pendukungnya.

Lihat kode sumber dan tinjau informasi berikut.

StockTrade kelas

Perdagangan saham individu diwakili oleh instance StockTrade kelas. Instance ini berisi atribut seperti simbol ticker, harga, jumlah saham, jenis perdagangan (beli atau jual), dan ID yang secara unik mengidentifikasi perdagangan. Kelas ini diimplementasikan untuk Anda.

Rekaman aliran

Aliran adalah urutan catatan. Rekaman adalah serialisasi `StockTrade` instance dalam format JSON. Misalnya:

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

StockTradeGenerator kelas

StockTradeGeneratormemiliki metode `getRandomTrade()` yang disebut yang mengembalikan perdagangan saham baru yang dihasilkan secara acak setiap kali dipanggil. Kelas ini diimplementasikan untuk Anda.

StockTradesWriter kelas

`main`Metode produser, StockTradesWriter terus mengambil perdagangan acak dan kemudian mengirimkannya ke Kinesis Data Streams dengan melakukan tugas-tugas berikut:

1. Membaca nama aliran data dan nama wilayah sebagai masukan.
2. Menggunakan `KinesisAsyncClientBuilder` untuk mengatur wilayah, kredensyal, dan konfigurasi klien.
3. Memeriksa bahwa aliran ada dan aktif (jika tidak, itu keluar dengan kesalahan).
4. Dalam loop kontinu, memanggil `StockTradeGenerator.getRandomTrade()` metode dan kemudian `sendStockTrade` metode untuk mengirim perdagangan ke aliran setiap 100 milidetik.

`sendStockTrade`Metode `StockTradesWriter` kelas memiliki kode berikut:

```
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);
        }
    }
```

Lihat rincian kode berikut:

- `PutRecord`API mengharapkan array byte, dan Anda perlu mengonversi perdagangan ke format JSON. Baris kode tunggal ini melakukan operasi itu:

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

- Sebelum Anda dapat mengirim perdagangan, Anda membuat `PutRecordRequest` instance baru (disebut permintaan dalam kasus ini). Masing-masing `request` membutuhkan nama stream, kunci partisi, dan gumpalan data.

  ```
  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();
```

Contoh menggunakan tiket saham sebagai kunci partisi, yang memetakan catatan ke pecahan tertentu. Dalam praktiknya, Anda harus memiliki ratusan atau ribuan kunci partisi per pecahan sehingga catatan tersebar merata di seluruh aliran Anda. Untuk informasi selengkapnya tentang cara menambahkan data ke stream, lihatMenentukan Amazon Kinesis Data Streams.

Sekarang `request` siap untuk mengirim ke klien (operasi put):

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

- Pemeriksaan kesalahan dan pencatatan selalu merupakan tambahan yang berguna. Kode ini log kondisi kesalahan:

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

Tambahkan blok try/catch di sekitar operasi: `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);
}
```

Hal ini karena Kinesis Data Streams menempatkan operasi dapat gagal karena kesalahan jaringan, atau karena aliran data mencapai batas throughput dan mendapatkan throttled. Dianjurkan agar Anda mempertimbangkan dengan cermat kebijakan coba ulang Anda untuk

`put` operasi untuk menghindari kehilangan data, seperti menggunakan percobaan ulang sederhana.

- Pencatatan status sangat membantu tetapi opsional:

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

Produser yang ditampilkan di sini menggunakan fungsionalitas rekaman tunggal API Kinesis Data Streams,. `PutRecord` Dalam prakteknya, jika produsen individu menghasilkan banyak catatan, seringkali lebih efisien untuk menggunakan beberapa fungsi catatan `PutRecords` dan mengirim batch catatan pada satu waktu. Untuk informasi selengkapnya, lihat Menentukan Amazon Kinesis Data Streams.

Untuk menjalankan produser

1. Verifikasi bahwa kunci akses dan pasangan kunci rahasia Langkah 2: Buat Kebijakan dan Pengguna IAM yang diambil disimpan dalam file`~/.aws/credentials`.
2. Jalankan `StockTradeWriter` kelas dengan argumen berikut:

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

Jika Anda membuat streaming di wilayah selain`us-west-2`, Anda harus menentukan wilayah itu di sini sebagai gantinya.

Anda akan melihat output yang serupa dengan yang berikut:

```
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
```

Perdagangan saham Anda sekarang sedang dicerna oleh Kinesis Data Streams.

Langkah Selanjutnya

Langkah 5: Menerapkan Konsumen

Langkah 5: Menerapkan Konsumen

Aplikasi konsumen dalam tutorial ini terus memproses perdagangan saham di aliran data Anda. Kemudian output saham paling populer yang dibeli dan dijual setiap menit. Aplikasi ini dibangun di atas Kinesis Client Library (KCL), yang melakukan banyak hal yang umum untuk aplikasi konsumen. Untuk informasi selengkapnya, lihat Menggunakan Perpustakaan Klien Kinesis.

Lihat kode sumber dan tinjau informasi berikut.

StockTradesProcessor kelas

Kelas utama konsumen, disediakan untuk Anda, yang melakukan tugas-tugas berikut:

- Membaca aplikasi, aliran data, dan nama wilayah, diteruskan sebagai argumen.
- Menciptakan sebuah `KinesisAsyncClient` instance dengan nama wilayah.
- Menciptakan sebuah `StockTradeRecordProcessorFactory` contoh yang melayani contoh`ShardRecordProcessor`, diimplementasikan oleh sebuah `StockTradeRecordProcessor` instance.
- Menciptakan sebuah `ConfigsBuilder` contoh dengan`KinesisAsyncClient`,`StreamName`, `ApplicationName` dan `StockTradeRecordProcessorFactory` contoh. Ini berguna untuk membuat semua konfigurasi dengan nilai default.
- Menciptakan scheduler KCL (sebelumnya, dalam versi KCL 1.x itu dikenal sebagai pekerja KCL) dengan contoh. `ConfigsBuilder`
- Scheduler membuat thread baru untuk setiap shard (ditugaskan untuk instance konsumen ini), yang terus loop untuk membaca catatan dari aliran data. Kemudian memanggil `StockTradeRecordProcessor` instance untuk memproses setiap batch catatan yang diterima.

StockTradeRecordProcessor kelas

Pelaksanaan `StockTradeRecordProcessor` contoh, yang pada gilirannya mengimplementasikan lima metode yang diperlukan:`initialize`,`processRecords`,, `leaseLost`shardEnded, dan`shutdownRequested`.

`shutdownRequested`Metode `initialize` dan digunakan oleh KCL untuk membiarkan prosesor rekaman tahu kapan harus siap untuk mulai menerima catatan dan kapan harus mengharapkan untuk berhenti menerima catatan, masing-masing, sehingga dapat melakukan pengaturan dan penghentian tugas khusus aplikasi. `leaseLost`dan `shardEnded` digunakan untuk menerapkan logika apa pun untuk apa yang harus dilakukan ketika sewa hilang atau pemrosesan telah mencapai akhir pecahan. Dalam contoh ini, kita cukup mencatat pesan yang menunjukkan peristiwa ini.

Kode untuk metode ini disediakan untuk Anda. Pemrosesan utama terjadi dalam `processRecords` metode, yang pada gilirannya digunakan `processRecord` untuk setiap catatan. Metode terakhir ini disediakan sebagai kode kerangka sebagian besar kosong bagi Anda untuk menerapkan pada langkah berikutnya, di mana dijelaskan secara lebih rinci.

Yang juga perlu diperhatikan adalah penerapan metode dukungan untuk`processRecord`:`reportStats`, dan`resetStats`, yang kosong dalam kode sumber asli.

`processRecords`Metode ini diimplementasikan untuk Anda, dan melakukan langkah-langkah berikut:

- Untuk setiap catatan yang diteruskan, ia `processRecord` memanggilnya.
- Jika setidaknya 1 menit telah berlalu sejak laporan terakhir, panggilan`reportStats()`, yang mencetak statistik terbaru, dan kemudian `resetStats()` yang menghapus statistik sehingga interval berikutnya hanya mencakup catatan baru.
- Menetapkan waktu pelaporan berikutnya.
- Jika setidaknya 1 menit telah berlalu sejak pos pemeriksaan terakhir, panggilan. `checkpoint()`
- Menetapkan waktu checkpointing berikutnya.

Metode ini menggunakan interval 60 detik untuk tingkat pelaporan dan checkpointing. Untuk informasi selengkapnya tentang checkpointing, lihat Menggunakan Kinesis Client Library.

StockStats kelas

Kelas ini menyediakan retensi data dan pelacakan statistik untuk saham paling populer dari waktu ke waktu. Kode ini disediakan untuk Anda dan berisi metode berikut:

- `addStockTrade(StockTrade)`: menyuntikkan yang diberikan `StockTrade` ke dalam statistik berjalan.
- `toString()`: mengembalikan statistik dalam string diformat.

Kelas ini melacak saham paling populer dengan menjaga hitungan berjalan dari jumlah total perdagangan untuk setiap saham dan jumlah maksimum. Ini memperbarui jumlah ini setiap kali perdagangan saham tiba.

Tambahkan kode ke metode `StockTradeRecordProcessor` kelas, seperti yang ditunjukkan pada langkah-langkah berikut.

Untuk mengimplementasikan konsumen

1. Menerapkan `processRecord` metode dengan instantiating `StockTrade` objek berukuran benar dan menambahkan data record untuk itu, log peringatan jika ada masalah.

```
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. Menerapkan `reportStats` metode sederhana. Jangan ragu untuk memodifikasi format keluaran agar sesuai dengan preferensi Anda.

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

3. Menerapkan `resetStats` metode, yang menciptakan `stockStats` contoh baru.

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

4. Menerapkan metode berikut yang diperlukan oleh `ShardRecordProcessor` antarmuka

```
@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);
        }
    }
```

Untuk menjalankan konsumen

1. Jalankan produser yang Anda tulis untuk menyuntikkan catatan perdagangan saham simulasi ke aliran Anda.
2. Verifikasi bahwa kunci akses dan pasangan kunci rahasia yang diambil sebelumnya (saat membuat pengguna IAM) disimpan dalam file. `~/.aws/credentials`
3. Jalankan `StockTradesProcessor` kelas dengan argumen berikut:

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

   Perhatikan bahwa jika Anda membuat streaming di wilayah selain`us-west-2`, Anda harus menentukan wilayah tersebut di sini sebagai gantinya.

Setelah satu menit, Anda akan melihat output seperti berikut, disegarkan setiap menit setelahnya:

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

Langkah Selanjutnya

[Langkah 6: (Opsional) Memperluas Konsumen]

Langkah 6: (Opsional) Memperluas Konsumen

Bagian opsional ini menunjukkan bagaimana Anda dapat memperluas kode konsumen untuk skenario yang sedikit lebih rumit.

Jika Anda ingin tahu tentang pesanan jual terbesar setiap menit, Anda dapat memodifikasi `StockStats` kelas di tiga tempat untuk mengakomodasi prioritas baru ini.

Untuk memperluas konsumen

1. Tambahkan variabel instance baru:

```
// 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. Tambahkan kode berikut ke `addStockTrade`:

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

3. Memodifikasi `toString` metode untuk mencetak informasi tambahan:

```
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);
}
```

Jika Anda menjalankan konsumen sekarang (ingat untuk menjalankan produser juga), Anda akan melihat output yang mirip dengan ini:

```
****** 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.
****************************************************************
```

Langkah Selanjutnya

Langkah 7: Finishing Up

Langkah 7: Finishing Up

Karena Anda membayar untuk menggunakan aliran data Kinesis, pastikan Anda menghapusnya dan tabel Amazon DynamoDB yang sesuai saat Anda selesai menggunakannya. Biaya nominal terjadi pada aliran aktif bahkan ketika Anda tidak mengirim dan mendapatkan catatan. Hal ini karena aliran aktif menggunakan sumber daya dengan terus “mendengarkan” untuk catatan masuk dan permintaan untuk mendapatkan catatan.

Untuk menghapus aliran dan tabel

1. Matikan produsen dan konsumen yang mungkin masih Anda jalankan.
2. Buka konsol Kinesis di https://console.aws.amazon.com/kinesis.
3. Pilih aliran yang Anda buat untuk aplikasi ini (`StockTradeStream`).
4. Pilih Hapus Stream.
5. Buka konsol DynamoDB di https://console.aws.amazon.com/dynamodb/.
6. Hapus tabel `StockTradesProcessor`.

Ringkasan

Memproses sejumlah besar data dalam waktu hampir nyata tidak memerlukan penulisan kode ajaib atau mengembangkan infrastruktur besar. Ini semudah menulis logika untuk memproses sejumlah kecil data (seperti menulis`processRecord(Record)`) tetapi menggunakan Kinesis Data Streams untuk skala sehingga bekerja untuk sejumlah besar data streaming. Anda tidak perlu khawatir tentang bagaimana pemrosesan Anda akan menskalakan karena Kinesis Data Streams menanganinya untuk Anda. Yang harus Anda lakukan adalah mengirim catatan streaming Anda ke Kinesis Data Streams dan menulis logika untuk memproses setiap catatan baru yang diterima.

Berikut adalah beberapa potensi peningkatan untuk aplikasi ini.

Agregat di semua pecahan

Saat ini, Anda mendapatkan statistik yang dihasilkan dari agregasi catatan data yang diterima oleh pekerja tunggal dari satu pecahan. (Shard tidak dapat diproses oleh lebih dari satu pekerja dalam satu aplikasi secara bersamaan.) Tentu saja, ketika Anda skala dan memiliki lebih dari satu pecahan, Anda mungkin ingin agregat di semua pecahan. Anda dapat melakukan ini dengan memiliki arsitektur pipeline di mana output dari setiap pekerja dimasukkan ke aliran lain dengan pecahan tunggal, yang diproses oleh pekerja yang mengumpulkan output dari tahap pertama. Karena data dari tahap pertama terbatas (satu sampel per menit per pecahan), maka akan mudah ditangani oleh satu pecahan.

Pemrosesan skala

Ketika aliran skala hingga memiliki banyak pecahan (karena banyak produsen mengirim data), cara untuk skala pemrosesan adalah dengan menambahkan lebih banyak pekerja. Anda dapat menjalankan pekerja di instans Amazon EC2 dan menggunakan grup Penskalaan Otomatis.

Gunakan konektor ke Amazon S3/Dynamodb/Amazon Redshift/Storm

Karena aliran terus diproses, hasilnya dapat dikirim ke tujuan lain. AWSmenyediakan konektor untuk mengintegrasikan Kinesis Data Streams dengan AWS layanan lain dan alat pihak ketiga.

Tutorial: Memproses Data Stock Real-Time Menggunakan KPL dan KCL 1.x

Skenario untuk tutorial ini melibatkan menelan perdagangan saham ke dalam aliran data dan menulis aplikasi Amazon Kinesis Data Streams sederhana yang melakukan perhitungan di stream. Anda akan

belajar cara mengirim aliran catatan ke Kinesis Data Streams dan mengimplementasikan aplikasi yang mengkonsumsi dan memproses catatan dalam waktu hampir nyata.

> **Important**
>
> Setelah Anda membuat stream, akun Anda dikenakan biaya nominal untuk penggunaan Kinesis Data Streams karena Kinesis Data Streams tidak memenuhi syarat untuk Tingkat Gratis. AWS Setelah aplikasi konsumen dimulai, aplikasi ini juga dikenakan biaya nominal untuk penggunaan Amazon DynamoDB. Aplikasi konsumen menggunakan DynamoDB untuk melacak status pemrosesan. Setelah selesai dengan aplikasi ini, hapus AWS sumber daya Anda untuk menghentikan biaya. Untuk informasi selengkapnya, lihat Langkah 7: Finishing Up.

Kode tidak mengakses data pasar saham aktual, tetapi mensimulasikan aliran perdagangan saham. Ia melakukannya dengan menggunakan generator perdagangan saham acak yang memiliki titik awal data pasar riil untuk 25 saham teratas berdasarkan kapitalisasi pasar per Februari 2015. Jika Anda memiliki akses ke aliran perdagangan saham secara real-time, Anda mungkin tertarik untuk mendapatkan statistik yang berguna dan tepat waktu dari aliran itu. Misalnya, Anda mungkin ingin melakukan analisis jendela geser di mana Anda menentukan stok paling populer yang dibeli dalam 5 menit terakhir. Atau Anda mungkin ingin pemberitahuan setiap kali ada pesanan jual yang terlalu besar (yaitu, ia memiliki terlalu banyak saham). Anda dapat memperluas kode dalam seri ini untuk menyediakan fungsionalitas tersebut.

Anda dapat mengerjakan langkah-langkah dalam tutorial ini di komputer desktop atau laptop Anda dan menjalankan kode produser dan konsumen pada mesin yang sama atau platform apa pun yang mendukung persyaratan yang ditentukan, seperti Amazon Elastic Compute Cloud (Amazon EC2).

Contoh yang ditampilkan menggunakan Wilayah AS Barat (Oregon), tetapi mereka bekerja pada salah satu AWSWilayah yang mendukung Kinesis Data Streams.

Tugas

- Prasyarat
- Langkah 1: Membuat Stream Data
- Langkah 2: Buat Kebijakan dan Pengguna IAM
- Langkah 3: Unduh dan Bangun Kode Implementasi
- Langkah 4: Menerapkan Produser

- Langkah 5: Menerapkan Konsumen
- Langkah 6: (Opsional) Memperluas Konsumen
- Langkah 7: Finishing Up

Prasyarat

Berikut ini adalah persyaratan untuk menyelesaikanTutorial: Memproses Data Stock Real-Time Menggunakan KPL dan KCL 1.x.

Akun Layanan Web Amazon

Sebelum memulai, pastikan Anda terbiasa dengan konsep yang dibahasTerminologi dan Konsep Aliran Data Kinesis Amazon, terutama aliran, pecahan, produsen, dan konsumen. Hal ini juga membantu untuk telah selesaiMenginstal dan MengonfigurasiAWS CLI.

Anda memerlukan AWS akun dan browser web untuk mengaksesAWS Management Console.

Untuk akses konsol, gunakan nama pengguna dan kata sandi IAM Anda untuk masuk ke halaman masuk AWS Management Consoledari IAM. Untuk informasi tentang kredensi AWS keamanan, termasuk akses terprogram dan alternatif untuk kredensi jangka panjang, lihat kredensi AWSkeamanan di Panduan Pengguna IAM. Untuk detail tentang masuk ke AndaAkun AWS, lihat Cara masuk ke AWS dalam Panduan AWS Sign-In Pengguna.

Untuk informasi selengkapnya tentang IAM dan petunjuk pengaturan kunci keamanan, lihat Membuat Pengguna IAM.

Persyaratan Perangkat Lunak Sistem

Sistem yang digunakan untuk menjalankan aplikasi harus memiliki Java 7 atau lebih tinggi diinstal. Untuk mengunduh dan menginstal Java Development Kit (JDK) terbaru, buka situs instalasi Java SE Oracle.

Jika Anda memiliki IDE Java, seperti Eclipse, Anda dapat membuka kode sumber, mengedit, membangun, dan menjalankannya.

Anda memerlukan AWS SDK for Javaversi terbaru. Jika Anda menggunakan Eclipse sebagai IDE Anda, Anda dapat menginstal AWSToolkit untuk Eclipse sebagai gantinya.

Aplikasi konsumen memerlukan Kinesis Client Library (KCL) versi 1.2.1 atau lebih tinggi, yang dapat Anda peroleh dari GitHub Kinesis Client Library (Java).

Langkah Selanjutnya

Langkah 1: Membuat Stream Data

Langkah 1: Membuat Stream Data

Pada langkah pertamaTutorial: Memproses Data Stock Real-Time Menggunakan KPL dan KCL 1.x, Anda membuat aliran yang akan Anda gunakan dalam langkah berikutnya.

Untuk membuat aliran

1. Masuk ke AWS Management Console dan buka konsol Kinesis di https://console.aws.amazon.com/kinesis.
2. Pilih Data Streams (Aliran Data) di panel navigasi.
3. Di bilah navigasi, perluas pemilih Region dan pilih Region.
4. Pilih Buat aliran Kinesis.
5. Masukkan nama untuk streaming Anda (misalnya,**`StockTradeStream`**).
6. Masukkan **1** jumlah pecahan, tetapi tinggalkan Perkirakan jumlah pecahan yang Anda perlukan untuk diciutkan.
7. Pilih Buat aliran Kinesis.

Pada halaman daftar aliran Kinesis, status streaming Anda adalah `CREATING` saat streaming sedang dibuat. Saat streaming siap digunakan, status akan berubah menjadi`ACTIVE`. Pilih nama streaming Anda. Di halaman yang muncul, tab Detail menampilkan ringkasan konfigurasi aliran Anda. Bagian Monitoring menampilkan informasi pemantauan untuk aliran.

Informasi Tambahan Tentang Pecahan

Saat Anda mulai menggunakan Kinesis Data Streams di luar tutorial ini, Anda mungkin perlu merencanakan proses pembuatan stream dengan lebih hati-hati. Anda harus merencanakan permintaan maksimum yang diharapkan saat Anda menyediakan pecahan. Dengan menggunakan skenario ini sebagai contoh, lalu lintas perdagangan pasar saham AS memuncak di siang hari (waktu Timur) dan perkiraan permintaan harus diambil sampel dari waktu itu. Anda kemudian memiliki pilihan untuk menyediakan permintaan maksimum yang diharapkan, atau skala aliran Anda naik dan turun dalam menanggapi fluktuasi permintaan.

Shard adalah unit kapasitas throughput. Pada halaman Create Kinesis stream, perluas Perkirakan jumlah pecahan yang Anda perlukan. Masukkan ukuran rekaman rata-rata, catatan maksimum yang ditulis per detik, dan jumlah aplikasi yang dikonsumsi, menggunakan panduan berikut:

Ukuran rekor rata-rata

Perkiraan ukuran rata-rata yang dihitung dari catatan Anda. Jika Anda tidak mengetahui nilai ini, gunakan perkiraan ukuran rekaman maksimum untuk nilai ini.

Max catatan ditulis

Memperhitungkan jumlah entitas yang menyediakan data dan perkiraan jumlah catatan per detik yang dihasilkan oleh masing-masing. Misalnya, jika Anda mendapatkan data perdagangan saham dari 20 server perdagangan dan masing-masing menghasilkan 250 perdagangan per detik, jumlah total perdagangan (catatan) per detik adalah 5000/detik.

Jumlah aplikasi yang mengonsumsi

Jumlah aplikasi yang secara independen membaca dari aliran untuk memproses aliran dengan cara yang berbeda dan menghasilkan output yang berbeda. Setiap aplikasi dapat memiliki beberapa instance yang berjalan pada mesin yang berbeda (yaitu, dijalankan dalam cluster) sehingga dapat mengikuti aliran volume tinggi.

Jika perkiraan jumlah pecahan yang ditampilkan melebihi batas shard Anda saat ini, Anda mungkin perlu mengirimkan permintaan untuk meningkatkan batas tersebut sebelum Anda dapat membuat stream dengan jumlah pecahan tersebut. Untuk meminta peningkatan batas shard Anda, gunakan formulir Batas Aliran Data Kinesis. Untuk informasi selengkapnya tentang aliran dan pecahan, lihat. Membuat dan Mengelola Streaming

Langkah Selanjutnya

Langkah 2: Buat Kebijakan dan Pengguna IAM

Langkah 2: Buat Kebijakan dan Pengguna IAM

Praktik terbaik keamanan untuk AWS menentukan penggunaan izin berbutir halus untuk mengontrol akses ke sumber daya yang berbeda. AWS Identity and Access Management (IAM) memungkinkan Anda untuk mengelola pengguna dan izin pengguna di. AWS Kebijakan IAM secara eksplisit mencantumkan tindakan yang diizinkan dan sumber daya di mana tindakan tersebut berlaku.

Berikut ini adalah izin minimum yang umumnya diperlukan untuk produsen dan konsumen Kinesis Data Streams.

Produser

Tindakan	Resource	Tujuan
`DescribeStream` , `DescribeStreamSummary` , `DescribeStreamConsumer`	Aliran data kinesis	Sebelum mencoba untuk menulis catatan, produsen meme dan aktif, dan jika pecahan yang terkandung dalam aliran, konsumen.
`SubscribeToShard` , `RegisterStreamConsumer`	Aliran data kinesis	Berlangganan dan mendaftarkan konsumen ke pecahan Ki
`PutRecord` , `PutRecords`	Aliran data kinesis	Tulis catatan ke Kinesis Data Streams.

Konsumen

Tindakan	Sumber Daya	Tujuan
`DescribeStream`	Aliran data kinesis	Sebelum mencoba membaca catatan, konsumen memeriks dan aktif, dan apakah pecahan terkandung dalam aliran.
`GetRecords` , `GetShardIterator`	Aliran data kinesis	Baca catatan dari pecahan Kinesis Data Streams.
`CreateTable` , `DescribeTable` , `GetItem`, `PutItem`, `Scan`, `UpdateItem`	Tabel Amazon DynamoDB	Jika konsumen dikembangkan menggunakan Kinesis Clien memerlukan izin ke tabel DynamoDB untuk melacak status Konsumen pertama mulai membuat tabel.
`DeleteItem`	Tabel Amazon DynamoDB	Untuk saat konsumen melakukan operasi split/merge pada Streams.
`PutMetricData`	CloudWatch Log Amazon	KCL juga mengunggah metrik keCloudWatch, yang bergun aplikasi.

Untuk aplikasi ini, Anda membuat kebijakan IAM tunggal yang memberikan semua izin sebelumnya. Dalam praktiknya, Anda mungkin ingin mempertimbangkan untuk membuat dua kebijakan, satu untuk produsen dan satu untuk konsumen.

Untuk membuat kebijakan IAM

1. Temukan Amazon Resource Name (ARN) untuk streaming baru. Anda dapat menemukan ARN ini terdaftar sebagai Streaming ARN di bagian atas tab Detail. Format ARN adalah sebagai berikut:

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

   wilayah

   Kode Region; misalnya,`us-west-2`. Untuk informasi selengkapnya, lihat Konsep Wilayah dan Availability Zone.

   akun

   ID AWS akun, seperti yang ditunjukkan pada Pengaturan Akun.

   nama

   Nama aliran dariLangkah 1: Membuat Stream Data, yang`StockTradeStream`.

2. Tentukan ARN untuk tabel DynamoDB yang akan digunakan oleh konsumen (dan dibuat oleh instance konsumen pertama). Itu harus dalam format berikut:

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

   Wilayah dan akun berasal dari tempat yang sama dengan langkah sebelumnya, tetapi nama waktu ini adalah nama tabel yang dibuat dan digunakan oleh aplikasi konsumen. KCL yang digunakan oleh konsumen menggunakan nama aplikasi sebagai nama tabel. Gunakan`StockTradesProcessor`, yang merupakan nama aplikasi yang digunakan nanti.

3. Di konsol IAM, di Kebijakan (https://console.aws.amazon.com/iam/home#policies), pilih Buat kebijakan. Jika ini adalah pertama kalinya Anda bekerja dengan kebijakan IAM, pilih Mulai, Buat Kebijakan.
4. Pilih Pilih di samping Pembuat Kebijakan.
5. Pilih Amazon Kinesis sebagai AWS layanan.

6. PilihDescribeStream,GetShardIterator,GetRecords,PutRecord,, dan PutRecords sebagai tindakan yang diizinkan.
7. Masukkan ARN yang Anda buat di Langkah 1.
8. Gunakan Tambahkan Pernyataan untuk masing-masing berikut ini:

AWSLayanan	Tindakan	ARN
Amazon DynamoDB	CreateTable , DeleteItem , DescribeTable , GetItem, PutItem, Scan, UpdateItem	ARN yang Anda buat di Langkah 2
Amazon CloudWatch	PutMetricData	*

   Tanda bintang (*) yang digunakan saat menentukan ARN tidak diperlukan. Dalam hal ini, itu karena tidak ada sumber daya khusus CloudWatch di mana PutMetricData tindakan dipanggil.
9. Pilih Langkah Selanjutnya.
10. Ubah Nama Kebijakan menjadiStockTradeStreamPolicy, tinjau kode, dan pilih Buat Kebijakan.

Dokumen kebijakan yang dihasilkan akan terlihat seperti berikut ini:

```
{
  "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": [
        "*"
      ]
    }
  ]
}
```

Untuk membuat pengguna IAM

1. Buka konsol IAM di https://console.aws.amazon.com/iam/.
2. Pada halaman Pengguna, pilih Tambah pengguna.
3. Untuk Nama pengguna, ketik `StockTradeStreamUser`.
4. Untuk Jenis akses, pilih Akses terprogram, lalu pilih Berikutnya: Izin.
5. Pilih Lampirkan kebijakan yang sudah ada secara langsung.
6. Cari berdasarkan nama untuk kebijakan yang Anda buat. Pilih kotak di sebelah kiri nama kebijakan, lalu pilih Berikutnya: Tinjauan.
7. Tinjau detail dan ringkasan, lalu pilih Buat pengguna.
8. Salin ID kunci akses, dan simpan secara pribadi. Di bawah Kunci akses rahasia, pilih Tampilkan, dan simpan kunci itu secara pribadi juga.
9. Tempel kunci akses dan rahasia ke file lokal di tempat aman yang hanya dapat Anda akses. Untuk aplikasi ini, buat file bernama `~/.aws/credentials` (dengan izin yang ketat). File harus dalam format berikut:

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

Untuk melampirkan kebijakan IAM ke pengguna

1. Di konsol IAM, buka Kebijakan dan pilih Tindakan Kebijakan.
2. Pilih `StockTradeStreamPolicy` dan Lampirkan.
3. Pilih `StockTradeStreamUser` dan Lampirkan Kebijakan.

Langkah Selanjutnya

Langkah 3: Unduh dan Bangun Kode Implementasi

Langkah 3: Unduh dan Bangun Kode Implementasi

Kode kerangka disediakan untukthe section called “Tutorial: Memproses Data Stock Real-Time Menggunakan KPL dan KCL 1.x”. Ini berisi implementasi rintisan untuk konsumsi aliran perdagangan

saham (produsen) dan pemrosesan data (konsumen). Prosedur berikut menunjukkan cara menyelesaikan implementasi.

Untuk mengunduh dan membuat kode implementasi

1. Unduh kode sumber ke komputer Anda.
2. Buat proyek di IDE favorit Anda dengan kode sumber, mengikuti struktur direktori yang disediakan.
3. Tambahkan pustaka berikut ke proyek:
   - Perpustakaan Klien Amazon Kinesis (KCL)
   - AWS SDK
   - Apache HttpCore
   - Apache HttpClient
   - Apache Commons
   - Apache Commons Logging
   - Jambu biji (Perpustakaan Inti Google Untuk Java)
   - Anotasi Jackson
   - Jackson Inti
   - Jackson Databind
   - Jackson Format Data: CBOR
   - Waktu Joda
4. Tergantung pada IDE Anda, proyek mungkin dibangun secara otomatis. Jika tidak, membangun proyek menggunakan langkah-langkah yang sesuai untuk IDE Anda.

Jika Anda menyelesaikan langkah-langkah ini dengan sukses, Anda sekarang siap untuk pindah ke bagian berikutnya,the section called “Langkah 4: Menerapkan Produser”. Jika build Anda menghasilkan kesalahan pada tahap apa pun, selidiki dan perbaiki sebelum melanjutkan.

Langkah Selanjutnya

Langkah 4: Menerapkan Produser

Aplikasi dalam Tutorial: Memproses Data Stock Real-Time Menggunakan KPL dan KCL 1.x menggunakan skenario dunia nyata dari pemantauan perdagangan pasar saham. Prinsip-prinsip berikut menjelaskan secara singkat bagaimana skenario ini memetakan ke produsen dan struktur kode pendukung.

Lihat kode sumber dan tinjau informasi berikut.

StockTrade kelas

Perdagangan saham individu diwakili oleh instance `StockTrade` kelas. Instance ini berisi atribut seperti simbol ticker, harga, jumlah saham, jenis perdagangan (beli atau jual), dan ID yang secara unik mengidentifikasi perdagangan. Kelas ini diimplementasikan untuk Anda.

Rekaman aliran

Aliran adalah urutan catatan. Rekaman adalah serialisasi `StockTrade` instance dalam format JSON. Misalnya:

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

StockTradeGenerator kelas

`StockTradeGenerator`memiliki metode `getRandomTrade()` yang disebut yang mengembalikan perdagangan saham baru yang dihasilkan secara acak setiap kali dipanggil. Kelas ini diimplementasikan untuk Anda.

StockTradesWriter kelas

`main`Metode produser, `StockTradesWriter` terus mengambil perdagangan acak dan kemudian mengirimkannya ke Kinesis Data Streams dengan melakukan tugas-tugas berikut:

1. Membaca nama aliran dan nama Region sebagai masukan.
2. Menciptakan sebuah`AmazonKinesisClientBuilder`.
3. Menggunakan pembuat klien untuk mengatur Region, kredenial, dan konfigurasi klien.
4. Membangun `AmazonKinesis` klien menggunakan pembangun klien.
5. Memeriksa bahwa aliran ada dan aktif (jika tidak, itu keluar dengan kesalahan).

6. Dalam loop kontinu, memanggil `StockTradeGenerator.getRandomTrade()` metode dan kemudian `sendStockTrade` metode untuk mengirim perdagangan ke aliran setiap 100 milidetik.

`sendStockTrade`Metode `StockTradesWriter` kelas memiliki kode berikut:

```
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);
    }
}
```

Lihat rincian kode berikut:

- `PutRecord`API mengharapkan array byte, dan Anda perlu mengkonversi `trade` ke format JSON. Baris kode tunggal ini melakukan operasi itu:

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

- Sebelum Anda dapat mengirim perdagangan, Anda membuat `PutRecordRequest` instance baru (disebut `putRecord` dalam kasus ini):

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

Setiap `PutRecord` panggilan membutuhkan nama stream, kunci partisi, dan gumpalan data. Kode berikut mengisi bidang ini dalam `putRecord` objek menggunakan `setXxxx()` metodenya:

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

Contoh menggunakan tiket saham sebagai kunci partisi, yang memetakan catatan ke pecahan tertentu. Dalam praktiknya, Anda harus memiliki ratusan atau ribuan kunci partisi per pecahan sehingga catatan tersebar merata di seluruh aliran Anda. Untuk informasi selengkapnya tentang cara menambahkan data ke stream, lihatMenambahkan Data ke Stream.

Sekarang `putRecord` siap untuk mengirim ke klien (`put`operasi):

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

- Pemeriksaan kesalahan dan pencatatan selalu merupakan tambahan yang berguna. Kode ini log kondisi kesalahan:

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

Tambahkan blok try/catch di sekitar operasi: `put`

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

Ini karena `put` operasi Kinesis Data Streams dapat gagal karena kesalahan jaringan, atau karena aliran mencapai batas throughputnya dan mendapatkan throttled. Kami merekomendasikan dengan cermat mempertimbangkan kebijakan coba ulang Anda untuk `put` operasi untuk menghindari kehilangan data, seperti menggunakan percobaan ulang sederhana.

- Pencatatan status sangat membantu tetapi opsional:

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

Produser yang ditampilkan di sini menggunakan fungsionalitas rekaman tunggal API Kinesis Data Streams,. `PutRecord` Dalam prakteknya, jika produsen individu menghasilkan banyak catatan, seringkali lebih efisien untuk menggunakan beberapa fungsi catatan `PutRecords` dan mengirim batch catatan pada satu waktu. Untuk informasi selengkapnya, lihat Menambahkan Data ke Stream.

Untuk menjalankan produser

1. Verifikasi bahwa kunci akses dan pasangan kunci rahasia yang diambil sebelumnya (saat membuat pengguna IAM) disimpan dalam file. `~/.aws/credentials`
2. Jalankan `StockTradeWriter` kelas dengan argumen berikut:

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

   Jika Anda membuat streaming di Wilayah selain`us-west-2`, Anda harus menentukan Wilayah tersebut di sini.

Anda akan melihat output yang serupa dengan yang berikut:

```
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
```

Aliran perdagangan saham Anda sekarang sedang dicerna oleh Kinesis Data Streams.

Langkah Selanjutnya

Langkah 5: Menerapkan Konsumen

Langkah 5: Menerapkan Konsumen

Aplikasi konsumen dalam Tutorial: Memproses Data Stock Real-Time Menggunakan KPL dan KCL 1.x terus memproses aliran perdagangan saham yang Anda buat. Kemudian output saham paling populer yang dibeli dan dijual setiap menit. Aplikasi ini dibangun di atas Kinesis Client Library (KCL), yang melakukan banyak hal yang umum untuk aplikasi konsumen. Untuk informasi selengkapnya, lihat Mengembangkan Konsumen KCL 1.x.

Lihat kode sumber dan tinjau informasi berikut.

StockTradesProcessor kelas

Kelas utama konsumen, disediakan untuk Anda, yang melakukan tugas-tugas berikut:

- Membaca aplikasi, aliran, dan nama Region, diteruskan sebagai argumen.
- Membaca mandat dari. `~/.aws/credentials`
- Menciptakan sebuah `RecordProcessorFactory` contoh yang melayani contoh`RecordProcessor`, diimplementasikan oleh sebuah `StockTradeRecordProcessor` instance.
- Membuat pekerja KCL dengan `RecordProcessorFactory` instance dan konfigurasi standar termasuk nama stream, kredenial, dan nama aplikasi.
- Worker membuat thread baru untuk setiap shard (ditugaskan ke instance konsumen ini), yang terus loop untuk membaca catatan dari Kinesis Data Streams. Kemudian memanggil `RecordProcessor` instance untuk memproses setiap batch catatan yang diterima.

StockTradeRecordProcessor kelas

Pelaksanaan `RecordProcessor` contoh, yang pada gilirannya mengimplementasikan tiga metode yang diperlukan:`initialize`,`processRecords`, dan`shutdown`.

Seperti namanya, `initialize` dan `shutdown` digunakan oleh Kinesis Client Library untuk memberi tahu prosesor rekaman kapan harus siap untuk mulai menerima catatan dan kapan harus berhenti menerima catatan, masing-masing, sehingga dapat melakukan pengaturan khusus aplikasi dan tugas penghentian. Kode untuk ini disediakan untuk Anda. Pemrosesan utama terjadi dalam `processRecords` metode, yang pada gilirannya digunakan `processRecord` untuk setiap

catatan. Metode terakhir ini disediakan sebagai kode kerangka sebagian besar kosong bagi Anda untuk menerapkan pada langkah berikutnya, di mana dijelaskan lebih lanjut.

Yang juga perlu diperhatikan adalah penerapan metode dukungan untuk`processRecord`:`reportStats`, dan`resetStats`, yang kosong dalam kode sumber asli.

`processRecords`Metode ini diimplementasikan untuk Anda, dan melakukan langkah-langkah berikut:

- Untuk setiap catatan yang diteruskan, panggilan `processRecord` di atasnya.
- Jika setidaknya 1 menit telah berlalu sejak laporan terakhir, panggilan`reportStats()`, yang mencetak statistik terbaru, dan kemudian `resetStats()` yang menghapus statistik sehingga interval berikutnya hanya mencakup catatan baru.
- Menetapkan waktu pelaporan berikutnya.
- Jika setidaknya 1 menit telah berlalu sejak pos pemeriksaan terakhir, panggilan. `checkpoint()`
- Menetapkan waktu checkpointing berikutnya.

Metode ini menggunakan interval 60 detik untuk tingkat pelaporan dan checkpointing. Untuk informasi selengkapnya tentang checkpointing, lihat. Informasi Tambahan Tentang Konsumen

StockStats kelas

Kelas ini menyediakan retensi data dan pelacakan statistik untuk saham paling populer dari waktu ke waktu. Kode ini disediakan untuk Anda dan berisi metode berikut:

- `addStockTrade(StockTrade)`: Menyuntikkan yang diberikan `StockTrade` ke dalam statistik berjalan.
- `toString()`: Mengembalikan statistik dalam string diformat.

Kelas ini melacak saham paling populer dengan menjaga hitungan berjalan dari jumlah total perdagangan untuk setiap saham dan jumlah maksimum. Ini memperbarui jumlah ini setiap kali perdagangan saham tiba.

Tambahkan kode ke metode `StockTradeRecordProcessor` kelas, seperti yang ditunjukkan pada langkah-langkah berikut.

Untuk mengimplementasikan konsumen

1. Menerapkan `processRecord` metode dengan instantiating `StockTrade` objek berukuran benar dan menambahkan data record untuk itu, log peringatan jika ada masalah.

```
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. Menerapkan `reportStats` metode sederhana. Jangan ragu untuk memodifikasi format output ke preferensi Anda.

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

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

3. Akhirnya, menerapkan `resetStats` metode, yang menciptakan `stockStats` contoh baru.

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

Untuk menjalankan konsumen

1. Jalankan produser yang Anda tulis untuk menyuntikkan catatan perdagangan saham simulasi ke aliran Anda.
2. Verifikasi bahwa kunci akses dan pasangan kunci rahasia yang diambil sebelumnya (saat membuat pengguna IAM) disimpan dalam file. `~/.aws/credentials`
3. Jalankan `StockTradesProcessor` kelas dengan argumen berikut:

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

   Perhatikan bahwa jika Anda membuat aliran Anda di Wilayah selain`us-west-2`, Anda harus menentukan Wilayah tersebut di sini sebagai gantinya.

Setelah satu menit, Anda akan melihat output seperti berikut, disegarkan setiap menit setelahnya:

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

Informasi Tambahan Tentang Konsumen

Jika Anda terbiasa dengan keuntungan dari Kinesis Client Library, yang dibahas di dalam Mengembangkan Konsumen KCL 1.x dan di tempat lain, Anda mungkin bertanya-tanya mengapa Anda harus menggunakannya di sini. Meskipun Anda hanya menggunakan aliran shard tunggal dan satu instance konsumen untuk memprosesnya, masih lebih mudah untuk mengimplementasikan konsumen menggunakan KCL. Bandingkan langkah-langkah implementasi kode di bagian produsen dengan konsumen, dan Anda dapat melihat kemudahan komparatif dalam menerapkan konsumen. Ini sebagian besar disebabkan oleh layanan yang disediakan KCL.

Dalam aplikasi ini, Anda fokus pada penerapan kelas prosesor rekaman yang dapat memproses catatan individu. Anda tidak perlu khawatir tentang bagaimana catatan diambil dari Kinesis Data Streams; KCL mengambil catatan dan memanggil prosesor rekaman setiap kali ada catatan baru yang tersedia. Selain itu, Anda tidak perlu khawatir tentang berapa banyak pecahan dan instance konsumen yang ada. Jika stream ditingkatkan, Anda tidak perlu menulis ulang aplikasi Anda untuk menangani lebih dari satu pecahan atau satu instance konsumen.

Istilah checkpointing berarti merekam titik dalam aliran hingga catatan data yang telah dikonsumsi dan diproses sejauh ini, sehingga jika aplikasi macet, aliran dibaca dari titik itu dan bukan dari awal aliran. Subjek checkpointing dan berbagai pola desain dan praktik terbaik untuk itu berada di luar lingkup Bab ini. Namun, itu adalah sesuatu yang mungkin Anda temui di lingkungan produksi.

Seperti yang Anda pelajari, `put` operasi di API Kinesis Data Streams mengambil kunci partisi sebagai input. Kinesis Data Streams menggunakan kunci partisi sebagai mekanisme untuk membagi catatan di beberapa pecahan (ketika ada lebih dari satu pecahan dalam aliran). Kunci partisi yang sama selalu rute ke pecahan yang sama. Hal ini memungkinkan konsumen yang memproses pecahan tertentu untuk dirancang dengan asumsi bahwa catatan dengan kunci partisi yang sama hanya dikirim ke konsumen itu, dan tidak ada catatan dengan kunci partisi yang sama berakhir di konsumen lain. Oleh karena itu, pekerja konsumen dapat menggabungkan semua catatan dengan kunci partisi yang sama tanpa khawatir bahwa itu mungkin kehilangan data yang dibutuhkan.

Dalam aplikasi ini, pemrosesan catatan konsumen tidak intensif, sehingga Anda dapat menggunakan satu pecahan dan melakukan pemrosesan di utas yang sama dengan utas KCL. Namun, dalam praktiknya, pertimbangkan terlebih dahulu meningkatkan jumlah pecahan. Dalam beberapa kasus, Anda mungkin ingin beralih pemrosesan ke thread yang berbeda, atau menggunakan kumpulan benang jika pemrosesan rekaman Anda diharapkan intensif. Dengan cara ini, KCL dapat mengambil catatan baru lebih cepat sementara thread lain dapat memproses catatan secara paralel. Desain multithreaded tidak sepele dan harus didekati dengan teknik canggih, sehingga meningkatkan jumlah pecahan Anda biasanya merupakan cara yang paling efektif dan termudah untuk meningkatkan skala.

Langkah Selanjutnya

Langkah 6: (Opsional) Memperluas Konsumen

Langkah 6: (Opsional) Memperluas Konsumen

Aplikasi di Tutorial: Memproses Data Stock Real-Time Menggunakan KPL dan KCL 1.x mungkin sudah cukup untuk tujuan Anda. Bagian opsional ini menunjukkan bagaimana Anda dapat memperluas kode konsumen untuk skenario yang sedikit lebih rumit.

Jika Anda ingin tahu tentang pesanan jual terbesar setiap menit, Anda dapat memodifikasi `StockStats` kelas di tiga tempat untuk mengakomodasi prioritas baru ini.

Untuk memperluas konsumen

1. Tambahkan variabel instance baru:

```
// 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. Tambahkan kode berikut ke `addStockTrade`:

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

```
}
```

3. Ubah `toString` metode untuk mencetak informasi tambahan:

```
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);
}
```

Jika Anda menjalankan konsumen sekarang (ingat untuk menjalankan produser juga), Anda akan melihat output yang mirip dengan ini:

```
****** 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.
 ************************************************************
```

Langkah Selanjutnya

Langkah 7: Finishing Up

Langkah 7: Finishing Up

Karena Anda membayar untuk menggunakan aliran data Kinesis, pastikan Anda menghapusnya dan tabel Amazon DynamoDB yang sesuai saat Anda selesai menggunakannya. Biaya nominal terjadi pada aliran aktif bahkan ketika Anda tidak mengirim dan mendapatkan catatan. Hal ini karena aliran aktif menggunakan sumber daya dengan terus “mendengarkan” untuk catatan masuk dan permintaan untuk mendapatkan catatan.

Untuk menghapus aliran dan tabel

1. Matikan produsen dan konsumen yang mungkin masih Anda jalankan.

2. Buka konsol Kinesis di https://console.aws.amazon.com/kinesis.
3. Pilih aliran yang Anda buat untuk aplikasi ini (`StockTradeStream`).
4. Pilih Hapus Stream.
5. Buka konsol DynamoDB di https://console.aws.amazon.com/dynamodb/.
6. Hapus tabel `StockTradesProcessor`.

Ringkasan

Memproses sejumlah besar data dalam waktu hampir nyata tidak memerlukan penulisan kode ajaib atau mengembangkan infrastruktur besar. Ini semudah menulis logika untuk memproses sejumlah kecil data (seperti menulis`processRecord(Record)`) tetapi menggunakan Kinesis Data Streams untuk skala sehingga bekerja untuk sejumlah besar data streaming. Anda tidak perlu khawatir tentang bagaimana pemrosesan Anda akan menskalakan karena Kinesis Data Streams menanganinya untuk Anda. Yang harus Anda lakukan adalah mengirim catatan streaming Anda ke Kinesis Data Streams dan menulis logika untuk memproses setiap catatan baru yang diterima.

Berikut adalah beberapa potensi peningkatan untuk aplikasi ini.

Agregat di semua pecahan

> Saat ini, Anda mendapatkan statistik yang dihasilkan dari agregasi catatan data yang diterima oleh pekerja tunggal dari satu pecahan. (Shard tidak dapat diproses oleh lebih dari satu pekerja dalam satu aplikasi pada saat yang bersamaan.) Tentu saja, ketika Anda skala dan memiliki lebih dari satu pecahan, Anda mungkin ingin agregat di semua pecahan. Anda dapat melakukan ini dengan memiliki arsitektur pipeline di mana output dari setiap pekerja dimasukkan ke aliran lain dengan pecahan tunggal, yang diproses oleh pekerja yang mengumpulkan output dari tahap pertama. Karena data dari tahap pertama terbatas (satu sampel per menit per pecahan), maka akan mudah ditangani oleh satu pecahan.

Pemrosesan skala

> Ketika skala aliran hingga memiliki banyak pecahan (karena banyak produsen mengirim data), cara untuk skala pengolahan adalah dengan menambahkan lebih banyak pekerja. Anda dapat menjalankan pekerja di instans Amazon EC2 dan menggunakan grup Penskalaan Otomatis.

Gunakan konektor ke Amazon S3/Dynamodb/Amazon Redshift/Storm

> Karena aliran terus diproses, hasilnya dapat dikirim ke tujuan lain. AWSmenyediakan konektor untuk mengintegrasikan Kinesis Data Streams dengan AWS layanan lain dan alat pihak ketiga.

Langkah Selanjutnya

- Untuk informasi selengkapnya tentang penggunaan operasi API Kinesis Data Streams, lihat Mengembangkan Produsen Menggunakan API Amazon Kinesis Data Streams denganAWS SDK for JavaMengembangkan Konsumen Kustom dengan Throughput Bersama MenggunakanAWS SDK for Java, dan. Membuat dan Mengelola Streaming
- Untuk informasi selengkapnya tentang Kinesis Client Library, lihatMengembangkan Konsumen KCL 1.x.
- Untuk informasi selengkapnya tentang cara mengoptimalkan aplikasi Anda, lihatTopik Lanjutan.

Tutorial: Menganalisis Data Stok Real-Time Menggunakan Managed Service untuk Apache Flink untuk Aplikasi Flink

Skenario untuk tutorial ini melibatkan menelan perdagangan saham ke dalam aliran data dan menulis Amazon Managed Service sederhana untuk aplikasi Apache Flink yang melakukan perhitungan pada aliran. Anda akan belajar cara mengirim aliran catatan ke Kinesis Data Streams dan mengimplementasikan aplikasi yang mengkonsumsi dan memproses catatan dalam waktu nyaris nyata.

Dengan Managed Service untuk Apache Flink untuk Aplikasi Aplikasi Flink, Anda dapat menggunakan Java atau Scala untuk memproses dan menganalisis data streaming. Layanan ini memungkinkan Anda untuk membuat dan menjalankan kode Java atau Scala terhadap sumber streaming untuk melakukan analitik deret waktu, memberi umpan dasbor waktu nyata, dan membuat metrik waktu nyata.

Anda dapat membangun aplikasi Flink di Managed Service untuk Apache Flink menggunakan pustaka open-source berdasarkan Apache Flink. Apache Flink adalah kerangka kerja dan mesin populer untuk memproses aliran data.

> **Important**
>
> Setelah Anda membuat dua aliran data dan aplikasi, akun Anda dikenakan biaya nominal untuk Kinesis Data Streams dan Layanan Terkelola untuk penggunaan Apache Flink karena tidak memenuhi syarat untuk Tingkat Gratis. AWS Setelah Anda selesai dengan aplikasi ini, hapus AWS sumber daya Anda untuk menghentikan biaya.

Kode tersebut tidak mengakses data pasar saham yang sebenarnya, melainkan mensimulasikan aliran perdagangan saham. Ia melakukannya dengan menggunakan generator perdagangan saham acak. Jika Anda memiliki akses ke aliran perdagangan saham secara real-time, Anda mungkin tertarik untuk mendapatkan statistik yang berguna dan tepat waktu dari aliran itu. Misalnya, Anda mungkin ingin melakukan analisis jendela geser di mana Anda menentukan saham paling populer yang dibeli dalam 5 menit terakhir. Atau Anda mungkin ingin pemberitahuan setiap kali ada pesanan jual yang terlalu besar (yaitu, memiliki terlalu banyak saham). Anda dapat memperluas kode dalam seri ini untuk menyediakan fungsionalitas tersebut.

Contoh yang ditampilkan menggunakan Wilayah Barat AS (Oregon), tetapi mereka bekerja di salah satu AWSWilayah yang mendukung Layanan Terkelola untuk Apache Flink.

Tugas

- Prasyarat untuk Menyelesaikan Latihan
- Langkah 1: Siapkan Akun AWS dan Buat Pengguna Administrator
- Langkah 2: Siapkan AWS Command Line Interface (AWS CLI)
- Langkah 3: Buat dan Jalankan Layanan Terkelola untuk Apache Flink untuk Aplikasi Flink

Prasyarat untuk Menyelesaikan Latihan

Untuk menyelesaikan langkah-langkah di panduan ini, Anda harus memiliki hal-hal berikut:

- Java Development Kit (JDK) versi 8. Atur variabel lingkungan `JAVA_HOME` untuk menunjuk ke lokasi penginstalan JDK Anda.
- Sebaiknya gunakan lingkungan pengembangan (seperti Eclipse Java Neon atau IntelliJ Idea) untuk mengembangkan dan mengompilasi aplikasi Anda.
- Klien Git. Instal klien Git jika Anda belum menginstalnya.
- Plugin Compiler Apache Maven. Maven harus berada di jalur kerja Anda. Untuk menguji instalasi Apache Maven Anda, masukkan hal berikut:

```
$ mvn -version
```

Untuk memulai, buka Langkah 1: Siapkan Akun AWS dan Buat Pengguna Administrator.

Langkah 1: Siapkan Akun AWS dan Buat Pengguna Administrator

Sebelum Anda menggunakan Amazon Managed Service untuk Apache Flink untuk Aplikasi Flink untuk pertama kalinya, selesaikan tugas-tugas berikut:

1. Mendaftar AWS
2. Membuat Pengguna IAM

Mendaftar AWS

Saat Anda mendaftar ke Amazon Web Services (AWS), AWS akun Anda secara otomatis mendaftar untuk semua layananAWS, termasuk Amazon Managed Service untuk Apache Flink. Anda hanya membayar biaya layanan yang Anda gunakan.

Dengan Managed Service for Apache Flink, Anda hanya membayar untuk sumber daya yang Anda gunakan. Jika Anda adalah AWS pelanggan baru, Anda dapat memulai dengan Managed Service untuk Apache Flink secara gratis. Untuk informasi selengkapnya, lihat AWS Tingkat Gratis.

Jika Anda sudah memiliki akun AWS, lanjutkan ke tugas berikutnya. Jika Anda tidak memiliki akun AWS, buat dengan mengikuti langkah-langkah ini.

Untuk membuat akun AWS

1. Buka https://portal.aws.amazon.com/billing/signup.
2. Ikuti petunjuk secara online.

   Anda akan diminta untuk menerima panggilan telepon dan memasukkan kode verifikasi pada keypad telepon sebagai bagian dari prosedur pendaftaran.

   Saat Anda mendaftar Akun AWS, Pengguna root akun AWS akan dibuat. Pengguna root memiliki akses ke semua Layanan AWS dan sumber daya dalam akun. Sebagai praktik terbaik keamanan, tetapkan akses administratif ke pengguna administratif, dan hanya gunakan pengguna root untuk melakukan tugas yang memerlukan akses pengguna root.

Catat ID akun AWS Anda karena Anda akan membutuhkannya untuk tugas berikutnya.

Membuat Pengguna IAM

Layanan diAWS, seperti Amazon Managed Service for Apache Flink, mengharuskan Anda memberikan kredensi saat mengaksesnya. Layanan kemudian dapat menentukan apakah Anda memiliki izin untuk mengakses sumber daya yang dimiliki oleh layanan tersebut. AWS Management Console mengharuskan Anda memasukkan kata sandi.

Anda dapat membuat access key untuk akun AWS Anda untuk mengakses AWS Command Line Interface (AWS CLI) atau API. Namun, sebaiknya jangan akses AWS menggunakan kredensial untuk akun AWS Anda. Sebagai gantinya, sebaiknya gunakan AWS Identity and Access Management (IAM). Buat pengguna IAM, tambahkan pengguna ke grup IAM dengan izin administratif, lalu berikan izin administratif ke pengguna IAM yang Anda buat. Anda kemudian dapat mengakses AWS menggunakan URL khusus dan kredensial pengguna IAM itu.

Jika Anda mendaftar AWS, tetapi belum membuat pengguna IAM untuk Anda sendiri, Anda dapat membuatnya menggunakan konsol IAM.

Latihan untuk memulai dalam panduan ini mengasumsikan bahwa Anda memiliki (`adminuser`) pengguna dengan izin administrator. Ikuti prosedur untuk membuat `adminuser` di akun Anda.

Untuk membuat grup untuk administrator

1. Masuk ke AWS Management Console dan buka konsol IAM di https://console.aws.amazon.com/iam/.
2. Di panel navigasi, pilih Groups (Grup), lalu pilih Create New Group (Buat Grup Baru).
3. Untuk Group Name (Nama Grup), masukkan nama untuk grup Anda, misalnya **`Administrators`**, lalu pilih Next Step (Langkah Selanjutnya).
4. Dalam daftar kebijakan, pilih kotak centang di sebelah AdministratorAccesskebijakan. Anda dapat menggunakan menu Filter dan kotak Search (Pencarian) untuk memfilter daftar kebijakan.
5. Pilih Next Step (Langkah Selanjutnya), lalu pilih Create Group (Buat Grup).

Grup baru Anda terdaftar dalam Group Name (Nama Grup).

Untuk membuat pengguna IAM untuk Anda sendiri, tambahkan ke grup Administrator, dan buat kata sandi

1. Pada panel navigasi, silakan pilih Pengguna, lalu pilih Tambahkan pengguna.

2. Di kotak User name (Nama pengguna), masukkan nama pengguna.
3. Pilih Akses terprogram dan Akses Konsol Manajeme AWS.
4. Pilih Selanjutnya: Izin.
5. Centang kotak centang di sebelah grup Administrators (Administrator). Selanjutnya pilih Next: Review (Selanjutnya: Tinjauan).
6. Pilih Create user (Buat pengguna).

Untuk masuk sebagai pengguna IAM baru

1. Keluar dari AWS Management Console.
2. Gunakan format URL berikut untuk masuk ke konsol:

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

   aws_account_number adalah ID akun AWS Anda tanpa tanda hubung. Misalnya, jika ID akun AWS Anda adalah 1234-5678-9012, ganti *aws_account_number* dengan **123456789012**. Untuk informasi selengkapnya tentang cara menemukan nomor akun Anda, lihat ID Akun AWS Anda dan Aliasnnya di Panduan Pengguna IAM.
3. Masukkan nama pengguna dan kata sandi IAM yang baru saja Anda buat. Saat Anda masuk, bilah navigasi menampilkan *your_user_name* @ *your_aws_account_id*.

> **Note**
>
> Jika Anda tidak ingin URL untuk halaman masuk Anda berisi ID akun AWS Anda, Anda dapat membuat alias akun.

Untuk membuat atau menghapus alias akun

1. Buka konsol IAM di https://console.aws.amazon.com/iam/.
2. Di panel navigasi, pilih Dashboard (Dasbor).
3. Temukan tautan masuk pengguna IAM.
4. Untuk membuat alias, pilih Customize (Sesuaikan). Masukkan nama yang ingin digunakan untuk alias Anda, lalu pilih Yes, Create (Ya, Buat).

5. Untuk menghapus alias, pilih Customize (Sesuaikan), lalu pilih Yes, Delete (Ya, hapus). URL masuk kembali ke menggunakan ID akun AWS Anda.

Untuk masuk setelah membuat alias akun, gunakan URL berikut:

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

Untuk memverifikasi tautan masuk pengguna IAM untuk akun Anda, buka konsol IAM dan periksa di Tautan masuk pengguna IAM di dasbor.

Untuk informasi selengkapnya tentang IAM, lihat hal berikut:

- AWS Identity and Access Management (IAM)
- Memulai
- Panduan Pengguna IAM

Langkah Selanjutnya

Langkah 2: Siapkan AWS Command Line Interface (AWS CLI)

Langkah 2: Siapkan AWS Command Line Interface (AWS CLI)

Pada langkah ini, Anda mengunduh dan mengonfigurasi AWS CLI untuk digunakan dengan Amazon Managed Service untuk Apache Flink untuk Aplikasi Flink.

> **Note**
>
> Latihan memulai dalam panduan ini mengasumsikan Anda menggunakan kredensial administrator (`adminuser`) di akun Anda untuk melakukan operasi.

> **Note**
>
> Jika sudah menginstal AWS CLI, Anda mungkin perlu meningkatkan untuk mendapatkan fungsionalitas terbaru. Untuk informasi selengkapnya, lihat Menginstal Antarmuka Baris AWS Perintah di Panduan AWS Command Line Interface Pengguna. Untuk memeriksa versi AWS CLI, jalankan perintah berikut.

```
aws --version
```

Latihan dalam tutorial ini memerlukan versi AWS CLI berikut atau yang lebih baru:

```
aws-cli/1.16.63
```

Untuk mengatur AWS CLI

1. Unduh dan konfigurasikan AWS CLI. Untuk instruksinya, lihat topik berikut di AWS Command Line InterfacePanduan Pengguna:
   - Menginstal AWS Command Line Interface
   - Mengonfigurasi AWS CLI
2. Tambahkan profil bernama untuk pengguna administrator di dalam file konfig AWS CLI. Anda dapat menggunakan profil ini saat menjalankan perintah AWS CLI. Untuk informasi selengkapnya tentang profil yang diberi nama, lihat Profil yang Diberi Nama dalam Panduan Pengguna 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
   ```

   Untuk daftar AWS Wilayah yang tersedia, lihat AWSWilayah dan Titik Akhir di. Referensi Umum Amazon Web Services
3. Verifikasikan penyiapan dengan memasukkan perintah bantuan berikut pada prompt perintah.

   ```
   aws help
   ```

Setelah Anda mengatur AWS akun danAWS CLI, Anda dapat mencoba latihan berikutnya, di mana Anda mengkonfigurasi aplikasi sampel dan menguji end-to-end pengaturan.

Langkah Selanjutnya

Langkah 3: Buat dan Jalankan Layanan Terkelola untuk Apache Flink untuk Aplikasi Flink

Langkah 3: Buat dan Jalankan Layanan Terkelola untuk Apache Flink untuk Aplikasi Flink

Dalam latihan ini, Anda membuat Layanan Terkelola untuk Apache Flink untuk aplikasi Flink dengan aliran data sebagai sumber dan wastafel.

Bagian ini berisi langkah-langkah berikut.

- Buat Dua Amazon Kinesis Data Streams
- Tulis Catatan Sampel ke Aliran Input
- Unduh dan Periksa Kode Java Streaming Apache Flink
- Kompilasi Kode Aplikasi
- Unggah Kode Java Streaming Apache Flink
- Buat dan Jalankan Managed Service untuk Apache Flink Application

Buat Dua Amazon Kinesis Data Streams

Sebelum Anda membuat Layanan Terkelola untuk Apache Flink untuk aplikasi Flink untuk latihan ini, buat dua aliran data Kinesis (dan). `ExampleInputStream` `ExampleOutputStream` Aplikasi Anda menggunakan aliran ini untuk sumber aplikasi dan aliran tujuan.

Anda dapat membuat aliran ini menggunakan konsol Amazon Kinesis atau perintah AWS CLI berikut. Untuk petunjuk konsol, lihat Membuat dan Memperbarui Aliran Data.

Untuk membuat aliran data AWS CLI

1. Untuk membuat aliran pertama (`ExampleInputStream`), gunakan perintah `create-stream` AWS CLI Amazon Kinesis berikut.

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

2. Untuk membuat aliran kedua yang digunakan aplikasi untuk menulis output, jalankan perintah yang sama, yang mengubah nama aliran menjadi `ExampleOutputStream`.

```
$ aws kinesis create-stream \
```

```
--stream-name ExampleOutputStream \
--shard-count 1 \
--region us-west-2 \
--profile adminuser
```

Tulis Catatan Sampel ke Aliran Input

Di bagian ini, Anda menggunakan script Python untuk menulis catatan sampel ke aliran untuk diproses aplikasi.

> **Note**
>
> Bagian ini memerlukan AWS SDK for Python (Boto).

1. Buat file bernama `stock.py` dengan konten berikut:

```
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. Selanjutnya dalam tutorial ini, Anda menjalankan skrip `stock.py` untuk mengirim data ke aplikasi.

```
$ python stock.py
```

Unduh dan Periksa Kode Java Streaming Apache Flink

Kode aplikasi Java untuk contoh ini tersedia dari GitHub. Untuk mengunduh kode aplikasi, lakukan hal berikut:

1. Klon repositori jarak jauh dengan perintah berikut:

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

2. Buka direktori `GettingStarted` tersebut.

Kode aplikasi terletak di file `CustomSinkStreamingJob.java` dan `CloudWatchLogSink.java`. Perhatikan hal tentang kode aplikasi berikut:

- Aplikasi menggunakan sumber Kinesis untuk membaca dari aliran sumber. Cuplikan berikut membuat sink Kinesis:

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

Kompilasi Kode Aplikasi

Di bagian ini, Anda menggunakan compiler Apache Maven untuk membuat kode Java untuk aplikasi. Untuk informasi tentang menginstal Apache Maven dan Java Development Kit (JDK), lihat Prasyarat untuk Menyelesaikan Latihan.

Aplikasi Java Anda memerlukan komponen-komponen berikut:

- File Model Objek Proyek (pom.xml). File ini berisi informasi tentang konfigurasi dan dependensi aplikasi, termasuk Layanan Terkelola untuk Apache Flink untuk pustaka Aplikasi Flink.
- `main`Metode yang berisi logika aplikasi.

> **Note**
>
> Untuk menggunakan konektor Kinesis untuk aplikasi berikut, Anda perlu mengunduh kode sumber untuk konektor dan membangunnya seperti yang dijelaskan dalam dokumentasi Apache Flink.

Untuk membuat dan mengkompilasi kode aplikasi

1. Buat aplikasi Java/Maven di lingkungan pengembangan Anda. Untuk informasi tentang membuat aplikasi, lihat dokumentasi untuk lingkungan pengembangan Anda:
   - Membuat proyek Java pertama Anda (Eclipse Java Neon)
   - Membuat, Menjalankan, dan Mengemas Aplikasi Java Pertama Anda (IntelliJ Idea)
2. Gunakan kode berikut untuk file bernama`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");
    }
}
```

Perhatikan hal berikut tentang contoh kode sebelumnya:

- File ini berisi `main` metode yang mendefinisikan fungsionalitas aplikasi.
- Aplikasi Anda membuat konektor sumber dan sink untuk mengakses sumber daya eksternal menggunakan objek `StreamExecutionEnvironment`.
- Aplikasi membuat konektor sumber dan sink menggunakan properti statis. Untuk menggunakan properti aplikasi dinamis, gunakan metode `createSourceFromApplicationProperties` dan `createSinkFromApplicationProperties` untuk membuat konektor. Metode ini membaca properti aplikasi untuk mengonfigurasi konektor.

3. Untuk menggunakan kode aplikasi Anda, Anda mengompilasi dan mengemasnya ke dalam file JAR. Anda dapat mengompilasi dan mengemas kode Anda dengan salah satu dari dua cara:

   - Gunakan alat Maven baris perintah. Buat file JAR Anda dengan menjalankan perintah berikut di direktori yang berisi file `pom.xml`:

     ```
     mvn package
     ```

   - Menyiapkan lingkungan pengembangan Anda. Lihat dokumentasi lingkungan pengembangan Anda untuk detail.

   Anda dapat mengunggah paket Anda sebagai file JAR, atau Anda dapat mengompresi paket Anda dan mengunggahnya sebagai file ZIP. Jika Anda membuat aplikasi Anda menggunakan AWS CLI, Anda menentukan tipe konten kode Anda (JAR atau ZIP).
4. Jika ada kesalahan saat mengompilasi, pastikan variabel lingkungan `JAVA_HOME`Anda diatur dengan benar.

Jika aplikasi berhasil mengompilasi, file berikut dibuat:

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

Unggah Kode Java Streaming Apache Flink

Pada bagian ini, Anda membuat bucket Amazon Simple Storage Service (Amazon S3) dan mengunggah kode aplikasi Anda.

Untuk mengunggah kode aplikasi

1. Buka konsol Amazon S3 di https://console.aws.amazon.com/s3/.
2. Pilih Buat bucket.
3. Masukkan **`ka-app-code-<username>`** di bidang Bucket name (Nama bucket). Tambahkan sufiks ke nama bucket, seperti nama pengguna Anda, untuk membuatnya unik secara global. Pilih Next (Selanjutnya).
4. Di langkah Konfigurasi opsi, jangan ubah pengaturan, dan pilih Next (Selanjutnya).
5. Di langkah Atur izin, jangan ubah pengaturan, dan pilih Next (Selanjutnya).
6. Pilih Create bucket (Buat bucket).
7. Di konsol Amazon S3, pilih bucket ka-app-code -, dan pilih Unggah. <username>

8. Di langkah Pilih file, pilih Add files (Tambahkan berkas). Navigasikan ke file `java-getting-started-1.0.jar` yang Anda buat di langkah sebelumnya. Pilih Berikutnya.
9. Di langkah Atur izin, jangan ubah pengaturan. Pilih Berikutnya.
10. Di langkah Atur properti, jangan ubah pengaturan. Pilih Upload (Unggah).

Kode aplikasi Anda sekarang disimpan di bucket Amazon S3 yang dapat diakses aplikasi Anda.

Buat dan Jalankan Managed Service untuk Apache Flink Application

Anda dapat membuat dan menjalankan Layanan Terkelola untuk Apache Flink untuk aplikasi Flink menggunakan konsol atau aplikasi. AWS CLI

> **Note**
>
> Saat Anda membuat aplikasi menggunakan konsol, sumber daya AWS Identity and Access Management (IAM) dan Amazon CloudWatch Logs dibuat untuk Anda. Saat Anda membuat aplikasi menggunakan AWS CLI, Anda membuat sumber daya ini secara terpisah.

Topik

- Buat dan Jalankan Aplikasi (Konsol)
- Buat dan Jalankan Aplikasi (AWS CLI)

Buat dan Jalankan Aplikasi (Konsol)

Ikuti langkah-langkah ini untuk membuat, mengonfigurasi, memperbarui, dan menjalankan aplikasi menggunakan konsol.

Buat Aplikasi

1. Buka konsol Kinesis di https://console.aws.amazon.com/kinesis.
2. Di dasbor Amazon Kinesis, pilih Buat aplikasi analitik.
3. Di halaman Kinesis Analytics - Buat aplikasi, masukkan detail aplikasi sebagai berikut:
   - Untuk Application name (Nama aplikasi), masukkan **`MyApplication`**.
   - Untuk Description (Deskripsi), masukkan **`My java test app`**.
   - Untuk Runtime, pilih Apache Flink 1.6.

4. Untuk Access permissions (Izin akses), pilih Create / update IAM role **kinesis-analytics-MyApplication-us-west-2** (Buat/perbarui IAM role).
5. Pilih Create application (Buat aplikasi).

> **Note**
>
> Saat Anda membuat Layanan Terkelola untuk Apache Flink untuk aplikasi Flink menggunakan konsol, Anda memiliki opsi untuk memiliki peran dan kebijakan IAM yang dibuat untuk aplikasi Anda. Aplikasi Anda menggunakan peran dan kebijakan ini untuk mengakses sumber daya dependen. Sumber daya IAM ini diberi nama menggunakan nama aplikasi dan Wilayah sebagai berikut:
>
> - Kebijakan: `kinesis-analytics-service-MyApplication-us-west-2`
> - Peran: `kinesis-analytics-MyApplication-us-west-2`

Edit Kebijakan IAM

Edit kebijakan IAM untuk menambahkan izin mengakses Kinesis data streams.

1. Buka konsol IAM di https://console.aws.amazon.com/iam/.
2. Pilih Policies (Kebijakan). Pilih kebijakan **kinesis-analytics-service-MyApplication-us-west-2** yang dibuat konsol untuk Anda di bagian sebelumnya.
3. Di halaman Ringkasan, pilih Edit policy (Edit kebijakan). Pilih tab JSON.
4. Tambahkan bagian yang disorot dari contoh kebijakan berikut ke kebijakan. Ganti ID akun sampel (*012345678901*) dengan ID akun Anda.

```
{
    "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"
        }
    ]
}
```

Konfigurasikan Aplikasi

1. Pada MyApplicationhalaman, pilih Konfigurasi.
2. Di halaman Konfigurasikan aplikasi, berikan Code location (Lokasi kode):
   - Untuk Bucket Amazon S3, masukkan **ka-app-code-*<username>***.
   - Untuk Jalur ke objek Amazon S3, masukkan **java-getting-started-1.0.jar**.
3. Di bawah Akses ke sumber daya aplikasi, untuk Access permissions (Izin akses), pilih Create / update IAM role **kinesis-analytics-MyApplication-us-west-2** (Pilih/perbarui IAM role).
4. Di bawah Properties (Properti), untuk Group ID (ID Grup), masukkan **ProducerConfigProperties**.
5. Masukkan properti dan nilai aplikasi berikut:

Kunci	Nilai
flink.inputstream.initpos	**LATEST**
aws:region	**us-west-2**
AggregationEnabled	**false**

6. Di bawah Pemantauan, pastikan Memantau tingkat metrik diatur ke Aplikasi.
7. Untuk CloudWatch logging, pilih kotak centang Aktifkan.
8. Pilih Perbarui.

> **Note**
>
> Saat Anda memilih untuk mengaktifkan CloudWatch logging, Managed Service for Apache Flink membuat grup log dan aliran log untuk Anda. Nama-nama sumber daya ini adalah sebagai berikut:
>
> - Grup log: `/aws/kinesis-analytics/MyApplication`
> - Aliran log: `kinesis-analytics-log-stream`

Jalankan Aplikasi

1. Pada MyApplicationhalaman, pilih Jalankan. Konfirmasikan tindakan.
2. Ketika aplikasi berjalan, refresh halaman. Konsol menunjukkan Grafik aplikasi.

Hentikan Aplikasi

Pada MyApplicationhalaman, pilih Berhenti. Konfirmasikan tindakan.

Perbarui Aplikasi

Dengan menggunakan konsol, Anda dapat memperbarui pengaturan aplikasi seperti properti aplikasi, pengaturan pemantauan, dan lokasi atau nama file dari JAR aplikasi. Anda juga dapat memuat ulang aplikasi JAR dari bucket Amazon S3 jika Anda perlu memperbarui kode aplikasi.

Pada MyApplicationhalaman, pilih Konfigurasi. Perbarui pengaturan aplikasi dan pilih Update (Perbarui).

Buat dan Jalankan Aplikasi (AWS CLI)

Di bagian ini, Anda menggunakan AWS CLI untuk membuat dan menjalankan aplikasi Managed Service for Apache Flink. Managed Service for Apache Flink for Flink Applications menggunakan `kinesisanalyticsv2` AWS CLI perintah untuk membuat dan berinteraksi dengan Managed Service untuk aplikasi Apache Flink.

Membuat Kebijakan Izin

Pertama, Anda membuat kebijakan izin dengan dua pernyataan: satu yang memberikan izin untuk tindakan `read` di aliran sumber, dan lainnya yang memberikan izin untuk tindakan `write` di aliran

sink. Anda selanjutnya melampirkan kebijakan ke IAM role (yang Anda buat di bagian berikutnya). Jadi, ketika Layanan Terkelola untuk Apache Flink mengasumsikan peran tersebut, layanan memiliki izin yang diperlukan untuk membaca dari aliran sumber dan menulis ke aliran wastafel.

Gunakan kode berikut untuk membuat kebijakan izin `KAReadSourceStreamWriteSinkStream`. Ganti *`username`* dengan nama pengguna yang Anda gunakan untuk membuat bucket Amazon S3 untuk menyimpan kode aplikasi. Ganti ID akun di Amazon Resource Names (ARN) (*`012345678901`*) dengan ID akun Anda.

```
{
    "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"
        }
    ]
}
```

Untuk step-by-step petunjuk membuat kebijakan izin, lihat Tutorial: Membuat dan Melampirkan Kebijakan Terkelola Pelanggan Pertama Anda di Panduan Pengguna IAM.

> **Note**
>
> Untuk mengakses AWS layanan lain, Anda dapat menggunakanAWS SDK for Java. Layanan Terkelola untuk Apache Flink secara otomatis menetapkan kredensional yang diperlukan oleh SDK ke peran IAM eksekusi layanan yang terkait dengan aplikasi Anda. Tidak ada langkah tambahan yang diperlukan.

Buat IAM Role

Di bagian ini, Anda membuat peran IAM yang dapat diasumsikan oleh Layanan Terkelola untuk Apache Flink untuk aplikasi Flink untuk membaca aliran sumber dan menulis ke aliran sink.

Layanan Terkelola untuk Apache Flink tidak dapat mengakses aliran Anda tanpa izin. Anda memberikan izin ini melalui IAM role. Setiap IAM role memiliki dua kebijakan yang dilampirkan. Kebijakan kepercayaan memberikan izin Layanan Terkelola untuk Apache Flink untuk mengambil peran, dan kebijakan izin menentukan apa yang dapat dilakukan Layanan Terkelola untuk Apache Flink setelah mengambil peran.

Anda melampirkan kebijakan izin yang Anda buat di bagian sebelumnya ke peran ini.

Untuk membuat IAM role

1. Buka konsol IAM di https://console.aws.amazon.com/iam/.
2. Dalam panel navigasi, pilih Roles (Peran), Create role (Buat Peran).
3. Di bawah Pilih tipe identitas tepercaya, pilih Layanan AWS. Di bawah Pilih layanan yang akan menggunakan peran ini, pilih Kinesis. Di bawah Pilih kasus penggunaan Anda, pilih Analitik Kinesis.

   Pilih Selanjutnya: Izin.
4. Di halaman Lampirkan kebijakan izin, pilih Next: Review (Selanjutnya: Tinjau). Anda melampirkan kebijakan izin setelah Anda membuat peran tersebut.
5. Di halaman Buat peran, masukkan **`KA-stream-rw-role`** untuk Role name (Nama peran). Pilih Create role (Buat peran).

   Sekarang Anda sudah membuat IAM role baru yang disebut `KA-stream-rw-role`. Berikutnya, Anda memperbarui kebijakan kepercayaan dan izin untuk peran.
6. Lampirkan kebijakan izin ke peran tersebut.

> **Note**
>
> Untuk latihan ini, Managed Service for Apache Flink mengasumsikan peran ini untuk membaca data dari aliran data Kinesis (sumber) dan menulis output ke aliran data Kinesis lain. Jadi, Anda melampirkan kebijakan yang Anda buat di langkah sebelumnya, the section called “Membuat Kebijakan Izin”.

a. Di halaman Ringkasan, pilih tab Permissions (Izin).

b. Pilih Attach Policies (Lampirkan Kebijakan).

c. Di kotak pencarian, masukkan **`KAReadSourceStreamWriteSinkStream`** (kebijakan yang Anda buat bagian sebelumnya).

d. Pilih ReadInputStreamWriteOutputStream kebijakan KA, dan pilih Lampirkan kebijakan.

Anda sekarang telah membuat peran eksekusi layanan yang digunakan aplikasi Anda untuk mengakses sumber daya. Buat catatan ARN peran baru.

Untuk step-by-step petunjuk cara membuat peran, lihat Membuat Peran IAM (Konsol) di Panduan Pengguna IAM.

Buat Managed Service untuk Apache Flink Application

1. Simpan kode JSON berikut ke file bernama `create_request.json`. Ganti ARN peran sampel dengan ARN untuk peran yang Anda buat sebelumnya. Ganti sufiks ARN bucket (*`username`*) dengan sufiks yang Anda pilih di bagian sebelumnya. Ganti ID akun sampel (*`012345678901`*) di peran eksekusi layanan dengan ID akun Anda.

```
{
    "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. Jalankan tindakan `CreateApplication` dengan permintaan sebelumnya untuk membuat aplikasi:

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

Aplikasi ini sekarang dibuat. Anda memulai aplikasi di langkah berikutnya.

Mulai Aplikasi

Di bagian ini, Anda menggunakan tindakan `StartApplication` untuk memulai aplikasi.

Untuk memulai aplikasi

1. Simpan kode JSON berikut ke file bernama `start_request.json`.

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

2. Jalankan tindakan StartApplication dengan permintaan sebelumnya untuk memulai aplikasi:

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

Aplikasi sekarang berjalan. Anda dapat memeriksa metrik Layanan Terkelola untuk Apache Flink di CloudWatch konsol Amazon untuk memverifikasi bahwa aplikasi berfungsi.

Hentikan Aplikasi

Di bagian ini, Anda menggunakan tindakan StopApplication untuk menghentikan aplikasi.

Untuk menghentikan aplikasi

1. Simpan kode JSON berikut ke file bernama `stop_request.json`.

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

2. Jalankan tindakan StopApplication dengan permintaan berikut untuk menghentikan aplikasi:

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

Aplikasi sekarang dihentikan.

Tutorial: MenggunakanAWSLambda dengan Amazon Kinesis Data Streams

Dalam tutorial ini, Anda membuat fungsi Lambda untuk menggunakan kejadian dari aliran data Kinesis. Dalam skenario contoh ini, aplikasi kustom menulis catatan ke stream data Kinesis.AWS Lambda kemudian melakukan polling terhadap aliran data ini dan memanggil fungsi Lambda Anda ketika mendeteksi rekaman data baru.AWS Lambda kemudian menjalankan fungsi Lambda dengan mengasumsikan peran eksekusi yang Anda tetapkan saat membuat fungsi Lambda.

Untuk petunjuk langkah demi langkah terperinci, lihatTutorial: MenggunakanAWSLambda dengan Amazon Kinesis.

> **Note**
>
> Tutorial ini mengasumsikan bahwa Anda memiliki pengetahuan tentang operasi Lambda dan konsol Lambda dasar. Jika belum, ikuti petunjuk diMemulai denganAWSLambdauntuk membuat fungsi Lambda pertama Anda.

Solusi Data Streaming AWS untuk Amazon Kinesis

Solusi Data Streaming AWS untuk Amazon Kinesis secara otomatis mengonfigurasi layanan AWS yang diperlukan untuk dengan mudah menangkap, menyimpan, memproses, dan mengirimkan data streaming. Solusi ini menyediakan beberapa opsi untuk menyelesaikan kasus penggunaan data streaming yang menggunakan beberapa AWS layanan termasuk Kinesis Data AWS Lambda Streams, Amazon API Gateway, dan Amazon Managed Service untuk Apache Flink.

Setiap solusi mencakup komponen berikut:

- Paket AWS CloudFormation untuk men-deploy contoh lengkap.
- CloudWatch Dasbor untuk menampilkan metrik aplikasi.
- CloudWatch alarm pada metrik aplikasi yang paling relevan.
- Semua IAM role dan kebijakan IAM yang diperlukan

Solusinya dapat ditemukan di sini: Solusi Data Streaming untuk Amazon Kinesis

Membuat dan Mengelola Streaming

Amazon Kinesis Data Streams menyerap sejumlah besar data secara real time, menyimpan data dengan tahan lama, dan membuat data tersedia untuk dikonsumsi. Unit data yang disimpan oleh Kinesis Data Streams adalah catatan data. Aliran data mewakili sekelompok catatan data. Catatan data dalam aliran data didistribusikan ke dalam pecahan.

Pecahan memiliki urutan catatan data dalam aliran. Ini berfungsi sebagai unit throughput dasar dari aliran data Kinesis. Sebuah pecahan mendukung 1 MB/s dan 1000 record per detik untuk menulis dan 2 MB/s untuk membaca baik dalam mode kapasitas on-demand maupun yang disediakan. Batas pecahan memastikan kinerja yang dapat diprediksi, membuatnya lebih mudah untuk merancang dan mengoperasikan alur kerja streaming data yang sangat andal.

Topik

Memilih Mode Kapasitas Aliran Data

Topik

Apa itu Mode Kapasitas Aliran Data?

Mode kapasitas menentukan bagaimana kapasitas aliran data dikelola dan bagaimana Anda dikenakan biaya untuk penggunaan aliran data Anda. Di Amazon Kinesis Data Streams, Anda dapat memilih antara mode sesuai permintaan dan mode yang disediakan untuk aliran data Anda.

- On-demand - aliran data dengan mode on-demand tidak memerlukan perencanaan kapasitas dan secara otomatis menskalakan untuk menangani gigabyte throughput tulis dan baca per menit. Dengan mode on-demand, Kinesis Data Streams secara otomatis mengelola pecahan untuk menyediakan throughput yang diperlukan.
- Disediakan - untuk aliran data dengan mode yang disediakan, Anda harus menentukan jumlah pecahan untuk aliran data. Kapasitas total aliran data adalah jumlah dari kapasitas pecahannya. Anda dapat menambah atau mengurangi jumlah pecahan dalam aliran data sesuai kebutuhan.

Anda dapat menggunakan Kinesis Data `PutRecords` Streams dan API untuk menulis `PutRecord` data ke dalam aliran data Anda baik dalam mode kapasitas sesuai permintaan maupun yang disediakan. Untuk mengambil data, kedua mode kapasitas mendukung konsumen default yang menggunakan `GetRecords` API dan konsumen Enhanced Fan-Out (EFO) yang menggunakan API. `SubscribeToShard`

Semua kemampuan Kinesis Data Streams, termasuk mode retensi, enkripsi, metrik pemantauan, dan lainnya, didukung untuk mode sesuai permintaan dan yang disediakan. Kinesis Data Streams memberikan daya tahan dan ketersediaan yang tinggi baik dalam mode kapasitas sesuai permintaan maupun yang disediakan.

Mode Sesuai Permintaan

Aliran data dalam mode on-demand tidak memerlukan perencanaan kapasitas dan secara otomatis menskalakan untuk menangani gigabyte throughput tulis dan baca per menit. Mode on-demand menyederhanakan pengambilan dan penyimpanan volume data yang besar pada latensi rendah karena menghilangkan penyediaan dan pengelolaan server, penyimpanan, atau throughput. Anda dapat menelan miliaran catatan per hari tanpa biaya operasional apa pun.

Mode on-demand sangat ideal untuk memenuhi kebutuhan lalu lintas aplikasi yang sangat bervariasi dan tidak dapat diprediksi. Anda tidak lagi harus menyediakan beban kerja ini untuk kapasitas puncak, yang dapat menghasilkan biaya yang lebih tinggi karena pemanfaatan yang rendah. Mode on-demand cocok untuk beban kerja dengan pola lalu lintas yang tidak dapat diprediksi dan sangat bervariasi.

Dengan mode kapasitas sesuai permintaan, Anda membayar per GB data yang ditulis dan dibaca dari aliran data Anda. Anda tidak perlu menentukan berapa banyak throughput baca dan tulis yang Anda harapkan untuk dilakukan aplikasi Anda. Kinesis Data Streams langsung mengakomodasi beban kerja Anda saat mereka naik atau turun. Untuk informasi selengkapnya, lihat Harga Amazon Kinesis Data Streams.

Anda dapat membuat aliran data baru dengan mode on-demand menggunakan konsol Kinesis Data Streams, API, atau perintah CLI.

Aliran data dalam mode on-demand mengakomodasi hingga dua kali lipat throughput penulisan puncak yang diamati dalam 30 hari sebelumnya. Saat throughput penulisan aliran data Anda mencapai puncak baru, Kinesis Data Streams menskalakan kapasitas aliran data secara otomatis. Misalnya, jika aliran data Anda memiliki throughput tulis yang bervariasi antara 10 MB/s dan 40 MB/s, maka Kinesis Data Streams memastikan bahwa Anda dapat dengan mudah melakukan burst untuk menggandakan throughput puncak sebelumnya, atau 80 MB/s. Jika aliran data yang sama mempertahankan throughput puncak baru sebesar 50 MB/s, Kinesis Data Streams memastikan bahwa ada kapasitas yang cukup untuk menelan 100 MB/s throughput tulis. Namun, pelambatan tulis dapat terjadi jika lalu lintas Anda meningkat menjadi lebih dari dua kali lipat puncak sebelumnya dalam durasi 15 menit. Anda perlu mencoba kembali permintaan yang dibatasi ini.

Kapasitas baca agregat dari aliran data dengan mode on-demand meningkat secara proporsional terhadap throughput penulisan. Ini membantu memastikan bahwa aplikasi konsumen selalu memiliki throughput baca yang memadai untuk memproses data yang masuk secara real time. Anda mendapatkan setidaknya dua kali throughput tulis dibandingkan dengan membaca data menggunakan `GetRecords` API. Kami menyarankan Anda menggunakan satu aplikasi konsumen dengan `GetRecord` API, sehingga memiliki cukup ruang untuk mengejar ketinggalan ketika aplikasi perlu pulih dari waktu henti. Disarankan agar Anda menggunakan kemampuan Enhanced Fan-Out dari Kinesis Data Streams untuk skenario yang memerlukan penambahan lebih dari satu aplikasi konsumen. Enhanced Fan-Out mendukung penambahan hingga 20 aplikasi konsumen ke aliran data menggunakan `SubscribeToShard` API, dengan setiap aplikasi konsumen memiliki throughput khusus.

Menangani Pengecualian Throughput Baca dan Tulis

Dengan mode kapasitas sesuai permintaan (sama dengan mode kapasitas yang disediakan), Anda harus menentukan kunci partisi dengan setiap catatan untuk menulis data ke dalam aliran data Anda. Kinesis Data Streams menggunakan kunci partisi Anda untuk mendistribusikan data di seluruh pecahan. Kinesis Data Streams memantau lalu lintas untuk setiap pecahan. Ketika lalu lintas masuk

melebihi 500 Kb/s per shard, itu membagi pecahan dalam waktu 15 menit. Nilai kunci hash shard induk didistribusikan kembali secara merata di seluruh pecahan anak.

Jika lalu lintas masuk Anda melebihi dua kali puncak sebelumnya, Anda dapat mengalami pengecualian baca atau tulis selama sekitar 15 menit, bahkan ketika data Anda didistribusikan secara merata di seluruh pecahan. Kami menyarankan Anda mencoba kembali semua permintaan tersebut sehingga semua catatan disimpan dengan benar di Kinesis Data Streams.

Anda mungkin mengalami pengecualian baca dan tulis jika Anda menggunakan kunci partisi yang mengarah ke distribusi data yang tidak merata, dan catatan yang ditetapkan ke pecahan tertentu melebihi batasnya. Dengan mode on-demand, aliran data secara otomatis beradaptasi untuk menangani pola distribusi data yang tidak merata kecuali satu kunci partisi melebihi throughput 1 MB/s pecahan dan batas 1000 record per detik.

Dalam mode on-demand, Kinesis Data Streams membagi pecahan secara merata saat mendeteksi peningkatan lalu lintas. Namun, itu tidak mendeteksi dan mengisolasi kunci hash yang mendorong bagian yang lebih tinggi dari lalu lintas masuk ke pecahan tertentu. Jika Anda menggunakan kunci partisi yang sangat tidak rata, Anda dapat terus menerima pengecualian tulis. Untuk kasus penggunaan seperti itu, kami menyarankan Anda menggunakan mode kapasitas yang disediakan yang mendukung pemisahan pecahan granular.

Mode yang Disediakan

Dengan mode yang disediakan, setelah Anda membuat aliran data, Anda dapat menskalakan kapasitas pecahan secara dinamis ke atas atau ke bawah menggunakan atau API. AWS Management Console UpdateShardCount Anda dapat melakukan pembaruan saat ada produsen Kinesis Data Streams atau aplikasi konsumen yang menulis atau membaca data dari aliran.

Mode yang disediakan cocok untuk lalu lintas yang dapat diprediksi dengan persyaratan kapasitas yang mudah diprediksi. Anda dapat menggunakan mode yang disediakan jika Anda ingin kontrol halus atas bagaimana data didistribusikan di seluruh pecahan.

Dengan mode yang disediakan, Anda harus menentukan jumlah pecahan untuk aliran data. Untuk menentukan ukuran aliran data dengan mode yang disediakan, Anda memerlukan nilai masukan berikut:

- Ukuran rata-rata catatan data yang ditulis ke aliran dalam kilobyte (KB), dibulatkan ke 1 KB (`average_data_size_in_KB`) terdekat.
- Jumlah catatan data yang ditulis dan dibaca dari aliran per detik (`records_per_second`).

- Jumlah konsumen, yaitu aplikasi Kinesis Data Streams yang mengkonsumsi data secara bersamaan dan independen dari stream (). `number_of_consumers`
- Bandwidth tulis masuk dalam KB (`incoming_write_bandwidth_in_KB`), yang sama dengan `average_data_size_in_KB` dikalikan dengan. `records_per_second`
- Bandwidth baca keluar dalam KB (`outgoing_read_bandwidth_in_KB`), yang sama dengan `incoming_write_bandwidth_in_KB` dikalikan dengan. `number_of_consumers`

Anda dapat menghitung jumlah pecahan (`number_of_shards`) yang dibutuhkan aliran Anda dengan menggunakan nilai input dalam rumus berikut.

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

Anda mungkin masih mengalami pengecualian throughput baca dan tulis dalam mode yang disediakan jika Anda tidak mengonfigurasi aliran data untuk menangani throughput puncak Anda. Dalam hal ini, Anda harus menskalakan aliran data Anda secara manual untuk mengakomodasi lalu lintas data Anda.

Anda mungkin juga mengalami pengecualian baca dan tulis jika Anda menggunakan kunci partisi yang mengarah ke distribusi data yang tidak merata dan catatan yang ditetapkan ke pecahan melebihi batasnya. Untuk mengatasi masalah ini dalam mode yang disediakan, identifikasi pecahan tersebut dan pisahkan secara manual untuk mengakomodasi lalu lintas Anda dengan lebih baik. Untuk informasi selengkapnya, lihat Menyusun Ulang Stream.

Beralih Antar Mode Kapasitas

Anda dapat mengalihkan mode kapasitas aliran data dari on-demand ke provisioned, atau dari provisioned ke on-demand. Untuk setiap aliran data di AWS akun Anda, Anda dapat beralih antara mode kapasitas sesuai permintaan dan yang disediakan dua kali dalam 24 jam.

Beralih di antara mode kapasitas aliran data tidak menyebabkan gangguan pada aplikasi Anda yang menggunakan aliran data ini. Anda dapat terus menulis dan membaca dari aliran data ini. Saat Anda beralih di antara mode kapasitas, baik dari on-demand ke provisioned atau dari provisioned ke on-demand, status stream diatur ke Update. Anda harus menunggu status aliran data untuk masuk ke Aktif sebelum Anda dapat mengubah propertinya lagi.

Saat Anda beralih dari mode kapasitas yang disediakan ke mode sesuai permintaan, aliran data Anda pada awalnya mempertahankan jumlah pecahan apa pun yang dimilikinya sebelum transisi,

dan mulai saat ini, Kinesis Data Streams memantau lalu lintas data Anda dan menskalakan jumlah pecahan aliran data sesuai permintaan ini tergantung pada throughput penulisan Anda.

Saat Anda beralih dari mode on-demand ke provisioned, aliran data Anda pada awalnya juga mempertahankan jumlah pecahan apa pun yang dimilikinya sebelum transisi, tetapi mulai saat ini, Anda bertanggung jawab untuk memantau dan menyesuaikan jumlah pecahan aliran data ini untuk mengakomodasi throughput tulis Anda dengan benar.

Membuat Stream melalui Konsol AWS Manajemen

Anda dapat membuat stream menggunakan konsol Kinesis Data Streams, Kinesis Data Streams API, atau (). AWS Command Line Interface AWS CLI

Untuk membuat aliran data menggunakan konsol

1. Masuk ke AWS Management Console dan buka konsol Kinesis di https://console.aws.amazon.com/kinesis.
2. Di bilah navigasi, perluas pemilih Wilayah dan pilih Wilayah.
3. Pilih Create data stream (Buat aliran data).
4. Pada halaman Create Kinesis stream, masukkan nama untuk aliran data Anda dan kemudian pilih mode kapasitas On-Demand atau Provisioned. Mode On-Demand dipilih secara default. Untuk informasi selengkapnya, lihat Memilih Mode Kapasitas Aliran Data.

   Dengan mode On-Demand, Anda kemudian dapat memilih Create Kinesis stream untuk membuat aliran data Anda. Dengan mode Provisioned, Anda kemudian harus menentukan jumlah pecahan yang Anda butuhkan, dan kemudian memilih Create Kinesis stream.

   Pada halaman Kinesis streams, Status streaming Anda adalah Membuat saat aliran sedang dibuat. Saat aliran siap digunakan, Status berubah menjadi Aktif.
5. Pilih nama streaming Anda. Halaman Detail Stream menampilkan ringkasan konfigurasi aliran Anda, bersama dengan informasi pemantauan.

Untuk membuat stream menggunakan Kinesis Data Streams API

- Untuk informasi tentang membuat stream menggunakan Kinesis Data Streams API, lihat. Membuat Stream melalui API

Untuk membuat aliran menggunakan AWS CLI

- Untuk informasi tentang membuat stream menggunakanAWS CLI, lihat perintah create-stream.

Membuat Stream melalui API

Gunakan langkah-langkah berikut untuk membuat aliran data Kinesis Anda.

Membangun Klien Kinesis Data Streams

Sebelum Anda dapat bekerja dengan aliran data Kinesis, Anda harus membangun objek klien. Kode Java berikut membuat instance pembuat klien dan menggunakannya untuk mengatur Region, kredensi, dan konfigurasi klien. Kemudian membangun objek klien.

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

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

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

Untuk informasi selengkapnya, lihat Wilayah dan Titik Akhir Kinesis Data Streams di. Referensi Umum AWS

Buat Stream

Sekarang setelah Anda membuat klien Kinesis Data Streams, Anda dapat membuat aliran untuk bekerja dengannya, yang dapat Anda capai dengan konsol Kinesis Data Streams, atau secara terprogram. Untuk membuat aliran secara terprogram, buat instance `CreateStreamRequest` objek dan tentukan nama untuk aliran dan (jika Anda ingin menggunakan mode yang disediakan) jumlah pecahan untuk aliran yang akan digunakan.

- Sesuai permintaan:

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

- Disediakan:

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

Nama aliran mengidentifikasi aliran. Nama tersebut dicakup ke AWS akun yang digunakan oleh aplikasi. Itu juga dicakup oleh Wilayah. Artinya, dua aliran dalam dua AWS akun berbeda dapat memiliki nama yang sama, dan dua aliran di AWS akun yang sama tetapi di dua Wilayah yang berbeda dapat memiliki nama yang sama, tetapi tidak dua aliran pada akun yang sama dan di Wilayah yang sama.

Throughput aliran adalah fungsi dari jumlah pecahan; lebih banyak pecahan diperlukan untuk throughput yang disediakan lebih besar. Lebih banyak pecahan juga meningkatkan biaya yang AWS dikenakan untuk aliran. Untuk informasi selengkapnya tentang menghitung jumlah pecahan yang sesuai untuk aplikasi Anda, lihatMemilih Mode Kapasitas Aliran Data.

Setelah `createStreamRequest` objek dikonfigurasi, buat aliran dengan memanggil `createStream` metode pada klien. Setelah meneleponcreateStream, tunggu hingga aliran mencapai `ACTIVE` status sebelum melakukan operasi apa pun di aliran. Untuk memeriksa status aliran, panggil `describeStream` metode. Namun, `describeStream` melempar pengecualian jika aliran tidak ada. Oleh karena itu, lampirkan `describeStream` panggilan dalam satu `try/catch` blok.

```
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" );
}
```

Memperbarui Stream

Anda dapat memperbarui detail aliran menggunakan konsol Kinesis Data Streams, Kinesis Data Streams API, atau file. AWS CLI

> **Note**
>
> Anda dapat mengaktifkan enkripsi sisi server untuk aliran yang ada, atau untuk aliran yang baru saja Anda buat.

Untuk memperbarui aliran data menggunakan konsol

1. Buka konsol Amazon Kinesis di https://console.aws.amazon.com/kinesis/.
2. Di bilah navigasi, perluas pemilih Wilayah dan pilih Wilayah.
3. Pilih nama streaming Anda dalam daftar. Halaman Detail Stream menampilkan ringkasan konfigurasi aliran dan informasi pemantauan Anda.
4. Untuk beralih antara mode kapasitas sesuai permintaan dan yang disediakan untuk aliran data, pilih Edit mode kapasitas di tab Konfigurasi. Untuk informasi selengkapnya, lihat Memilih Mode Kapasitas Aliran Data.

> **Important**
>
> Untuk setiap aliran data di AWS akun Anda, Anda dapat beralih antara mode sesuai permintaan dan yang disediakan dua kali dalam 24 jam.

5. Untuk aliran data dengan mode yang disediakan, untuk mengedit jumlah pecahan, pilih Edit pecahan yang disediakan di tab Konfigurasi, lalu masukkan jumlah pecahan baru.
6. Untuk mengaktifkan enkripsi data data sisi server, pilih Edit di bagian enkripsi sisi server. Pilih kunci KMS untuk digunakan sebagai kunci utama untuk enkripsi, atau gunakan kunci master default, aws/kinesis, yang dikelola oleh Kinesis. Jika Anda mengaktifkan enkripsi untuk aliran dan menggunakan kunci AWS KMS master Anda sendiri, pastikan bahwa produsen dan aplikasi konsumen Anda memiliki akses ke AWS KMS kunci utama yang Anda gunakan. Untuk menetapkan izin ke aplikasi untuk mengakses kunci yang dibuat penggunaAWS KMS, lihat. the section called “Izin untuk Menggunakan Kunci Master KMS Buatan Pengguna”
7. Untuk mengedit periode penyimpanan data, pilih Edit di bagian Periode penyimpanan data, lalu masukkan periode penyimpanan data baru.
8. Jika Anda telah mengaktifkan metrik khusus di akun Anda, pilih Edit di bagian Metrik tingkat pecahan, lalu tentukan metrik untuk aliran Anda. Untuk informasi selengkapnya, lihat the section called “Memantau Layanan dengan CloudWatch”.

Memperbarui Stream Menggunakan API

Untuk memperbarui detail streaming menggunakan API, lihat metode berikut:

- AddTagsToStream
- DecreaseStreamRetentionPeriod
- DisableEnhancedMonitoring
- EnableEnhancedMonitoring
- IncreaseStreamRetentionPeriod
- RemoveTagsFromStream
- StartStreamEncryption
- StopStreamEncryption
- UpdateShardCount

Memperbarui Stream Menggunakan AWS CLI

Untuk informasi tentang memperbarui aliran menggunakanAWS CLI, lihat referensi Kinesis CLI.

Daftar Aliran

Seperti yang dijelaskan di bagian sebelumnya, aliran dicakup ke AWS akun yang terkait dengan AWS kredensil yang digunakan untuk membuat instance klien Kinesis Data Streams dan juga ke Wilayah yang ditentukan untuk klien. Sebuah AWS akun dapat memiliki banyak aliran aktif pada satu waktu. Anda dapat mencantumkan aliran Anda di konsol Kinesis Data Streams, atau secara terprogram. Kode di bagian ini menunjukkan cara membuat daftar semua aliran untuk AWS akun Anda.

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

Contoh kode ini pertama kali membuat instance baru `ListStreamsRequest` dan memanggil `setLimit` metodenya untuk menentukan bahwa maksimum 20 aliran harus dikembalikan untuk setiap panggilan ke`listStreams`. Jika Anda tidak menentukan nilai untuk`setLimit`, Kinesis Data Streams mengembalikan sejumlah aliran kurang dari atau sama dengan nomor di akun. Kode kemudian diteruskan `listStreamsRequest` ke `listStreams` metode klien. Nilai kembali `listStreams` disimpan dalam sebuah `ListStreamsResult` objek. Kode memanggil `getStreamNames` metode pada objek ini dan menyimpan nama aliran yang dikembalikan dalam `streamNames` daftar. Perhatikan bahwa Kinesis Data Streams mungkin mengembalikan aliran lebih sedikit daripada yang ditentukan oleh batas yang ditentukan meskipun ada lebih banyak aliran daripada yang ada di akun dan Wilayah. Untuk memastikan bahwa Anda mengambil semua aliran, gunakan `getHasMoreStreams` metode seperti yang dijelaskan dalam contoh kode berikutnya.

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

Kode ini memanggil `getHasMoreStreams` metode `listStreamsRequest` untuk memeriksa apakah ada aliran tambahan yang tersedia di luar yang dikembalikan dalam panggilan awal `kelistStreams`. Jika demikian, kode memanggil `setExclusiveStartStreamName` metode dengan nama aliran terakhir yang dikembalikan dalam panggilan sebelumnya `kelistStreams`. `setExclusiveStartStreamName`Metode ini menyebabkan `listStreams` panggilan berikutnya dimulai setelah aliran itu. Grup nama aliran yang dikembalikan oleh panggilan itu kemudian ditambahkan ke `streamNames` daftar. Proses ini berlanjut hingga semua nama aliran dikumpulkan dalam daftar.

Aliran yang dikembalikan oleh `listStreams` dapat berada di salah satu status berikut:

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

Anda dapat memeriksa status aliran menggunakan `describeStream` metode, seperti yang ditunjukkan pada bagian sebelumnya,Membuat Stream melalui API.

Daftar Pecahan

Aliran data dapat memiliki satu atau lebih pecahan. Ada dua metode untuk daftar (atau mengambil) pecahan dari aliran data.

Topik

- ListShards API - Direkomendasikan
- DescribeStream API - Usang

ListShards API - Direkomendasikan

Metode yang disarankan untuk membuat daftar atau mengambil pecahan dari aliran data adalah dengan menggunakan API. ListShards Contoh berikut menunjukkan bagaimana Anda bisa mendapatkan daftar pecahan dalam aliran data. Untuk deskripsi lengkap tentang operasi utama yang digunakan dalam contoh ini dan semua parameter yang dapat Anda atur untuk operasi, lihat 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());
        }
    }
}
```

Untuk menjalankan contoh kode sebelumnya Anda dapat menggunakan file POM seperti yang berikut.

```
<?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>
```

Dengan `ListShards` API, Anda dapat menggunakan ShardFilterparameter untuk memfilter respons API. Anda hanya dapat menentukan satu filter pada satu waktu.

Jika Anda menggunakan `ShardFilter` parameter saat menjalankan ListShards API, `Type` adalah properti wajib dan harus ditentukan. Jika Anda menentukan`AT_TRIM_HORIZON`,`FROM_TRIM_HORIZON`, atau `AT_LATEST` jenis, Anda tidak perlu menentukan properti `ShardId` atau `Timestamp` opsional.

Jika Anda menentukan `AFTER_SHARD_ID` jenisnya, Anda juga harus memberikan nilai untuk `ShardId` properti opsional. `ShardId`Properti identik dalam fungsionalitas dengan `ExclusiveStartShardId` parameter ListShards API. Ketika `ShardId` properti ditentukan, responsnya mencakup pecahan yang dimulai dengan pecahan yang ID-nya segera mengikuti `ShardId` yang Anda berikan.

Jika Anda menentukan `AT_TIMESTAMP` atau `FROM_TIMESTAMP_ID` jenis, Anda juga harus memberikan nilai untuk `Timestamp` properti opsional. Jika Anda menentukan `AT_TIMESTAMP` jenisnya, maka semua pecahan yang terbuka pada stempel waktu yang disediakan dikembalikan. Jika Anda menentukan `FROM_TIMESTAMP` jenisnya, maka semua pecahan mulai dari stempel waktu yang disediakan hingga TIP dikembalikan.

Important

`DescribeStreamSummary`dan `ListShard` API menyediakan cara yang lebih skalabel untuk mengambil informasi tentang aliran data Anda. Lebih khusus lagi, kuota untuk DescribeStream API dapat menyebabkan pelambatan. Untuk informasi selengkapnya, lihat Kuota dan Batas. Perhatikan juga bahwa `DescribeStream` kuota dibagikan di semua aplikasi yang berinteraksi dengan semua aliran data di akun AndaAWS. Kuota untuk ListShards API, di sisi lain, khusus untuk satu aliran data. Jadi, Anda tidak hanya mendapatkan TPS yang lebih tinggi dengan ListShards API, tetapi skala tindakannya lebih baik saat Anda membuat lebih banyak aliran data.
Kami menyarankan Anda memigrasikan semua produsen dan konsumen yang memanggil DescribeStream API untuk menjalankan API DescribeStreamSummary dan API. ListShard Untuk mengidentifikasi produsen dan konsumen ini, sebaiknya gunakan Athena untuk mengurai CloudTrail log karena agen pengguna untuk KPL dan KCL ditangkap dalam panggilan 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
```

Kami juga merekomendasikan bahwa integrasi AWS Lambda dan Amazon Firehose dengan Kinesis Data Streams yang memanggil API dikonfigurasi ulang sehingga integrasi sebagai gantinya dipanggil `DescribeStream` dan. `DescribeStreamSummary` `ListShards` Khususnya, untuk AWS Lambda, Anda harus memperbarui pemetaan sumber acara Anda. Untuk Amazon Firehose, izin IAM yang sesuai harus diperbarui sehingga menyertakan izin IAM. `ListShards`

DescribeStream API - Usang

> **Important**
>
> Informasi di bawah ini menjelaskan cara yang saat ini tidak digunakan lagi untuk mengambil pecahan dari aliran data melalui API. DescribeStream Saat ini sangat disarankan agar Anda menggunakan `ListShards` API untuk mengambil pecahan yang terdiri dari aliran data.

Objek respons yang dikembalikan oleh `describeStream` metode memungkinkan Anda untuk mengambil informasi tentang pecahan yang terdiri dari aliran. Untuk mengambil pecahan, panggil `getShards` metode pada objek ini. Metode ini mungkin tidak mengembalikan semua pecahan dari aliran dalam satu panggilan. Dalam kode berikut, kami memeriksa `getHasMoreShards` metode `getStreamDescription` untuk melihat apakah ada pecahan tambahan yang tidak dikembalikan. Jika demikian, yaitu, jika metode ini kembali`true`, kami terus memanggil `getShards` dalam satu lingkaran, menambahkan setiap batch baru pecahan yang dikembalikan ke daftar pecahan kami. Loop keluar saat `getHasMoreShards` kembali`false`; yaitu, semua pecahan telah dikembalikan. Perhatikan bahwa `getShards` tidak mengembalikan pecahan yang ada di `EXPIRED` negara bagian. Untuk informasi lebih lanjut tentang status pecahan, termasuk `EXPIRED` negara bagian, lihatPerutean Data, Persistensi Data, dan Status Shard setelah Reshard.

```
DescribeStreamRequest describeStreamRequest = new DescribeStreamRequest();
describeStreamRequest.setStreamName( myStreamName );
List<Shard> shards = new ArrayList<>();
String exclusiveStartShardId = null;
do {
    describeStreamRequest.setExclusiveStartShardId( exclusiveStartShardId );
    DescribeStreamResult describeStreamResult =
 client.describeStream( describeStreamRequest );
    shards.addAll( describeStreamResult.getStreamDescription().getShards() );
    if (describeStreamResult.getStreamDescription().getHasMoreShards() && shards.size()
 > 0) {
        exclusiveStartShardId = shards.get(shards.size() - 1).getShardId();
    } else {
        exclusiveStartShardId = null;
    }
} while ( exclusiveStartShardId != null );
```

Menghapus Stream

Anda dapat menghapus aliran dengan konsol Kinesis Data Streams, atau secara terprogram. Untuk menghapus aliran secara terprogram, gunakan`DeleteStreamRequest`, seperti yang ditunjukkan pada kode berikut.

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

Matikan aplikasi apa pun yang beroperasi di streaming sebelum Anda menghapusnya. Jika aplikasi mencoba beroperasi pada aliran yang dihapus, ia menerima `ResourceNotFound` pengecualian. Selain itu, jika Anda kemudian membuat aliran baru yang memiliki nama yang sama dengan aliran sebelumnya, dan aplikasi yang beroperasi pada aliran sebelumnya masih berjalan, aplikasi ini mungkin mencoba berinteraksi dengan aliran baru seolah-olah itu adalah aliran sebelumnya—dengan hasil yang tidak dapat diprediksi.

Membagikan Ulang Aliran

> **Important**
>
> Anda dapat mengubah streaming Anda menggunakan API. UpdateShardCount Jika tidak, Anda dapat terus melakukan pemisahan dan penggabungan seperti yang dijelaskan di sini.

Amazon Kinesis Data Streams mendukung resharding, yang memungkinkan Anda menyesuaikan jumlah pecahan dalam aliran Anda untuk beradaptasi dengan perubahan laju aliran data melalui aliran. Resharding dianggap sebagai operasi lanjutan. Jika Anda baru mengenal Kinesis Data Streams, kembali ke subjek ini setelah Anda terbiasa dengan semua aspek lain dari Kinesis Data Streams.

Ada dua jenis operasi resharding: shard split dan shard merge. Dalam pecahan pecahan, Anda membagi satu pecahan menjadi dua pecahan. Dalam penggabungan pecahan, Anda menggabungkan dua pecahan menjadi satu pecahan. Resharding selalu berpasangan dalam arti bahwa Anda tidak dapat membagi menjadi lebih dari dua pecahan dalam satu operasi, dan Anda tidak dapat menggabungkan lebih dari dua pecahan dalam satu operasi. Pecahan atau sepasang pecahan tempat operasi resharding bekerja disebut sebagai pecahan induk. Pecahan atau sepasang pecahan yang dihasilkan dari operasi resharding disebut sebagai pecahan anak.

Pemisahan meningkatkan jumlah pecahan dalam aliran Anda dan karenanya meningkatkan kapasitas data aliran. Karena Anda dikenakan biaya per shard, pemisahan meningkatkan biaya streaming Anda. Demikian pula, penggabungan mengurangi jumlah pecahan dalam aliran Anda dan karenanya mengurangi kapasitas data—dan biaya—aliran.

Resharding biasanya dilakukan oleh aplikasi administratif yang berbeda dari aplikasi produsen (put) dan aplikasi konsumen (get). Aplikasi administratif semacam itu memantau kinerja keseluruhan aliran berdasarkan metrik yang disediakan oleh Amazon CloudWatch atau berdasarkan metrik yang dikumpulkan dari produsen dan konsumen. Aplikasi administratif juga membutuhkan seperangkat izin IAM yang lebih luas daripada konsumen atau produsen karena konsumen dan produsen biasanya tidak memerlukan akses ke API yang digunakan untuk resharding. Untuk informasi selengkapnya tentang izin IAM untuk Kinesis Data Streams, lihat. Mengontrol Akses ke Sumber Daya Amazon Kinesis Data Streams Menggunakan IAM

Untuk informasi selengkapnya tentang resharding, lihat Bagaimana cara mengubah jumlah pecahan terbuka di Kinesis Data Streams?

Topik

- Strategi untuk Resharding
- Memisahkan Pecahan
- Menggabungkan Dua Pecahan
- Setelah Resharding

Strategi untuk Resharding

Tujuan resharding di Amazon Kinesis Data Streams adalah untuk memungkinkan streaming Anda beradaptasi dengan perubahan laju aliran data. Anda membagi pecahan untuk meningkatkan kapasitas (dan biaya) aliran Anda. Anda menggabungkan pecahan untuk mengurangi biaya (dan kapasitas) aliran Anda.

Salah satu pendekatan untuk resharding adalah dengan membagi setiap pecahan di aliran—yang akan menggandakan kapasitas aliran. Namun, ini mungkin memberikan lebih banyak kapasitas tambahan daripada yang sebenarnya Anda butuhkan dan karenanya menghasilkan biaya yang tidak perlu.

Anda juga dapat menggunakan metrik untuk menentukan pecahan panas atau dingin Anda, yaitu pecahan yang menerima lebih banyak data, atau data yang jauh lebih sedikit, dari yang diharapkan.

Anda kemudian dapat secara selektif membagi pecahan panas untuk meningkatkan kapasitas kunci hash yang menargetkan pecahan tersebut. Demikian pula, Anda dapat menggabungkan pecahan dingin untuk memanfaatkan kapasitas yang tidak terpakai dengan lebih baik.

Anda dapat memperoleh beberapa data performa untuk streaming Anda dari CloudWatch metrik Amazon yang diterbitkan oleh Kinesis Data Streams. Namun, Anda juga dapat mengumpulkan beberapa metrik Anda sendiri untuk aliran Anda. Salah satu pendekatannya adalah dengan mencatat nilai kunci hash yang dihasilkan oleh kunci partisi untuk catatan data Anda. Ingat bahwa Anda menentukan kunci partisi pada saat Anda menambahkan catatan ke aliran.

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

Kinesis Data Streams menggunakan MD5 untuk menghitung kunci hash dari kunci partisi. Karena Anda menentukan kunci partisi untuk catatan, Anda dapat menggunakan MD5 untuk menghitung nilai kunci hash untuk catatan itu dan mencatatnya.

Anda juga bisa mencatat ID pecahan tempat catatan data Anda ditetapkan. ID pecahan tersedia dengan menggunakan `getShardId` metode `putRecordResults` objek dikembalikan oleh `putRecords` metode, dan `putRecordResult` objek dikembalikan oleh `putRecord` metode.

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

Dengan ID shard dan nilai kunci hash, Anda dapat menentukan pecahan dan kunci hash mana yang menerima lalu lintas paling banyak atau paling sedikit. Anda kemudian dapat menggunakan resharding untuk memberikan kapasitas yang lebih atau kurang, yang sesuai untuk kunci ini.

Memisahkan Pecahan

Untuk membagi pecahan di Amazon Kinesis Data Streams, Anda perlu menentukan bagaimana nilai kunci hash dari shard induk harus didistribusikan kembali ke pecahan turunan. Ketika Anda menambahkan catatan data ke aliran, itu ditetapkan ke pecahan berdasarkan nilai kunci hash. Nilai kunci hash adalah hash MD5 dari kunci partisi yang Anda tentukan untuk catatan data pada saat Anda menambahkan catatan data ke aliran. Catatan data yang memiliki kunci partisi yang sama juga memiliki nilai kunci hash yang sama.

Nilai kunci hash yang mungkin untuk pecahan tertentu merupakan satu set bilangan bulat non-negatif berurutan yang bersebelahan. Rentang kemungkinan nilai kunci hash ini diberikan oleh yang berikut:

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

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

Saat Anda membagi pecahan, Anda menentukan nilai dalam rentang ini. Nilai kunci hash dan semua nilai kunci hash yang lebih tinggi didistribusikan ke salah satu pecahan anak. Semua nilai kunci hash yang lebih rendah didistribusikan ke pecahan anak lainnya.

Kode berikut menunjukkan operasi pecahan pecahan yang mendistribusikan kembali kunci hash secara merata di antara masing-masing pecahan anak, pada dasarnya membelah pecahan induk menjadi dua. Ini hanyalah salah satu cara yang mungkin untuk membagi pecahan induk. Anda dapat, misalnya, membagi pecahan sehingga sepertiga bagian bawah kunci dari orang tua pergi ke satu pecahan anak dan dua pertiga bagian atas kunci pergi ke pecahan anak lainnya. Namun, untuk banyak aplikasi, membelah pecahan menjadi dua adalah pendekatan yang efektif.

Kode mengasumsikan bahwa `myStreamName` memegang nama aliran Anda dan variabel objek `shard` menahan pecahan untuk dipecah. Mulailah dengan membuat instance `splitShardRequest` objek baru dan mengatur nama aliran dan ID shard.

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

Tentukan nilai kunci hash yang berada di tengah-tengah antara nilai terendah dan tertinggi dalam pecahan. Ini adalah nilai kunci hash awal untuk pecahan anak yang akan berisi bagian atas kunci hash dari pecahan induk. Tentukan nilai ini dalam `setNewStartingHashKey` metode. Anda hanya perlu tentukan nilai ini. Kinesis Data Streams secara otomatis mendistribusikan kunci hash di bawah nilai ini ke pecahan anak lain yang dibuat oleh split. Langkah terakhir adalah memanggil `splitShard` metode pada klien Kinesis Data Streams.

```
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);
```

Langkah pertama setelah prosedur ini ditunjukkan diMenunggu Streaming Menjadi Aktif Lagi.

Menggabungkan Dua Pecahan

Operasi penggabungan shard membutuhkan dua pecahan yang ditentukan dan menggabungkannya menjadi satu pecahan. Setelah penggabungan, pecahan anak tunggal menerima data untuk semua nilai kunci hash yang dicakup oleh dua pecahan induk.

Ketinggian Pecahan

Untuk menggabungkan dua pecahan, pecahan harus berdekatan. Dua pecahan dianggap berdekatan jika penyatuan rentang kunci hash untuk dua pecahan membentuk satu set yang berdekatan tanpa celah. Misalnya, Anda memiliki dua pecahan, satu dengan rentang kunci hash 276... 381 dan yang lainnya dengan rentang kunci hash 382... 454. Anda dapat menggabungkan dua pecahan ini menjadi pecahan tunggal yang akan memiliki rentang kunci hash 276... 454.

Untuk mengambil contoh lain, misalkan Anda memiliki dua pecahan, satu dengan rentang kunci hash 276.. 381 dan yang lainnya dengan rentang kunci hash 455... 560. Anda tidak dapat menggabungkan dua pecahan ini karena akan ada satu atau lebih pecahan di antara keduanya yang mencakup kisaran 382.. 454.

Himpunan semua `OPEN` pecahan dalam aliran — sebagai grup — selalu mencakup seluruh rentang nilai kunci hash MD5. Untuk informasi lebih lanjut tentang status pecahan — seperti —lihat. `CLOSED` Perutean Data, Persistensi Data, dan Status Shard setelah Reshard

Untuk mengidentifikasi pecahan yang merupakan kandidat untuk digabungkan, Anda harus menyaring semua pecahan yang berada dalam keadaan. `CLOSED` Pecahan yang `OPEN` —yaitu, bukan `CLOSED` —memiliki nomor urut akhir. `null` Anda dapat menguji nomor urut akhir untuk pecahan menggunakan:

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

Setelah memfilter pecahan tertutup, urutkan pecahan yang tersisa dengan nilai kunci hash tertinggi yang didukung oleh setiap pecahan. Anda dapat mengambil nilai ini menggunakan:

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

Jika dua pecahan berdekatan dalam daftar yang difilter dan diurutkan ini, mereka dapat digabungkan.

Kode untuk Operasi Gabungan

Kode berikut menggabungkan dua pecahan. Kode mengasumsikan bahwa `myStreamName` memegang nama aliran Anda dan variabel objek `shard1` dan `shard2` menahan dua pecahan yang berdekatan untuk digabungkan.

Untuk operasi penggabungan, mulailah dengan membuat instance objek baru. `mergeShardsRequest` Tentukan nama aliran dengan `setStreamName` metode. Kemudian tentukan dua pecahan untuk digabungkan menggunakan metode `setShardToMerge` and`setAdjacentShardToMerge`. Terakhir, panggil `mergeShards` metode pada klien Kinesis Data Streams untuk melakukan operasi.

```
MergeShardsRequest mergeShardsRequest = new MergeShardsRequest();
mergeShardsRequest.setStreamName(myStreamName);
mergeShardsRequest.setShardToMerge(shard1.getShardId());
mergeShardsRequest.setAdjacentShardToMerge(shard2.getShardId());
client.mergeShards(mergeShardsRequest);
```

Langkah pertama setelah prosedur ini ditunjukkan di[Menunggu Streaming Menjadi Aktif Lagi](#).

Setelah Resharding

Setelah segala jenis prosedur resharding di Amazon Kinesis Data Streams, dan sebelum pemrosesan rekaman normal dilanjutkan, prosedur dan pertimbangan lain diperlukan. Bagian berikut menjelaskan ini.

Topik

- [Menunggu Streaming Menjadi Aktif Lagi](#)
- [Perutean Data, Persistensi Data, dan Status Shard setelah Reshard](#)

Menunggu Streaming Menjadi Aktif Lagi

Setelah Anda memanggil operasi resharding, salah satu `splitShard` atau`mergeShards`, Anda harus menunggu streaming menjadi aktif kembali. Kode yang akan digunakan sama seperti ketika Anda menunggu aliran menjadi aktif setelah [membuat aliran](#). Kode itu adalah sebagai berikut:

```
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" );
}
```

Perutean Data, Persistensi Data, dan Status Shard setelah Reshard

Kinesis Data Streams adalah layanan streaming data real-time, yang berarti bahwa aplikasi Anda harus mengasumsikan bahwa data mengalir terus menerus melalui pecahan di aliran Anda. Saat Anda melakukan reshard, catatan data yang mengalir ke pecahan induk dirutekan ulang untuk mengalir ke pecahan anak berdasarkan nilai kunci hash yang dipetakan oleh kunci partisi rekam data. Namun, catatan data apa pun yang ada di pecahan induk sebelum reshard tetap berada di pecahan tersebut. Dengan kata lain, pecahan induk tidak hilang ketika reshard terjadi. Mereka bertahan bersama dengan data yang mereka isi sebelum reshard. Rekaman data dalam pecahan induk dapat diakses menggunakan `getShardIterator`dan `getRecords` operasi di Kinesis Data Streams API, atau melalui Perpustakaan Klien Kinesis.

> **Note**
>
> Catatan data dapat diakses sejak ditambahkan ke aliran ke periode retensi saat ini. Ini berlaku terlepas dari perubahan apa pun pada pecahan dalam aliran selama periode waktu tersebut. Untuk informasi selengkapnya tentang periode retensi aliran, lihatMengubah Periode Retensi Data.

Dalam proses resharding, pecahan induk bertransisi dari suatu `OPEN` keadaan ke keadaan ke negara `CLOSED` bagian. `EXPIRED`

- BUKA: Sebelum operasi reshard, pecahan induk berada dalam `OPEN` status, yang berarti bahwa catatan data dapat ditambahkan ke pecahan dan diambil dari pecahan.
- DITUTUP: Setelah operasi reshard, pecahan induk bertransisi ke status. `CLOSED` Ini berarti bahwa catatan data tidak lagi ditambahkan ke pecahan. Catatan data yang akan ditambahkan ke pecahan ini sekarang ditambahkan ke pecahan anak sebagai gantinya. Namun, catatan data masih dapat diambil dari pecahan untuk waktu yang terbatas.
- KEDALUWARSA: Setelah periode retensi streaming berakhir, semua catatan data di pecahan induk telah kedaluwarsa dan tidak lagi dapat diakses. Pada titik ini, pecahan itu sendiri beralih ke suatu `EXPIRED` keadaan. Panggilan `getStreamDescription().getShards` untuk menghitung pecahan dalam aliran tidak termasuk pecahan dalam `EXPIRED` pecahan daftar yang dikembalikan. Untuk informasi selengkapnya tentang periode retensi aliran, lihatMengubah Periode Retensi Data.

Setelah reshard terjadi dan aliran kembali dalam `ACTIVE` keadaan, Anda dapat segera mulai membaca data dari pecahan anak. Namun, pecahan induk yang tersisa setelah reshard masih dapat berisi data yang belum Anda baca yang ditambahkan ke aliran sebelum reshard. Jika Anda membaca data dari pecahan anak sebelum membaca semua data dari pecahan induk, Anda dapat membaca data untuk kunci hash tertentu di luar urutan yang diberikan oleh nomor urut catatan data. Oleh karena itu, dengan asumsi bahwa urutan data itu penting, Anda harus, setelah reshard, selalu terus membaca data dari pecahan induk hingga habis. Hanya dengan begitu Anda harus mulai membaca data dari pecahan anak. Ketika `getRecordsResult.getNextShardIterator` kembali`null`, ini menunjukkan bahwa Anda telah membaca semua data di pecahan induk. Jika Anda membaca data menggunakan Perpustakaan Klien Kinesis, pustaka memastikan bahwa Anda menerima data secara berurutan meskipun reshard terjadi.

Mengubah Periode Retensi Data

Amazon Kinesis Data Streams mendukung perubahan pada periode retensi rekaman data aliran data Anda. Aliran data Kinesis adalah urutan urutan catatan data yang dimaksudkan untuk ditulis dan dibaca secara real time. Oleh karena itu, catatan data disimpan dalam pecahan di aliran Anda untuk sementara. Periode waktu dari saat catatan ditambahkan ke saat tidak lagi dapat diakses disebut periode retensi. Aliran data Kinesis menyimpan catatan dari 24 jam secara default, hingga 8760 jam (365 hari).

Anda dapat memperbarui periode retensi melalui konsol Kinesis Data Streams atau dengan IncreaseStreamRetentionPeriodmenggunakan dan operasi. DecreaseStreamRetentionPeriod Dengan konsol Kinesis Data Streams, Anda dapat mengedit periode retensi lebih dari satu aliran data secara massal secara bersamaan. Anda dapat meningkatkan periode retensi hingga maksimum 8760 jam (365 hari) menggunakan IncreaseStreamRetentionPeriodoperasi atau konsol Kinesis Data Streams. Anda dapat mengurangi periode retensi hingga minimal 24 jam menggunakan DecreaseStreamRetentionPeriodoperasi atau konsol Kinesis Data Streams. Sintaks permintaan untuk kedua operasi mencakup nama aliran dan periode retensi dalam jam. Terakhir, Anda dapat memeriksa periode retensi aliran saat ini dengan memanggil DescribeStreamoperasi.

Berikut ini adalah contoh perubahan periode retensi menggunakanAWS CLI:

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

Kinesis Data Streams berhenti membuat catatan tidak dapat diakses pada periode retensi lama dalam beberapa menit setelah meningkatkan periode retensi. Misalnya, mengubah periode retensi dari 24 jam menjadi 48 jam berarti rekaman yang ditambahkan ke aliran 23 jam 55 menit sebelumnya masih tersedia setelah 24 jam.

Kinesis Data Streams segera membuat catatan yang lebih tua dari periode retensi baru tidak dapat diakses setelah penurunan periode retensi. Karena itu, berhati-hatilah saat memanggil DecreaseStreamRetentionPeriodoperasi.

Tetapkan periode retensi data Anda untuk memastikan bahwa konsumen Anda dapat membaca data sebelum kedaluwarsa, jika terjadi masalah. Anda harus mempertimbangkan dengan cermat semua kemungkinan, seperti masalah dengan logika pemrosesan catatan Anda atau ketergantungan hilir yang turun untuk jangka waktu yang lama. Pikirkan periode retensi sebagai jaring pengaman untuk memungkinkan lebih banyak waktu bagi konsumen data Anda untuk pulih. Operasi API periode

retensi memungkinkan Anda mengaturnya secara proaktif atau merespons peristiwa operasional secara reaktif.

Biaya tambahan berlaku untuk streaming dengan periode retensi yang ditetapkan di atas 24 jam. Untuk informasi selengkapnya, lihat Harga Amazon Kinesis Data Streams.

Menandai Aliran di Amazon Kinesis Data Streams

Anda dapat menetapkan metadata Anda sendiri ke aliran yang Anda buat di Amazon Kinesis Data Streams dalam bentuktag. Tanda adalah pasangan nilai-kunci yang Anda tetapkan untuk sebuah aliran. Menggunakan tanda adalah cara sederhana tetapi andal untuk mengelola sumber daya AWS dan mengatur data, termasuk data penagihan.

Isi

Dasar-Dasar Tanda

Anda menggunakan konsol Kinesis Data StreamsAWS CLI, atau API Kinesis Data Streams untuk menyelesaikan tugas-tugas berikut:

- Menambahkan tag ke aliran
- Membuat daftar tanda
- Menghapus tag dari aliran

Anda dapat menggunakan tanda untuk mengategorikan aliran Anda. Misalnya, Anda dapat mengategorikan aliran berdasarkan tujuan, pemilik, pemilik, atau lingkungan. Karena Anda menentukan kunci dan nilai untuk setiap tanda, Anda dapat membuat serangkaian kategori khusus untuk memenuhi kebutuhan spesifik Anda. Misalnya, Anda dapat menentukan satu set tanda yang membantu Anda melacak aliran berdasarkan pemilik dan aplikasi terkait. Berikut adalah beberapa contoh tanda:

- Proyek: Nama Proyek
- Pemilik: Nama
- Tujuan: Pengujian beban
- Aplikasi: Nama aplikasi
- Lingkungan: Produksi

Melacak Biaya Menggunakan Penandaan

Anda dapat menggunakan tanda untuk mengategorikan dan melacak biaya AWS. Saat Anda menerapkan tag keAWSsumber daya, termasuk aliran,AWSLaporan alokasi biaya mencakup penggunaan dan biaya yang dikumpulkan berdasarkan tag. Anda dapat menerapkan tag yang mewakili kategori bisnis (seperti pusat biaya, nama aplikasi, atau pemilik) untuk mengatur biaya Anda di berbagai layanan. Untuk informasi selengkapnya, lihat Menggunakan Tanda Alokasi Biaya untuk Laporan Penagihan Khusus dalam Panduan Pengguna AWS Billing.

Pembatasan Tag

Batasan berikut berlaku untuk tanda.

Batasan dasar

- Jumlah maksimum tanda per sumber daya (aliran) adalah 50.
- Kunci dan nilai tanda peka huruf besar dan kecil.
- Anda tidak dapat mengubah atau mengedit tanda untuk aliran yang dihapus.

Batasan kunci tanda

- Setiap kunci tanda harus unik. Jika Anda menambahkan tanda dengan kunci yang sudah digunakan, tanda baru akan menimpa pasangan nilai-kunci yang sudah ada.
- Anda tidak dapat memulai kunci tanda dengan `aws:` karena prefiks ini disimpan untuk digunakan oleh AWS. AWS membuat tanda yang dimulai dengan prefiks ini atas nama Anda, tetapi Anda tidak dapat mengedit atau menghapusnya.
- Kunci tanda harus memiliki panjang antara 1 dan 128 karakter Unicode.
- Kunci tag harus terdiri dari karakter berikut: Unicode huruf, digit, ruang putih, dan karakter khusus berikut:`_ . / = + - @`.

Batasan nilai tanda

- Panjang nilai tanda harus antara 0 dan 255 karakter Unicode.
- Nilai tanda dapat kosong. Jika tidak, mereka harus terdiri dari karakter berikut: Huruf Unicode, angka, spasi, dan salah satu karakter khusus berikut:`_ . / = + - @`.

Penandaan Aliran Menggunakan Konsol Kinesis Data Streams

Anda dapat menambahkan, mendaftar, dan menghapus tanda menggunakan konsol Kinesis Data Streams.

Untuk melihat tanda untuk aliran

1. Buka konsol Kinesis Data Streams Di bilah navigasi, memperluas pemilih wilayah dan pilih wilayah.
2. PadaDaftar Aliranhalaman, pilih stream.
3. PadaRincian Streamhalaman, klikTagtab.

Untuk menambahkan tag ke aliran

1. Buka konsol Kinesis Data Streams Di bilah navigasi, memperluas pemilih wilayah dan pilih wilayah.
2. PadaDaftar Aliranhalaman, pilih stream.
3. PadaRincian Streamhalaman, klikTagtab.
4. Tentukan kunci tag diKuncibidang, opsional menentukan nilai tag diNilaibidang, dan kemudian klikTambahkan tag.

   JikaTambahkan tagTombol tidak aktif, kunci tanda atau nilai tanda yang Anda tentukan tidak memenuhi pembatasan tag. Untuk informasi selengkapnya, lihat Pembatasan Tag.
5. Untuk melihat tag baru Anda dalam daftar diTagtab, klik ikon refresh.

Untuk menghapus tag dari aliran

1. Buka konsol Kinesis Data Streams Di bilah navigasi, memperluas pemilih wilayah dan pilih wilayah.
2. Pada halaman Daftar Stream, pilih stream.

3. Pada halaman Detail Stream, klikTagtab, dan kemudian klikMenghapusikon untuk tag.
4. DiMenghapus Tagkotak dialog, klikYa, Hapus.

Penandaan Aliran MenggunakanAWS CLI

Anda dapat menambahkan, mendaftar, dan menghapus tanda menggunakan AWS CLI. Untuk contoh, lihat dokumentasi berikut.

add-tags-to-Aliran

Menambahkan atau memperbarui tanda untuk aliran tertentu.

list-tags-for-Aliran

Mendaftar tanda untuk aliran tertentu.

remove-tags-from-Aliran

Menghapus tanda dari aliran tertentu.

Penandaan Aliran Menggunakan API Kinesis Data Streams

Anda dapat menambahkan, mendaftar, dan menghapus tanda menggunakan API Kinesis Data Streams. Untuk contoh, lihat dokumentasi berikut:

AddTagsToStream

Menambahkan atau memperbarui tanda untuk aliran tertentu.

ListTagsForStream

Mendaftar tanda untuk aliran tertentu.

RemoveTagsFromStream

Menghapus tanda dari aliran tertentu.

Menentukan Amazon Kinesis Data Streams

Produser adalah aplikasi yang menulis data ke Amazon Kinesis Data Streams. Anda dapat membangun produsen untuk Kinesis Data Streams menggunakan Kinesis Producary.AWS SDK for Java

Jika Anda baru mengenal Kinesis Data Streams, mulailah memahami konsep dan terminologi yang disajikan dalamApa itu Amazon Kinesis Data Streams? danMemulai dengan Amazon Kinesis Data Streams.

> **Important**
>
> Kinesis Data Streams mendukung perubahan pada periode retensi catatan data stream Anda. Untuk informasi selengkapnya, lihat Mengubah Periode Retensi Data.

Untuk memasukkan data ke dalam aliran, Anda harus menentukan nama aliran, kunci partisi, dan gumpalan data yang akan ditambahkan ke aliran. Kunci partisi digunakan untuk menentukan pecahan dalam aliran catatan data ditambahkan ke.

Semua data dalam pecahan dikirim ke pekerja yang sama yang memproses pecahan. Kunci partisi mana yang Anda gunakan tergantung pada logika aplikasi Anda. Jumlah kunci partisi biasanya harus jauh lebih besar daripada jumlah pecahan. Ini karena kunci partisi digunakan untuk menentukan bagaimana memetakan catatan data ke pecahan tertentu. Jika Anda memiliki cukup kunci partisi, data dapat didistribusikan secara merata di seluruh pecahan dalam aliran.

Konten

Mengembangkan Produsen Menggunakan Perpustakaan Produsen Amazon Kinesis

Produsen Amazon Kinesis Data Streams adalah aplikasi yang menempatkan catatan data pengguna ke dalam aliran data Kinesis (juga disebut konsumsi data). Perpustakaan Produser Kinesis (KPL) menyederhanakan pengembangan aplikasi produser, memungkinkan pengembang untuk mencapai throughput tulis yang tinggi ke aliran data Kinesis.

Anda dapat memantau KPL dengan Amazon CloudWatch. Untuk informasi selengkapnya, lihat Memantau Perpustakaan Produser Kinesis dengan Amazon CloudWatch.

Daftar Isi

> **Note**
>
> Disarankan agar Anda meningkatkan ke versi KPL terbaru. KPL diperbarui secara berkala dengan rilis yang lebih baru yang mencakup ketergantungan terbaru dan patch keamanan, perbaikan bug, dan fitur baru yang kompatibel ke belakang. Untuk informasi lebih lanjut, lihat https://github.com/awslabs/ amazon-kinesis-producer /releases/.

Peran KPL

KPL adalah pustaka yang easy-to-use sangat dapat dikonfigurasi yang membantu Anda menulis ke aliran data Kinesis. Ini bertindak sebagai perantara antara kode aplikasi produser Anda dan tindakan API Kinesis Data Streams. KPL melakukan tugas-tugas utama berikut:

- Menulis ke satu atau lebih aliran data Kinesis dengan mekanisme coba ulang otomatis dan dapat dikonfigurasi
- Mengumpulkan catatan dan menggunakan `PutRecords` untuk menulis beberapa catatan ke beberapa pecahan per permintaan
- Mengagregat catatan pengguna untuk meningkatkan ukuran payload dan meningkatkan throughput
- Terintegrasi secara mulus dengan Kinesis Client Library (KCL) untuk melakukan de-agregasi batch record pada konsumen
- Mengirimkan CloudWatch metrik Amazon atas nama Anda untuk memberikan visibilitas ke kinerja produsen

Perhatikan bahwa KPL berbeda dari Kinesis Data Streams API yang tersedia di SDK.AWS Kinesis Data Streams API membantu Anda mengelola banyak aspek Kinesis Data Streams (termasuk membuat stream, resharding, dan menempatkan dan mendapatkan record), sementara KPL menyediakan lapisan abstraksi khusus untuk menyerap data. Untuk informasi tentang Kinesis Data Streams API, lihat Referensi API Amazon Kinesis.

Keuntungan Menggunakan KPL

Daftar berikut merupakan beberapa keuntungan utama menggunakan KPL untuk mengembangkan produsen Kinesis Data Streams.

KPL dapat digunakan dalam kasus penggunaan sinkron atau asinkron. Kami menyarankan untuk menggunakan kinerja antarmuka asinkron yang lebih tinggi kecuali ada alasan khusus untuk menggunakan perilaku sinkron. Untuk informasi selengkapnya tentang dua kasus penggunaan dan kode contoh ini, lihatMenulis ke Aliran Data Kinesis Anda Menggunakan KPL.

Manfaat Kinerja

KPL dapat membantu membangun produsen berkinerja tinggi. Pertimbangkan situasi di mana instans Amazon EC2 Anda berfungsi sebagai proxy untuk mengumpulkan peristiwa 100 byte dari

ratusan atau ribuan perangkat berdaya rendah dan menulis catatan ke dalam aliran data Kinesis. Instans EC2 ini masing-masing harus menulis ribuan peristiwa per detik ke aliran data Anda. Untuk mencapai throughput yang dibutuhkan, produsen harus menerapkan logika yang rumit, seperti batching atau multithreading, selain logika coba ulang dan de-agregasi rekaman di sisi konsumen. KPL melakukan semua tugas ini untuk Anda.

Kemudahan Penggunaan Sisi Konsumen

Untuk pengembang sisi konsumen yang menggunakan KCL di Jawa, KPL terintegrasi tanpa usaha tambahan. Ketika KCL mengambil rekaman Kinesis Data Streams agregat yang terdiri dari beberapa catatan pengguna KPL, secara otomatis memanggil KPL untuk mengekstrak catatan pengguna individu sebelum mengembalikannya ke pengguna.

Untuk pengembang sisi konsumen yang tidak menggunakan KCL tetapi menggunakan operasi API `GetRecords` secara langsung, perpustakaan Java KPL tersedia untuk mengekstrak catatan pengguna individu sebelum mengembalikannya ke pengguna.

Pemantauan Produsen

Anda dapat mengumpulkan, memantau, dan menganalisis produsen Kinesis Data Streams menggunakan CloudWatch Amazon dan KPL. KPL memancarkan throughput, kesalahan, dan metrik lainnya untuk CloudWatch atas nama Anda, dan dapat dikonfigurasi untuk memantau pada tingkat aliran, pecahan, atau produsen.

Arsitektur Asinkron

Karena KPL dapat menyangga catatan sebelum mengirimnya ke Kinesis Data Streams, itu tidak memaksa aplikasi pemanggil untuk memblokir dan menunggu konfirmasi bahwa catatan telah tiba di server sebelum melanjutkan eksekusi. Panggilan untuk memasukkan catatan ke dalam KPL selalu segera kembali dan tidak menunggu catatan dikirim atau respons diterima dari server. Sebaliknya, `Future` objek dibuat yang menerima hasil pengiriman catatan ke Kinesis Data Streams di lain waktu. Ini adalah perilaku yang sama dengan klien asinkron di SDK. AWS

Kapan Tidak Menggunakan KPL

KPL dapat menimbulkan penundaan pemrosesan tambahan hingga di `RecordMaxBufferedTime` dalam perpustakaan (dapat dikonfigurasi pengguna). Nilai yang lebih besar `RecordMaxBufferedTime` menghasilkan efisiensi pengepakan yang lebih tinggi dan kinerja yang lebih baik. Aplikasi yang tidak dapat mentolerir penundaan tambahan ini mungkin perlu menggunakan AWS SDK secara langsung. Untuk informasi selengkapnya tentang penggunaan

AWS SDK dengan Kinesis Data Streams, lihat. Mengembangkan Produsen Menggunakan API Amazon Kinesis Data Streams denganAWS SDK for Java Untuk informasi selengkapnya tentang `RecordMaxBufferedTime` dan properti KPL yang dapat dikonfigurasi pengguna lainnya, lihat. Mengkonfigurasi Perpustakaan Produser Kinesis

Instalasi KPL

Amazon menyediakan binari C++ Kinesis Producer Library (KPL) bawaan untuk macOS, Windows, dan distribusi Linux terbaru (untuk detail platform yang didukung, lihat bagian selanjutnya). Binari ini dikemas sebagai bagian dari file Java .jar dan secara otomatis dipanggil dan digunakan jika Anda menggunakan Maven untuk menginstal paket. Untuk menemukan versi terbaru KPL dan KCL, gunakan tautan pencarian Maven berikut:

- KPL
- KCL

Binari Linux telah dikompilasi dengan GNU Compiler Collection (GCC) dan ditautkan secara statis terhadap libstdc++ di Linux. Mereka diharapkan untuk bekerja pada distribusi Linux 64-bit yang mencakup glibc versi 2.5 atau lebih tinggi.

Pengguna distribusi Linux yang lebih lama dapat membangun KPL menggunakan instruksi build yang disediakan bersama dengan sumbernya. GitHub Untuk mengunduh KPL dari GitHub, lihat Perpustakaan Produser Kinesis.

Transisi ke Sertifikat Amazon Trust Services (ATS) untuk Perpustakaan Produsen Kinesis

Pada tanggal 9 Februari 2018, pukul 09.00 PST, Amazon Kinesis Data Streams memasang sertifikat ATS. Untuk terus dapat menulis catatan ke Kinesis Data Streams menggunakan Kinesis Producer Library (KPL), Anda harus meng-upgrade instalasi KPL Anda ke versi 0.12.6 atau yang lebih baru. Perubahan ini mempengaruhi semua AWS Wilayah.

Untuk informasi tentang perpindahan ke ATS, silakan lihat Cara AWSMempersiapkan Pindah ke Otoritas Sertifikat Sendiri.

Jika Anda mengalami masalah dan membutuhkan dukungan teknis, buat kasus dengan AWS Support Center.

Platform yang Didukung KPL

Kinesis Producer Library (KPL) ditulis dalam C++ dan berjalan sebagai proses anak ke proses pengguna utama. Binari asli 64-bit yang dikompilasi dibundel dengan rilis Java dan dikelola oleh pembungkus Java.

Paket Java berjalan tanpa perlu menginstal pustaka tambahan pada sistem operasi berikut:

- Distribusi Linux dengan kernel 2.6.18 (September 2006) dan kemudian
- Apple OS X 10.9 dan yang lebih baru
- Windows Server 2008 dan yang lebih baru

> **Important**
>
> Windows Server 2008 dan yang lebih baru didukung untuk semua versi KPL hingga versi 0.14.0.
> Platform Windows TIDAK didukung dimulai dengan KPL versi 0.14.0 atau lebih tinggi.

Perhatikan bahwa KPL hanya 64-bit.

Kode Sumber

Jika binari yang disediakan dalam instalasi KPL tidak cukup untuk lingkungan Anda, inti KPL ditulis sebagai modul C++. Kode sumber untuk modul C++ dan antarmuka Java dirilis di bawah Lisensi Publik Amazon dan tersedia GitHub di Perpustakaan Produser Kinesis. Meskipun KPL dapat digunakan pada platform apa pun di mana kompiler C++ dan JRE yang sesuai standar baru-baru ini tersedia, Amazon tidak secara resmi mendukung platform apa pun yang tidak ada dalam daftar platform yang didukung.

Konsep Kunci KPL

Bagian berikut berisi konsep dan terminologi yang diperlukan untuk memahami dan mendapatkan manfaat dari Perpustakaan Produsen Kinesis (KPL).

Topik

- Catatan
- Batching

- Agregasi
- Koleksi

Catatan

Dalam panduan ini, kami membedakan antara catatan pengguna KPL dan catatan Kinesis Data Streams. Saat kami menggunakan catatan istilah tanpa kualifikasi, kami merujuk ke catatan pengguna KPL. Ketika kami merujuk ke catatan Kinesis Data Streams, kami secara eksplisit mengatakan catatan Kinesis Data Streams.

Catatan pengguna KPL adalah gumpalan data yang memiliki arti khusus bagi pengguna. Contohnya termasuk gumpalan JSON yang mewakili peristiwa UI di situs web, atau entri log dari server web.

Catatan Kinesis Data Streams adalah instance `Record` dari struktur data yang ditentukan oleh API layanan Kinesis Data Streams. Ini berisi kunci partisi, nomor urut, dan gumpalan data.

Batching

Batching mengacu pada melakukan satu tindakan pada beberapa item alih-alih berulang kali melakukan tindakan pada setiap item individu.

Dalam konteks ini, “item” adalah catatan, dan tindakan mengirimkannya ke Kinesis Data Streams. Dalam situasi non-batching, Anda akan menempatkan setiap rekaman dalam catatan Kinesis Data Streams terpisah dan membuat satu permintaan HTTP untuk mengirimkannya ke Kinesis Data Streams. Dengan batching, setiap permintaan HTTP dapat membawa beberapa catatan, bukan hanya satu.

KPL mendukung dua jenis batching:

- Agregasi — Menyimpan beberapa catatan dalam satu catatan Kinesis Data Streams.
- Pengumpulan — Menggunakan operasi API `PutRecords` untuk mengirim beberapa catatan Kinesis Data Streams ke satu atau beberapa pecahan dalam aliran data Kinesis Anda.

Kedua jenis batching KPL dirancang untuk hidup berdampingan dan dapat dihidupkan atau dimatikan secara independen satu sama lain. Secara default, keduanya dihidupkan.

Agregasi

Agregasi mengacu pada penyimpanan beberapa catatan dalam catatan Kinesis Data Streams. Agregasi memungkinkan pelanggan untuk meningkatkan jumlah catatan yang dikirim per panggilan API, yang secara efektif meningkatkan throughput produsen.

Pecahan Kinesis Data Streams mendukung hingga 1.000 rekaman Kinesis Data Streams per detik, atau throughput 1 MB. Catatan Kinesis Data Streams per detik mengikat pelanggan dengan catatan yang lebih kecil dari 1 KB. Agregasi rekaman memungkinkan pelanggan untuk menggabungkan beberapa catatan ke dalam satu catatan Kinesis Data Streams. Hal ini memungkinkan pelanggan untuk meningkatkan throughput per shard mereka.

Pertimbangkan kasus satu pecahan di wilayah us-timur-1 yang saat ini berjalan pada tingkat konstan 1.000 catatan per detik, dengan catatan yang masing-masing 512 byte. Dengan agregasi KPL, Anda dapat mengemas 1.000 catatan hanya ke dalam 10 catatan Kinesis Data Streams, mengurangi RPS menjadi 10 (masing-masing 50 KB).

Koleksi

Collection mengacu pada pengelompokan beberapa data Kinesis Data Streams dan mengirimkannya dalam satu permintaan HTTP dengan panggilan ke `PutRecords` operasi API, alih-alih mengirim setiap catatan Kinesis Data Streams dalam permintaan HTTP-nya sendiri.

Ini meningkatkan throughput dibandingkan dengan tidak menggunakan koleksi karena mengurangi overhead membuat banyak permintaan HTTP terpisah. Bahkan, `PutRecords` dirinya sendiri dirancang khusus untuk tujuan ini.

Koleksi berbeda dari agregasi karena bekerja dengan kelompok catatan Kinesis Data Streams. Rekaman Kinesis Data Streams yang dikumpulkan masih dapat berisi beberapa catatan dari pengguna. Hubungan tersebut dapat divisualisasikan seperti:

```
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--|
```

Mengintegrasikan KPL dengan Kode Produser

Kinesis Producer Library (KPL) berjalan dalam proses terpisah, dan berkomunikasi dengan proses pengguna induk Anda menggunakan IPC. Arsitektur ini kadang-kadang disebut microservice, dan dipilih karena dua alasan utama:

1) Proses pengguna Anda tidak akan macet bahkan jika KPL mogok

Proses Anda dapat memiliki tugas yang tidak terkait dengan Kinesis Data Streams, dan mungkin dapat melanjutkan operasi bahkan jika KPL mogok. Dimungkinkan juga bagi proses pengguna induk Anda untuk memulai ulang KPL dan memulihkan ke keadaan berfungsi penuh (fungsi ini ada di pembungkus resmi).

Contohnya adalah server web yang mengirimkan metrik ke Kinesis Data Streams; server dapat terus melayani halaman bahkan jika bagian Kinesis Data Streams telah berhenti bekerja. Merusak seluruh server karena bug di KPL akan menyebabkan pemadaman yang tidak perlu.

2) Klien arbitrer dapat didukung

Selalu ada pelanggan yang menggunakan bahasa selain yang didukung secara resmi. Pelanggan ini juga harus dapat menggunakan KPL dengan mudah.

Matriks Penggunaan yang Disarankan

Matriks penggunaan berikut menyebutkan pengaturan yang disarankan untuk pengguna yang berbeda dan memberi tahu Anda tentang apakah dan bagaimana Anda harus menggunakan KPL. Perlu diingat bahwa jika agregasi diaktifkan, de-agregasi juga harus digunakan untuk mengekstrak catatan Anda di sisi konsumen.

Bahasa sampingan produser	Bahasa sisi konsumen	Versi KCL	Logika titik pemeriksaan	Bisakah Anda menggunakan KPL?	Peringatan
Apa pun kecuali Jawa	*	*	*	Tidak	N/A
Java	Java	Menggunakan Java SDK secara langsung	N/A	Ya	Jika agregasi digunakan, Anda harus menggunakan pustaka de-agregasi yang disediakan setelah panggilan. `GetRecords`
Java	Apa pun kecuali Jawa	Menggunakan SDK secara langsung	N/A	Ya	Harus menonaktifkan agregasi.
Java	Java	1.3.x	N/A	Ya	Harus menonaktifkan agregasi.
Java	Java	1.4.x	Panggilan pos pemeriksaan tanpa argumen apa pun	Ya	Tidak ada

Bahasa sampingan produser	Bahasa sisi konsumen	Versi KCL	Logika titik pemeriksaan	Bisakah Anda menggunakan KPL?	Peringatan
Java	Java	1.4.x	Memanggil pos pemeriksaan dengan nomor urut eksplisit	Ya	Nonaktifkan agregasi, atau ubah kode untuk menggunakan nomor urutan yang diperluas untuk pos pemeriksaan.
Java	Apa pun kecuali Jawa	1.3.x +daemon multibahasa+pembungkus khusus bahasa	N/A	Ya	Harus menonaktifkan agregasi.

Menulis ke Aliran Data Kinesis Anda Menggunakan KPL

Bagian berikut menunjukkan kode sampel dalam perkembangan dari produsen “bare-bones” yang paling sederhana hingga kode asinkron sepenuhnya.

Kode Produsen Barebones

Kode berikut adalah semua yang diperlukan untuk menulis produsen kerja minimal. Catatan pengguna Kinesis Producer Library (KPL) diproses di latar belakang.

```
// 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 ...
```

Menanggapi hasil secara sinkron

Pada contoh sebelumnya, kode tidak memeriksa apakah catatan pengguna KPL berhasil. KPL melakukan percobaan ulang yang diperlukan untuk memperhitungkan kegagalan. Tetapi jika Anda ingin memeriksa hasilnya, Anda dapat memeriksanya menggunakan `Future` objek yang dikembalikan`addUserRecord`, seperti pada contoh berikut (contoh sebelumnya ditampilkan untuk konteks):

```
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
        }
    }
}
```

Menanggapi Hasil Secara Asinkron

Contoh sebelumnya adalah memanggil `get()` `Future` objek, yang memblokir eksekusi. Jika Anda tidak ingin memblokir eksekusi, Anda dapat menggunakan callback asinkron, seperti yang ditunjukkan pada contoh berikut:

```
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);
}
```

Mengkonfigurasi Perpustakaan Produser Kinesis

Meskipun pengaturan default harus berfungsi dengan baik untuk sebagian besar kasus penggunaan, Anda mungkin ingin mengubah beberapa pengaturan default untuk menyesuaikan perilaku dengan `KinesisProducer` kebutuhan Anda. Sebuah instance dari `KinesisProducerConfiguration` kelas dapat diteruskan ke `KinesisProducer` konstruktor untuk melakukannya, misalnya:

```
KinesisProducerConfiguration config = new KinesisProducerConfiguration()
        .setRecordMaxBufferedTime(3000)
        .setMaxConnections(1)
        .setRequestTimeout(60000)
        .setRegion("us-west-1");

final KinesisProducer kinesisProducer = new KinesisProducer(config);
```

Anda juga dapat memuat konfigurasi dari file properti:

```
KinesisProducerConfiguration config =
 KinesisProducerConfiguration.fromPropertiesFile("default_config.properties");
```

Anda dapat mengganti jalur dan nama file apa pun yang dapat diakses oleh proses pengguna. Anda juga dapat memanggil metode set pada `KinesisProducerConfiguration` instance yang dibuat dengan cara ini untuk menyesuaikan konfigurasi.

File properti harus menentukan parameter menggunakan nama mereka di PascalCase. Nama-nama cocok dengan yang digunakan dalam metode set di `KinesisProducerConfiguration` kelas. Sebagai contoh:

```
RecordMaxBufferedTime = 100
MaxConnections = 4
RequestTimeout = 6000
Region = us-west-1
```

Untuk informasi selengkapnya tentang aturan penggunaan parameter konfigurasi dan batas nilai, lihat contoh file properti konfigurasi GitHub.

Perhatikan bahwa setelah `KinesisProducer` diinisialisasi, mengubah `KinesisProducerConfiguration` instance yang digunakan tidak memiliki efek lebih lanjut. `KinesisProducer`saat ini tidak mendukung konfigurasi ulang dinamis.

De-agregasi Konsumen

Dimulai dengan rilis 1.4.0, KCL mendukung de-agregasi otomatis catatan pengguna KPL. Kode aplikasi konsumen yang ditulis dengan versi KCL sebelumnya akan dikompilasi tanpa modifikasi apa pun setelah Anda memperbarui KCL. Namun, jika agregasi KPL digunakan di sisi produsen, ada kehalusan yang melibatkan checkpointing: semua subrecord dalam catatan agregat memiliki nomor urut yang sama, sehingga data tambahan harus disimpan dengan pos pemeriksaan jika Anda perlu membedakan antara subrecord. Data tambahan ini disebut sebagai nomor urutan.

Migrasi dari Versi Sebelumnya dari KCL

Anda tidak diharuskan mengubah panggilan yang ada untuk melakukan checkpointing bersamaan dengan agregasi. Masih dijamin bahwa Anda dapat mengambil semua catatan yang berhasil disimpan di Kinesis Data Streams. KCL sekarang menyediakan dua operasi pos pemeriksaan baru untuk mendukung kasus penggunaan tertentu, yang dijelaskan di bawah ini.

Jika kode Anda yang ada ditulis untuk KCL sebelum dukungan KPL, dan operasi pos pemeriksaan Anda dipanggil tanpa argumen, itu setara dengan memeriksa nomor urut catatan pengguna KPL terakhir dalam batch. Jika operasi pos pemeriksaan Anda dipanggil dengan string nomor urut, itu setara dengan pemeriksaan nomor urut yang diberikan dari batch bersama dengan nomor urutan implisit 0 (nol).

Memanggil operasi pos pemeriksaan KCL baru `checkpoint()` tanpa argumen apa pun secara semantik setara dengan pemeriksaan nomor urut `Record` panggilan terakhir dalam batch, bersama dengan nomor urutan implisit 0 (nol).

Memanggil operasi pos pemeriksaan KCL baru `checkpoint(Record record)` secara semantik setara dengan pemeriksaan nomor urut yang diberikan `Record` bersama dengan nomor urutan implisit 0 (nol). Jika `Record` panggilan sebenarnya a`UserRecord`, nomor `UserRecord` urut dan nomor urutan diperiksa.

Memanggil operasi pos pemeriksaan KCL baru `checkpoint(String sequenceNumber, long subSequenceNumber)` secara eksplisit memeriksa nomor urut yang diberikan bersama dengan nomor urutan yang diberikan.

Dalam salah satu kasus ini, setelah pos pemeriksaan disimpan di tabel pos pemeriksaan Amazon DynamoDB, KCL dapat melanjutkan pengambilan catatan dengan benar bahkan ketika aplikasi mogok dan restart. Jika lebih banyak catatan terkandung dalam urutan, pengambilan terjadi dimulai dengan catatan nomor urutan berikutnya dalam catatan dengan nomor urut yang terakhir diperiksa. Jika pos pemeriksaan terbaru menyertakan nomor urutan terakhir dari catatan nomor urut sebelumnya, pengambilan terjadi dimulai dengan catatan dengan nomor urut berikutnya.

Bagian selanjutnya membahas rincian urutan dan urutan checkpointing untuk konsumen yang perlu menghindari melewatkan dan duplikasi catatan. Jika melewatkan (atau duplikasi) catatan saat menghentikan dan memulai ulang pemrosesan catatan konsumen Anda tidak penting, Anda dapat menjalankan kode yang ada tanpa modifikasi.

Ekstensi KCL untuk De-agregasi KPL

Seperti yang telah dibahas sebelumnya, de-agregasi KPL dapat melibatkan pos pemeriksaan selanjutnya. Untuk memfasilitasi penggunaan checkpointing sequence, sebuah `UserRecord` kelas telah ditambahkan ke 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 */
    }
}
```

Kelas ini sekarang digunakan sebagai penggantiRecord. Ini tidak merusak kode yang ada karena merupakan subkelas dari. Record UserRecordKelas mewakili subrecord aktual dan standar, catatan non-agregat. Catatan non-agregat dapat dianggap sebagai catatan agregat dengan tepat satu subrecord.

Selain itu, dua operasi baru ditambahkan keIRecordProcessorCheckpointer:

```
public void checkpoint(Record record);
public void checkpoint(String sequenceNumber, long subSequenceNumber);
```

Untuk mulai menggunakan checkpointing nomor urutan, Anda dapat melakukan konversi berikut. Ubah kode formulir berikut:

```
checkpointer.checkpoint(record.getSequenceNumber());
```

Kode formulir baru:

```
checkpointer.checkpoint(record);
```

Kami menyarankan Anda menggunakan checkpoint(Record record) formulir untuk checkpointing berikutnya. Namun, jika Anda sudah menyimpan sequenceNumbers dalam string untuk digunakan untuk checkpointing, Anda sekarang juga harus menyimpansubSequenceNumber, seperti yang ditunjukkan pada contoh berikut:

```
String sequenceNumber = record.getSequenceNumber();
long subSequenceNumber = ((UserRecord) record).getSubSequenceNumber();  // ... do other
 processing
```

```
checkpointer.checkpoint(sequenceNumber, subSequenceNumber);
```

Pemeran dari `Record` untuk `UserRecord` selalu berhasil karena implementasinya selalu menggunakan `UserRecord` di bawah tenda. Kecuali ada kebutuhan untuk melakukan aritmatika pada nomor urut, pendekatan ini tidak disarankan.

Saat memproses catatan pengguna KPL, KCL menulis nomor urutan ke Amazon DynamoDB sebagai bidang tambahan untuk setiap baris. Versi KCL sebelumnya digunakan `AFTER_SEQUENCE_NUMBER` untuk mengambil catatan saat melanjutkan pos pemeriksaan. KCL saat ini dengan dukungan KPL menggunakan `AT_SEQUENCE_NUMBER` sebagai gantinya. Ketika catatan pada nomor urut yang diperiksa diambil, nomor urutan yang diperiksa diperiksa, dan subrecord dijatuhkan sebagaimana mestinya (yang mungkin semuanya, jika subrecord terakhir adalah yang diperiksa). Sekali lagi, catatan non-agregat dapat dianggap sebagai catatan agregat dengan satu subrecord, sehingga algoritma yang sama berfungsi untuk catatan agregat dan non-agregat.

Menggunakan GetRecords Langsung

Anda juga dapat memilih untuk tidak menggunakan KCL tetapi menjalankan operasi API `GetRecords` secara langsung untuk mengambil catatan Kinesis Data Streams. Untuk membongkar catatan yang diambil ini ke dalam catatan pengguna KPL asli Anda, panggil salah satu operasi statis berikut di: `UserRecord.java`

```
public static List<Record> deaggregate(List<Record> records)

public static List<UserRecord> deaggregate(List<UserRecord> records, BigInteger
 startingHashKey, BigInteger endingHashKey)
```

Operasi pertama menggunakan nilai default `0` (nol) untuk `startingHashKey` dan nilai default `2^128 -1` untuk`endingHashKey`.

Masing-masing operasi ini melakukan de-agregasi daftar catatan Kinesis Data Streams yang diberikan ke dalam daftar catatan pengguna KPL. Setiap catatan pengguna KPL yang kunci hash eksplisit atau kunci partisi berada di luar jangkauan `startingHashKey` (inklusif) dan `endingHashKey` (inklusif) dibuang dari daftar catatan yang dikembalikan.

Menggunakan KPL dengan Firehose

Jika Anda menggunakan Kinesis Producer Library (KPL) untuk menulis data ke aliran data Kinesis, Anda dapat menggunakan agregasi untuk menggabungkan catatan yang Anda tulis ke aliran data

Kinesis tersebut. Jika Anda kemudian menggunakan aliran data tersebut sebagai sumber untuk aliran pengiriman Firehose Anda, Firehose menghapus agregasi catatan sebelum mengirimkannya ke tujuan. Jika Anda mengonfigurasi aliran pengiriman untuk mengubah data, Firehose menghapus agregasi catatan sebelum mengirimkannya. AWS LambdaUntuk informasi selengkapnya, lihat Menulis ke Amazon Firehose Menggunakan Kinesis Data Streams.

Menggunakan KPL dengan AWS Glue Schema Registry

Anda dapat mengintegrasikan aliran data Kinesis Anda dengan registri skema AWS Glue. Registri skema AWS Glue memungkinkan Anda menemukan, mengontrol, dan mengembangkan skema secara terpusat, sambil memastikan data yang dihasilkan terus divalidasi oleh skema terdaftar. Sebuah skema mendefinisikan struktur dan format catatan data. Sebuah skema adalah sebuah spesifikasi berversi untuk publikasi data yang handal, konsumsi, atau penyimpanan. AWS Glue Schema Registry memungkinkan Anda untuk meningkatkan kualitas end-to-end data dan tata kelola data dalam aplikasi streaming Anda. Untuk informasi selengkapnya, lihat AWS Glue Schema Registry. Salah satu cara untuk mengatur integrasi ini adalah melalui perpustakaan KPL dan Kinesis Client Library (KCL) di Jawa.

> **Important**
>
> Saat ini, integrasi registri skema Kinesis Data AWS Streams dan Glue hanya didukung untuk aliran data Kinesis yang menggunakan produsen KPL yang diimplementasikan di Jawa. Support multi-bahasa tidak tersedia.

Untuk petunjuk terperinci tentang cara mengatur integrasi Kinesis Data Streams dengan Schema Registry menggunakan KPL, lihat bagian “Berinteraksi dengan Data Menggunakan Perpustakaan KPL/KCL” di Kasus Penggunaan: Mengintegrasikan Aliran Data Kinesis Amazon dengan Registri Skema Glue. AWS

Konfigurasi Proxy KPL

Untuk aplikasi yang tidak dapat terhubung langsung ke internet, semua klien AWS SDK mendukung penggunaan proxy HTTP atau HTTPS. Dalam lingkungan perusahaan yang khas, semua lalu lintas jaringan keluar harus melalui server proxy. Jika aplikasi Anda menggunakan Kinesis Producer Library (KPL) untuk mengumpulkan dan mengirim data ke AWS lingkungan yang menggunakan server proxy, aplikasi Anda akan memerlukan konfigurasi proxy KPL. KPL adalah perpustakaan tingkat tinggi yang dibangun di atas AWS Kinesis SDK. Ini dibagi menjadi proses asli dan pembungkus.

Proses asli melakukan semua pekerjaan pemrosesan dan pengiriman catatan, sementara pembungkus mengelola proses asli dan berkomunikasi dengannya. Untuk informasi selengkapnya, lihat Menerapkan Produsen yang Efisien dan Andal dengan Perpustakaan Produsen Amazon Kinesis.

Pembungkus ditulis dalam Java dan proses asli ditulis dalam C ++ dengan menggunakan Kinesis SDK. KPL versi 0.14.7 dan yang lebih tinggi sekarang mendukung konfigurasi proxy di pembungkus Java yang dapat meneruskan semua konfigurasi proxy ke proses asli. Untuk informasi lebih lanjut, lihat https://github.com/awslabs/ amazon-kinesis-producer /releases/tag/v0.14.7.

Anda dapat menggunakan kode berikut untuk menambahkan konfigurasi proxy ke aplikasi KPL Anda.

```
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);
```

Mengembangkan Produsen Menggunakan API Amazon Kinesis Data Streams denganAWS SDK for Java

Anda dapat mengembangkan produsen menggunakan Amazon Kinesis Data Streams API denganAWSSDK for Java. Jika Anda baru mengenal Kinesis Data Streams, mulailah dengan memahami konsep dan terminologi yang disajikan dalamApa itu Amazon Kinesis Data Streams?danMemulai dengan Amazon Kinesis Data Streams.

Contoh-contoh ini membahasAPI Kinesis Data Streamsdan gunakanAWSSDK for Javauntuk menambahkan (menempatkan) data ke stream. Namun, untuk sebagian besar kasus penggunaan, Anda harus memilih perpustakaan Kinesis Data Streams KPL. Untuk informasi selengkapnya, lihat Mengembangkan Produsen Menggunakan Perpustakaan Produsen Amazon Kinesis.

Java contoh kode dalam Bab ini menunjukkan bagaimana melakukan operasi dasar Kinesis Data Streams API, dan dibagi secara logis oleh jenis operasi. Contoh-contoh ini tidak mewakili

kode siap produksi, karena contoh ini tidak memeriksa semua kemungkinan pengecualian, atau memperhitungkan semua kemungkinan pertimbangan keamanan atau performa. Juga, Anda dapat menghubungiAPI Kinesis Data Streamsmenggunakan bahasa pemrograman lainnya. Untuk informasi lebih lanjut tentang semua yang tersediaAWSSDK, lihatMulai Berkembang dengan Amazon Web Services.

Setiap tugas memiliki prasyarat; misalnya, Anda tidak dapat menambahkan data ke stream sampai Anda membuat stream, yang mengharuskan Anda untuk membuat klien. Untuk informasi selengkapnya, lihat Membuat dan Mengelola Streaming.

Topik

- Menambahkan Data ke Stream
- Berinteraksi dengan Data MenggunakanAWSRegistri Skema

Menambahkan Data ke Stream

Setelah stream dibuat, Anda dapat menambahkan data ke dalamnya dalam bentuk catatan. Catatan adalah struktur data yang berisi data yang akan diproses dalam bentuk gumpalan data. Setelah Anda menyimpan data dalam catatan, Kinesis Data Streams tidak memeriksa, menafsirkan, atau mengubah data dengan cara apa pun. Setiap catatan juga memiliki nomor urut dan kunci partisi yang terkait.

Ada dua operasi yang berbeda dalam Kinesis Data Streams API yang menambahkan data ke stream,`PutRecords`dan`PutRecord`. Parameter`PutRecords`operasi mengirimkan beberapa catatan ke aliran Anda per permintaan HTTP, dan tunggal`PutRecord`operasi mengirimkan catatan ke aliran Anda satu per satu (permintaan HTTP terpisah diperlukan untuk setiap catatan). Anda harus lebih memilih menggunakan`PutRecords`untuk sebagian besar aplikasi karena akan mencapai throughput yang lebih tinggi per produsen data. Untuk informasi selengkapnya tentang masing-masing operasi ini, lihat subbagian terpisah di bawah ini.

Topik

- Menambahkan Beberapa Rekaman denganPutRecords
- Menambahkan Rekam Tunggal denganPutRecord

Selalu ingat bahwa, karena aplikasi sumber Anda menambahkan data ke stream menggunakan Kinesis Data Streams API, kemungkinan besar ada satu atau lebih aplikasi konsumen yang secara

bersamaan memproses data dari aliran. Untuk informasi tentang cara konsumen mendapatkan data menggunakan Kinesis Data Streams API, lihatMendapatkan Data dari Stream.

> **Important**
>
> Mengubah Periode Retensi Data

Menambahkan Beberapa Rekaman denganPutRecords

ParameterPutRecordsoperasi mengirimkan beberapa catatan ke Kinesis Data Streams dalam satu permintaan. Dengan menggunakanPutRecords, produsen dapat mencapai throughput yang lebih tinggi saat mengirim data ke aliran data Kinesis mereka. MASINGPutRecordspermintaan dapat mendukung hingga 500 catatan. Setiap catatan dalam permintaan dapat sebesar 1 MB, hingga batas 5 MB untuk seluruh permintaan, termasuk kunci partisi. Seperti dengan singlePutRecordoperasi dijelaskan di bawah ini,PutRecordsmenggunakan nomor urut dan tombol partisi. Namun,PutRecordparameterSequenceNumberForOrderingtidak termasuk dalamPutRecordspanggilan. ParameterPutRecordsupaya untuk memproses semua catatan dalam urutan alami permintaan.

Setiap catatan data memiliki nomor urut yang unik. Nomor urut ditugaskan oleh Kinesis Data Streams setelah Anda meneleponclient.putRecordsuntuk menambahkan catatan data ke sungai. Nomor urutan untuk kunci partisi yang sama umumnya meningkat dari waktu ke waktu; semakin lama periode waktu antaraPutRecordspermintaan, semakin besar nomor urut menjadi.

> **Note**
>
> Nomor urutan tidak dapat digunakan sebagai indeks untuk set data dalam aliran yang sama. Untuk memisahkan set data secara logis, gunakan tombol partisi atau buat aliran terpisah untuk setiap kumpulan data.

SEBUAHPutRecordspermintaan dapat mencakup catatan dengan kunci partisi yang berbeda. Ruang lingkup permintaan adalah aliran; setiap permintaan dapat mencakup kombinasi kunci partisi dan catatan hingga batas permintaan. Permintaan yang dibuat dengan banyak kunci partisi yang berbeda ke aliran dengan banyak pecahan yang berbeda umumnya lebih cepat daripada permintaan dengan sejumlah kecil kunci partisi ke sejumlah kecil pecahan. Jumlah kunci partisi harus jauh lebih besar dari jumlah pecahan untuk mengurangi latensi dan memaksimalkan throughput.

PutRecordsContoh

Kode berikut membuat 100 catatan data dengan kunci partisi berurutan dan menempatkan mereka dalam aliran yang disebut`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);
```

Parameter`PutRecords`respon termasuk array respon`Records`. Setiap catatan dalam susunan respons secara langsung berkorelasi dengan catatan dalam susunan permintaan dengan urutan asal, dari permintaan dan respons paling atas hingga paling bawah. Respons`Records`Array selalu mencakup jumlah yang sama dengan susunan permintaan.

Menangani Kegagalan Saat MenggunakanPutRecords

Secara default, kegagalan catatan individu dalam permintaan tidak menghentikan pemrosesan catatan berikutnya dalam`PutRecords`permintaan. Ini berarti bahwa respons`Records`Array

mencakup catatan berhasil maupun tidak berhasil diproses. Anda harus mendeteksi catatan yang tidak berhasil diproses dan menyertakannya ke dalam panggilan berikutnya.

Catatan berhasil mencakup`SequenceNumber`dan`ShardID`nilai, dan catatan yang tidak berhasil termasuk`ErrorCode`dan`ErrorMessage`nilai. Parameter`ErrorCode`mencerminkan jenis kesalahan dan dapat berupa salah satu nilai berikut:`ProvisionedThroughputExceededException`atau`InternalFailure`.`ErrorMessage`memberi informasi lebih rinci tentang`ProvisionedThroughputExceededException`pengecualian termasuk ID akun, nama stream, dan ID shard dari record yang throttled. Contoh di bawah ini memiliki tiga catatan dalam`PutRecords`permintaan. Catatan kedua gagal dan tercermin dalam respons.

Example PutRecordsSintaks Permintaan

```
{
    "Records": [
        {
     "Data": "XzxkYXRhPl8w",
     "PartitionKey": "partitionKey1"
        },
        {
     "Data": "AbceddeRFfg12asd",
     "PartitionKey": "partitionKey1"
        },
        {
     "Data": "KFpcd98*7nd1",
     "PartitionKey": "partitionKey3"
        }
    ],
    "StreamName": "myStream"
}
```

Example PutRecordsSintaksis Respons

```
{
    "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"
        }
    ]
}
```

Catatan yang tidak berhasil diproses dapat disertakan dalam`PutRecords`permintaan. Pertama, periksa parameter `FailedRecordCount` dalam `putRecordsResult` untuk mengonfirmasi apakah ada catatan kegagalan dalam permintaan. Jika ya, masing-masing`putRecordsEntry`yang memiliki`ErrorCode`yang tidak`null`harus ditambahkan ke permintaan berikutnya. Untuk contoh jenis handler ini, lihat kode berikut.

Example PutRecordshandler kegagalan

```
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);
}
```

Menambahkan Rekam Tunggal denganPutRecord

Setiap panggilan kePutRecordberoperasi pada satu rekor. Lebih sukaPutRecordsdijelaskan dalamMenambahkan Beberapa Rekaman denganPutRecordskecuali aplikasi Anda secara khusus perlu selalu mengirim catatan tunggal per permintaan, atau alasan lainPutRecordsTidak dapat digunakan.

Setiap catatan data memiliki nomor urut yang unik. Nomor urut ditugaskan oleh Kinesis Data Streams setelah Anda meneleponclient.putRecorduntuk menambahkan catatan data ke sungai. Nomor urutan untuk kunci partisi yang sama umumnya meningkat dari waktu ke waktu; semakin lama periode waktu antaraPutRecordpermintaan, semakin besar nomor urut menjadi.

Ketika menempatkan terjadi dalam suksesi cepat, nomor urut kembali tidak dijamin meningkat karena operasi put muncul pada dasarnya sebagai simultan untuk Kinesis Data Streams. Untuk menjamin jumlah urutan yang meningkat secara ketat untuk kunci partisi yang sama, gunakanSequenceNumberForOrderingparameter, seperti yang ditunjukkan dalamPutRecordContohsampel kode.

Apakah Anda menggunakanSequenceNumberForOrdering, mencatat bahwa Kinesis Data Streams menerima melaluiGetRecordspanggilan secara ketat diperintahkan oleh nomor urut.

> **Note**
>
> Nomor urutan tidak dapat digunakan sebagai indeks untuk set data dalam aliran yang sama. Untuk memisahkan set data secara logis, gunakan tombol partisi atau buat aliran terpisah untuk setiap kumpulan data.

Kunci partisi digunakan untuk mengelompokkan data dalam aliran. Rekaman data ditugaskan ke pecahan dalam aliran berdasarkan kunci partisi. Secara khusus, Kinesis Data Streams menggunakan

tombol partisi sebagai masukan ke fungsi hash yang memetakan kunci partisi (dan data terkait) ke pecahan tertentu.

Sebagai hasil dari mekanisme hashing ini, semua catatan data dengan peta kunci partisi yang sama ke pecahan yang sama dalam aliran. Namun, jika jumlah kunci partisi melebihi jumlah pecahan, beberapa pecahan harus berisi catatan dengan kunci partisi yang berbeda. Dari sudut pandang desain, untuk memastikan bahwa semua pecahan Anda dimanfaatkan dengan baik, jumlah pecahan (ditentukan olehsetShardCountmetodeCreateStreamRequest) harus secara substansial kurang dari jumlah kunci partisi yang unik, dan jumlah data yang mengalir ke kunci partisi tunggal harus secara substansial kurang dari kapasitas pecahan.

PutRecordContoh

Kode berikut membuat sepuluh catatan data, didistribusikan di dua kunci partisi, dan menempatkan mereka dalam aliran yang disebutmyStreamName.

```
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();
}
```

Contoh kode sebelumnyasetSequenceNumberForOrderinguntuk menjamin ketat meningkatkan pemesanan dalam setiap kunci partisi. Untuk menggunakan parameter ini secara efektif, aturSequenceNumberForOrderingdari catatan saat ini (catatann) ke nomor urutan catatan sebelumnya (recordn-1). Untuk mendapatkan nomor urutan rekaman yang telah ditambahkan ke aliran, panggilgetSequenceNumberpada hasilputRecord.

ParameterSequenceNumberForOrderingparameter memastikan secara ketat meningkatkan nomor urutan untuk kunci partisi yang sama.SequenceNumberForOrderingtidak menyediakan pemesanan catatan di beberapa kunci partisi.

Berinteraksi dengan Data MenggunakanAWSRegistri Skema

Anda dapat mengintegrasikan aliran data Kinesis Anda denganAWSRegistri skema Glue. ParameterAWSRegistri skema memungkinkan Anda untuk menemukan, mengontrol, dan mengembangkan skema secara terpusat. Sebuah skema mendefinisikan struktur dan format catatan data. Sebuah skema adalah sebuah spesifikasi berversi untuk publikasi data yang handal, konsumsi, atau penyimpanan. ParameterAWSGlue Skema Registry memungkinkan Anda untuk meningkatkanend-to-endkualitas data dan tata kelola data dalam aplikasi streaming Anda. Untuk informasi selengkapnya, lihatAWSRegistri Skema. Salah satu cara untuk mengatur integrasi ini adalah melaluiPutRecordsdanPutRecordAPI Kinesis Data Streams tersedia dalamAWSSDK Java.

Untuk petunjuk rinci tentang cara mengatur Kinesis Data Streams dengan Registri Skema menggunakanPutRecordsdanPutRecordAPI Kinesis Data Streams, lihat bagian “Berinteraksi dengan Data Menggunakan API Kinesis Data Streams”Kasus Penggunaan: Mengintegrasikan Amazon Kinesis Data Streams denganAWSRegistri Skema.

Menulis ke Amazon Kinesis Data Streams Menggunakan Agen Kinesis

Kinesis Agent adalah aplikasi perangkat lunak Java yang berdiri sendiri yang menawarkan cara mudah untuk mengumpulkan dan mengirim data ke Kinesis Data Streams. Agen terus memantau serangkaian file dan mengirimkan data baru ke aliran Anda. Agen menangani rotasi file, checkpointing, dan coba lagi pada kegagalan. Agen memberikan semua data Anda dengan cara yang andal, tepat waktu, dan sederhana. Ini juga memancarkan CloudWatch metrik Amazon untuk membantu Anda memantau dan memecahkan masalah proses streaming dengan lebih baik.

Secara default, catatan diurai dari setiap file berdasarkan karakter baris baru (`'\n'`). Namun, agen juga dapat dikonfigurasi untuk mengurai catatan multi-baris (lihat Pengaturan Konfigurasi Agen).

Anda dapat menginstal agen di lingkungan server berbasis Linux seperti server web, server log, dan server basis data. Setelah menginstal agen, konfigurasikan dengan menentukan file yang akan dipantau dan aliran untuk data. Setelah agen dikonfigurasi, agen akan mengumpulkan data dari file dengan andal dan mengirimkannya ke aliran dengan andal.

Topik

- Prasyarat
- Mengunduh dan Menginstal Agen

- Mengonfigurasi dan Memulai Agen
- Pengaturan Konfigurasi Agen
- Memantau Beberapa Direktori File dan Menulis ke Beberapa Aliran
- Gunakan Agen untuk Memproses Data
- Perintah Agen CLI
- Pertanyaan yang Sering Diajukan

Prasyarat

- Sistem operasi Anda harus Amazon Linux AMI dengan versi 2015.09 atau yang lebih baru, atau Red Hat Enterprise Linux versi 7 atau yang lebih baru.
- Jika Anda menggunakan Amazon EC2 untuk menjalankan agen Anda, luncurkan instans EC2 Anda.
- Kelola AWS kredensyal Anda menggunakan salah satu metode berikut:
  - Tentukan IAM role ketika Anda meluncurkan instans EC2 Anda.
  - Tentukan AWS kredensil saat Anda mengonfigurasi agen (lihat awsAccessKeyId dan awsSecretAccessKunci).
  - Edit `/etc/sysconfig/aws-kinesis-agent` untuk menentukan wilayah dan kunci AWS akses Anda.
  - Jika instans EC2 Anda berada di AWS akun yang berbeda, buat peran IAM untuk menyediakan akses ke layanan Kinesis Data Streams, dan tentukan peran tersebut saat Anda mengonfigurasi agen (lihat AssumeroLearn dan Id). assumeRoleExternal Gunakan salah satu metode sebelumnya untuk menentukan AWS kredensi pengguna di akun lain yang memiliki izin untuk mengambil peran ini.
- Peran IAM atau AWS kredenal yang Anda tentukan harus memiliki izin untuk melakukan operasi Kinesis Data PutRecordsStreams agar agen dapat mengirim data ke aliran Anda. Jika Anda mengaktifkan CloudWatch pemantauan untuk agen, izin untuk melakukan CloudWatch PutMetricDataoperasi juga diperlukan. Untuk informasi selengkapnyaMengontrol Akses ke Sumber Daya Amazon Kinesis Data Streams Menggunakan IAM, lihatMemantau Kinesis Data Streams Agen Kesehatan dengan Amazon CloudWatch, dan Kontrol CloudWatch Akses.

Mengunduh dan Menginstal Agen

Pertama-tama, hubungkan ke instans Anda. Untuk informasi selengkapnya, lihat Menghubungkan ke Instans Anda dalam Panduan Pengguna Amazon EC2 untuk Instans Linux. Jika Anda mengalami masalah saat menyambungkan, lihat Memecahkan Masalah Penyambungan ke Instans Anda di Panduan Pengguna Amazon EC2 untuk Instans Linux.

Untuk mengatur agen menggunakan Amazon Linux AMI

Gunakan perintah berikut untuk mengunduh dan menginstal agen:

```
sudo yum install -y aws-kinesis-agent
```

Untuk mengatur agen menggunakan Red Hat Enterprise Linux

Gunakan perintah berikut untuk mengunduh dan menginstal agen:

```
sudo yum install -y https://s3.amazonaws.com/streaming-data-agent/aws-kinesis-agent-
latest.amzn2.noarch.rpm
```

Untuk mengatur agen menggunakan GitHub

1. Unduh agen dari amazon-kinesis-agentawlabs/.
2. Instal agen dengan menavigasi ke direktori unduhan dan menjalankan perintah berikut:

   ```
   sudo ./setup --install
   ```

Untuk mengatur agen dalam wadah Docker

Agen Kinesis dapat dijalankan dalam wadah juga melalui basis wadah amazonlinux. Gunakan Dockerfile berikut dan kemudian jalankan. `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"]
```

Mengonfigurasi dan Memulai Agen

Untuk mengonfigurasi dan memulai agen

1. Buka dan edit file konfigurasi (sebagai pengguna super jika menggunakan izin akses file default): `/etc/aws-kinesis/agent.json`

   Dalam file konfigurasi ini, tentukan file (`"filePattern"`) dari mana agen mengumpulkan data, dan nama stream (`"kinesisStream"`) tempat agen mengirim data. Perhatikan bahwa nama file adalah pola, dan agen mengenali rotasi file. Anda dapat memutar file atau membuat file baru satu kali per detik. Agen menggunakan stempel waktu pembuatan file untuk menentukan file mana yang akan dilacak dan diekor ke aliran Anda; membuat file baru atau memutar file lebih sering dari sekali per detik tidak memungkinkan agen untuk membedakan dengan benar di antara mereka.

   ```
   {
      "flows": [
           {
               "filePattern": "/tmp/app.log*",
               "kinesisStream": "yourkinesisstream"
           }
      ]
   }
   ```

2. Mulailah agen secara manual:

   ```
   sudo service aws-kinesis-agent start
   ```

3. (Opsional) Konfigurasikan agen untuk memulai pada startup sistem:

   ```
   sudo chkconfig aws-kinesis-agent on
   ```

Agen sekarang berjalan sebagai layanan sistem di latar belakang. Ini terus memantau file yang ditentukan dan mengirim data ke aliran yang ditentukan. Aktivitas agen masuk di `/var/log/aws-kinesis-agent/aws-kinesis-agent.log`.

Pengaturan Konfigurasi Agen

Agen mendukung dua pengaturan konfigurasi wajib, `filePattern` dan`kinesisStream`, ditambah pengaturan konfigurasi opsional untuk fitur tambahan. Anda dapat menentukan konfigurasi wajib dan opsional di`/etc/aws-kinesis/agent.json`.

Setiap kali mengubah file konfigurasi, Anda harus menghentikan dan memulai agen, menggunakan perintah berikut:

```
sudo service aws-kinesis-agent stop
sudo service aws-kinesis-agent start
```

Atau, Anda dapat menggunakan perintah berikut:

```
sudo service aws-kinesis-agent restart
```

Berikut ini adalah pengaturan konfigurasi umum.

Pengaturan Konfigurasi	Deskripsi
`assumeRoleARN`	ARN dari peran yang akan diasumsikan oleh pengguna. Untuk informasi selengkapnya, lihat Mendelegasikan Akses di Seluruh AWS Akun Menggunakan Peran IAM di Panduan Pengguna IAM.
`assumeRoleExternalId`	Pengidentifikasi opsional yang menentukan siapa yang dapat mengambil peran tersebut. Untuk informasi selengkapnya, lihat Cara Menggunakan ID Eksternal di Panduan Pengguna IAM.
`awsAccessKeyId`	AWS ID kunci akses yang mengesampingkan kredensyal default. Pengaturan ini diutamakan daripada semua penyedia kredensial lainnya.
`awsSecretAccessKey`	AWS kunci rahasia yang mengesampingkan kredensi default. Pengaturan ini diutamakan daripada semua penyedia kredensial lainnya.
`cloudwatch.emitMetrics`	Memungkinkan agen untuk memancarkan metrik ke CloudWatch if set (true). Default: betul

Pengaturan Konfigurasi	Deskripsi
`cloudwatch.endpoint`	Titik akhir regional untuk CloudWatch. Default: `monitoring.us-east-1.amazonaws.com`
`kinesis.endpoint`	Titik akhir regional untuk Kinesis Data Streams. Default: `kinesis.us-east-1.amazonaws.com`

Berikut ini adalah pengaturan konfigurasi aliran.

Pengaturan Konfigurasi	Deskripsi
`dataProcessingOptions`	Daftar opsi pemrosesan diterapkan ke setiap catatan yang diuraikan sebelum dikirim ke aliran. Pilihan pemrosesan dilakukan dalam urutan yang ditentukan. Untuk informasi selengkapnya, lihat Gunakan Agen untuk Memproses Data.
`kinesisStream`	[Wajib] Nama aliran.
`filePattern`	[Wajib] Direktori dan pola file yang harus dicocokkan untuk diambil oleh agen. Untuk semua file yang cocok dengan pola ini, izin baca harus diberikan`aws-kinesis-agent-user` . Untuk direktori yang berisi file, izin baca dan eksekusi harus diberikan kepada`aws-kinesis-agent-user` .
`initialPosition`	Posisi awal dari mana file mulai diurai. Nilai yang valid adalah `START_OF_FILE` dan `END_OF_FILE` . Default: `END_OF_FILE`
`maxBufferAgeMillis`	Waktu maksimum, dalam milidetik, di mana agen menyangga data sebelum mengirimnya ke aliran. Rentang nilai: 1.000 hingga 900.000 (1 detik hingga 15 menit)

Pengaturan Konfigurasi	Deskripsi
	Default: 60.000 (1 menit)
`maxBufferSizeBytes`	Ukuran maksimum, dalam byte, di mana agen buffer data sebelum mengirimnya ke aliran. Rentang nilai: 1 hingga 4.194.304 (4 MB) Default: 4.194.304 (4 MB)
`maxBufferSizeRecords`	Jumlah maksimum catatan yang agen buffer data sebelum mengirimnya ke aliran. Rentang nilai: 1 hingga 500 Default: 500
`minTimeBetweenFilePollsMillis`	Interval waktu, dalam milidetik, saat agen melakukan polling dan mengurai file yang dipantau untuk data baru. Kisaran nilai: 1 atau lebih Default: 100
`multiLineStartPattern`	Pola untuk mengidentifikasi awal catatan. Catatan dibuat dari baris yang cocok dengan pola tersebut dan baris berikutnya yang tidak cocok dengan pola tersebut. Nilai-nilai yang benar adalah ekspresi reguler. Secara default, setiap baris baru dalam file log diurai sebagai satu catatan.
`partitionKeyOption`	Metode untuk menghasilkan kunci partisi. Nilai yang valid adalah `RANDOM` (bilangan bulat yang dihasilkan secara acak) dan `DETERMINISTIC` (nilai hash dihitung dari data). Default: `RANDOM`

Pengaturan Konfigurasi	Deskripsi
`skipHeaderLines`	Jumlah baris yang dilewati agen untuk diurai di awal file yang dipantau. Kisaran nilai: 0 atau lebih Default: 0 (nol)
`truncatedRecordTerminator`	String yang digunakan agen untuk memotong rekaman yang diuraikan ketika ukuran rekaman melebihi batas ukuran rekaman Kinesis Data Streams. (1.000 KB) Default: `'\n'` (baris baru)

Memantau Beberapa Direktori File dan Menulis ke Beberapa Aliran

Dengan menentukan beberapa pengaturan konfigurasi aliran, Anda dapat mengonfigurasi agen untuk memantau beberapa direktori file dan mengirim data ke beberapa aliran. Dalam contoh konfigurasi berikut, agen memonitor dua direktori file dan mengirimkan data ke aliran Kinesis dan aliran pengiriman Firehose masing-masing. Perhatikan bahwa Anda dapat menentukan titik akhir yang berbeda untuk Kinesis Data Streams dan Firehose sehingga aliran Kinesis dan aliran pengiriman Firehose tidak perlu berada di wilayah yang sama.

```
{
    "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"
        }
    ]
}
```

Untuk informasi lebih lanjut tentang penggunaan agen dengan Firehose, lihat Menulis ke Amazon Data Firehose dengan Agen Kinesis.

Gunakan Agen untuk Memproses Data

Agen dapat memproses catatan yang diuraikan dari file yang dipantau sebelum mengirimnya ke streaming Anda. Anda dapat mengaktifkan fitur ini dengan menambahkan pengaturan konfigurasi `dataProcessingOptions` ke aliran file Anda. Satu atau lebih opsi pemrosesan dapat ditambahkan dan mereka akan dilakukan dalam urutan yang ditentukan.

Agen mendukung opsi pemrosesan berikut yang tercantum. Karena agen bersifat open-source, Anda dapat mengembangkan dan memperluas opsi pemrosesannya lebih lanjut. Anda dapat mengunduh agen dari Agen Kinesis.

Opsi Pemrosesan

`SINGLELINE`

Mengonversi rekaman multi-baris menjadi catatan baris tunggal dengan menghapus karakter baris baru, spasi utama, dan spasi tambahan.

```
{
    "optionName": "SINGLELINE"
}
```

`CSVTOJSON`

Mengkonversi rekaman dari pembatas format dipisahkan ke format JSON.

```
{
    "optionName": "CSVTOJSON",
    "customFieldNames": [ "field1", "field2", ... ],
    "delimiter": "yourdelimiter"
}
```

`customFieldNames`

[Diperlukan] Nama-nama field yang digunakan sebagai kunci dalam setiap pasangan nilai kunci JSON. Misalnya, jika Anda menentukan`["f1", "f2"]`, catatan “v1, v2" akan dikonversi ke. `{"f1":"v1","f2":"v2"}`

`delimiter`

String yang digunakan sebagai pembatas dalam catatan. Default adalah koma (,).

`LOGTOJSON`

Mengonversi catatan dari format log ke format JSON. Format log yang didukung adalah Apache Common Log, Apache Combined Log, Apache Error Log, dan RFC3164 Syslog.

```
{
    "optionName": "LOGTOJSON",
    "logFormat": "logformat",
    "matchPattern": "yourregexpattern",
    "customFieldNames": [ "field1", "field2", … ]
}
```

`logFormat`

[Diperlukan] Format entri log. Berikut adalah nilai yang mungkin:

- `COMMONAPACHELOG` — Format Log Umum Apache. Setiap entri log memiliki pola berikut secara default: `"%{host} %{ident} %{authuser} [%{datetime}] \"%{request}\" %{response} %{bytes}"`.
- `COMBINEDAPACHELOG` — Format Log Gabungan Apache. Setiap entri log memiliki pola berikut secara default: `"%{host} %{ident} %{authuser} [%{datetime}] \"%{request}\" %{response} %{bytes} %{referrer} %{agent}"`.
- `APACHEERRORLOG` — Format Log Kesalahan Apache. Setiap entri log memiliki pola berikut secara default: `"[%{timestamp}] [%{module}:%{severity}] [pid %{processid}:tid %{threadid}] [client: %{client}] %{message}"`.
- `SYSLOG` — Format Syslog RFC3164. Setiap entri log memiliki pola berikut secara default: `"%{timestamp} %{hostname} %{program}[%{processid}]: %{message}"`.

`matchPattern`

Pola ekspresi reguler digunakan untuk mengekstrak nilai dari entri log. Pengaturan ini digunakan jika entri log Anda tidak dalam salah satu format log yang telah ditentukan. Jika pengaturan ini digunakan, Anda juga harus menentukan`customFieldNames`.

`customFieldNames`

Nama bidang khusus digunakan sebagai kunci dalam setiap pasangan nilai kunci JSON. Anda dapat menggunakan pengaturan ini untuk menentukan nama bidang untuk nilai-nilai yang

diekstraksi dari `matchPattern`, atau menimpa nama bidang default dari format log yang telah ditetapkan sebelumnya.

Example : Konfigurasi LOGTOJSON

Berikut adalah salah satu contoh konfigurasi `LOGTOJSON` untuk entri Log Umum Apache yang dikonversi ke format JSON:

```
{
    "optionName": "LOGTOJSON",
    "logFormat": "COMMONAPACHELOG"
}
```

Sebelum konversi:

```
64.242.88.10 - - [07/Mar/2004:16:10:02 -0800] "GET /mailman/listinfo/hsdivision
 HTTP/1.1" 200 6291
```

Setelah konversi:

```
{"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 : Konfigurasi LOGTOJSON dengan Bidang Khusus

Berikut adalah contoh lain konfigurasi `LOGTOJSON`:

```
{
    "optionName": "LOGTOJSON",
    "logFormat": "COMMONAPACHELOG",
    "customFieldNames": ["f1", "f2", "f3", "f4", "f5", "f6", "f7"]
}
```

Dengan pengaturan konfigurasi ini, entri Log Umum Apache yang sama dari contoh sebelumnya dikonversi ke format JSON sebagai berikut:

```
{"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 : Mengonversi Entri Log Umum Apache

Konfigurasi alur berikut mengonversi entri Apache Common Log ke catatan baris tunggal dalam format JSON:

```
{
    "flows": [
        {
            "filePattern": "/tmp/app.log*",
            "kinesisStream": "my-stream",
            "dataProcessingOptions": [
                {
                    "optionName": "LOGTOJSON",
                    "logFormat": "COMMONAPACHELOG"
                }
            ]
        }
    ]
}
```

Example : Mengonversi Catatan Multi-Baris

Konfigurasi aliran berikut mengurai catatan multi-baris yang baris pertamanya dimulai dengan "`[SEQUENCE=`". Setiap catatan pertama kali dikonversi ke catatan baris tunggal. Kemudian, nilai-nilai diekstraksi dari catatan tersebut berdasarkan pembatas tab. Nilai yang diekstraksi dipetakan ke nilai `customFieldNames` yang ditentukan untuk membentuk catatan baris tunggal dalam format JSON.

```
{
    "flows": [
        {
            "filePattern": "/tmp/app.log*",
            "kinesisStream": "my-stream",
            "multiLineStartPattern": "\\[SEQUENCE=",
            "dataProcessingOptions": [
                {
                    "optionName": "SINGLELINE"
                },
                {
                    "optionName": "CSVTOJSON",
                    "customFieldNames": [ "field1", "field2", "field3" ],
                    "delimiter": "\\t"
                }
```

```
            ]
        }
    ]
}
```

Example : Konfigurasi LOGTOJSON dengan Pola Pencocokan

Berikut adalah salah satu contoh konfigurasi `LOGTOJSON` untuk entri Log Umum Apache yang dikonversi ke format JSON, dengan bidang terakhir (byte) dihilangkan:

```
{
    "optionName": "LOGTOJSON",
    "logFormat": "COMMONAPACHELOG",
    "matchPattern": "^([\\d.]+) (\\S+) (\\S+) \\[([\\w:/]+\\s[+\\-]\\d{4})\\] \"(.
+?)\" (\\d{3})",
    "customFieldNames": ["host", "ident", "authuser", "datetime", "request",
 "response"]
}
```

Sebelum konversi:

```
123.45.67.89 - - [27/Oct/2000:09:27:09 -0400] "GET /java/javaResources.html HTTP/1.0"
 200
```

Setelah konversi:

```
{"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"}
```

Perintah Agen CLI

Secara otomatis memulai agen pada startup sistem:

```
sudo chkconfig aws-kinesis-agent on
```

Memeriksa status agen:

```
sudo service aws-kinesis-agent status
```

Menghentikan agen:

```
sudo service aws-kinesis-agent stop
```

Membaca file log agen dari lokasi ini:

```
/var/log/aws-kinesis-agent/aws-kinesis-agent.log
```

Menghapus instalasi agen:

```
sudo yum remove aws-kinesis-agent
```

Pertanyaan yang Sering Diajukan

Apakah ada Agen Kinesis untuk Windows?

Kinesis Agent untuk Windows adalah perangkat lunak yang berbeda dari Kinesis Agent untuk platform Linux.

Mengapa Agen Kinesis melambat dan/atau meningkat? `RecordSendErrors`

Ini biasanya karena pelambatan dari Kinesis. Periksa `WriteProvisionedThroughputExceeded` metrik untuk Kinesis Data Streams `ThrottledRecords` atau metrik untuk Firehose Delivery Streams. Setiap peningkatan dari 0 dalam metrik ini menunjukkan bahwa batas aliran perlu ditingkatkan. Untuk informasi selengkapnya, lihat batas Kinesis Data Stream dan Amazon Firehose Delivery Streams.

Setelah Anda mengesampingkan pembatasan, lihat apakah Agen Kinesis dikonfigurasi untuk mengekor sejumlah besar file kecil. Ada penundaan ketika Agen Kinesis mengekor file baru, jadi Agen Kinesis harus membuntuti sejumlah kecil file yang lebih besar. Coba konsolidasikan file log Anda ke file yang lebih besar.

Mengapa saya mendapatkan `java.lang.OutOfMemoryError` pengecualian?

Agen Kinesis tidak memiliki cukup memori untuk menangani beban kerjanya saat ini. Cobalah meningkatkan `JAVA_START_HEAP` dan `JAVA_MAX_HEAP` masuk `/usr/bin/start-aws-kinesis-agent` dan memulai kembali agen.

Mengapa saya mendapatkan **`IllegalStateException : connection pool shut down`** pengecualian?

Agen Kinesis tidak memiliki koneksi yang cukup untuk menangani beban kerjanya saat ini. Coba tingkatkan `maxConnections` dan `maxSendingThreads` dalam pengaturan konfigurasi agen umum Anda di`/etc/aws-kinesis/agent.json`. Nilai default untuk bidang ini adalah 12 kali prosesor runtime yang tersedia. Lihat AgentConfiguration.java untuk mengetahui selengkapnya tentang pengaturan konfigurasi agen lanjutan.

Bagaimana saya bisa men-debug masalah lain dengan Agen Kinesis?

`DEBUG`log level dapat diaktifkan di`/etc/aws-kinesis/log4j.xml`.

Bagaimana cara mengonfigurasi Agen Kinesis?

Semakin kecil`maxBufferSizeBytes`, semakin sering Agen Kinesis akan mengirim data. Ini bisa bagus karena mengurangi waktu pengiriman catatan, tetapi juga meningkatkan permintaan per detik ke Kinesis.

Mengapa Agen Kinesis mengirimkan catatan duplikat?

Ini terjadi karena kesalahan konfigurasi dalam file tailing. Pastikan masing-masing hanya `fileFlow's filePattern` cocok dengan satu file. Ini juga dapat terjadi jika `logrotate` mode yang digunakan dalam `copytruncate` mode. Coba ubah mode ke mode default atau buat untuk menghindari duplikasi. Untuk informasi selengkapnya tentang penanganan rekaman duplikat, lihat Menangani Rekaman Duplikat.

Menulis ke Kinesis Data AWS Streams menggunakan Layanan lain

Berikut ini adalah daftar AWS layanan lain yang dapat berintegrasi langsung dengan Kinesis Data Streams untuk menulis data ke Kinesis Data Streams:

Topik

- AWS Amplify
- Amazon Aurora
- Amazon CloudFront
- CloudWatch Log Amazon
- Amazon Connect

- AWS Database Migration Service
- Amazon DynamoDB
- Amazon EventBridge
- AWS IoT Core
- Layanan Basis Data Relasional Amazon
- Amazon Pinpoint
- Database Buku Besar Amazon Quantum

AWS Amplify

Anda dapat menggunakan Amazon Kinesis Data Streams untuk mengalirkan data dengan mudah dari aplikasi seluler yang dibuat AWS dengan Amplify untuk pemrosesan waktu nyata. Anda kemudian dapat membuat dasbor real-time, menangkap pengecualian dan menghasilkan peringatan, mendorong rekomendasi, dan membuat keputusan bisnis atau operasional real-time lainnya. Anda juga dapat dengan mudah mengirim data ke layanan lain seperti Amazon Simple Storage Service, Amazon DynamoDB, dan Amazon Redshift.

Untuk informasi selengkapnya, lihat Menggunakan Amazon Kinesis di Pusat Pengembang AWSAmplify.

Amazon Aurora

Anda dapat menggunakan Amazon Kinesis Data Streams untuk memantau aktivitas di kluster Amazon Aurora DB Anda. Menggunakan Aliran Aktivitas Database, klaster Aurora DB Anda mendorong aktivitas ke Aliran Data Amazon Kinesis secara real-time. Anda kemudian dapat membangun aplikasi untuk manajemen kepatuhan yang menggunakan aktivitas ini, mengaudit mereka dan menghasilkan peringatan. Anda juga dapat menggunakan Amazon Amazon Firehose untuk menyimpan data.

Untuk informasi selengkapnya, lihat Aliran Aktivitas Database di Panduan Pengembang Amazon Aurora.

Amazon CloudFront

Anda dapat menggunakan Amazon Kinesis Data CloudFront Streams dengan log real-time dan mendapatkan informasi tentang permintaan yang dibuat ke distribusi secara real time. Anda

kemudian dapat membangun konsumen aliran data Kinesis Anda sendiri, atau menggunakan Amazon Amazon Firehose untuk mengirim data log ke Amazon Simple Storage Service (Amazon S3), Amazon Redshift, Amazon Service (Service), atau OpenSearch layanan pemrosesan OpenSearch log pihak ketiga.

Untuk informasi selengkapnya, lihat Log waktu nyata di Panduan CloudFront Pengembang Amazon.

CloudWatch Log Amazon

Anda dapat menggunakan CloudWatch langganan untuk mendapatkan akses ke umpan real-time peristiwa log dari Amazon CloudWatch Logs dan mengirimkannya ke Amazon Kinesis Data Stream untuk diproses, dianalisis, dan dimuat ke sistem lain.

Untuk informasi selengkapnya, lihat Pemrosesan data log secara real-time dengan langganan di Panduan Pengguna Amazon CloudWatch Logs.

Amazon Connect

Anda dapat menggunakan Kinesis Data Streams untuk mengekspor catatan kontak dan peristiwa agen secara real-time dari instans Amazon Connect. Anda juga dapat mengaktifkan streaming data dari Profil Pelanggan Amazon Connect untuk secara otomatis menerima pembaruan ke Aliran Data Kinesis tentang pembuatan profil baru atau perubahan pada profil yang sudah ada.

Anda kemudian dapat membangun aplikasi konsumen untuk memproses dan menganalisis data secara real-time. Misalnya, menggunakan catatan kontak dan data profil pelanggan, Anda dapat menyimpan data sistem sumber Anda, seperti CRM dan alat otomatisasi pemasaran, up-to-date dengan informasi terbaru. Dengan menggunakan data peristiwa agen, Anda dapat membuat dasbor yang menampilkan informasi dan peristiwa agen, dan memicu pemberitahuan khusus tentang aktivitas agen tertentu.

Untuk informasi selengkapnya, lihat streaming data untuk instans Anda, atur ekspor waktu nyata, dan aliran peristiwa agen di Panduan Administrator Amazon Connect.

AWS Database Migration Service

Anda dapat menggunakan AWS Database Migration Service untuk memigrasikan data ke Amazon Kinesis Data Stream. Anda dapat membangun aplikasi konsumen yang memproses catatan data secara real time. Anda juga dapat dengan mudah mengirim data ke hilir ke layanan lain seperti Amazon Simple Storage Service, Amazon DynamoDB, dan Amazon Redshift.

Untuk informasi selengkapnya, lihat Menggunakan Kinesis Data Streams AWS Database Migration Servicedi Panduan Pengguna.

Amazon DynamoDB

Anda dapat menggunakan Amazon Kinesis Data Streams untuk menangkap perubahan pada Amazon DynamoDB. Kinesis Data Streams menangkap modifikasi tingkat item dalam tabel DynamoDB dan mereplikasi mereka ke Kinesis Data Stream. Aplikasi konsumen Anda dapat mengakses aliran ini untuk melihat perubahan tingkat item secara real time dan mengirimkan perubahan tersebut ke hilir atau mengambil tindakan berdasarkan konten.

Untuk informasi selengkapnya, lihat cara Kinesis Data Streams bekerja dengan DynamoDB di Panduan Pengembang Amazon DynamoDB.

Amazon EventBridge

Menggunakan Kinesis Data Streams, Anda AWS dapat mengirim EventBridge peristiwa panggilan API ke aliran, membangun aplikasi konsumen, dan memproses data dalam jumlah besar. Anda juga dapat menggunakan Kinesis Data Streams sebagai EventBridge target di Pipes dan mengirimkan rekaman aliran dari salah satu sumber yang tersedia setelah pemfilteran dan pengayaan opsional.

Untuk informasi selengkapnya, lihat Mengirim peristiwa ke aliran Amazon Kinesis dan EventBridge Pipa di EventBridge Panduan Pengguna Amazon.

AWS IoT Core

Anda dapat menulis data secara real-time dari pesan MQTT di IoT AWS Core dengan menggunakan tindakan Aturan IoT. AWS Anda kemudian dapat membangun aplikasi yang memproses data, menganalisis kontennya dan menghasilkan peringatan, dan mengirimkannya ke aplikasi analitik atau AWS layanan lainnya,

Untuk informasi selengkapnya, lihat Kinesis Data Streams di Panduan AWSPengembang Inti IoT.

Layanan Basis Data Relasional Amazon

Anda dapat menggunakan Amazon Kinesis Data Streams untuk memantau aktivitas di instans Amazon RDS Anda. Menggunakan Aliran Aktivitas Database, Amazon RDS mendorong aktivitas ke Aliran Data Amazon Kinesis secara real-time. Anda kemudian dapat membangun aplikasi untuk manajemen kepatuhan yang menggunakan aktivitas ini, mengaudit mereka dan menghasilkan peringatan. Anda juga dapat menggunakan Amazon Amazon Firehose untuk menyimpan data.

Untuk informasi selengkapnya, lihat Aliran Aktivitas Database di Panduan Pengembang Amazon RDS.

Amazon Pinpoint

Anda dapat mengatur Amazon Pinpoint untuk mengirim data peristiwa ke Amazon Kinesis Data Streams. Amazon Pinpoint dapat mengirim data acara untuk kampanye, perjalanan, dan email transaksional dan pesan SMS. Anda kemudian dapat menyerap data ke dalam aplikasi analitik atau membangun aplikasi konsumen Anda sendiri yang mengambil tindakan berdasarkan konten acara.

Untuk informasi selengkapnya, lihat Acara Streaming di Panduan Pengembang Amazon Pinpoint.

Database Buku Besar Amazon Quantum

Anda dapat membuat aliran di QLDB yang menangkap setiap revisi dokumen yang berkomitmen pada jurnal Anda dan mengirimkan data ini ke Amazon Kinesis Data Streams secara real time. Aliran QLDB adalah aliran data yang berkelanjutan dari jurnal buku besar Anda ke sumber daya aliran data Kinesis. Kemudian, Anda dapat menggunakan platform streaming Kinesis atau Perpustakaan Klien Kinesis untuk menggunakan streaming Anda, memproses catatan data, dan menganalisis konten data. Aliran QLDB menulis data Anda ke Kinesis Data Streams dalam tiga jenis rekaman:, , dan. `control` `block summary` `revision details`

Untuk informasi selengkapnya, lihat Streaming di Panduan pengembang Amazon QLDB.

Menggunakan integrasi pihak ketiga

Anda dapat menulis data ke aliran Data Kinesis menggunakan salah satu opsi pihak ketiga berikut yang terintegrasi dengan Kinesis Data Streams:

Topik

- Apache Flink
- Fasih
- Debezium
- Oracle GoldenGate
- Mengublikasi Kafka
- Pengalaman Adobe
- Striim

Apache Flink

Apache Flink adalah sebuah kerangka kerja dan mesin pengolahan terdistribusi untuk komputasi stateful atas aliran data yang tak terbatas dan dibatasi. Untuk informasi selengkapnya tentang cara menulis ke Kinesis Data Streams dari Apache Flink, lihat Konektor Amazon Kinesis Data Streams.

Fasih

Fluentd adalah kolektor data open source untuk lapisan logging terpadu. Untuk informasi lebih lanjut tentang menulis ke Kinesis Data Streams dari Fluentd. Untuk informasi selengkapnya, lihat Pemrosesan aliran dengan Kinesis.

Debezium

Debezium adalah platform terdistribusi open source untuk pengambilan data perubahan. Untuk informasi selengkapnya tentang menulis ke Kinesis Data Streams dari Debezium, lihat Streaming Perubahan Data MySQL ke Amazon Kinesis.

Oracle GoldenGate

Oracle GoldenGate adalah produk perangkat lunak yang memungkinkan Anda untuk mereplikasi, menyaring, dan mengubah data dari satu database ke database lain. Untuk informasi lebih lanjut tentang menulis ke Kinesis Data Streams dari Oracle GoldenGate, lihat Replikasi data ke Kinesis Data Stream menggunakan Oracle GoldenGate.

Mengublikasi Kafka

Kafka Connect adalah alat untuk streaming data yang dapat diskalakan dan andal antara Apache Kafka dan sistem lainnya. Untuk informasi lebih lanjut tentang menulis data dari Apache Kafka ke Kinesis Data Streams, lihat Kinesis kafka konektor.

Pengalaman Adobe

Adobe Experience Platform memungkinkan organisasi untuk memusatkan dan menstandarisasi data pelanggan dari sistem apa pun. Ini kemudian menerapkan ilmu data dan pembelajaran mesin untuk secara dramatis meningkatkan desain dan penyampaian pengalaman yang kaya dan dipersonalisasi. Untuk informasi selengkapnya tentang menulis data dari Adobe Experience Platform ke Kinesis Data Streams. lihat cara membuat koneksi Amazon Kinesis.

Striim

Striim adalah platform lengkap dalam memori untuk mengumpulkan, memfilter, mengubah, memperkaya, menggabungkan, menganalisis, dan mengirimkan data secara real time. end-to-end Untuk informasi lebih lanjut tentang cara menulis data ke Kinesis Data Streams dari Striim, lihat Kinesis Writer.

Pemecahan Masalah Produsen Amazon Kinesis Data Streams

Bagian berikut menawarkan solusi untuk beberapa masalah umum yang mungkin Anda temukan saat bekerja dengan produsen Amazon Kinesis Data Streams.

- Aplikasi Produser Menulis pada Tingkat Lebih Lambat dari yang Diharapkan
- Kesalahan izin kunci utama KMS
- Masalah umum, pertanyaan, dan ide pemecahan masalah bagi produsen

Aplikasi Produser Menulis pada Tingkat Lebih Lambat dari yang Diharapkan

Alasan paling umum untuk menulis throughput yang lebih lambat dari yang diharapkan adalah sebagai berikut.

- Batas Layanan Melebihi
- Pengoptimalan produsen

Batas Layanan Melebihi

Untuk mengetahui apakah batas layanan terlampaui, periksa apakah produsen Anda melempar pengecualian throughput dari layanan, dan validasi operasi API apa yang sedang dilekang. Perlu diingat bahwa ada batas yang berbeda berdasarkan panggilan, lihatKuota dan Batas. Sebagai contoh, selain batas shard-level untuk menulis dan membaca yang paling umum dikenal, ada batas tingkat aliran berikut:

- CreateStream
- DeleteStream
- ListStreams
- GetShardIterator

- MergeShards
- DescribeStream
- DescribeStreamRingkasan

OperasiCreateStream,DeleteStream,ListStreams,GetShardIterator, danMergeShardsdibatasi hingga 5 panggilan per detik. ParameterDescribeStreamoperasi terbatas hingga 10 panggilan per detik. ParameterDescribeStreamSummaryoperasi terbatas pada 20 panggilan per detik.

Jika panggilan ini bukan masalah, pastikan Anda telah memilih kunci partisi yang memungkinkan Anda untuk mendistribusikanmenempatkanoperasi secara merata di semua pecahan, dan bahwa Anda tidak memiliki kunci partisi tertentu yang menabrak batas layanan ketika sisanya tidak. Hal ini mengharuskan Anda mengukur throughput puncak dan memperhitungkan jumlah pecahan dalam aliran Anda. Untuk informasi selengkapnya tentang mengelola aliran, lihatMembuat dan Mengelola Streaming.

> **Tip**
>
> Ingatlah untuk mengumpulkan kilobyte terdekat untuk perhitungan throttling throughput saat menggunakan operasi single-recordPutRecord, sedangkan operasi multi-recordPutRecordsputaran pada jumlah kumulatif dari catatan di setiap panggilan. Misalnya, aPutRecordspermintaan dengan 600 catatan yang berukuran 1,1 KB tidak akan mendapatkan throttled.

Pengoptimalan produsen

Sebelum Anda mulai mengoptimalkan produser Anda, ada beberapa tugas utama yang harus diselesaikan. Pertama, identifikasi throughput puncak yang Anda inginkan dalam hal ukuran rekaman dan catatan per detik. Selanjutnya, mengesampingkan kapasitas aliran sebagai faktor pembatas (Batas Layanan Melebihi). Jika Anda telah mengesampingkan kapasitas streaming, gunakan tips pemecahan masalah berikut dan pedoman optimasi untuk dua jenis produsen yang umum.

Produsen Besar

Produsen besar biasanya berjalan dari server lokal atau instans Amazon EC2. Pelanggan yang membutuhkan throughput yang lebih tinggi dari produsen besar biasanya peduli dengan latensi per-record. Strategi untuk menangani latensi meliputi: Jika pelanggan dapat micro-batch/buffer record,

gunakanPerpustakaan Produser Kinesis(yang memiliki logika agregasi lanjutan), operasi multi-recordPutRecords, atau catatan agregat ke file yang lebih besar sebelum menggunakan operasi single-recordPutRecord. Jika Anda tidak dapat batch/buffer, gunakan beberapa thread untuk menulis ke layanan Kinesis Data Streams pada saat yang bersamaan. ParameterAWS SDK for Javadan SDK lainnya termasuk klien async yang dapat melakukan ini dengan kode yang sangat sedikit.

Produsen Kecil

Produser kecil biasanya merupakan aplikasi seluler, perangkat IoT, atau klien web. Jika itu adalah aplikasi seluler, sebaiknya gunakan`PutRecords`operasi atau Perekam Kinesis diAWSMobile SDK. Untuk informasi selengkapnya, lihatAWS Mobile SDK for AndroidPanduan MemulaiAWS Mobile SDK for iOSPanduan Memulai Aplikasi seluler harus menangani koneksi intermiten secara inheren dan memerlukan semacam batch put, seperti`PutRecords`. Jika Anda tidak dapat melakukan batch untuk beberapa alasan, lihat informasi Large Producer di atas. Jika produser Anda adalah browser, jumlah data yang dihasilkan biasanya sangat kecil. Namun, Anda menempatkanmenempatkanoperasi di jalur kritis aplikasi, yang tidak kami rekomendasikan.

Kesalahan izin kunci utama KMS

Kesalahan ini terjadi ketika aplikasi produser menulis ke aliran terenkripsi tanpa izin pada kunci master KMS. Untuk menetapkan izin ke aplikasi untuk mengakses kunci KMS, lihatMenggunakan Kebijakan Kunci diAWSKMSdanMenggunakan Kebijakan IAMAWSKMS.

Masalah umum, pertanyaan, dan ide pemecahan masalah bagi produsen

- Mengapa aliran data Kinesis saya mengembalikan Kesalahan Server Internal 500?
- Bagaimana cara memecahkan masalah kesalahan batas waktu saat menulis dari Flink ke Kinesis Data Streams?
- Bagaimana cara memecahkan masalah kesalahan throttling di Kinesis Data Streams?
- Mengapa throttling aliran data Kinesis saya?
- Bagaimana cara memasukkan data data ke dalam aliran data Kinesis menggunakan KPL?

Topik Tingkat Lanjut untuk Produsen Kinesis Data Streams

Bagian ini membahas cara mengoptimalkan produsen Amazon Kinesis Data Streams Anda.

Topik

KPL Coba Ulang dan Pembatasan Tingkat

Saat Anda menambahkan catatan pengguna Kinesis Producer Library (KPL) menggunakan KPL`addUserRecord()`operasi, catatan diberikan cap waktu dan ditambahkan ke buffer dengan tenggat waktu yang ditetapkan oleh`RecordMaxBufferedTime`parameter konfigurasi Kombinasi stamp/tenggat waktu ini menetapkan prioritas penyangga. Catatan memerah dari buffer berdasarkan kriteria berikut:

- Prioritas buffer
- Konfigurasi agregasi
- Konfigurasi koleksi

Parameter konfigurasi agregasi dan koleksi yang mempengaruhi perilaku penyangga adalah sebagai berikut:

- `AggregationMaxCount`
- `AggregationMaxSize`
- `CollectionMaxCount`
- `CollectionMaxSize`

Rekaman yang disiram kemudian dikirim ke aliran data Kinesis Anda sebagai catatan Amazon Kinesis Data Streams menggunakan panggilan ke operasi API Kinesis Data Streams`PutRecords`. Parameter`PutRecords`operasi mengirimkan permintaan ke aliran Anda yang kadang-kadang menunjukkan kegagalan penuh atau sebagian. Rekaman yang gagal secara otomatis ditambahkan kembali ke buffer KPL. Batas waktu baru ditetapkan berdasarkan minimum dua nilai ini:

- Setengah arus`RecordMaxBufferedTime`konfigurasi
- Rekortime-to-livenilai

Strategi ini memungkinkan rekaman pengguna KPL yang dicoba ulang untuk dimasukkan dalam panggilan API Kinesis Data Streams berikutnya, untuk meningkatkan throughput dan mengurangi kompleksitas sambil menegakkan rekaman Kinesis Data Streamstime-to-livenilai Tidak ada algoritma

backoff, membuat ini strategi coba ulang yang relatif agresif. Spamming karena retries berlebihan dicegah dengan membatasi tingkat, dibahas di bagian berikutnya.

Pembatasan Laju

KPL menyertakan fitur pembatas tarif, yang membatasi throughput per-shard yang dikirim dari satu produsen. Pembatasan tingkat diimplementasikan menggunakan algoritma bucket token dengan ember terpisah untuk catatan Kinesis Data Streams dan byte. Setiap menulis sukses ke aliran data Kinesis menambahkan token (atau beberapa token) ke setiap bucket, hingga ambang batas tertentu. Ambang batas ini dapat dikonfigurasi tetapi secara default ditetapkan 50 persen lebih tinggi dari batas pecahan yang sebenarnya, untuk memungkinkan saturasi pecahan dari satu produsen.

Anda dapat menurunkan batas ini untuk mengurangi spamming karena percobaan ulang yang berlebihan. Namun, praktik terbaik adalah agar setiap produsen mencoba lagi untuk throughput maksimum secara agresif dan menangani setiap throttling yang dihasilkan ditentukan sebagai berlebihan dengan memperluas kapasitas aliran dan menerapkan strategi kunci partisi yang sesuai.

Pertimbangan Saat Menggunakan Agregasi KPL

Meskipun skema nomor urut dari data Amazon Kinesis Data Streams yang dihasilkan tetap sama, agregasi menyebabkan pengindeksan data pengguna Kinesis Producer Library (KPL) yang terkandung dalam rekaman Kinesis Data Streams agregat dimulai dari 0 (nol); namun, selama Anda tidak bergantung pada urutan angka-angka untuk mengidentifikasi catatan pengguna KPL Anda secara unik, kode Anda dapat mengabaikan ini, karena agregasi (catatan pengguna KPL Anda ke dalam rekaman Kinesis Data Streams) dan de-agregasi berikutnya (dari data Kinesis Data Streams ke dalam catatan pengguna KPL Anda) secara otomatis menangani hal ini untuk Anda. Hal ini berlaku apakah konsumen Anda menggunakan KCL atauAWSSDK Untuk menggunakan fungsionalitas agregasi ini, Anda harus menarik bagian Java dari KPL ke dalam build Anda jika konsumen Anda ditulis menggunakan API yang disediakan diAWSSDK

Jika Anda berniat untuk menggunakan nomor urut sebagai pengidentifikasi unik untuk catatan pengguna KPL Anda, kami sarankan Anda menggunakan kontrak yang taat`public int hashCode()`dan`public boolean equals(Object obj)`operasi yang disediakan dalam`Record`dan`UserRecord`untuk mengaktifkan perbandingan catatan pengguna KPL Anda. Selain itu, jika Anda ingin memeriksa nomor subsequence dari catatan pengguna KPL Anda, Anda dapat melemparkannya ke`UserRecord`contoh dan mengambil nomor subsequence nya.

Untuk informasi selengkapnya, lihat De-agregasi Konsumen.

Membaca Data dari Amazon Kinesis Data Streams

Konsumen adalah aplikasi yang memproses semua data dari aliran data Kinesis. Ketika konsumen menggunakan fan-out yang ditingkatkan, ia mendapatkan peruntukan throughput baca 2 MB/detik, memungkinkan banyak konsumen untuk membaca data dari aliran yang sama secara paralel, tanpa bersaing untuk throughput baca dengan konsumen lain. Untuk menggunakan kemampuan fan-out yang ditingkatkan dari pecahan, lihat. Mengembangkan Konsumen Khusus dengan Throughput Khusus (Peningkatan Fan-Out)

Secara default, pecahan dalam aliran menyediakan 2 MB/detik throughput baca per pecahan. Throughput ini dibagikan ke semua konsumen yang membaca dari pecahan tertentu. Dengan kata lain, default 2 MB/detik throughput per shard adalah tetap, bahkan jika ada beberapa konsumen yang membaca dari pecahan. Untuk menggunakan throughput default pecahan ini, lihat,. Mengembangkan Konsumen Khusus dengan Throughput Bersama

Tabel berikut membandingkan throughput default dengan fan-out yang ditingkatkan. Penundaan propagasi pesan didefinisikan sebagai waktu yang dibutuhkan dalam milidetik untuk muatan yang dikirim menggunakan API pengiriman muatan (seperti PutRecord dan PutRecords) untuk menjangkau aplikasi konsumen melalui API yang memakan muatan (seperti dan). GetRecords SubscribeToShard

Karakteristik	Konsumen Tidak Terdaftar tanpa Peningkatan Fan-Out	Konsumen Terdaftar dengan Peningkatan Fan-Out
Throughput Baca Shard	Diperbaiki pada total 2 MB/detik per pecahan. Jika ada banyak konsumen yang membaca dari pecahan yang sama, mereka semua berbagi throughput ini. Jumlah throughput yang mereka terima dari pecahan tidak melebihi 2 MB/detik.	Timbangan saat konsumen mendaftar untuk menggunakan fan-out yang disempurnakan. Setiap konsumen yang terdaftar untuk menggunakan fan-out yang ditingkatkan menerima throughpu t baca sendiri per pecahan, hingga 2 MB/detik, terlepas dari konsumen lain.
Penundaan propagasi pesan	Rata-rata sekitar 200 ms jika Anda memiliki satu konsumen yang membaca dari aliran. Rata-rata ini	Biasanya rata-rata 70 ms apakah Anda memiliki satu konsumen atau lima konsumen.

Karakteristik	Konsumen Tidak Terdaftar tanpa Peningkatan Fan-Out	Konsumen Terdaftar dengan Peningkatan Fan-Out
	naik menjadi sekitar 1000 ms jika Anda memiliki lima konsumen.	
Biaya	N/A	Ada biaya pengambilan data dan biaya jam pecahan konsumen. Untuk informasi selengkapnya, lihat Harga Amazon Kinesis Data Streams.
Model pengiriman catatan	Tarik model melalui HTTP menggunakan GetRecords.	Kinesis Data Streams mendorong catatan kepada Anda melalui HTTP/2 menggunakan. SubscribeToShard

Topik

- Menggunakan Penampil Data di Konsol Kinesis
- Mengkueri aliran data Anda di Konsol Kinesis
- Mengembangkan Konsumen Menggunakan AWS Lambda
- Mengembangkan Konsumen Menggunakan Amazon Managed Service untuk Apache Flink
- Mengembangkan Konsumen Menggunakan Amazon Data Firehose
- Menggunakan Perpustakaan Klien Kinesis
- Mengembangkan Konsumen Khusus dengan Throughput Bersama
- Mengembangkan Konsumen Khusus dengan Throughput Khusus (Peningkatan Fan-Out)
- Migrasi Konsumen dari KCL 1.x ke KCL 2.x
- Menggunakan AWS Layanan lain untuk membaca data dari Kinesis Data Streams
- Menggunakan integrasi pihak ketiga
- Pemecahan Masalah Kinesis Data Streams Konsumen
- Topik Lanjutan untuk Konsumen Amazon Kinesis Data Streams

Menggunakan Penampil Data di Konsol Kinesis

Penampil Data di Konsol Manajemen Kinesis memungkinkan Anda untuk melihat catatan data dalam pecahan tertentu dari aliran data Anda tanpa harus mengembangkan aplikasi konsumen. Untuk menggunakan Penampil Data, ikuti langkah-langkah berikut:

1. Masuk ke AWS Management Console dan buka konsol Kinesis di https://console.aws.amazon.com/kinesis.
2. Pilih aliran data aktif yang catatannya ingin Anda lihat dengan Penampil Data dan kemudian pilih tab Penampil Data.
3. Di tab Penampil Data untuk aliran data aktif yang dipilih, pilih pecahan yang catatannya ingin Anda lihat, pilih Posisi Awal, lalu klik Dapatkan catatan. Anda dapat mengatur posisi awal ke salah satu nilai berikut:

   - Pada nomor urut: Tampilkan catatan dari posisi yang dilambangkan dengan nomor urut yang ditentukan dalam bidang nomor urut.
   - Setelah nomor urut: Tampilkan catatan tepat setelah posisi dilambangkan dengan nomor urut yang ditentukan dalam bidang nomor urut.
   - Pada stempel waktu: Tampilkan catatan dari posisi yang dilambangkan dengan cap waktu yang ditentukan dalam bidang stempel waktu.
   - Potong cakrawala: Tampilkan catatan pada catatan terakhir yang belum dipangkas di pecahan, yang merupakan catatan data tertua di pecahan.
   - Terbaru: Tampilkan catatan tepat setelah catatan terbaru di pecahan, sehingga Anda selalu membaca data terbaru di pecahan.

   Catatan data yang dihasilkan yang cocok dengan ID pecahan yang ditentukan dan posisi awal kemudian ditampilkan dalam tabel catatan di konsol. Maksimal 50 catatan ditampilkan pada satu waktu. Untuk melihat kumpulan catatan berikutnya, klik tombol Berikutnya.
4. Klik setiap catatan individu untuk melihat muatan rekaman itu dalam data mentah atau format JSON di jendela terpisah.

Perhatikan bahwa saat Anda mengklik tombol Dapatkan catatan atau Berikutnya di Penampil Data, ini akan memanggil GetRecordsAPI dan ini berlaku terhadap batas GetRecordsAPI 5 transaksi per detik.

Mengkueri aliran data Anda di Konsol Kinesis

Tab Analisis Data di Konsol Aliran Data Kinesis memungkinkan Anda melakukan kueri aliran data menggunakan SQL. Untuk menggunakan kemampuan ini, ikuti langkah-langkah berikut:

1. Masuk ke AWS Management Console dan buka konsol Kinesis di https://console.aws.amazon.com/kinesis.
2. Pilih aliran data aktif yang ingin Anda kueri dengan SQL dan kemudian pilih tab Analisis data.
3. Di tab Analisis data, Anda dapat melakukan inspeksi dan visualisasi aliran dengan notebook Managed Apache Flink Studio. Anda dapat melakukan kueri SQL ad-hoc untuk memeriksa aliran data Anda dan melihat hasil dalam hitungan detik menggunakan Apache Zeppelin. Di tab Analisis data, pilih Saya setuju dan kemudian pilih Buat buku catatan untuk membuat buku catatan.
4. Setelah notebook dibuat, pilih Buka di Apache Zeppelin. Ini akan membuka buku catatan Anda di tab baru. Notebook adalah antarmuka interaktif tempat Anda dapat mengirimkan kueri SQL Anda. Pilih catatan yang berisi nama aliran Anda.
5. Anda akan melihat catatan dengan `SELECT` kueri sampel untuk menampilkan data dalam aliran yang sudah berjalan. Ini memungkinkan Anda melihat skema untuk aliran data Anda.
6. Untuk mencoba kueri lain seperti jendela jatuh atau geser, pilih Lihat contoh kueri di tab Analisis data. Salin kueri, ubah agar sesuai dengan skema aliran data Anda, lalu jalankan dalam paragraf baru di catatan Zeppelin Anda.

Mengembangkan Konsumen Menggunakan AWS Lambda

Anda dapat menggunakan AWS Lambda fungsi untuk memproses catatan dalam aliran data. AWS Lambda adalah layanan komputasi yang memungkinkan Anda menjalankan kode tanpa menyediakan atau mengelola server. Ini mengeksekusi kode Anda hanya bila diperlukan dan skala secara otomatis, dari beberapa permintaan per hari hingga ribuan per detik. Anda hanya membayar untuk waktu komputasi yang Anda konsumsi. Tidak ada biaya ketika kode Anda tidak berjalan. Dengan AWS Lambda, Anda dapat menjalankan kode untuk hampir semua jenis aplikasi atau layanan backend, semua dengan administrasi nol. Ini menjalankan kode Anda pada infrastruktur komputasi ketersediaan tinggi dan melakukan semua administrasi sumber daya komputasi, termasuk pemeliharaan server dan sistem operasi, penyediaan kapasitas dan penskalaan otomatis, pemantauan kode, dan pencatatan. Untuk informasi selengkapnya, lihat Menggunakan AWS Lambda dengan Amazon Kinesis.

Untuk informasi pemecahan masalah, lihat Mengapa pemicu Kinesis Data Streams tidak dapat menjalankan fungsi Lambda saya?

Mengembangkan Konsumen Menggunakan Amazon Managed Service untuk Apache Flink

Anda dapat menggunakan Amazon Managed Service untuk aplikasi Apache Flink untuk memproses dan menganalisis data dalam aliran Kinesis menggunakan SQL, Java, atau Scala. Layanan Terkelola untuk aplikasi Apache Flink dapat memperkaya data menggunakan sumber referensi, mengumpulkan data dari waktu ke waktu, atau menggunakan pembelajaran mesin untuk menemukan anomali data. Kemudian Anda dapat menulis hasil analisis ke aliran Kinesis lain, aliran pengiriman Firehose, atau fungsi Lambda. Untuk informasi selengkapnya, lihat Managed Service for Apache Flink Developer Guide for SQL Applications atau Managed Service for Apache Flink Developer Guide for Flink Applications.

Mengembangkan Konsumen Menggunakan Amazon Data Firehose

Anda dapat menggunakan Firehose untuk membaca dan memproses catatan dari aliran Kinesis. Firehose adalah layanan yang dikelola sepenuhnya untuk mengirimkan data streaming real-time ke tujuan seperti Amazon S3, Amazon Redshift, Amazon OpenSearch Service, dan Splunk. Firehose juga mendukung endpoint HTTP kustom atau endpoint HTTP yang dimiliki oleh penyedia layanan pihak ketiga yang didukung, termasuk Datadog, MongoDB, dan New Relic. Anda juga dapat mengonfigurasi Firehose untuk mengubah catatan data Anda dan mengonversi format rekaman sebelum mengirimkan data ke tujuannya. Untuk informasi selengkapnya, lihat Menulis ke Firehose Menggunakan Kinesis Data Streams.

Menggunakan Perpustakaan Klien Kinesis

Salah satu metode pengembangan aplikasi konsumen kustom yang dapat memproses data dari aliran data KDS adalah dengan menggunakan Kinesis Client Library (KCL).

Topik

- Apa itu Perpustakaan Klien Kinesis?
- Versi Tersedia KCL
- Konsep KCL
- Menggunakan Meja Sewa untuk Melacak Pecahan yang Diproses oleh Aplikasi Konsumen KCL

- Memproses Beberapa Aliran Data dengan KCL 2.x yang sama untuk Aplikasi Konsumen Java
- Menggunakan Perpustakaan Klien Kinesis dengan Registri Skema AWS Glue

> **Note**
>
> Untuk KCL 1.x dan KCL 2.x, Anda disarankan untuk meningkatkan ke versi KCL 1.x terbaru atau versi KCL 2.x, tergantung pada skenario penggunaan Anda. Baik KCL 1.x dan KCL 2.x diperbarui secara berkala dengan rilis yang lebih baru yang mencakup dependensi dan patch keamanan terbaru, perbaikan bug, dan fitur baru yang kompatibel ke belakang. Untuk informasi lebih lanjut, lihat https://github.com/awslabs/ amazon-kinesis-client /releases.

Apa itu Perpustakaan Klien Kinesis?

KCL membantu Anda mengkonsumsi dan memproses data dari aliran data Kinesis dengan menangani banyak tugas kompleks yang terkait dengan komputasi terdistribusi. Ini termasuk load balancing di beberapa instance aplikasi konsumen, menanggapi kegagalan instans aplikasi konsumen, memeriksa catatan yang diproses, dan bereaksi terhadap resharding. KCL menangani semua subtugas ini sehingga Anda dapat memfokuskan upaya Anda untuk menulis logika pemrosesan catatan khusus Anda.

KCL berbeda dari Kinesis Data Streams API yang tersedia di SDK. AWS Kinesis Data Streams API membantu Anda mengelola banyak aspek Kinesis Data Streams, termasuk membuat stream, resharding, dan menempatkan serta mendapatkan rekaman. KCL menyediakan lapisan abstraksi di sekitar semua subtugas ini, khususnya sehingga Anda dapat fokus pada logika pemrosesan data kustom aplikasi konsumen Anda. Untuk informasi tentang Kinesis Data Streams API, lihat Referensi API Amazon Kinesis.

> **Important**
>
> KCL adalah perpustakaan Java. Support untuk bahasa selain Java disediakan menggunakan antarmuka multi-bahasa yang disebut. MultiLangDaemon Daemon ini berbasis Java dan berjalan di latar belakang saat Anda menggunakan bahasa KCL selain Java. Misalnya, jika Anda menginstal KCL untuk Python dan menulis aplikasi konsumen Anda sepenuhnya dengan Python, Anda masih memerlukan Java diinstal pada sistem Anda karena itu. MultiLangDaemon Selanjutnya, MultiLangDaemon memiliki beberapa pengaturan default yang mungkin perlu Anda sesuaikan untuk kasus penggunaan Anda, misalnya, AWS wilayah

yang terhubung dengannya. Untuk informasi selengkapnya tentang MultiLangDaemon on GitHub, lihat MultiLangDaemon proyek KCL.

KCL bertindak sebagai perantara antara logika pemrosesan rekaman Anda dan Kinesis Data Streams. KCL melakukan tugas-tugas berikut:

- Terhubung ke aliran data
- Menghitung pecahan dalam aliran data
- Menggunakan sewa untuk mengoordinasikan asosiasi pecahan dengan pekerjanya
- Membuat instance pemroses rekaman untuk setiap pecahan yang dikelolanya
- Menarik catatan data dari aliran data
- Mendorong rekaman ke pemroses rekaman yang sesuai
- Catatan yang diproses di pos pemeriksaan
- Menyeimbangkan asosiasi shard-worker (leases) saat jumlah instans pekerja berubah atau saat aliran data di-sharded (pecahan dibagi atau digabungkan)

Versi Tersedia KCL

Saat ini, Anda dapat menggunakan salah satu dari versi KCL yang didukung berikut ini untuk membangun aplikasi konsumen kustom Anda:

- KCL 1.x

  Lihat informasi yang lebih lengkap di Mengembangkan Konsumen KCL 1.x
- KCL 2.x

  Lihat informasi yang lebih lengkap di Mengembangkan Konsumen KCL 2.x

Anda dapat menggunakan KCL 1.x atau KCL 2.x untuk membangun aplikasi konsumen yang menggunakan throughput bersama. Untuk informasi selengkapnya, lihat Mengembangkan Konsumen Kustom dengan Throughput Bersama Menggunakan KCL.

Untuk membangun aplikasi konsumen yang menggunakan throughput khusus (konsumen fan-out yang disempurnakan), Anda hanya dapat menggunakan KCL 2.x. Untuk informasi selengkapnya, lihat Mengembangkan Konsumen Khusus dengan Throughput Khusus (Peningkatan Fan-Out).

Untuk informasi tentang perbedaan antara KCL 1.x dan KCL 2.x, dan petunjuk tentang cara bermigrasi dari KCL 1.x ke KCL 2.x, lihat. Migrasi Konsumen dari KCL 1.x ke KCL 2.x

Konsep KCL

- Aplikasi konsumen KCL — aplikasi yang dibuat khusus menggunakan KCL dan dirancang untuk membaca dan memproses catatan dari aliran data.
- Contoh aplikasi konsumen - Aplikasi konsumen KCL biasanya didistribusikan, dengan satu atau lebih instance aplikasi berjalan secara bersamaan untuk mengoordinasikan kegagalan dan pemrosesan catatan data keseimbangan beban secara dinamis.
- Worker — kelas tingkat tinggi yang digunakan instance aplikasi konsumen KCL untuk mulai memproses data.

> **Important**
>
> Setiap instance aplikasi konsumen KCL memiliki satu pekerja.

Pekerja menginisialisasi dan mengawasi berbagai tugas, termasuk menyinkronkan informasi pecahan dan sewa, melacak tugas pecahan, dan memproses data dari pecahan. Seorang pekerja memberi KCL informasi konfigurasi untuk aplikasi konsumen, seperti nama aliran data yang datanya mencatat aplikasi konsumen KCL ini akan diproses dan AWS kredensyal yang diperlukan untuk mengakses aliran data ini. Pekerja juga memulai instance aplikasi konsumen KCL tertentu untuk mengirimkan catatan data dari aliran data ke prosesor rekaman.

> **Important**
>
> Dalam KCL 1.x kelas ini disebut Worker. Untuk informasi lebih lanjut, (ini adalah repositori Java KCL), lihat https://github.com/awslabs/ /blob/v1.x/src/main/java/com/amazonaws/services/kinesis/clientlibrary/lib/worker/worker.java. amazon-kinesis-client Dalam KCL 2.x, kelas ini disebut Scheduler. Tujuan Scheduler di KCL 2.x identik dengan tujuan Pekerja di KCL 1.x. Untuk informasi selengkapnya tentang kelas Scheduler di KCL 2.x, lihat https://github.com/awslabs/ amazon-kinesis-client /blob/master/ /src/main/java/software/amazon/kinesis/coordinator/Scheduler.java. amazon-kinesis-client

- Sewa — data yang mendefinisikan pengikatan antara pekerja dan pecahan. Aplikasi konsumen KCL terdistribusi menggunakan sewa untuk mempartisi pemrosesan catatan data di seluruh

armada pekerja. Pada waktu tertentu, setiap pecahan catatan data terikat pada pekerja tertentu dengan sewa yang diidentifikasi oleh variabel LeaseKey.

Secara default, seorang pekerja dapat memegang satu atau lebih sewa (tunduk pada nilai variabel maxLeasesForWorker) pada saat yang sama.

> **Important**
>
> Setiap pekerja akan bersaing untuk memegang semua sewa yang tersedia untuk semua pecahan yang tersedia dalam aliran data. Tetapi hanya satu pekerja yang akan berhasil memegang setiap sewa pada satu waktu.

Misalnya, jika Anda memiliki instance aplikasi konsumen A dengan pekerja A yang memproses aliran data dengan 4 pecahan, pekerja A dapat menyimpan sewa ke pecahan 1, 2, 3, dan 4 secara bersamaan. Tetapi jika Anda memiliki dua instance aplikasi konsumen: A dan B dengan pekerja A dan pekerja B, dan instance ini memproses aliran data dengan 4 pecahan, pekerja A dan pekerja B tidak dapat menahan sewa untuk shard 1 secara bersamaan. Seorang pekerja memegang sewa ke pecahan tertentu sampai siap untuk berhenti memproses catatan data pecahan ini atau sampai gagal. Ketika satu pekerja berhenti memegang sewa, pekerja lain mengambil dan memegang sewa.

Untuk informasi lebih lanjut, (ini adalah repositori Java KCL), lihat https://github.com/awslabs/amazon-kinesis-client /blob/v1.x/src/main/java/com/amazonaws/services/kinesis/leases/impl/lease.java untuk KCL 1.x dan https://github.com/awslabs/ /blob/master/ /src/main/java/software/amazon/kinesis/leases/Lease.java untuk KCL 2.x. amazon-kinesis-client amazon-kinesis-client

- Tabel sewa - tabel Amazon DynamoDB unik yang digunakan untuk melacak pecahan dalam aliran data KDS yang disewakan dan diproses oleh pekerja aplikasi konsumen KCL. Tabel sewa harus tetap sinkron (di dalam pekerja dan di semua pekerja) dengan informasi pecahan terbaru dari aliran data saat aplikasi konsumen KCL berjalan. Untuk informasi selengkapnya, lihat Menggunakan Meja Sewa untuk Melacak Pecahan yang Diproses oleh Aplikasi Konsumen KCL.
- Rekam prosesor — logika yang mendefinisikan bagaimana aplikasi konsumen KCL Anda memproses data yang didapatnya dari aliran data. Saat runtime, instance aplikasi konsumen KCL membuat instance pekerja, dan pekerja ini membuat instance satu prosesor rekaman untuk setiap pecahan yang disewanya.

Menggunakan Meja Sewa untuk Melacak Pecahan yang Diproses oleh Aplikasi Konsumen KCL

Topik

Apa itu Meja Sewa

Untuk setiap aplikasi Amazon Kinesis Data Streams, KCL menggunakan tabel sewa unik (disimpan dalam tabel Amazon DynamoDB) untuk melacak pecahan dalam aliran data KDS yang disewakan dan diproses oleh pekerja aplikasi konsumen KCL.

> **Important**
>
> KCL menggunakan nama aplikasi konsumen untuk membuat nama tabel sewa yang digunakan aplikasi konsumen ini, oleh karena itu setiap nama aplikasi konsumen harus unik.

Anda dapat melihat tabel sewa menggunakan konsol Amazon DynamoDB saat aplikasi konsumen sedang berjalan.

Jika tabel sewa untuk aplikasi konsumen KCL Anda tidak ada saat aplikasi dimulai, salah satu pekerja membuat tabel sewa untuk aplikasi ini.

> **Important**
>
> Akun Anda dikenakan biaya untuk biaya yang terkait dengan tabel DynamoDB, selain biaya yang terkait dengan Kinesis Data Streams itu sendiri.

Setiap baris dalam tabel sewa mewakili pecahan yang sedang diproses oleh pekerja aplikasi konsumen Anda. Jika aplikasi konsumen KCL Anda hanya memproses satu aliran data`leaseKey`, maka kunci hash untuk tabel sewa adalah ID pecahan. Jika yaMemproses Beberapa Aliran Data dengan KCL 2.x yang sama untuk Aplikasi Konsumen Java, maka struktur LeaseKey terlihat

seperti ini:. `account-id:StreamName:streamCreationTimestamp:ShardId` Misalnya, `111111111:multiStreamTest-1:12345:shardId-000000000336`.

Selain ID pecahan, setiap baris juga menyertakan data berikut:

- pos pemeriksaan: Nomor urutan pos pemeriksaan terbaru untuk pecahan. Nilai ini unik di semua pecahan dalam aliran data.
- checkpointSubSequenceNomor: Saat menggunakan fitur agregasi Perpustakaan Produser Kinesis, ini adalah ekstensi ke pos pemeriksaan yang melacak catatan pengguna individu dalam catatan Kinesis.
- LeaseCounter: Digunakan untuk pembuatan versi sewa sehingga pekerja dapat mendeteksi bahwa sewa mereka telah diambil oleh pekerja lain.
- LeaseKey: Pengidentifikasi unik untuk sewa. Setiap sewa khusus untuk pecahan dalam aliran data dan dipegang oleh satu pekerja pada satu waktu.
- LeaseOwner: Pekerja yang memegang sewa ini.
- ownerSwitchesSincePos pemeriksaan: Berapa kali sewa ini telah berganti pekerja sejak terakhir kali pos pemeriksaan ditulis.
- parentShardId: Digunakan untuk memastikan bahwa pecahan induk diproses sepenuhnya sebelum pemrosesan dimulai pada pecahan anak. Ini memastikan bahwa catatan diproses dalam urutan yang sama dengan yang dimasukkan ke dalam aliran.
- hashrange: Digunakan oleh `PeriodicShardSyncManager` untuk menjalankan sinkronisasi berkala untuk menemukan pecahan yang hilang di tabel sewa dan membuat sewa untuk mereka jika diperlukan.

> **Note**
>
> Data ini hadir dalam tabel sewa untuk setiap pecahan dimulai dengan KCL 1.14 dan KCL 2.3. Untuk informasi lebih lanjut tentang `PeriodicShardSyncManager` dan sinkronisasi berkala antara sewa dan pecahan, lihat. Bagaimana Tabel Sewa Disinkronkan dengan Pecahan dalam Aliran Data KDS

- childshards: Digunakan oleh `LeaseCleanupManager` untuk meninjau status pemrosesan pecahan anak dan memutuskan apakah pecahan induk dapat dihapus dari tabel sewa.

> **Note**
>
> Data ini hadir dalam tabel sewa untuk setiap pecahan dimulai dengan KCL 1.14 dan KCL 2.3.

- ShardID: ID pecahan.

> **Note**
>
> Data ini hanya ada di tabel sewa jika AndaMemproses Beberapa Aliran Data dengan KCL 2.x yang sama untuk Aplikasi Konsumen Java. Ini hanya didukung di KCL 2.x untuk Java, dimulai dengan KCL 2.3 untuk Java dan seterusnya.

- nama aliran Pengidentifikasi aliran data dalam format berikut:`account-id:StreamName:streamCreationTimestamp`.

> **Note**
>
> Data ini hanya ada di tabel sewa jika AndaMemproses Beberapa Aliran Data dengan KCL 2.x yang sama untuk Aplikasi Konsumen Java. Ini hanya didukung di KCL 2.x untuk Java, dimulai dengan KCL 2.3 untuk Java dan seterusnya.

Throughput

Jika aplikasi Amazon Kinesis Data Streams menerima pengecualian throughput yang disediakan, Anda harus meningkatkan throughput yang disediakan untuk tabel DynamoDB. KCL membuat tabel dengan throughput yang disediakan 10 pembacaan per detik dan 10 penulisan per detik, tetapi ini mungkin tidak cukup untuk aplikasi Anda. Misalnya, jika aplikasi Amazon Kinesis Data Streams sering melakukan pemeriksaan atau beroperasi pada aliran yang terdiri dari banyak pecahan, Anda mungkin memerlukan lebih banyak throughput.

Untuk informasi tentang throughput yang disediakan di DynamoDB, lihat Mode Kapasitas Baca/Tulis dan Bekerja dengan Tabel dan Data di Panduan Pengembang Amazon DynamoDB.

Bagaimana Tabel Sewa Disinkronkan dengan Pecahan dalam Aliran Data KDS

Pekerja dalam aplikasi konsumen KCL menggunakan sewa untuk memproses pecahan dari aliran data tertentu. Informasi tentang pekerja apa yang menyewakan pecahan apa pada waktu tertentu disimpan dalam tabel sewa. Tabel sewa harus tetap sinkron dengan informasi pecahan terbaru dari aliran data saat aplikasi konsumen KCL berjalan. KCL menyinkronkan tabel sewa dengan informasi pecahan yang diperoleh dari layanan Kinesis Data Streams selama bootstrap aplikasi konsumen (baik ketika aplikasi konsumen diinisialisasi atau dimulai ulang) dan juga setiap kali pecahan yang sedang diproses mencapai akhir (resharding). Dengan kata lain, pekerja atau aplikasi konsumen KCL disinkronkan dengan aliran data yang mereka proses selama bootstrap aplikasi konsumen awal dan setiap kali aplikasi konsumen menemukan peristiwa reshard aliran data.

Topik

Sinkronisasi di KCL 1.0 - 1.13 dan KCL 2.0 - 2.2

Di KCL 1.0 - 1.13 dan KCL 2.0 - 2.2, selama bootstrap aplikasi konsumen dan juga selama setiap peristiwa reshard aliran data, KCL menyinkronkan tabel sewa dengan informasi pecahan yang diperoleh dari layanan Kinesis Data Streams dengan menjalankan atau API penemuan. `ListShards` `DescribeStream` Dalam semua versi KCL yang tercantum di atas, setiap pekerja aplikasi konsumen KCL menyelesaikan langkah-langkah berikut untuk melakukan proses sinkronisasi sewa/shard selama bootstrap aplikasi konsumen dan pada setiap acara reshard aliran:

- Mengambil semua pecahan untuk data aliran yang sedang diproses
- Mengambil semua sewa pecahan dari tabel sewa
- Menyaring setiap pecahan terbuka yang tidak memiliki sewa di tabel sewa
- Mengulangi semua pecahan terbuka yang ditemukan dan untuk setiap pecahan terbuka tanpa induk terbuka:
  - Melintasi pohon hierarki melalui jalur leluhurnya untuk menentukan apakah pecahan itu adalah keturunan. Pecahan dianggap sebagai keturunan, jika pecahan leluhur sedang diproses (entri sewa untuk pecahan leluhur ada di tabel sewa) atau jika pecahan leluhur harus diproses (misalnya, jika posisi awal adalah atau) `TRIM_HORIZON` `AT_TIMESTAMP`

- Jika pecahan terbuka dalam konteks adalah keturunan, KCL memeriksa pecahan berdasarkan posisi awal dan membuat sewa untuk orang tuanya, jika diperlukan

Sinkronisasi di KCL 2.x, Dimulai dengan KCL 2.3 dan Beyond

Dimulai dengan versi terbaru yang didukung dari KCL 2.x (KCL 2.3) dan seterusnya, perpustakaan sekarang mendukung perubahan berikut pada proses sinkronisasi. Perubahan sinkronisasi lease/shard ini secara signifikan mengurangi jumlah panggilan API yang dilakukan oleh aplikasi konsumen KCL ke layanan Kinesis Data Streams dan mengoptimalkan manajemen sewa di aplikasi konsumen KCL Anda.

- Selama bootstrap aplikasi, jika tabel sewa kosong, KCL menggunakan opsi pemfilteran `ListShard` API (parameter permintaan `ShardFilter` opsional) untuk mengambil dan membuat sewa hanya untuk snapshot pecahan yang terbuka pada waktu yang ditentukan oleh parameter. `ShardFilter` `ShardFilter`Parameter ini memungkinkan Anda untuk memfilter respons `ListShards` API. Satu-satunya properti yang diperlukan dari `ShardFilter` parameter adalah`Type`. KCL menggunakan properti `Type` filter dan berikut nilai validnya untuk mengidentifikasi dan mengembalikan snapshot pecahan terbuka yang mungkin memerlukan sewa baru:
  - `AT_TRIM_HORIZON`- Responsnya mencakup semua pecahan yang terbuka di`TRIM_HORIZON`.
  - `AT_LATEST`- Respons hanya mencakup pecahan aliran data yang saat ini terbuka.
  - `AT_TIMESTAMP`- respons mencakup semua pecahan yang stempel waktu awalnya kurang dari atau sama dengan stempel waktu yang diberikan dan stempel waktu akhir lebih besar dari atau sama dengan stempel waktu yang diberikan atau masih terbuka.

  `ShardFilter`digunakan saat membuat sewa untuk tabel sewa kosong untuk menginisialisasi sewa untuk snapshot pecahan yang ditentukan di. `RetrievalConfig#initialPositionInStreamExtended`

  Untuk informasi selengkapnya tentang `ShardFilter`, lihat https://docs.aws.amazon.com/kinesis/latest/APIReference/API_ShardFilter.html.
- Alih-alih semua pekerja melakukan sinkronisasi sewa/pecahan untuk menjaga tabel sewa tetap mutakhir dengan pecahan terbaru dalam aliran data, satu pemimpin pekerja terpilih melakukan sinkronisasi sewa/pecahan.
- KCL 2.3 menggunakan parameter `ChildShards` pengembalian `GetRecords` dan `SubscribeToShard` API untuk melakukan sinkronisasi sewa/pecahan yang terjadi pada

pecahan tertutup, memungkinkan pekerja KCL `SHARD_END` untuk hanya membuat sewa untuk pecahan anak dari pecahan yang selesai diproses. Untuk dibagikan di seluruh aplikasi konsumen, pengoptimalan sinkronisasi sewa/pecahan ini menggunakan parameter API. `ChildShards` `GetRecords` Untuk aplikasi konsumen throughput khusus (fan-out yang ditingkatkan), pengoptimalan sinkronisasi lease/shard ini menggunakan parameter API. `ChildShards` `SubscribeToShard` Lihat informasi selengkapnya di GetRecords, SubscribeToShards, dan ChildShard.

- Dengan perubahan di atas, perilaku KCL bergerak dari model semua pekerja yang belajar tentang semua pecahan yang ada ke model pekerja yang hanya belajar tentang pecahan pecahan anak-anak yang dimiliki setiap pekerja. Oleh karena itu, selain sinkronisasi yang terjadi selama bootstraping aplikasi konsumen dan peristiwa reshard, KCL sekarang juga melakukan pemindaian shard/lease berkala tambahan untuk mengidentifikasi lubang potensial dalam tabel sewa (dengan kata lain, untuk mempelajari semua pecahan baru) untuk memastikan rentang hash lengkap dari aliran data sedang diproses dan membuat sewa untuk mereka jika diperlukan. `PeriodicShardSyncManager`adalah komponen yang bertanggung jawab untuk menjalankan pemindaian lease/shard berkala.

  Untuk informasi lebih lanjut tentang `PeriodicShardSyncManager` di KCL 2.3, lihat https:// github.com/awslabs/ amazon-kinesis-client /blob/master/ /src/main/java/software/amazon/kinesis/ leases/ amazon-kinesis-client .java #L201 -L213. LeaseManagementConfig

  Di KCL 2.3, opsi konfigurasi baru tersedia untuk dikonfigurasi `PeriodicShardSyncManager` di`LeaseManagementConfig`:

Nama	Nilai default	Deskripsi
leasesRecoveryAuditorExecutionFrequencyMillis	120000 (2 menit)	Frekuensi (dalam millis) pekerjaan auditor untuk memindai sewa sebagian dalam tabel sewa. Jika auditor mendeteksi lubang apa pun dalam

Nama	Nilai default	Deskripsi
		sewa untuk aliran, maka itu akan memicu sinkronis asi pecahan berdasarkan. `leasesRecoveryAuditorInconsistencyConfidenceThreshold`

Nama	Nilai default	Deskripsi
leasesRecoveryAuditorInconsistencyConfidenceThreshold	3	Ambang batas kepercayaan untuk pekerjaan auditor periodik untuk menentukan apakah sewa untuk aliran data dalam tabel sewa tidak konsisten. Jika auditor menemukan kumpulan inkonsistensi yang sama secara berurutan untuk aliran data untuk ini berkali-kali, maka itu akan memicu sinkronisasi pecahan.

CloudWatch Metrik baru juga sekarang dipancarkan untuk memantau kesehatan. `PeriodicShardSyncManager` Untuk informasi selengkapnya, lihat PeriodicShardSyncManager.

- Termasuk optimasi `HierarchicalShardSyncer` untuk hanya membuat sewa untuk satu lapisan pecahan.

Sinkronisasi di KCL 1.x, Dimulai dengan KCL 1.14 dan Beyond

Dimulai dengan versi terbaru yang didukung dari KCL 1.x (KCL 1.14) dan seterusnya, perpustakaan sekarang mendukung perubahan berikut pada proses sinkronisasi. Perubahan sinkronisasi lease/

shard ini secara signifikan mengurangi jumlah panggilan API yang dilakukan oleh aplikasi konsumen KCL ke layanan Kinesis Data Streams dan mengoptimalkan manajemen sewa di aplikasi konsumen KCL Anda.

- Selama bootstrap aplikasi, jika tabel sewa kosong, KCL menggunakan opsi pemfilteran `ListShard` API (parameter permintaan `ShardFilter` opsional) untuk mengambil dan membuat sewa hanya untuk snapshot pecahan yang terbuka pada waktu yang ditentukan oleh parameter. `ShardFilter` `ShardFilter`Parameter ini memungkinkan Anda untuk memfilter respons `ListShards` API. Satu-satunya properti yang diperlukan dari `ShardFilter` parameter adalah`Type`. KCL menggunakan properti `Type` filter dan berikut nilai validnya untuk mengidentifikasi dan mengembalikan snapshot pecahan terbuka yang mungkin memerlukan sewa baru:
  - `AT_TRIM_HORIZON`- Responsnya mencakup semua pecahan yang terbuka di`TRIM_HORIZON`.
  - `AT_LATEST`- Respons hanya mencakup pecahan aliran data yang saat ini terbuka.
  - `AT_TIMESTAMP`- respons mencakup semua pecahan yang stempel waktu awalnya kurang dari atau sama dengan stempel waktu yang diberikan dan stempel waktu akhir lebih besar dari atau sama dengan stempel waktu yang diberikan atau masih terbuka.

  `ShardFilter`digunakan saat membuat sewa untuk tabel sewa kosong untuk menginisialisasi sewa untuk snapshot pecahan yang ditentukan di. `KinesisClientLibConfiguration#initialPositionInStreamExtended`

  Untuk informasi selengkapnya tentang `ShardFilter`, lihat https://docs.aws.amazon.com/kinesis/latest/APIReference/API_ShardFilter.html.
- Alih-alih semua pekerja melakukan sinkronisasi sewa/pecahan untuk menjaga tabel sewa tetap mutakhir dengan pecahan terbaru dalam aliran data, satu pemimpin pekerja terpilih melakukan sinkronisasi sewa/pecahan.
- KCL 1.14 menggunakan parameter `ChildShards` pengembalian `GetRecords` dan `SubscribeToShard` API untuk melakukan sinkronisasi sewa/pecahan yang terjadi pada pecahan tertutup, memungkinkan pekerja KCL `SHARD_END` untuk hanya membuat sewa untuk pecahan anak dari pecahan pecahan yang selesai diproses. Lihat informasi yang lebih lengkap di GetRecords dan ChildShard.
- Dengan perubahan di atas, perilaku KCL bergerak dari model semua pekerja yang belajar tentang semua pecahan yang ada ke model pekerja yang hanya belajar tentang pecahan pecahan anak-anak yang dimiliki setiap pekerja. Oleh karena itu, selain sinkronisasi yang terjadi selama bootstraping aplikasi konsumen dan peristiwa reshard, KCL sekarang juga melakukan

pemindaian shard/lease berkala tambahan untuk mengidentifikasi lubang potensial dalam tabel sewa (dengan kata lain, untuk mempelajari semua pecahan baru) untuk memastikan rentang hash lengkap dari aliran data sedang diproses dan membuat sewa untuk mereka jika diperlukan. `PeriodicShardSyncManager`adalah komponen yang bertanggung jawab untuk menjalankan pemindaian lease/shard berkala.

Kapan `KinesisClientLibConfiguration#shardSyncStrategyType` diatur ke`ShardSyncStrategyType.SHARD_END`, `PeriodicShardSync` `leasesRecoveryAuditorInconsistencyConfidenceThreshold` digunakan untuk menentukan ambang batas untuk jumlah pemindaian berturut-turut yang berisi lubang di tabel sewa setelah itu untuk menegakkan sinkronisasi pecahan. Kapan `KinesisClientLibConfiguration#shardSyncStrategyType` diatur ke`ShardSyncStrategyType.PERIODIC`, `leasesRecoveryAuditorInconsistencyConfidenceThreshold` diabaikan.

Untuk informasi lebih lanjut tentang `PeriodicShardSyncManager` di KCL 1.14, lihat https:// github.com/awslabs/ amazon-kinesis-client /blob/v1.x/src/main/java/com/amazonaws/services/ kinesis/clientlibrary/lib/worker/ KinesisClientLibConfiguration .java #L987 -L999.

Di KCL 1.14, opsi konfigurasi baru tersedia untuk dikonfigurasi `PeriodicShardSyncManager` di: `LeaseManagementConfig`

Nama	Nilai default	Deskripsi
leasesRecoveryAuditorInconsistencyConfidenceThreshold	3	Ambang batas kepercayaan untuk pekerjaan auditor periodik untuk menentukan apakah sewa untuk aliran data dalam tabel sewa tidak konsisten. Jika auditor menemukan kumpulan inkonsistensi yang sama secara berurutan untuk aliran data untuk ini berkali-kali, maka itu akan memicu sinkronisasi pecahan.

CloudWatch Metrik baru juga sekarang dipancarkan untuk memantau kesehatan. `PeriodicShardSyncManager` Untuk informasi selengkapnya, lihat PeriodicShardSyncManager.

- KCL 1.14 sekarang juga mendukung pembersihan sewa yang ditangguhkan. Sewa dihapus secara asinkron `LeaseCleanupManager` pada saat mencapai`SHARD_END`, ketika pecahan telah kedaluwarsa melewati periode retensi aliran data atau ditutup sebagai hasil dari operasi resharding.

  Opsi konfigurasi baru tersedia untuk dikonfigurasi`LeaseCleanupManager`.

Nama	Nilai default	Deskripsi
leaseClea nupIntervalMillis	1 menit	Interval untuk menjalank an thread pembersihan sewa.
completed LeaseClea nupIntervalMillis	5 menit	Interval untuk memeriksa apakah sewa selesai atau tidak.
garbageLe aseCleanu pIntervalMillis	30 menit	Interval untuk memeriksa apakah sewa adalah sampah (yaitu dipangkas melewati periode retensi aliran data) atau tidak.

- Termasuk optimasi `KinesisShardSyncer` untuk hanya membuat sewa untuk satu lapisan pecahan.

Memproses Beberapa Aliran Data dengan KCL 2.x yang sama untuk Aplikasi Konsumen Java

Bagian ini menjelaskan perubahan berikut dalam KCL 2.x untuk Java yang memungkinkan Anda membuat aplikasi konsumen KCL yang dapat memproses lebih dari satu aliran data pada saat yang bersamaan.

> **Important**
>
> Pemrosesan multistream hanya didukung di KCL 2.x untuk Java, dimulai dengan KCL 2.3 untuk Java dan seterusnya.
> Pemrosesan multistream TIDAK didukung untuk bahasa lain di mana KCL 2.x dapat diimplementasikan.
> Pemrosesan multistream TIDAK didukung dalam versi KCL 1.x apa pun.

- MultistreamTracker antarmuka

  Untuk membangun aplikasi konsumen yang dapat memproses beberapa aliran pada saat yang sama, Anda harus menerapkan antarmuka baru yang disebut MultistreamTracker. Antarmuka ini mencakup `streamConfigList` metode yang mengembalikan daftar aliran data dan konfigurasinya untuk diproses oleh aplikasi konsumen KCL. Perhatikan bahwa aliran data yang sedang diproses dapat diubah selama runtime aplikasi konsumen. `streamConfigList`disebut secara berkala oleh KCL untuk mempelajari tentang perubahan aliran data untuk diproses.

  `streamConfigList`Metode ini mengisi StreamConfigdaftar.

```
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;
}
```

  Perhatikan bahwa bidang `StreamIdentifier` dan `InitialPositionInStreamExtended` wajib, sementara `consumerArn` adalah opsional. Anda harus menyediakan `consumerArn` satu-satunya jika Anda menggunakan KCL 2.x untuk mengimplementasikan aplikasi konsumen fan-out yang disempurnakan.

Untuk informasi lebih lanjut tentang`StreamIdentifier`, lihat https://github.com/awslabs/ amazon-kinesis-client /blob/0c5042dadf794fe988438436252a5a8fe70b6b0b/amazon-kinesis-client /src/main/java/software/amazon/kinesis/common/ .java #L29. StreamIdentifier Anda dapat membuat instance multistream untuk `StreamIdentifier` dari pengenal aliran serial. Pengidentifikasi aliran serial harus dari format berikut:. `account-id:StreamName:streamCreationTimestamp`

```
 * @param streamIdentifierSer
    * @return StreamIdentifier
    */
   public static StreamIdentifier multiStreamInstance(String streamIdentifierSer) {
       if (PATTERN.matcher(streamIdentifierSer).matches()) {
           final String[] split = streamIdentifierSer.split(DELIMITER);
           return new StreamIdentifier(split[0], split[1],
Long.parseLong(split[2]));
       } else {
           throw new IllegalArgumentException("Unable to deserialize
StreamIdentifier from " + streamIdentifierSer);
       }
   }
```

`MultistreamTracker`juga mencakup strategi untuk menghapus sewa aliran lama di tabel sewa (). `formerStreamsLeasesDeletionStrategy` Perhatikan bahwa strategi TIDAK DAPAT diubah selama runtime aplikasi konsumen. Untuk informasi lebih lanjut, lihat https://github.com/awslabs/ amazon-kinesis-client /blob/0c5042dadf794fe988438436252a5a8fe70b6b0b/amazon-kinesis-client /src/main/java/software/amazon/kinesis/processor/ .java FormerStreamsLeasesDeletionStrategy

- ConfigsBuilderadalah kelas seluruh aplikasi yang dapat Anda gunakan untuk menentukan semua pengaturan konfigurasi KCL 2.x yang akan digunakan saat membangun aplikasi konsumen KCL Anda. `ConfigsBuilder`kelas sekarang memiliki dukungan untuk `MultistreamTracker` antarmuka. Anda dapat menginisialisasi ConfigsBuilder baik dengan nama satu aliran data untuk menggunakan catatan dari:

```
/**
    * 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;
    }
```

Atau Anda dapat menginisialisasi ConfigsBuilder dengan `MultiStreamTracker` jika Anda ingin mengimplementasikan aplikasi konsumen KCL yang memproses beberapa aliran secara bersamaan.

```

* 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;
    }
```

- Dengan dukungan multistream yang diterapkan untuk aplikasi konsumen KCL Anda, setiap baris tabel sewa aplikasi sekarang berisi ID pecahan dan nama aliran dari beberapa aliran data yang diproses aplikasi ini.
- Ketika dukungan multistream untuk aplikasi konsumen KCL Anda diimplementasikan, LeaseKey mengambil struktur berikut: `account-id:StreamName:streamCreationTimestamp:ShardId` Misalnya, `111111111:multiStreamTest-1:12345:shardId-000000000336`.

> **Important**
>
> Ketika aplikasi konsumen KCL Anda yang ada dikonfigurasi untuk memproses hanya satu aliran data, LeaseKey (yang merupakan kunci hash untuk tabel sewa) adalah ID pecahan. Jika Anda mengkonfigurasi ulang aplikasi konsumen KCL yang ada ini untuk memproses beberapa aliran data, itu merusak tabel sewa Anda, karena dengan dukungan multistream, struktur LeaseKey harus sebagai berikut:. `account-id:StreamName:StreamCreationTimestamp:ShardId`

Menggunakan Perpustakaan Klien Kinesis dengan Registri Skema AWS Glue

Anda dapat mengintegrasikan aliran data Kinesis Anda dengan registri skema AWS Glue. Registri skema AWS Glue memungkinkan Anda menemukan, mengontrol, dan mengembangkan skema secara terpusat, sambil memastikan data yang dihasilkan terus divalidasi oleh skema terdaftar. Sebuah skema mendefinisikan struktur dan format catatan data. Sebuah skema adalah sebuah spesifikasi berversi untuk publikasi data yang handal, konsumsi, atau penyimpanan. AWSGlue Schema Registry memungkinkan Anda untuk meningkatkan kualitas end-to-end data dan tata kelola

data dalam aplikasi streaming Anda. Untuk informasi selengkapnya, lihat AWSGlue Schema Registry. Salah satu cara untuk mengatur integrasi ini adalah melalui KCL di Jawa.

> **Important**
>
> Saat ini, integrasi registri skema Kinesis Data AWS Streams dan Glue hanya didukung untuk aliran data Kinesis yang menggunakan konsumen KCL 2.3 yang diterapkan di Jawa. Support multi-bahasa tidak tersedia. Konsumen KCL 1.0 tidak didukung. Konsumen KCL 2.x sebelum KCL 2.3 tidak didukung.

Untuk petunjuk terperinci tentang cara mengatur integrasi Kinesis Data Streams dengan Schema Registry menggunakan KCL, lihat bagian “Berinteraksi dengan Data Menggunakan Perpustakaan KPL/KCL” di Kasus Penggunaan: Mengintegrasikan Aliran Data Kinesis Amazon dengan Registri Skema Glue. AWS

Mengembangkan Konsumen Khusus dengan Throughput Bersama

Jika Anda tidak memerlukan throughput khusus saat menerima data dari Kinesis Data Streams, dan jika Anda tidak memerlukan penundaan propagasi baca di bawah 200 ms, Anda dapat membuat aplikasi konsumen seperti yang dijelaskan dalam topik berikut.

Topik

- Mengembangkan Konsumen Kustom dengan Throughput Bersama Menggunakan KCL
- Mengembangkan Konsumen Kustom dengan Throughput Bersama MenggunakanAWS SDK for Java

Untuk informasi tentang membangun konsumen yang dapat menerima catatan dari aliran data Kinesis dengan throughput khusus, lihatMengembangkan Konsumen Khusus dengan Throughput Khusus (Peningkatan Fan-Out).

Mengembangkan Konsumen Kustom dengan Throughput Bersama Menggunakan KCL

Salah satu metode pengembangan aplikasi konsumen kustom dengan throughput bersama adalah dengan menggunakan Kinesis Client Library (KCL).

Topik

- Mengembangkan Konsumen KCL 1.x
- Mengembangkan Konsumen KCL 2.x

Mengembangkan Konsumen KCL 1.x

Anda dapat mengembangkan aplikasi konsumen untuk Amazon Kinesis Data Streams menggunakan Kinesis Client Library (KCL). Untuk informasi lebih lanjut tentang KCL, lihatApa itu Perpustakaan Klien Kinesis? .

Daftar Isi

- Mengembangkan Konsumen Perpustakaan Klien Kinesis di Jawa
- Mengembangkan Konsumen Perpustakaan Klien Kinesis di Node.js
- Mengembangkan Konsumen Perpustakaan Klien Kinesis di .NET
- Mengembangkan Konsumen Perpustakaan Klien Kinesis dengan Python
- Mengembangkan Konsumen Perpustakaan Klien Kinesis di Ruby

Mengembangkan Konsumen Perpustakaan Klien Kinesis di Jawa

Anda dapat menggunakan Kinesis Client Library (KCL) untuk membangun aplikasi yang memproses data dari aliran data Kinesis Anda. Perpustakaan Klien Kinesis tersedia dalam berbagai bahasa. Topik ini membahas Java. Untuk melihat referensi Javadoc, lihat topik AWSJavadoc untuk Kelas. AmazonKinesisClient

Untuk mengunduh Java KCL dari GitHub, buka Perpustakaan Klien Kinesis (Java). Untuk menemukan Java KCL di Apache Maven, buka halaman hasil pencarian KCL. Untuk mengunduh kode sampel untuk aplikasi konsumen Java KCL dari GitHub, buka halaman proyek sampel KCL untuk Java. GitHub

Aplikasi sampel menggunakan Apache Commons Logging. Anda dapat mengubah konfigurasi logging dalam `configure` metode statis yang ditentukan dalam `AmazonKinesisApplicationSample.java` file. Untuk informasi selengkapnya tentang cara menggunakan Apache Commons Logging dengan aplikasi Log4j dan AWS Java, lihat Logging with Log4j di Panduan Pengembang. AWS SDK for Java

Anda harus menyelesaikan tugas-tugas berikut saat menerapkan aplikasi konsumen KCL di Jawa:

Tugas

- Menerapkan RecordProcessor Metode I
- Menerapkan Pabrik Kelas untuk RecordProcessor Antarmuka I
- Buat Pekerja
- Ubah Properti Konfigurasi
- Migrasi ke Versi 2 dari Antarmuka Prosesor Rekam

Menerapkan RecordProcessor Metode I

KCL saat ini mendukung dua versi antarmuka`IRecordProcessor`: Antarmuka asli tersedia dengan versi pertama KCL, dan versi 2 tersedia dimulai dengan KCL versi 1.5.0. Kedua antarmuka didukung penuh. Pilihan Anda tergantung pada persyaratan skenario spesifik Anda. Lihat Javadocs yang dibuat secara lokal atau kode sumber untuk melihat semua perbedaannya. Bagian berikut menguraikan implementasi minimal untuk memulai.

RecordProcessor Versi I

- Antarmuka Asli (Versi 1)
- Antarmuka Diperbarui (Versi 2)

Antarmuka Asli (Versi 1)

`IRecordProcessor`Antarmuka asli (`package com.amazonaws.services.kinesis.clientlibrary.interfaces`) memperlihatkan metode prosesor rekaman berikut yang harus diterapkan konsumen Anda. Sampel menyediakan implementasi yang dapat Anda gunakan sebagai titik awal (lihat`AmazonKinesisApplicationSampleRecordProcessor.java`).

```
public void initialize(String shardId)
public void processRecords(List<Record> records, IRecordProcessorCheckpointer
 checkpointer)
public void shutdown(IRecordProcessorCheckpointer checkpointer, ShutdownReason reason)
```

menginisialisasi

KCL memanggil `initialize` metode ketika prosesor rekaman dipakai, melewati ID pecahan tertentu sebagai parameter. Prosesor rekaman ini hanya memproses pecahan ini dan biasanya, kebalikannya juga benar (pecahan ini hanya diproses oleh prosesor rekaman ini). Namun, konsumen

Anda harus memperhitungkan kemungkinan bahwa catatan data dapat diproses lebih dari satu kali. Kinesis Data Streams memiliki semantik setidaknya sekali, artinya setiap catatan data dari pecahan diproses setidaknya satu kali oleh pekerja di konsumen Anda. Untuk informasi lebih lanjut tentang kasus di mana pecahan tertentu dapat diproses oleh lebih dari satu pekerja, lihatResharding, Scaling, dan Pengolahan Paralel.

```
public void initialize(String shardId)
```

processRecords

KCL memanggil `processRecords` metode, melewati daftar catatan data dari pecahan yang ditentukan oleh metode. `initialize(shardId)` Prosesor rekaman memproses data dalam catatan ini sesuai dengan semantik konsumen. Misalnya, pekerja mungkin melakukan transformasi pada data dan kemudian menyimpan hasilnya di bucket Amazon Simple Storage Service (Amazon S3).

```
public void processRecords(List<Record> records, IRecordProcessorCheckpointer
 checkpointer)
```

Selain data itu sendiri, catatan juga berisi nomor urut dan kunci partisi. Pekerja dapat menggunakan nilai-nilai ini saat memproses data. Misalnya, pekerja dapat memilih bucket S3 untuk menyimpan data berdasarkan nilai kunci partisi. `Record`Kelas mengekspos metode berikut yang menyediakan akses ke data catatan, nomor urut, dan kunci partisi.

```
record.getData()
record.getSequenceNumber()
record.getPartitionKey()
```

Dalam sampel, metode privat `processRecordsWithRetries` memiliki kode yang menunjukkan bagaimana seorang pekerja dapat mengakses data rekaman, nomor urut, dan kunci partisi.

Kinesis Data Streams membutuhkan prosesor rekaman untuk melacak catatan yang telah diproses dalam pecahan. KCL menangani pelacakan ini untuk Anda dengan meneruskan checkpointer () `IRecordProcessorCheckpointer` ke. `processRecords` Prosesor rekaman memanggil `checkpoint` metode pada antarmuka ini untuk menginformasikan KCL tentang seberapa jauh perkembangannya dalam memproses catatan di pecahan. Jika pekerja gagal, KCL menggunakan informasi ini untuk memulai kembali pemrosesan pecahan pada catatan diproses terakhir yang diketahui.

Untuk operasi split atau merge, KCL tidak akan mulai memproses pecahan baru sampai prosesor untuk pecahan asli dipanggil `checkpoint` untuk memberi sinyal bahwa semua pemrosesan pada pecahan asli selesai.

Jika Anda tidak melewati parameter, KCL mengasumsikan bahwa panggilan ke `checkpoint` berarti bahwa semua catatan telah diproses, hingga catatan terakhir yang diteruskan ke prosesor rekaman. Oleh karena itu, prosesor rekaman harus memanggil `checkpoint` hanya setelah memproses semua catatan dalam daftar yang diteruskan ke sana. Prosesor rekaman tidak perlu memanggil `checkpoint` setiap panggilan ke`processRecords`. Prosesor dapat, misalnya, memanggil `checkpoint` setiap panggilan ketiga ke`processRecords`. Anda dapat secara opsional menentukan nomor urut yang tepat dari catatan sebagai parameter untuk`checkpoint`. Dalam hal ini, KCL mengasumsikan bahwa semua catatan telah diproses hingga catatan itu saja.

Dalam sampel, metode pribadi `checkpoint` menunjukkan cara memanggil `IRecordProcessorCheckpointer.checkpoint` menggunakan penanganan pengecualian yang sesuai dan logika coba lagi.

KCL bergantung pada `processRecords` untuk menangani pengecualian apa pun yang timbul dari pemrosesan catatan data. Jika pengecualian dilemparkan`processRecords`, KCL melompati catatan data yang dilewatkan sebelum pengecualian. Artinya, catatan ini tidak dikirim kembali ke prosesor rekaman yang melemparkan pengecualian atau ke prosesor rekaman lainnya di konsumen.

penonaktifan

KCL memanggil `shutdown` metode baik saat pemrosesan berakhir (alasan shutdown adalah`TERMINATE`) atau pekerja tidak lagi merespons (alasan shutdown adalah). `ZOMBIE`

```
public void shutdown(IRecordProcessorCheckpointer checkpointer, ShutdownReason reason)
```

Pemrosesan berakhir ketika prosesor rekaman tidak menerima catatan lebih lanjut dari pecahan, karena pecahan dipecah atau digabungkan, atau aliran dihapus.

KCL juga meneruskan `IRecordProcessorCheckpointer` antarmuka ke`shutdown`. Jika alasan shutdown adalah`TERMINATE`, prosesor rekaman harus menyelesaikan pemrosesan catatan data apa pun, dan kemudian memanggil `checkpoint` metode pada antarmuka ini.

Antarmuka Diperbarui (Versi 2)

`IRecordProcessor`Antarmuka (`package com.amazonaws.services.kinesis.clientlibrary.interfaces.v2`) yang diperbarui memperlihatkan metode prosesor rekaman berikut yang harus diterapkan konsumen Anda:

```
void initialize(InitializationInput initializationInput)
void processRecords(ProcessRecordsInput processRecordsInput)
void shutdown(ShutdownInput shutdownInput)
```

Semua argumen dari versi asli antarmuka dapat diakses melalui metode get pada objek kontainer. Misalnya, untuk mengambil daftar catatan di`processRecords()`, Anda dapat menggunakan`processRecordsInput.getRecords()`.

Pada versi 2 antarmuka ini (KCL 1.5.0 dan yang lebih baru), input baru berikut tersedia selain input yang disediakan oleh antarmuka asli:

nomor urut awal

Dalam `InitializationInput` objek yang diteruskan ke `initialize()` operasi, nomor urut awal dari mana catatan akan diberikan ke instance prosesor rekaman. Ini adalah nomor urut yang terakhir diperiksa oleh instance prosesor rekaman yang sebelumnya memproses pecahan yang sama. Ini disediakan jika aplikasi Anda membutuhkan informasi ini.

nomor urut pos pemeriksaan yang tertunda

Dalam `InitializationInput` objek yang diteruskan ke `initialize()` operasi, nomor urutan pos pemeriksaan yang tertunda (jika ada) yang tidak dapat dilakukan sebelum instance prosesor rekaman sebelumnya berhenti.

Menerapkan Pabrik Kelas untuk RecordProcessor Antarmuka I

Anda juga perlu mengimplementasikan pabrik untuk kelas yang mengimplementasikan metode prosesor rekaman. Ketika konsumen Anda membuat instance pekerja, ia meneruskan referensi ke pabrik ini.

Sampel mengimplementasikan kelas pabrik dalam file `AmazonKinesisApplicationSampleRecordProcessorFactory.java` menggunakan antarmuka prosesor rekaman asli. Jika Anda ingin pabrik kelas membuat prosesor rekaman versi 2, gunakan nama paket`com.amazonaws.services.kinesis.clientlibrary.interfaces.v2`.

```
public class SampleRecordProcessorFactory implements IRecordProcessorFactory {
    /**
    * Constructor.
    */
    public SampleRecordProcessorFactory() {
        super();
    }
    /**
    * {@inheritDoc}
    */
    @Override
    public IRecordProcessor createProcessor() {
        return new SampleRecordProcessor();
    }
}
```

Buat Pekerja

Seperti dibahas dalamMenerapkan RecordProcessor Metode I, ada dua versi antarmuka prosesor rekaman KCL untuk dipilih, yang memengaruhi cara Anda membuat pekerja. Antarmuka prosesor rekaman asli menggunakan struktur kode berikut untuk membuat pekerja:

```
final KinesisClientLibConfiguration config = new KinesisClientLibConfiguration(...)
final IRecordProcessorFactory recordProcessorFactory = new RecordProcessorFactory();
final Worker worker = new Worker(recordProcessorFactory, config);
```

Dengan versi 2 dari antarmuka prosesor rekaman, Anda dapat menggunakan `Worker.Builder` untuk membuat pekerja tanpa perlu khawatir tentang konstruktor mana yang akan digunakan dan urutan argumen. Antarmuka prosesor rekaman yang diperbarui menggunakan struktur kode berikut untuk membuat pekerja:

```
final KinesisClientLibConfiguration config = new KinesisClientLibConfiguration(...)
final IRecordProcessorFactory recordProcessorFactory = new RecordProcessorFactory();
final Worker worker = new Worker.Builder()
    .recordProcessorFactory(recordProcessorFactory)
    .config(config)
    .build();
```

Ubah Properti Konfigurasi

Sampel memberikan nilai default untuk properti konfigurasi. Data konfigurasi untuk pekerja ini kemudian dikonsolidasikan dalam sebuah `KinesisClientLibConfiguration` objek. Objek ini dan referensi ke pabrik kelas untuk `IRecordProcessor` diteruskan dalam panggilan yang membuat instance pekerja. Anda dapat mengganti salah satu properti ini dengan nilai Anda sendiri menggunakan file properti Java (lihat`AmazonKinesisApplicationSample.java`).

Nama Aplikasi

KCL memerlukan nama aplikasi yang unik di seluruh aplikasi Anda, dan di seluruh tabel Amazon DynamoDB di Wilayah yang sama. Ini menggunakan nilai konfigurasi nama aplikasi dengan cara berikut:

- Semua pekerja yang terkait dengan nama aplikasi ini diasumsikan bekerja sama pada aliran yang sama. Pekerja ini dapat didistribusikan pada beberapa contoh. Jika Anda menjalankan instance tambahan dari kode aplikasi yang sama, tetapi dengan nama aplikasi yang berbeda, KCL memperlakukan instance kedua sebagai aplikasi yang sepenuhnya terpisah yang juga beroperasi pada aliran yang sama.
- KCL membuat tabel DynamoDB dengan nama aplikasi dan menggunakan tabel untuk mempertahankan informasi status (seperti pos pemeriksaan dan pemetaan pecahan pekerja) untuk aplikasi. Setiap aplikasi memiliki tabel DynamoDB sendiri. Untuk informasi selengkapnya, lihat Menggunakan Meja Sewa untuk Melacak Pecahan yang Diproses oleh Aplikasi Konsumen KCL.

Mengatur Kredensial

Anda harus membuat AWS kredensil Anda tersedia untuk salah satu penyedia kredensi dalam rantai penyedia kredensi default. Misalnya, jika Anda menjalankan konsumen pada instans EC2, sebaiknya Anda meluncurkan instans dengan peran IAM. AWSkredensyal yang mencerminkan izin yang terkait dengan peran IAM ini tersedia untuk aplikasi pada instance melalui metadata instance-nya. Ini adalah cara paling aman untuk mengelola kredensional bagi konsumen yang berjalan pada instans EC2.

Aplikasi sampel pertama kali mencoba untuk mengambil kredenal IAM dari metadata instance:

```
credentialsProvider = new InstanceProfileCredentialsProvider();
```

Jika aplikasi sampel tidak dapat memperoleh kredensil dari metadata instance, aplikasi tersebut mencoba mengambil kredensil dari file properti:

```
credentialsProvider = new ClasspathPropertiesFileCredentialsProvider();
```

Untuk informasi selengkapnya tentang metadata instans, lihat Metadata Instans di Panduan Pengguna Amazon EC2 untuk Instans Linux.

Menggunakan ID Pekerja untuk Beberapa Instance

Kode inisialisasi sampel membuat ID untuk pekerja`workerId`, menggunakan nama komputer lokal dan menambahkan pengidentifikasi unik global seperti yang ditunjukkan dalam cuplikan kode berikut. Pendekatan ini mendukung skenario beberapa contoh aplikasi konsumen yang berjalan pada satu komputer.

```
String workerId = InetAddress.getLocalHost().getCanonicalHostName() + ":" +
 UUID.randomUUID();
```

Migrasi ke Versi 2 dari Antarmuka Prosesor Rekam

Jika Anda ingin memigrasikan kode yang menggunakan antarmuka asli, selain langkah-langkah yang dijelaskan sebelumnya, langkah-langkah berikut diperlukan:

1. Ubah kelas prosesor rekaman Anda untuk mengimpor antarmuka prosesor rekaman versi 2:

   ```
   import com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor;
   ```

2. Ubah referensi ke input untuk menggunakan `get` metode pada objek kontainer. Misalnya, dalam `shutdown()` operasi, ubah "`checkpointer`" menjadi "`shutdownInput.getCheckpointer()`".
3. Ubah kelas pabrik prosesor rekaman Anda untuk mengimpor antarmuka pabrik prosesor rekaman versi 2:

   ```
   import
    com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessorFactory;
   ```

4. Ubah konstruksi pekerja yang akan digunakan`Worker.Builder`. Sebagai contoh:

   ```
   final Worker worker = new Worker.Builder()
       .recordProcessorFactory(recordProcessorFactory)
       .config(config)
       .build();
   ```

Mengembangkan Konsumen Perpustakaan Klien Kinesis di Node.js

Anda dapat menggunakan Kinesis Client Library (KCL) untuk membangun aplikasi yang memproses data dari aliran data Kinesis Anda. Perpustakaan Klien Kinesis tersedia dalam berbagai bahasa. Topik ini membahas Node.js.

KCL adalah perpustakaan Java; dukungan untuk bahasa selain Java disediakan menggunakan antarmuka multi-bahasa yang disebut. MultiLangDaemon Daemon ini berbasis Java dan berjalan di latar belakang saat Anda menggunakan bahasa KCL selain Java. Oleh karena itu, jika Anda menginstal KCL untuk Node.js dan menulis aplikasi konsumen Anda sepenuhnya di Node.js, Anda masih memerlukan Java diinstal pada sistem Anda karena itu MultiLangDaemon. Selanjutnya, MultiLangDaemon memiliki beberapa pengaturan default yang mungkin perlu Anda sesuaikan untuk kasus penggunaan Anda, misalnya, AWS Wilayah yang terhubung dengannya. Untuk informasi lebih lanjut tentang MultiLangDaemon on GitHub, buka halaman MultiLangDaemon proyek KCL.

Untuk mengunduh Node.js KCL dari GitHub, buka Perpustakaan Klien Kinesis (Node.js).

Unduhan Kode Sampel

Ada dua contoh kode yang tersedia untuk KCL di Node.js:

- sampel dasar

  Digunakan di bagian berikut untuk menggambarkan dasar-dasar membangun aplikasi konsumen KCL di Node.js.

- click-stream-sample

  Sedikit lebih maju dan menggunakan skenario dunia nyata, setelah Anda membiasakan diri dengan kode sampel dasar. Sampel ini tidak dibahas di sini tetapi memiliki file README dengan informasi lebih lanjut.

Anda harus menyelesaikan tugas-tugas berikut saat menerapkan aplikasi konsumen KCL di Node.js:

Tugas

- Menerapkan Record Processor
- Ubah Properti Konfigurasi

Menerapkan Record Processor

Konsumen paling sederhana yang mungkin menggunakan KCL untuk Node.js harus mengimplementasikan `recordProcessor` fungsi, yang pada gilirannya berisi fungsi`initialize`,`processRecords`, dan`shutdown`. Sampel menyediakan implementasi yang dapat Anda gunakan sebagai titik awal (lihat`sample_kcl_app.js`).

```
function recordProcessor() {
  // return an object that implements initialize, processRecords and shutdown
 functions.}
```

menginisialisasi

KCL memanggil `initialize` fungsi ketika prosesor rekaman dimulai. Prosesor rekaman ini hanya memproses ID pecahan yang diteruskan sebagai`initializeInput.shardId`, dan biasanya, kebalikannya juga benar (pecahan ini hanya diproses oleh prosesor rekaman ini). Namun, konsumen Anda harus memperhitungkan kemungkinan bahwa catatan data dapat diproses lebih dari satu kali. Ini karena Kinesis Data Streams memiliki semantik setidaknya sekali, artinya setiap catatan data dari pecahan diproses setidaknya satu kali oleh pekerja di konsumen Anda. Untuk informasi lebih lanjut tentang kasus di mana pecahan tertentu dapat diproses oleh lebih dari satu pekerja, lihat[Resharding, Scaling, dan Pengolahan Paralel](#).

```
initialize: function(initializeInput, completeCallback)
```

processRecords

KCL memanggil fungsi ini dengan input yang berisi daftar catatan data dari pecahan yang ditentukan ke fungsi tersebut`initialize`. Prosesor rekaman yang Anda terapkan memproses data dalam catatan ini sesuai dengan semantik konsumen Anda. Misalnya, pekerja mungkin melakukan transformasi pada data dan kemudian menyimpan hasilnya di bucket Amazon Simple Storage Service (Amazon S3).

```
processRecords: function(processRecordsInput, completeCallback)
```

Selain data itu sendiri, catatan juga berisi nomor urut dan kunci partisi, yang dapat digunakan pekerja saat memproses data. Misalnya, pekerja dapat memilih bucket S3 untuk menyimpan data berdasarkan nilai kunci partisi. `record`Kamus mengekspos pasangan kunci-nilai berikut untuk mengakses data catatan, nomor urut, dan kunci partisi:

```
record.data
record.sequenceNumber
record.partitionKey
```

Perhatikan bahwa data tersebut dikodekan oleh Base64.

Dalam sampel dasar, fungsi `processRecords` memiliki kode yang menunjukkan bagaimana seorang pekerja dapat mengakses data rekaman, nomor urut, dan kunci partisi.

Kinesis Data Streams membutuhkan prosesor rekaman untuk melacak catatan yang telah diproses dalam pecahan. KCL menangani pelacakan ini dengan `checkpointer` objek yang dilewatkan sebagai`processRecordsInput.checkpointer`. Prosesor rekaman Anda memanggil `checkpointer.checkpoint` fungsi untuk memberi tahu KCL seberapa jauh perkembangannya dalam memproses catatan di pecahan. Jika pekerja gagal, KCL menggunakan informasi ini saat Anda memulai ulang pemrosesan pecahan sehingga berlanjut dari catatan olahan terakhir yang diketahui.

Untuk operasi split atau merge, KCL tidak mulai memproses pecahan baru sampai prosesor untuk pecahan asli dipanggil `checkpoint` untuk memberi sinyal bahwa semua pemrosesan pada pecahan asli selesai.

Jika Anda tidak meneruskan nomor urut ke `checkpoint` fungsi, KCL mengasumsikan bahwa panggilan ke `checkpoint` berarti bahwa semua catatan telah diproses, hingga catatan terakhir yang diteruskan ke prosesor rekaman. Oleh karena itu, prosesor rekaman harus memanggil `checkpoint` hanya setelah memproses semua catatan dalam daftar yang diteruskan ke sana. Prosesor rekaman tidak perlu memanggil `checkpoint` setiap panggilan ke`processRecords`. Prosesor dapat, misalnya, memanggil setiap panggilan ketiga, atau beberapa peristiwa `checkpoint` di luar prosesor rekaman Anda, seperti layanan verifikasi/validasi khusus yang telah Anda terapkan.

Anda dapat secara opsional menentukan nomor urut yang tepat dari catatan sebagai parameter untuk`checkpoint`. Dalam hal ini, KCL mengasumsikan bahwa semua catatan telah diproses hingga catatan itu saja.

Aplikasi sampel dasar menunjukkan panggilan sesederhana mungkin ke `checkpointer.checkpoint` fungsi tersebut. Anda dapat menambahkan logika checkpointing lain yang Anda butuhkan untuk konsumen Anda pada titik ini dalam fungsi.

penonaktifan

KCL memanggil `shutdown` fungsi baik saat pemrosesan berakhir (`shutdownInput.reason`is`TERMINATE`) atau pekerja tidak lagi merespons (`shutdownInput.reason`is`ZOMBIE`).

```
shutdown: function(shutdownInput, completeCallback)
```

Pemrosesan berakhir ketika prosesor rekaman tidak menerima catatan lebih lanjut dari pecahan, karena pecahan dipecah atau digabungkan, atau aliran dihapus.

KCL juga meneruskan `shutdownInput.checkpointer` objek ke`shutdown`. Jika alasan shutdown adalah`TERMINATE`, Anda harus memastikan bahwa prosesor rekaman telah selesai memproses catatan data apa pun, dan kemudian memanggil `checkpoint` fungsi pada antarmuka ini.

Ubah Properti Konfigurasi

Sampel memberikan nilai default untuk properti konfigurasi. Anda dapat mengganti salah satu properti ini dengan nilai Anda sendiri (lihat `sample.properties` di sampel dasar).

Nama Aplikasi

KCL memerlukan aplikasi yang unik di antara aplikasi Anda, dan di antara tabel Amazon DynamoDB di Wilayah yang sama. Ini menggunakan nilai konfigurasi nama aplikasi dengan cara berikut:

- Semua pekerja yang terkait dengan nama aplikasi ini diasumsikan bekerja sama pada aliran yang sama. Pekerja ini dapat didistribusikan pada beberapa contoh. Jika Anda menjalankan instance tambahan dari kode aplikasi yang sama, tetapi dengan nama aplikasi yang berbeda, KCL memperlakukan instance kedua sebagai aplikasi yang sepenuhnya terpisah yang juga beroperasi pada aliran yang sama.
- KCL membuat tabel DynamoDB dengan nama aplikasi dan menggunakan tabel untuk mempertahankan informasi status (seperti pos pemeriksaan dan pemetaan pecahan pekerja) untuk aplikasi. Setiap aplikasi memiliki tabel DynamoDB sendiri. Untuk informasi selengkapnya, lihat Menggunakan Meja Sewa untuk Melacak Pecahan yang Diproses oleh Aplikasi Konsumen KCL.

Mengatur Kredensial

Anda harus membuat AWS kredensil Anda tersedia untuk salah satu penyedia kredensi dalam rantai penyedia kredensi default. Anda dapat menggunakan `AWSCredentialsProvider` properti untuk menetapkan penyedia kredensial. `sample.properties`File harus membuat kredensyal Anda tersedia untuk salah satu penyedia kredensyal dalam rantai penyedia kredensi default. Jika Anda

menjalankan konsumen di instans Amazon EC2, sebaiknya Anda mengonfigurasi instans dengan peran IAM. AWSkredensyal yang mencerminkan izin yang terkait dengan peran IAM ini tersedia untuk aplikasi pada instance melalui metadata instance-nya. Ini adalah cara paling aman untuk mengelola kredensil untuk aplikasi konsumen yang berjalan pada instans EC2.

Contoh berikut mengkonfigurasi KCL untuk memproses aliran data Kinesis bernama `kclnodejssample` menggunakan prosesor rekaman yang disediakan di: `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
```

Mengembangkan Konsumen Perpustakaan Klien Kinesis di .NET

Anda dapat menggunakan Kinesis Client Library (KCL) untuk membangun aplikasi yang memproses data dari aliran data Kinesis Anda. Perpustakaan Klien Kinesis tersedia dalam berbagai bahasa. Topik ini membahas .NET.

KCL adalah perpustakaan Java; dukungan untuk bahasa selain Java disediakan menggunakan antarmuka multi-bahasa yang disebut. MultiLangDaemon Daemon ini berbasis Java dan berjalan di latar belakang saat Anda menggunakan bahasa KCL selain Java. Oleh karena itu, jika Anda menginstal KCL untuk.NET dan menulis aplikasi konsumen Anda sepenuhnya di .NET, Anda masih perlu Java diinstal pada sistem Anda karena itu MultiLangDaemon. Selanjutnya, MultiLangDaemon memiliki beberapa pengaturan default yang mungkin perlu Anda sesuaikan untuk kasus penggunaan Anda, misalnya, AWS Wilayah yang terhubung dengannya. Untuk informasi lebih lanjut tentang MultiLangDaemon on GitHub, buka halaman MultiLangDaemon proyek KCL.

Untuk mengunduh .NET KCL dari GitHub, buka Kinesis Client Library (.NET). Untuk mengunduh kode sampel untuk aplikasi konsumen.NET KCL, buka halaman proyek konsumen sampel KCL untuk .NET di. GitHub

Anda harus menyelesaikan tugas-tugas berikut saat menerapkan aplikasi konsumen KCL di .NET:

Tugas

- Menerapkan Metode RecordProcessor Kelas I
- Ubah Properti Konfigurasi

Menerapkan Metode RecordProcessor Kelas I

Konsumen harus menerapkan metode berikut untuk`IRecordProcessor`. Konsumen sampel menyediakan implementasi yang dapat Anda gunakan sebagai titik awal (lihat `SampleRecordProcessor` kelas di`SampleConsumer/AmazonKinesisSampleConsumer.cs`).

```
public void Initialize(InitializationInput input)
public void ProcessRecords(ProcessRecordsInput input)
public void Shutdown(ShutdownInput input)
```

Inisialisasi

KCL memanggil metode ini ketika prosesor rekaman dipakai, melewati ID pecahan tertentu dalam parameter (). `input` `input.ShardId` Prosesor rekaman ini hanya memproses pecahan ini, dan biasanya, kebalikannya juga benar (pecahan ini hanya diproses oleh prosesor rekaman ini). Namun, konsumen Anda harus memperhitungkan kemungkinan bahwa catatan data dapat diproses lebih dari satu kali. Ini karena Kinesis Data Streams memiliki semantik setidaknya sekali, artinya setiap catatan data dari pecahan diproses setidaknya satu kali oleh pekerja di konsumen Anda. Untuk informasi lebih lanjut tentang kasus di mana pecahan tertentu dapat diproses oleh lebih dari satu pekerja, lihatResharding, Scaling, dan Pengolahan Paralel.

```
public void Initialize(InitializationInput input)
```

ProcessRecords

KCL memanggil metode ini, melewati daftar catatan data dalam `input` parameter (`input.Records`) dari pecahan yang ditentukan oleh metode. `Initialize` Prosesor rekaman yang Anda terapkan memproses data dalam catatan ini sesuai dengan semantik konsumen Anda. Misalnya, pekerja mungkin melakukan transformasi pada data dan kemudian menyimpan hasilnya di bucket Amazon Simple Storage Service (Amazon S3).

```
public void ProcessRecords(ProcessRecordsInput input)
```

Selain data itu sendiri, catatan juga berisi nomor urut dan kunci partisi. Pekerja dapat menggunakan nilai-nilai ini saat memproses data. Misalnya, pekerja dapat memilih bucket S3 untuk menyimpan data

berdasarkan nilai kunci partisi. `Record`Kelas mengekspos berikut ini untuk mengakses data catatan, nomor urut, dan kunci partisi:

```
byte[] Record.Data
string Record.SequenceNumber
string Record.PartitionKey
```

Dalam sampel, metode ini `ProcessRecordsWithRetries` memiliki kode yang menunjukkan bagaimana seorang pekerja dapat mengakses data rekaman, nomor urut, dan kunci partisi.

Kinesis Data Streams membutuhkan prosesor rekaman untuk melacak catatan yang telah diproses dalam pecahan. KCL menangani pelacakan ini untuk Anda dengan meneruskan `Checkpointer` objek ke `ProcessRecords` (`input.Checkpointer`). Prosesor rekaman memanggil `Checkpointer.Checkpoint` metode untuk menginformasikan KCL tentang seberapa jauh perkembangannya dalam memproses catatan di pecahan. Jika pekerja gagal, KCL menggunakan informasi ini untuk memulai kembali pemrosesan pecahan pada catatan diproses terakhir yang diketahui.

Untuk operasi split atau merge, KCL tidak mulai memproses pecahan baru sampai prosesor untuk pecahan asli dipanggil `Checkpointer.Checkpoint` untuk memberi sinyal bahwa semua pemrosesan pada pecahan asli selesai.

Jika Anda tidak melewati parameter, KCL mengasumsikan bahwa panggilan untuk `Checkpointer.Checkpoint` menandakan bahwa semua catatan telah diproses, hingga catatan terakhir yang diteruskan ke prosesor rekaman. Oleh karena itu, prosesor rekaman harus memanggil `Checkpointer.Checkpoint` hanya setelah memproses semua catatan dalam daftar yang diteruskan ke sana. Prosesor rekaman tidak perlu memanggil `Checkpointer.Checkpoint` setiap panggilan ke`ProcessRecords`. Prosesor dapat, misalnya, memanggil `Checkpointer.Checkpoint` setiap panggilan ketiga atau keempat. Anda dapat secara opsional menentukan nomor urut yang tepat dari catatan sebagai parameter untuk`Checkpointer.Checkpoint`. Dalam hal ini, KCL mengasumsikan bahwa catatan telah diproses hanya hingga catatan itu.

Dalam sampel, metode pribadi `Checkpoint(Checkpointer checkpointer)` menunjukkan cara memanggil `Checkpointer.Checkpoint` metode menggunakan penanganan pengecualian yang sesuai dan logika coba lagi.

KCL untuk.NET menangani pengecualian secara berbeda dari pustaka bahasa KCL lainnya karena tidak menangani pengecualian apa pun yang muncul dari pemrosesan catatan data. Setiap pengecualian yang tidak tertangkap dari kode pengguna akan merusak program.

Shutdown

KCL memanggil `Shutdown` metode baik saat pemrosesan berakhir (alasan shutdown adalah`TERMINATE`) atau pekerja tidak lagi merespons (nilai shutdown `input.Reason` adalah). `ZOMBIE`

```
public void Shutdown(ShutdownInput input)
```

Pemrosesan berakhir ketika prosesor rekaman tidak menerima catatan lebih lanjut dari pecahan, karena pecahan dipecah atau digabungkan, atau aliran dihapus.

KCL juga meneruskan `Checkpointer` objek ke`shutdown`. Jika alasan shutdown adalah`TERMINATE`, prosesor rekaman harus menyelesaikan pemrosesan catatan data apa pun, dan kemudian memanggil `checkpoint` metode pada antarmuka ini.

Ubah Properti Konfigurasi

Konsumen sampel memberikan nilai default untuk properti konfigurasi. Anda dapat mengganti salah satu properti ini dengan nilai Anda sendiri (lihat`SampleConsumer/kcl.properties`).

Nama Aplikasi

KCL memerlukan aplikasi yang unik di antara aplikasi Anda, dan di antara tabel Amazon DynamoDB di Wilayah yang sama. Ini menggunakan nilai konfigurasi nama aplikasi dengan cara berikut:

- Semua pekerja yang terkait dengan nama aplikasi ini diasumsikan bekerja sama pada aliran yang sama. Pekerja ini dapat didistribusikan pada beberapa contoh. Jika Anda menjalankan instance tambahan dari kode aplikasi yang sama, tetapi dengan nama aplikasi yang berbeda, KCL memperlakukan instance kedua sebagai aplikasi yang sepenuhnya terpisah yang juga beroperasi pada aliran yang sama.
- KCL membuat tabel DynamoDB dengan nama aplikasi dan menggunakan tabel untuk mempertahankan informasi status (seperti pos pemeriksaan dan pemetaan pecahan pekerja) untuk aplikasi. Setiap aplikasi memiliki tabel DynamoDB sendiri. Untuk informasi selengkapnya, lihat Menggunakan Meja Sewa untuk Melacak Pecahan yang Diproses oleh Aplikasi Konsumen KCL.

Mengatur Kredensial

Anda harus membuat AWS kredensil Anda tersedia untuk salah satu penyedia kredensi dalam rantai penyedia kredensi default. Anda dapat menggunakan `AWSCredentialsProvider` properti untuk menetapkan penyedia kredensial. Sample.properties harus membuat kredensyal Anda tersedia untuk salah satu penyedia kredensyal dalam rantai penyedia kredensyal default. Jika Anda menjalankan aplikasi konsumen pada instans EC2, sebaiknya Anda mengonfigurasi instans dengan peran IAM. AWSkredensyal yang mencerminkan izin yang terkait dengan peran IAM ini tersedia untuk aplikasi pada instance melalui metadata instance-nya. Ini adalah cara paling aman untuk mengelola kredensional bagi konsumen yang berjalan pada instans EC2.

File properti sampel mengonfigurasi KCL untuk memproses aliran data Kinesis yang disebut “kata-kata” menggunakan prosesor rekaman yang disertakan. `AmazonKinesisSampleConsumer.cs`

Mengembangkan Konsumen Perpustakaan Klien Kinesis dengan Python

Anda dapat menggunakan Kinesis Client Library (KCL) untuk membangun aplikasi yang memproses data dari aliran data Kinesis Anda. Perpustakaan Klien Kinesis tersedia dalam berbagai bahasa. Topik ini membahas Python.

KCL adalah perpustakaan Java; dukungan untuk bahasa selain Java disediakan menggunakan antarmuka multi-bahasa yang disebut. MultiLangDaemon Daemon ini berbasis Java dan berjalan di latar belakang saat Anda menggunakan bahasa KCL selain Java. Oleh karena itu, jika Anda menginstal KCL untuk Python dan menulis aplikasi konsumen Anda sepenuhnya dengan Python, Anda masih memerlukan Java diinstal pada sistem Anda karena itu. MultiLangDaemon Selanjutnya, MultiLangDaemon memiliki beberapa pengaturan default yang mungkin perlu Anda sesuaikan untuk kasus penggunaan Anda, misalnya, AWS Wilayah yang terhubung dengannya. Untuk informasi lebih lanjut tentang MultiLangDaemon on GitHub, buka halaman MultiLangDaemon proyek KCL.

Untuk mengunduh Python KCL dari GitHub, pergi ke Kinesis Client Library (Python). Untuk mengunduh kode sampel untuk aplikasi konsumen Python KCL, buka halaman proyek sampel KCL untuk Python di. GitHub

Anda harus menyelesaikan tugas-tugas berikut saat menerapkan aplikasi konsumen KCL dengan Python:

Tugas

- Menerapkan Metode RecordProcessor Kelas
- Ubah Properti Konfigurasi

Menerapkan Metode RecordProcessor Kelas

`RecordProcess`Kelas harus memperluas `RecordProcessorBase` untuk menerapkan metode berikut. Sampel menyediakan implementasi yang dapat Anda gunakan sebagai titik awal (lihat`sample_kclpy_app.py`).

```
def initialize(self, shard_id)
def process_records(self, records, checkpointer)
def shutdown(self, checkpointer, reason)
```

menginisialisasi

KCL memanggil `initialize` metode ketika prosesor rekaman dipakai, melewati ID pecahan tertentu sebagai parameter. Prosesor rekaman ini hanya memproses pecahan ini, dan biasanya, kebalikannya juga benar (pecahan ini hanya diproses oleh prosesor rekaman ini). Namun, konsumen Anda harus memperhitungkan kemungkinan bahwa catatan data dapat diproses lebih dari satu kali. Ini karena Kinesis Data Streams memiliki semantik setidaknya sekali, artinya setiap catatan data dari pecahan diproses setidaknya satu kali oleh pekerja di konsumen Anda. Untuk informasi lebih lanjut tentang kasus di mana pecahan tertentu dapat diproses oleh lebih dari satu pekerja, lihatResharding, Scaling, dan Pengolahan Paralel.

```
def initialize(self, shard_id)
```

process_records

KCL memanggil metode ini, melewati daftar catatan data dari pecahan yang ditentukan oleh metode. `initialize` Prosesor rekaman yang Anda terapkan memproses data dalam catatan ini sesuai dengan semantik konsumen Anda. Misalnya, pekerja mungkin melakukan transformasi pada data dan kemudian menyimpan hasilnya di bucket Amazon Simple Storage Service (Amazon S3).

```
def process_records(self, records, checkpointer)
```

Selain data itu sendiri, catatan juga berisi nomor urut dan kunci partisi. Pekerja dapat menggunakan nilai-nilai ini saat memproses data. Misalnya, pekerja dapat memilih bucket S3 untuk menyimpan data berdasarkan nilai kunci partisi. `record`Kamus mengekspos pasangan kunci-nilai berikut untuk mengakses data catatan, nomor urut, dan kunci partisi:

```
record.get('data')
```

```
record.get('sequenceNumber')
record.get('partitionKey')
```

Perhatikan bahwa data tersebut dikodekan oleh Base64.

Dalam sampel, metode ini `process_records` memiliki kode yang menunjukkan bagaimana seorang pekerja dapat mengakses data rekaman, nomor urut, dan kunci partisi.

Kinesis Data Streams membutuhkan prosesor rekaman untuk melacak catatan yang telah diproses dalam pecahan. KCL menangani pelacakan ini untuk Anda dengan mengirimkan `Checkpointer` objek ke`process_records`. Prosesor rekaman memanggil `checkpoint` metode pada objek ini untuk menginformasikan KCL tentang seberapa jauh perkembangannya dalam memproses catatan di pecahan. Jika pekerja gagal, KCL menggunakan informasi ini untuk memulai kembali pemrosesan pecahan pada catatan diproses terakhir yang diketahui.

Untuk operasi split atau merge, KCL tidak mulai memproses pecahan baru sampai prosesor untuk pecahan asli dipanggil `checkpoint` untuk memberi sinyal bahwa semua pemrosesan pada pecahan asli selesai.

Jika Anda tidak melewati parameter, KCL mengasumsikan bahwa panggilan ke `checkpoint` berarti bahwa semua catatan telah diproses, hingga catatan terakhir yang diteruskan ke prosesor rekaman. Oleh karena itu, prosesor rekaman harus memanggil `checkpoint` hanya setelah memproses semua catatan dalam daftar yang diteruskan ke sana. Prosesor rekaman tidak perlu memanggil `checkpoint` setiap panggilan ke`process_records`. Prosesor dapat, misalnya, memanggil `checkpoint` setiap panggilan ketiga. Anda dapat secara opsional menentukan nomor urut yang tepat dari catatan sebagai parameter untuk`checkpoint`. Dalam hal ini, KCL mengasumsikan bahwa semua catatan telah diproses hingga catatan itu saja.

Dalam sampel, metode pribadi `checkpoint` menunjukkan cara memanggil `Checkpointer.checkpoint` metode menggunakan penanganan pengecualian yang sesuai dan logika coba lagi.

KCL bergantung pada `process_records` untuk menangani pengecualian apa pun yang timbul dari pemrosesan catatan data. Jika pengecualian dilemparkan`process_records`, KCL melompati catatan data yang diteruskan `process_records` sebelum pengecualian. Artinya, catatan ini tidak dikirim kembali ke prosesor rekaman yang melemparkan pengecualian atau ke prosesor rekaman lainnya di konsumen.

penonaktifan

KCL memanggil `shutdown` metode baik saat pemrosesan berakhir (alasan shutdown adalah`TERMINATE`) atau pekerja tidak lagi merespons (shutdown `reason` adalah). `ZOMBIE`

```
def shutdown(self, checkpointer, reason)
```

Pemrosesan berakhir ketika prosesor rekaman tidak menerima catatan lebih lanjut dari pecahan, karena pecahan dipecah atau digabungkan, atau aliran dihapus.

KCL juga meneruskan `Checkpointer` objek ke`shutdown`. Jika shutdown `reason`adalah`TERMINATE`, prosesor rekaman harus menyelesaikan pemrosesan catatan data apa pun, dan kemudian memanggil `checkpoint` metode pada antarmuka ini.

Ubah Properti Konfigurasi

Sampel memberikan nilai default untuk properti konfigurasi. Anda dapat mengganti salah satu properti ini dengan nilai Anda sendiri (lihat`sample.properties`).

Nama Aplikasi

KCL memerlukan nama aplikasi yang unik di antara aplikasi Anda, dan di antara tabel Amazon DynamoDB di Wilayah yang sama. Ini menggunakan nilai konfigurasi nama aplikasi dengan cara berikut:

- Semua pekerja yang terkait dengan nama aplikasi ini diasumsikan bekerja sama pada aliran yang sama. Pekerja ini dapat didistribusikan pada beberapa contoh. Jika Anda menjalankan instance tambahan dari kode aplikasi yang sama, tetapi dengan nama aplikasi yang berbeda, KCL memperlakukan instance kedua sebagai aplikasi yang sepenuhnya terpisah yang juga beroperasi pada aliran yang sama.
- KCL membuat tabel DynamoDB dengan nama aplikasi dan menggunakan tabel untuk mempertahankan informasi status (seperti pos pemeriksaan dan pemetaan pecahan pekerja) untuk aplikasi. Setiap aplikasi memiliki tabel DynamoDB sendiri. Untuk informasi selengkapnya, lihat Menggunakan Meja Sewa untuk Melacak Pecahan yang Diproses oleh Aplikasi Konsumen KCL.

Mengatur Kredensial

Anda harus membuat AWS kredensil Anda tersedia untuk salah satu penyedia kredensi dalam rantai penyedia kredensi default. Anda dapat menggunakan `AWSCredentialsProvider` properti untuk menetapkan penyedia kredensial. Sample.properties harus membuat kredensyal Anda tersedia untuk salah satu penyedia kredensyal dalam rantai penyedia kredensyal default. Jika Anda menjalankan

aplikasi konsumen di instans Amazon EC2, sebaiknya Anda mengonfigurasi instans dengan peran IAM. AWSkredensyal yang mencerminkan izin yang terkait dengan peran IAM ini tersedia untuk aplikasi pada instance melalui metadata instance-nya. Ini adalah cara paling aman untuk mengelola kredensil untuk aplikasi konsumen yang berjalan pada instans EC2.

File properti sampel mengonfigurasi KCL untuk memproses aliran data Kinesis yang disebut "kata-kata" menggunakan prosesor rekaman yang disertakan. `sample_kclpy_app.py`

Mengembangkan Konsumen Perpustakaan Klien Kinesis di Ruby

Anda dapat menggunakan Kinesis Client Library (KCL) untuk membangun aplikasi yang memproses data dari aliran data Kinesis Anda. Perpustakaan Klien Kinesis tersedia dalam berbagai bahasa. Topik ini membahas Ruby.

KCL adalah perpustakaan Java; dukungan untuk bahasa selain Java disediakan menggunakan antarmuka multi-bahasa yang disebut. MultiLangDaemon Daemon ini berbasis Java dan berjalan di latar belakang saat Anda menggunakan bahasa KCL selain Java. Oleh karena itu, jika Anda menginstal KCL untuk Ruby dan menulis aplikasi konsumen Anda sepenuhnya di Ruby, Anda masih memerlukan Java diinstal pada sistem Anda karena itu. MultiLangDaemon Selanjutnya, MultiLangDaemon memiliki beberapa pengaturan default yang mungkin perlu Anda sesuaikan untuk kasus penggunaan Anda, misalnya, AWS Wilayah yang terhubung dengannya. Untuk informasi lebih lanjut tentang MultiLangDaemon on GitHub, buka halaman MultiLangDaemon proyek KCL.

Untuk mengunduh Ruby KCL dari GitHub, buka Perpustakaan Klien Kinesis (Ruby). Untuk mengunduh kode sampel untuk aplikasi konsumen Ruby KCL, buka halaman proyek sampel KCL untuk Ruby di. GitHub

Untuk informasi selengkapnya tentang pustaka dukungan KCL Ruby, lihat Dokumentasi Permata Ruby KCL.

Mengembangkan Konsumen KCL 2.x

Topik ini menunjukkan cara menggunakan versi 2.0 dari Kinesis Client Library (KCL). Untuk informasi lebih lanjut tentang KCL, lihat ikhtisar yang disediakan dalam Mengembangkan Konsumen Menggunakan Perpustakaan Klien Kinesis 1.x.

Daftar Isi

Mengembangkan Konsumen Perpustakaan Klien Kinesis di Jawa

Kode berikut menunjukkan contoh implementasi di Java dari `ProcessorFactory` dan`RecordProcessor`. Jika Anda ingin memanfaatkan fitur fan-out yang disempurnakan, lihat Menggunakan Konsumen dengan Enhanced Fan-Out.

```
/*
 *  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);
            }
        }
    }

}
```

Mengembangkan Konsumen Perpustakaan Klien Kinesis dengan Python

Anda dapat menggunakan Kinesis Client Library (KCL) untuk membangun aplikasi yang memproses data dari aliran data Kinesis Anda. Perpustakaan Klien Kinesis tersedia dalam berbagai bahasa. Topik ini membahas Python.

KCL adalah perpustakaan Java; dukungan untuk bahasa selain Java disediakan menggunakan antarmuka multi-bahasa yang disebut. MultiLangDaemon Daemon ini berbasis Java dan berjalan di latar belakang saat Anda menggunakan bahasa KCL selain Java. Oleh karena itu, jika Anda menginstal KCL untuk Python dan menulis aplikasi konsumen Anda sepenuhnya dengan Python, Anda masih memerlukan Java diinstal pada sistem Anda karena itu. MultiLangDaemon Selanjutnya, MultiLangDaemon memiliki beberapa pengaturan default yang mungkin perlu Anda sesuaikan untuk kasus penggunaan Anda, misalnya, AWS Wilayah yang terhubung dengannya. Untuk informasi lebih lanjut tentang MultiLangDaemon on GitHub, buka halaman MultiLangDaemon proyek KCL.

Untuk mengunduh Python KCL dari GitHub, pergi ke Kinesis Client Library (Python). Untuk mengunduh kode sampel untuk aplikasi konsumen Python KCL, buka halaman proyek sampel KCL untuk Python. GitHub

Anda harus menyelesaikan tugas-tugas berikut saat menerapkan aplikasi konsumen KCL dengan Python:

Tugas

- Menerapkan Metode RecordProcessor Kelas
- Ubah Properti Konfigurasi

Menerapkan Metode RecordProcessor Kelas

`RecordProcess`Kelas harus memperluas `RecordProcessorBase` kelas untuk mengimplementasikan metode berikut:

```
initialize
```

```
process_records
shutdown_requested
```

Contoh ini menyediakan implementasi yang dapat Anda gunakan sebagai titik awal.

```
#!/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()
```

Ubah Properti Konfigurasi

Sampel memberikan nilai default untuk properti konfigurasi, seperti yang ditunjukkan pada skrip berikut. Anda dapat mengganti salah satu properti ini dengan nilai Anda sendiri.

```
# 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
```

Nama Aplikasi

KCL memerlukan nama aplikasi yang unik di antara aplikasi Anda dan di antara tabel Amazon DynamoDB di Wilayah yang sama. Ini menggunakan nilai konfigurasi nama aplikasi dengan cara berikut:

- Semua pekerja yang terkait dengan nama aplikasi ini diasumsikan bekerja sama pada aliran yang sama. Pekerja ini dapat didistribusikan di beberapa instance. Jika Anda menjalankan instance tambahan dari kode aplikasi yang sama, tetapi dengan nama aplikasi yang berbeda, KCL

memperlakukan instance kedua sebagai aplikasi yang sepenuhnya terpisah yang juga beroperasi pada aliran yang sama.

- KCL membuat tabel DynamoDB dengan nama aplikasi dan menggunakan tabel untuk mempertahankan informasi status (seperti pos pemeriksaan dan pemetaan pecahan pekerja) untuk aplikasi. Setiap aplikasi memiliki tabel DynamoDB sendiri. Untuk informasi selengkapnya, lihat Menggunakan Meja Sewa untuk Melacak Pecahan yang Diproses oleh Aplikasi Konsumen KCL.

Kredensial

Anda harus membuat AWS kredensyal Anda tersedia untuk salah satu penyedia kredensi dalam rantai penyedia kredensyal default. Anda dapat menggunakan `AWSCredentialsProvider` properti untuk menyetel penyedia kredensyal. Jika Anda menjalankan aplikasi konsumen di instans Amazon EC2, sebaiknya Anda mengonfigurasi instans dengan peran IAM. AWSkredensyal yang mencerminkan izin yang terkait dengan peran IAM ini tersedia untuk aplikasi pada instance melalui metadata instance-nya. Ini adalah cara paling aman untuk mengelola kredensyal untuk aplikasi konsumen yang berjalan pada instans EC2.

Mengembangkan Konsumen Kustom dengan Throughput Bersama MenggunakanAWS SDK for Java

Salah satu metode untuk mengembangkan Kinesis Data Streams kustom konsumen dengan bersama di seluruh adalah dengan menggunakan Amazon Kinesis Data Streams API. Bagian ini menjelaskan menggunakan API Kinesis Data Streams denganAWS SDK for Java. Contoh kode Java di bagian ini menunjukkan bagaimana melakukan operasi API KDS dasar, dan dibagi secara logis berdasarkan jenis operasi.

Contoh-contoh ini tidak mewakili kode siap produksi. Mereka tidak memeriksa semua kemungkinan pengecualian atau memperhitungkan semua kemungkinan pertimbangan keamanan atau performa.

Anda dapat memanggil API Kinesis Data Streams menggunakan bahasa pemrograman lain yang berbeda. Untuk informasi selengkapnya tentang semuaAWS SDK yang tersedia, lihat Mulai Pengembangan dengan Amazon Web Services.

> **Important**
>
> Metode yang disarankan untuk mengembangkan Kinesis Data Streams dengan shared seluruh adalah dengan menggunakan Kinesis Client Library (KCL). KCL membantu Anda

mengkonsumsi dan memproses data dari aliran data Kinesis dengan mengurus banyak tugas kompleks yang terkait dengan komputasi terdistribusi. Untuk informasi selengkapnya, lihat Mengembangkan Konsumen Kustom dengan Throughput Bersama Menggunakan KCL.

Topik

- Mendapatkan Data dari Stream
- Menggunakan Iterator Shard
- Menggunakan GetRecords
- Beradaptasi dengan Reshard
- Berinteraksi dengan Data Menggunakan Registry SkemaAWS Glue

Mendapatkan Data dari Stream

API Kinesis Data Streams menyertakan`getShardIterator` dan`getRecords` metode yang dapat Anda panggil untuk mengambil catatan dari aliran data. Ini adalah model tarik, di mana kode Anda menarik catatan data langsung dari pecahan aliran data.

Important

Kami menyarankan Anda menggunakan dukungan prosesor rekaman yang disediakan oleh KCL untuk mengambil catatan dari aliran data Anda. Ini adalah model push, di mana Anda menerapkan kode yang memproses data. KCL mengambil catatan data dari aliran data dan mengirimkannya ke kode aplikasi Anda. Selain itu, KCL menyediakan fungsionalitas failover, recovery, dan load balancing. Untuk informasi selengkapnya, lihat Mengembangkan Konsumen Kustom dengan Throughput Bersama Menggunakan KCL.

Namun, dalam beberapa kasus Anda mungkin lebih suka menggunakan API Kinesis Data Streams. Misalnya, untuk menerapkan alat kustom untuk memantau atau men-debug aliran data Anda.

Important

Kinesis Data Streams mendukung perubahan pada periode penyimpanan data data stream Anda. Untuk informasi selengkapnya, lihat Mengubah Periode Retensi Data.

Menggunakan Iterator Shard

Anda mengambil catatan dari sungai pada basis per-pecahan. Untuk setiap pecahan, dan untuk setiap batch catatan yang Anda ambil dari pecahan itu, Anda harus mendapatkan iterator pecahan. The iterator pecahan digunakan dalam`getRecordsRequest` objek untuk menentukan pecahan dari mana catatan yang akan diambil. Jenis yang terkait dengan iterator pecahan menentukan titik dalam pecahan dari mana catatan harus diambil (lihat nanti di bagian ini untuk lebih jelasnya). Sebelum Anda dapat bekerja dengan iterator pecahan, Anda perlu mengambil pecahan, seperti yang dibahas dalamDescribeStream API - Usang.

Dapatkan iterator pecahan awal menggunakan`getShardIterator` metode ini. Mendapatkan iterator pecahan untuk batch tambahan catatan menggunakan`getNextShardIterator` metode`getRecordsResult` objek dikembalikan oleh`getRecords` metode. Iterator shard berlaku selama 5 menit. Jika Anda menggunakan iterator shard saat valid, Anda mendapatkan yang baru. Setiap iterator shard tetap berlaku selama 5 menit, bahkan setelah digunakan.

Untuk mendapatkan iterator pecahan awal, instantiate`GetShardIteratorRequest` dan menyebarkannya ke`getShardIterator` metode. Untuk mengkonfigurasi permintaan, tentukan aliran dan ID pecahan. Untuk informasi selengkapnya tentang cara mendapatkan aliran diAWS akun Anda, lihatDaftar Aliran. Untuk informasi selengkapnya tentang cara mendapatkan pecahan dalam sebuah aliran, lihatDescribeStream API - Usang.

```
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();
```

Kode contoh ini menentukan`TRIM_HORIZON` sebagai jenis iterator ketika mendapatkan iterator pecahan awal. Jenis iterator ini berarti bahwa catatan harus dikembalikan dimulai dengan catatan pertama yang ditambahkan ke shard—bukan dimulai dengan catatan yang paling baru ditambahkan, juga dikenal sebagai tip. Berikut adalah kemungkinan jenis iterator:

- `AT_SEQUENCE_NUMBER`
- `AFTER_SEQUENCE_NUMBER`

- `AT_TIMESTAMP`
- `TRIM_HORIZON`
- `LATEST`

Untuk informasi lebih lanjut, lihat ShardIteratorType.

Beberapa jenis iterator mengharuskan Anda menentukan nomor urut selain jenis; misalnya:

```
getShardIteratorRequest.setShardIteratorType("AT_SEQUENCE_NUMBER");
getShardIteratorRequest.setStartingSequenceNumber(specialSequenceNumber);
```

Setelah Anda mendapatkan catatan menggunakan`getRecords`, Anda bisa mendapatkan nomor urut untuk catatan dengan memanggil`getSequenceNumber` metode catatan.

```
record.getSequenceNumber()
```

Selain itu, kode yang menambahkan catatan ke aliran data bisa mendapatkan nomor urut untuk catatan tambahan dengan memanggil`getSequenceNumber` hasil`putRecord`.

```
lastSequenceNumber = putRecordResult.getSequenceNumber();
```

Anda dapat menggunakan nomor urut untuk menjamin pemesanan catatan yang meningkat secara ketat. Untuk informasi lebih lanjut, lihat contoh kode diPutRecordContoh.

Menggunakan GetRecords

Setelah Anda mendapatkan iterator pecahan, instantiate`GetRecordsRequest` objek. Tentukan iterator untuk permintaan menggunakan`setShardIterator` metode.

Opsional, Anda juga dapat mengatur jumlah catatan untuk mengambil menggunakan`setLimit` metode. Jumlah catatan yang dikembalikan oleh`getRecords` selalu sama dengan atau kurang dari batas ini. Jika Anda tidak menentukan batas ini,`getRecords` mengembalikan 10 MB catatan diambil. Kode contoh di bawah menetapkan batas ini menjadi 25 catatan.

Jika tidak ada catatan yang dikembalikan, itu berarti tidak ada catatan data yang saat ini tersedia dari pecahan ini pada nomor urut yang direferensikan oleh iterator shard. Dalam situasi ini, aplikasi Anda harus menunggu sejumlah waktu yang sesuai untuk sumber data untuk streaming.

Kemudian cobalah untuk mendapatkan data dari pecahan lagi menggunakan iterator pecahan yang dikembalikan oleh panggilan sebelumnya kegetRecords.

LulusgetRecordsRequest kegetRecords metode, dan menangkap nilai kembali sebagaigetRecordsResult objek. Untuk mendapatkan catatan data, memanggilgetRecords metode padagetRecordsResult objek.

```
GetRecordsRequest getRecordsRequest = new GetRecordsRequest();
getRecordsRequest.setShardIterator(shardIterator);
getRecordsRequest.setLimit(25);

GetRecordsResult getRecordsResult = client.getRecords(getRecordsRequest);
List<Record> records = getRecordsResult.getRecords();
```

Untuk mempersiapkan panggilan laingetRecords, dapatkan iterator pecahan berikutnya darigetRecordsResult.

```
shardIterator = getRecordsResult.getNextShardIterator();
```

Untuk hasil terbaik, tidur setidaknya 1 detik (1.000 milidetik) di antara panggilangetRecords untuk menghindari melebihi batasgetRecords frekuensi.

```
try {
  Thread.sleep(1000);
}
catch (InterruptedException e) {}
```

Biasanya, Anda harus menelepongetRecords dalam satu lingkaran, bahkan ketika Anda mengambil satu catatan dalam skenario pengujian. Panggilan tunggal untukgetRecords dapat mengembalikan daftar catatan kosong, bahkan ketika pecahan berisi lebih banyak catatan pada nomor urut nanti. Ketika ini terjadi,NextShardIterator kembali bersama dengan daftar catatan kosong referensi nomor urut kemudian dalam pecahan, dangetRecords panggilan berturut-turut akhirnya mengembalikan catatan. Contoh berikut menunjukkan penggunaan loop.

Contoh: GetRecords

Contoh kode berikut mencerminkan`getRecords` tips di bagian ini, termasuk melakukan panggilan dalam satu lingkaran.

```
// 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();
}
```

Jika Anda menggunakan Kinesis Client Library, mungkin melakukan beberapa panggilan sebelum mengembalikan data. Perilaku ini adalah dengan desain dan tidak menunjukkan masalah dengan KCL atau data Anda.

Beradaptasi dengan Reshard

Jika`getRecordsResult.getNextShardIterator` kembali`null`, ini menunjukkan bahwa pecahan pecahan atau gabungan telah terjadi yang melibatkan pecahan ini. Shard ini sekarang

dalamCLOSED keadaan dan Anda telah membaca semua catatan data yang tersedia dari pecahan ini.

Dalam skenario ini, Anda dapat menggunakangetRecordsResult.childShards untuk mempelajari tentang pecahan anak baru dari pecahan yang sedang diproses yang dibuat oleh split atau merge. Untuk informasi lebih lanjut, lihat ChildShard.

Dalam kasus split, kedua pecahan baru keduanya memilikiparentShardId sama dengan ID pecahan pecahan yang Anda proses sebelumnya. NilaiadjacentParentShardId untuk kedua pecahan ini adalahnull.

Dalam kasus penggabungan, shard baru tunggal yang dibuat oleh gabungan memilikiparentShardId ID pecahan dari salah satu pecahan induk danadjacentParentShardId sama dengan ID pecahan dari pecahan induk lainnya. Aplikasi Anda telah membaca semua data dari salah satu pecahan ini. Ini adalah pecahan yanggetRecordsResult.getNextShardIterator dikembalikannull. Jika urutan data penting untuk aplikasi Anda, pastikan bahwa itu juga membaca semua data dari pecahan induk lainnya sebelum membaca data baru dari pecahan anak yang dibuat oleh gabungan.

Jika Anda menggunakan beberapa prosesor untuk mengambil data dari aliran (katakanlah, satu prosesor per pecahan), dan pecahan pecahan atau penggabungan terjadi, sesuaikan jumlah prosesor ke atas atau bawah untuk beradaptasi dengan perubahan jumlah pecahan.

Untuk informasi lebih lanjut tentang resharding, termasuk diskusi tentang shards states—sepertiCLOSED —seeMembagikan Ulang Aliran.

Berinteraksi dengan Data Menggunakan Registry SkemaAWS Glue

Anda dapat mengintegrasikan aliran data Kinesis Anda dengan registri skemaAWS Glue. RegistriAWS Glue skema memungkinkan Anda untuk menemukan, mengontrol, dan mengembangkan skema secara terpusat, sambil memastikan data yang dihasilkan terus divalidasi oleh skema terdaftar. Sebuah skema mendefinisikan struktur dan format catatan data. Sebuah skema adalah sebuah spesifikasi berversi untuk publikasi data yang handal, konsumsi, atau penyimpanan. Registri SkemaAWS Glue memungkinkan Anda untuk meningkatkan kualitas end-to-end data dan tata kelola data dalam aplikasi streaming Anda. Untuk informasi selengkapnya, lihat AWSGlue Skema. Salah satu cara untuk mengatur integrasi ini adalah melaluiGetRecords Kinesis Data Streams API yang tersedia diAWS Java SDK.

Untuk petunjuk terperinci tentang cara mengatur integrasi Kinesis Data Streams dengan Schema Registry menggunakan APIGetRecords Kinesis Data Streams, lihat bagian “Berinteraksi dengan

Data Menggunakan API Kinesis Data Streams” dalam Kasus Penggunaan: Mengintegrasikan Amazon Kinesis Data Streams dengan RegistryAWS Glue Schema.

Mengembangkan Konsumen Khusus dengan Throughput Khusus (Peningkatan Fan-Out)

Di Amazon Kinesis Data Streams, Anda dapat membangun konsumen yang menggunakan fitur bernamaditingkatkan fan-out. Fitur ini memungkinkan konsumen untuk menerima catatan dari aliran dengan throughput hingga 2 MB data per detik per pecahan. Throughput ini didedikasikan, yang berarti bahwa konsumen yang menggunakan fan-out yang ditingkatkan tidak harus bersaing dengan konsumen lain yang menerima data dari aliran. Kinesis Data Streams mendorong catatan data dari aliran ke konsumen yang menggunakan fan-out yang ditingkatkan. Oleh karena itu, konsumen ini tidak perlu melakukan polling untuk data.

> **Important**
>
> Anda dapat mendaftarkan hingga dua puluh konsumen per aliran untuk menggunakan fan-out yang ditingkatkan.

Diagram berikut menunjukkan arsitektur fan-out yang disempurnakan. Jika Anda menggunakan versi 2.0 atau versi lebih baru dari Amazon Kinesis Client Library (KCL) untuk membangun konsumen, KCL mengatur konsumen untuk menggunakan fan-out yang disempurnakan untuk menerima data dari semua pecahan aliran. Jika Anda menggunakan API untuk membangun konsumen yang menggunakan fan-out yang disempurnakan, maka Anda dapat berlangganan pecahan individu.

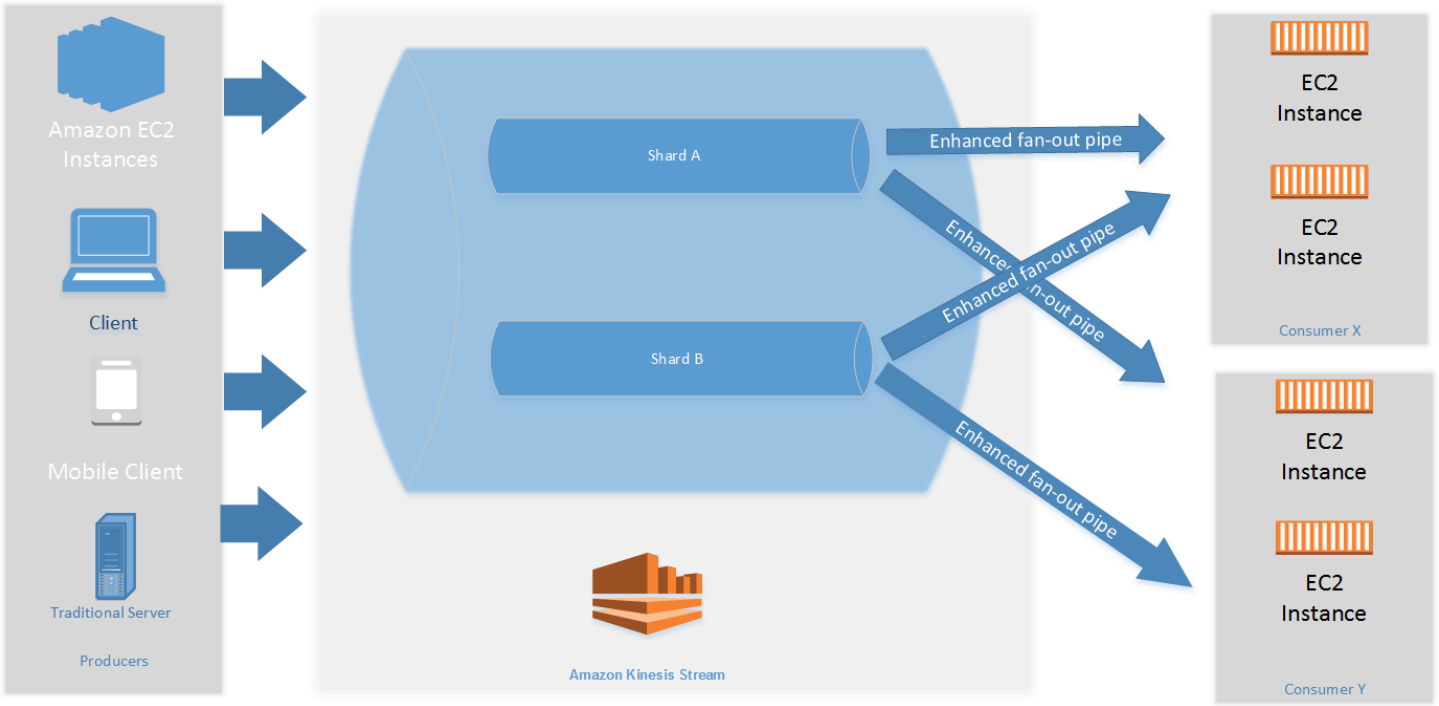


Diagram menunjukkan hal berikut:

- Sebuah aliran dengan dua pecahan.
- Dua konsumen yang menggunakan fan-out yang disempurnakan untuk menerima data dari aliran: Konsumen X dan Konsumen Y. masing-masing dari dua konsumen berlangganan semua pecahan dan semua catatan sungai. Jika Anda menggunakan versi 2.0 atau yang lebih baru dari KCL untuk membangun konsumen, KCL secara otomatis berlangganan konsumen tersebut ke semua pecahan aliran. Di sisi lain, jika Anda menggunakan API untuk membangun konsumen, Anda dapat berlangganan pecahan individu.
- Panah yang mewakili pipa kipas yang disempurnakan yang digunakan konsumen untuk menerima data dari sungai. Pipa kipas yang disempurnakan menyediakan data hingga 2 MB/detik per pecahan, terlepas dari pipa lain atau dari jumlah konsumen.

Topik

- Mengembangkan Peningkatan Fan-Out Konsumen dengan KCL 2.x
- Mengembangkan Konsumen Fan-Out yang Ditingkatkan dengan API Kinesis Data Streams
- Mengelola Peningkatan Fan-Out Konsumen denganAWS Management Console

Mengembangkan Peningkatan Fan-Out Konsumen dengan KCL 2.x

Konsumen yang menggunakanditingkatkan fan-outdi Amazon Kinesis Data Streams dapat menerima rekaman dari aliran data dengan throughput khusus hingga 2 MB data per detik per pecahan. Jenis konsumen tidak harus bersaing dengan konsumen lain yang menerima data dari aliran. Untuk informasi selengkapnya, lihat Mengembangkan Konsumen Khusus dengan Throughput Khusus (Peningkatan Fan-Out).

Anda dapat menggunakan versi 2.0 atau lebih baru dari Kinesis Client Library (KCL) untuk mengembangkan aplikasi yang menggunakan fan-out yang disempurnakan untuk menerima data dari stream. KCL secara otomatis berlangganan aplikasi Anda ke semua pecahan aliran, dan memastikan bahwa aplikasi konsumen Anda dapat membaca dengan nilai throughput 2 MB/detik per pecahan. Jika Anda ingin menggunakan KCL tanpa menyalakan fan-out yang disempurnakan, lihatMengembangkan Konsumen Menggunakan Kinesis Client Library 2.0.

Topik

- Mengembangkan Peningkatan Fan-Out Konsumen Menggunakan KCL 2.x di Jawa

Mengembangkan Peningkatan Fan-Out Konsumen Menggunakan KCL 2.x di Jawa

Anda dapat menggunakan versi 2.0 atau versi lebih baru dari Kinesis Client Library (KCL) untuk mengembangkan aplikasi di Amazon Kinesis Data Streams untuk menerima data dari stream menggunakan fan-out yang disempurnakan. Kode berikut menunjukkan contoh implementasi di Java`ProcessorFactory`dan`RecordProcessor`.

Dianjurkan agar Anda menggunakan`KinesisClientUtil`untuk membuat`KinesisAsyncClient`dan mengonfigurasi`maxConcurrency`di`KinesisAsyncClient`.

> **Important**
>
> Klien Amazon Kinesis mungkin melihat latensi yang meningkat secara signifikan, kecuali jika Anda mengonfigurasi`KinesisAsyncClient`untuk memiliki`maxConcurrency`cukup tinggi untuk memungkinkan semua sewa ditambah penggunaan tambahan`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);
            }
        }
    }

}
```

Mengembangkan Konsumen Fan-Out yang Ditingkatkan dengan API Kinesis Data Streams

Ditingkatkan fan-outadalah fitur Amazon Kinesis Data Streams yang memungkinkan konsumen menerima rekaman dari aliran data dengan throughput khusus hingga 2 MB data per detik per pecahan. Konsumen yang menggunakan fan-out yang disempurnakan tidak harus bersaing dengan konsumen lain yang menerima data dari aliran. Untuk informasi selengkapnya, lihat Mengembangkan Konsumen Khusus dengan Throughput Khusus (Peningkatan Fan-Out).

Anda dapat menggunakan operasi API untuk membangun konsumen yang menggunakan fan-out yang disempurnakan di Kinesis Data Streams.

Untuk mendaftarkan konsumen dengan fan-out yang disempurnakan menggunakan Kinesis Data Streams API

1. PANGLANRegisterStreamConsumeruntuk mendaftarkan aplikasi Anda sebagai konsumen yang menggunakan fan-out yang disempurnakan. Kinesis Data Streams menghasilkan Amazon Resource Name (ARN) untuk konsumen dan mengembalikannya dalam respons.
2. Untuk mulai mendengarkan pecahan tertentu, lulus ARN konsumen dalam panggilan keSubscribeToShard. Kinesis Data Streams kemudian mulai mendorong catatan dari pecahan itu kepada Anda, dalam bentuk peristiwa tipeSubscribeToShardEventmelalui koneksi HTTP/2. Sambungan tetap terbuka hingga 5 menit. PANGLANSubscribeToShardlagi jika Anda ingin terus menerima catatan dari beling setelah`future`yang dikembalikan oleh panggilan keSubscribeToShardmelengkapi secara normal atau sangat.

   > **Note**
   >
   > `SubscribeToShard`API juga mengembalikan daftar pecahan anak dari pecahan saat ini ketika akhir pecahan saat ini tercapai.

3. Untuk membatalkan pendaftaran konsumen yang menggunakan fan-out ditingkatkan, panggilanDeregisterStreamConsumer.

Kode berikut adalah contoh bagaimana Anda dapat berlangganan konsumen Anda ke pecahan, memperbarui langganan secara berkala, dan menangani acara.

```
    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);
        }
    }
```

Jika`event.ContinuationSequenceNumber`pulang`null`, ini menunjukkan bahwa pecahan pecahan atau penggabungan telah terjadi yang melibatkan pecahan ini. beling ini sekarang dalam`CLOSED`negara, dan Anda telah membaca semua catatan data yang tersedia dari beling ini. Dalam skenario ini, per contoh di atas, Anda dapat menggunakan`event.childShards`untuk belajar tentang pecahan anak baru dari beling yang sedang diproses yang diciptakan oleh split atau merge. Untuk informasi selengkapnya, lihatChildShard.

Berinteraksi dengan Data MenggunakanAWSRegistri Skema

Anda dapat mengintegrasikan aliran data Kinesis Anda denganAWSRegistri skema Glue. ParameterAWSRegistri skema Glue memungkinkan Anda untuk menemukan, mengontrol, dan mengembangkan skema secara terpusat, sambil memastikan data yang dihasilkan terus divalidasi oleh skema yang terdaftar. Sebuah skema mendefinisikan struktur dan format catatan data. Sebuah skema adalah sebuah spesifikasi berversi untuk publikasi data yang handal, konsumsi, atau penyimpanan. ParameterAWSGlue Skema Registry memungkinkan Anda untuk meningkatkan end-to-end kualitas data dan tata kelola data dalam aplikasi streaming Anda. Untuk informasi selengkapnya, lihatAWSRegistri Skema. Salah satu cara untuk mengatur integrasi ini adalah melalui`GetRecords`Kinesis Data Streams API tersedia di`AWSSDK Java`.

Untuk petunjuk rinci tentang cara mengatur integrasi Kinesis Data Streams dengan Registri Skema dengan menggunakan`GetRecords`API Kinesis Data Streams, lihat bagian “Berinteraksi dengan Data Menggunakan API Kinesis Data Streams” diKasus Penggunaan: Mengintegrasikan Amazon Kinesis Data Streams denganAWSRegistri Skema.

Mengelola Peningkatan Fan-Out Konsumen denganAWS Management Console

Konsumen yang menggunakanditingkatkan fan-outdi Amazon Kinesis Data Streams dapat menerima rekaman dari aliran data dengan throughput khusus hingga 2 MB data per detik per pecahan. Untuk informasi selengkapnya, lihat Mengembangkan Konsumen Khusus dengan Throughput Khusus (Peningkatan Fan-Out).

Anda dapat menggunakanAWS Management Consoleuntuk melihat daftar semua konsumen yang terdaftar untuk menggunakan ditingkatkan fan-out dengan aliran tertentu. Untuk setiap konsumen tersebut, Anda dapat melihat detail seperti ARN, status, tanggal pembuatan, dan metrik pemantauan.

Untuk melihat konsumen yang terdaftar untuk menggunakan fan-out yang disempurnakan, status, tanggal pembuatan, dan metrik di konsol

1. Masuk ke AWS Management Console dan buka konsol Kinesis di https://console.aws.amazon.com/kinesis.
2. Pilih Data Streams (Aliran Data) di panel navigasi.
3. Pilih Kinesis data stream untuk melihat detailnya.
4. Pada halaman detail untuk streaming, pilihDitingkatkan fan-outTab.
5. Pilih konsumen untuk melihat nama, status, dan tanggal pendaftarannya.

Untuk membatalkan pendaftaran konsumen

1. Buka konsol Kinesis di https://console.aws.amazon.com/kinesis.
2. Pilih Data Streams (Aliran Data) di panel navigasi.
3. Pilih Kinesis data stream untuk melihat detailnya.
4. Pada halaman detail untuk streaming, pilihDitingkatkan fan-outTab.
5. Centang kotak di sebelah kiri nama setiap konsumen yang ingin Anda daftarkan.
6. PilihKonsumen deregister.

Migrasi Konsumen dari KCL 1.x ke KCL 2.x

Topik ini menjelaskan perbedaan antara versi 1.x dan 2.x Kinesis Client Library (KCL). Ini juga menunjukkan cara memigrasi konsumen Anda dari versi 1.x ke versi 2.x dari KCL. Setelah Anda memigrasikan klien Anda, itu akan mulai memproses catatan dari lokasi checkpointed terakhir.

Versi 2.0 dari KCL memperkenalkan perubahan antarmuka berikut:

Perubahan Antarmuka

Antarmuka KCL 1.x	Antarmuka 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`	Dilipat ke`software.amazon.kinesis.processor.ShardRecordProcessor`

Topik

Migrasi Prosesor Rekam

Contoh berikut menunjukkan prosesor rekaman diimplementasikan untuk 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();
        }
    }
```

```
}
```

Untuk memigrasi kelas prosesor rekaman

1. Mengubah antarmuka daricom.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcesso sebagai berikut:

```
// 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. Perbaruiimportuntukinitializedanprocessrecordsmetode.

```
// 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. Gantishutdownmetode dengan metode baru berikut:leaseLost,shardEnded, danshutdownRequested.

```
//    @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();
        }
```

```
    }
```

Berikut ini adalah versi terbaru dari kelas prosesor rekaman.

```
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();
        }
    }
}
```

Migrasi Pabrik Prosesor Rekam

Pabrik prosesor rekaman bertanggung jawab untuk membuat prosesor rekaman saat sewa diperoleh. Berikut ini adalah contoh pabrik 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();
    }
}
```

Untuk memigrasi pabrik prosesor rekaman

1. Ubah antarmuka yang diimplementasikan daricom.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcesso sebagai berikut.

   ```
   // 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. Ubah tanda tangan kembali untuk`createProcessor`.

```
// public IRecordProcessor createProcessor() {
public ShardRecordProcessor shardRecordProcessor() {
```

Berikut ini adalah contoh pabrik prosesor catatan di 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();
    }
}
```

Migrasi Pekerja

Dalam versi 2.0 dari KCL, kelas baru, disebut`Scheduler`, menggantikan`Worker`kelas. Berikut ini adalah contoh pekerja 1.x.

```
final KinesisClientLibConfiguration config = new KinesisClientLibConfiguration(...)
final IRecordProcessorFactory recordProcessorFactory = new RecordProcessorFactory();
final Worker worker = new Worker.Builder()
    .recordProcessorFactory(recordProcessorFactory)
    .config(config)
    .build();
```

Untuk memigrasikan pekerja

1. Mengubah`import`untuk`Worker`kelas untuk pernyataan impor untuk`Scheduler`dan`ConfigsBuilder`kelas.

```
// import com.amazonaws.services.kinesis.clientlibrary.lib.worker.Worker;
import software.amazon.kinesis.coordinator.Scheduler;
import software.amazon.kinesis.common.ConfigsBuilder;
```

2. Buat`ConfigsBuilder`dan a`Scheduler`seperti yang ditunjukkan dalam contoh berikut.

   Dianjurkan agar Anda menggunakan`KinesisClientUtil`untuk membuat`KinesisAsyncClient`dan untuk mengonfigurasi`maxConcurrency`di`KinesisAsyncClient`.

> **Important**
>
> Klien Amazon Kinesis mungkin melihat latensi yang meningkat secara signifikan, kecuali jika Anda mengonfigurasi`KinesisAsyncClient`untuk memiliki`maxConcurrency`cukup tinggi untuk memungkinkan semua sewa ditambah penggunaan tambahan`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()
    );
```

Mengkonfigurasi Klien Amazon Kinesis

Dengan rilis 2.0 dari Kinesis Client Library, konfigurasi klien dipindahkan dari kelas konfigurasi tunggal (`KinesisClientLibConfiguration`) ke enam kelas konfigurasi. Tabel berikut menjelaskan migrasi.

Bidang Konfigurasi dan Kelas Baru mereka

Bidang asli	Kelas Konfigurasi	Deskripsi
`applicationName`	`ConfigsBuilder`	Nama untuk ini aplikasi KCL. Digunakan sebagai default untuk`tableName` dan`consumerName` .
`tableName`	`ConfigsBuilder`	Memungkinkan mengganti nama tabel yang digunakan untuk tabel sewa Amazon DynamoDB.
`streamName`	`ConfigsBuilder`	Nama aliran yang diproses aplikasi ini catatan dari.
`kinesisEndpoint`	`ConfigsBuilder`	Opsi ini telah dihapus. Lihat Penghapusan Konfigurasi Klien.

Bidang asli	Kelas Konfigurasi	Deskripsi
dynamoDBEndpoint	ConfigsBuilder	Opsi ini telah dihapus. Lihat Penghapusan Konfigurasi Klien.
initialPositionInStreamExtended	RetrievalConfig	Lokasi di beling dari mana KCL mulai mengambil catatan, dimulai dengan menjalankan awal aplikasi.
kinesisCredentialsProvider	ConfigsBuilder	Opsi ini telah dihapus. Lihat Penghapusan Konfigurasi Klien.
dynamoDBCredentialsProvider	ConfigsBuilder	Opsi ini telah dihapus. Lihat Penghapusan Konfigurasi Klien.
cloudWatchCredentialsProvider	ConfigsBuilder	Opsi ini telah dihapus. Lihat Penghapusan Konfigurasi Klien.
failoverTimeMillis	LeaseManagementConfig	Jumlah milidetik yang harus lulus sebelum Anda dapat mempertimbangkan pemilik sewa telah gagal.
workerIdentifier	ConfigsBuilder	Pengenal unik yang mewakili instansiasi prosesor aplikasi ini. Ini harus unik.
shardSyncIntervalMillis	LeaseManagementConfig	Waktu antara panggilan sinkronisasi beling.
maxRecords	PollingConfig	Memungkinkan pengaturan jumlah maksimum catatan yang Kinesis kembali.
idleTimeBetweenReadsInMillis	CoordinatorConfig	Opsi ini telah dihapus. Lihat Penghapusan Waktu Idle.

Bidang asli	Kelas Konfigurasi	Deskripsi
`callProcessRecordsEvenForEmptyRecordList`	`ProcessorConfig`	Ketika diatur, prosesor rekaman disebut bahkan ketika tidak ada catatan yang disediakan dari Kinesis.
`parentShardPollIntervalMillis`	`CoordinatorConfig`	Seberapa sering prosesor rekaman harus memilih untuk melihat apakah pecahan induk telah selesai.
`cleanupLeasesUponShardCompletion`	`LeaseManagementConfig`	Ketika diatur, sewa akan dihapus segera setelah sewa anak telah mulai diproses.
`ignoreUnexpectedChildShards`	`LeaseManagementConfig`	Ketika diatur, pecahan anak yang memiliki pecahan terbuka diabaikan. Ini terutama untuk DynamoDB Streams.
`kinesisClientConfig`	`ConfigsBuilder`	Opsi ini telah dihapus. Lihat Penghapusan Konfigurasi Klien.
`dynamoDBClientConfig`	`ConfigsBuilder`	Opsi ini telah dihapus. Lihat Penghapusan Konfigurasi Klien.
`cloudWatchClientConfig`	`ConfigsBuilder`	Opsi ini telah dihapus. Lihat Penghapusan Konfigurasi Klien.
`taskBackoffTimeMillis`	`LifecycleConfig`	Waktu untuk menunggu untuk mencoba kembali tugas gagal.
`metricsBufferTimeMillis`	`MetricsConfig`	KendaliCloudWatchpenerbitan metrik.
`metricsMaxQueueSize`	`MetricsConfig`	KendaliCloudWatchpenerbitan metrik.

Bidang asli	Kelas Konfigurasi	Deskripsi
metricsLevel	MetricsConfig	KendaliCloudWatchpenerbitan metrik.
metricsEnabledDimensions	MetricsConfig	KendaliCloudWatchpenerbitan metrik.
validateSequenceNumberBeforeCheckpointing	CheckpointConfig	Opsi ini telah dihapus. Lihat Checkpoint Urutan Nomor Validasi.
regionName	ConfigsBuilder	Opsi ini telah dihapus. Lihat Penghapusan Konfigurasi Klien.
maxLeasesForWorker	LeaseManagementConfig	Jumlah maksimum sewa satu contoh dari aplikasi harus menerima.
maxLeasesToStealAtOneTime	LeaseManagementConfig	Jumlah maksimum sewa aplikasi harus mencoba untuk mencuri pada satu waktu.
initialLeaseTableReadCapacity	LeaseManagementConfig	DynamoDB membaca IOPS yang digunakan jika Kinesis Client Library perlu membuat tabel sewa DynamoDB baru.
initialLeaseTableWriteCapacity	LeaseManagementConfig	DynamoDB membaca IOPS yang digunakan jika Kinesis Client Library perlu membuat tabel sewa DynamoDB baru.
initialPositionInStreamExtended	LeaseManagementConfig	Posisi awal dalam aliran bahwa aplikasi harus dimulai pada. Ini hanya digunakan selama pembuatan sewa awal.

Bidang asli	Kelas Konfigurasi	Deskripsi
`skipShardSyncAtWorkerInitializationIfLeasesExist`	`CoordinatorConfig`	Nonaktifkan sinkronisasi data shard jika tabel sewa berisi sewa yang ada. TODO: KinesisEco-438
`shardPrioritization`	`CoordinatorConfig`	Yang shard prioritisasi untuk digunakan.
`shutdownGraceMillis`	N/A	Opsi ini telah dihapus. LihatMultiLangPenghapusan.
`timeoutInSeconds`	N/A	Opsi ini telah dihapus. LihatMultiLangPenghapusan.
`retryGetRecordsInSeconds`	`PollingConfig`	Mengkonfigurasi penundaan antaraGetRecordsupaya untuk kegagalan.
`maxGetRecordsThreadPool`	`PollingConfig`	Ukuran kolam benang yang digunakan untukGetRecords.
`maxLeaseRenewalThreads`	`LeaseManagementConfig`	Mengontrol ukuran kolam ulir penyewaan pembaruan. Semakin banyak sewa yang bisa diambil aplikasi Anda, semakin besar kolam ini seharusnya.
`recordsFetcherFactory`	`PollingConfig`	Memungkinkan mengganti pabrik yang digunakan untuk membuat fetcher yang mengambil dari aliran.
`logWarningForTaskAfterMillis`	`LifecycleConfig`	Berapa lama menunggu sebelum peringatan dicatat jika tugas belum selesai.

Bidang asli	Kelas Konfigurasi	Deskripsi
`listShardsBackoffTimeInMillis`	`RetrievalConfig`	Jumlah milidetik untuk menunggu antara panggilan ke`ListShards` saat terjadi kegagalan.
`maxListShardsRetryAttempts`	`RetrievalConfig`	Jumlah maksimum kali`ListShards` mencoba kembali sebelum menyerah.

Penghapusan Waktu

Dalam versi 1.x dari KCL,`idleTimeBetweenReadsInMillis`berhubungan dengan dua jumlah:

- Jumlah waktu antara pemeriksaan pengiriman tugas. Anda sekarang dapat mengkonfigurasi waktu ini antara tugas dengan menetapkan`CoordinatorConfig#shardConsumerDispatchPollIntervalMillis`.
- Jumlah waktu untuk tidur ketika tidak ada catatan dikembalikan dari Kinesis Data Streams. Dalam versi 2.0, dalam catatan fan-out ditingkatkan didorong dari retriever masing-masing. Aktivitas pada konsumen beling hanya terjadi ketika permintaan didorong tiba.

Penghapusan Konfigurasi

Dalam versi 2.0, KCL tidak lagi menciptakan klien. Hal ini tergantung pada pengguna untuk menyediakan klien yang valid. Dengan perubahan ini, semua parameter konfigurasi yang mengendalikan pembuatan klien telah dihapus. Jika Anda membutuhkan parameter ini, Anda dapat mengaturnya pada klien sebelum memberikan klien untuk`ConfigsBuilder`.

Bidang Dihapus	Konfigurasi Setara
`kinesisEndpoint`	Konfigurasikan SDK`KinesisAsyncClient` dengan titik akhir yang disukai:`KinesisAsyncClient.builder().endpointOverride(URI.create("https://<kinesis endpoint>")).build()` .

Bidang Dihapus	Konfigurasi Setara
`dynamoDBEndpoint`	Konfigurasikan SDK`DynamoDbAsyncClient` dengan titik akhir yang disukai:`DynamoDbAsyncClient.builder().endpointOverride(URI.create("https://<dynamodb endpoint>")).build()` .
`kinesisClientConfig`	Konfigurasikan SDK`KinesisAsyncClient` dengan konfigurasi yang dibutuhkan:`KinesisAsyncClient.builder().overrideConfiguration(<your configuration>).build()` .
`dynamoDBClientConfig`	Konfigurasikan SDK`DynamoDbAsyncClient` dengan konfigurasi yang dibutuhkan:`DynamoDbAsyncClient.builder().overrideConfiguration(<your configuration>).build()` .
`cloudWatchClientConfig`	Konfigurasikan SDK`CloudWatchAsyncClient` dengan konfigurasi yang dibutuhkan:`CloudWatchAsyncClient.builder().overrideConfiguration(<your configuration>).build()` .
`regionName`	Konfigurasikan SDK dengan Wilayah yang disukai. Ini sama untuk semua klien SDK. Misalnya, `KinesisAsyncClient.builder().region(Region.US_WEST_2).build()` .

Menggunakan AWS Layanan lain untuk membaca data dari Kinesis Data Streams

Berikut ini adalah daftar AWS layanan lain yang dapat langsung diintegrasikan dengan Kinesis Data Streams untuk membaca data Kinesis Data Streams:

Topik

- Menggunakan Amazon EMR
- Menggunakan EventBridge Pipa Amazon
- Menggunakan AWS Glue
- Menggunakan Amazon Redshift

Menggunakan Amazon EMR

Cluster EMR Amazon dapat membaca dan memproses aliran Amazon Kinesis secara langsung, menggunakan alat yang sudah dikenal di ekosistem Hadoop seperti Hive, Pig,, Hadoop Streaming API, dan Cascading. MapReduce Anda juga dapat menggabungkan dengan data waktu nyata dari Amazon Kinesis dengan data yang ada di Amazon S3, Amazon DynamoDB, dan HDFS dalam sebuah klaster yang berjalan. Anda dapat langsung memuat data dari Amazon EMR ke Amazon S3 atau DynamoDB untuk kegiatan setelah pemrosesan.

Untuk informasi selengkapnya, lihat Amazon Kinesis di Panduan Rilis EMR Amazon.

Menggunakan EventBridge Pipa Amazon

Amazon EventBridge Pipes mendukung Amazon Kinesis Data Streams sebagai sumber. Amazon EventBridge Pipes membantu Anda membuat point-to-point integrasi antara produsen acara dan konsumen dengan langkah transformasi, filter, dan pengayaan opsional. Anda dapat menggunakan EventBridge Pipes untuk menerima catatan dalam Aliran Data Kinesis dan secara opsional memfilter atau menyempurnakan catatan ini sebelum mengirimnya ke salah satu tujuan yang tersedia untuk diproses, termasuk Kinesis Data Streams.

Untuk informasi selengkapnya, lihat aliran Amazon Kinesis sebagai sumber di Panduan EventBridge Rilis Amazon.

Menggunakan AWS Glue

Dengan menggunakan AWS Glue streaming ETL, Anda dapat membuat pekerjaan ekstrak, transformasi, dan pemuatan streaming (ETL) yang berjalan terus menerus dan menggunakan data dari Amazon Kinesis Data Streams. Tugas tersebut membersihkan dan mengubah data, dan kemudian memuat hasil ke danau data Amazon S3 atau penyimpanan data JDBC.

Untuk informasi selengkapnya, lihat pekerjaan Streaming ETL AWS Glue di Panduan AWS Glue Rilis.

Menggunakan Amazon Redshift

Amazon Redshift mendukung konsumsi streaming dari Amazon Kinesis Data Streams. Fitur konsumsi streaming Amazon Redshift menyediakan latensi rendah, konsumsi data streaming berkecepatan tinggi dari Amazon Kinesis Data Streams ke tampilan terwujud Amazon Redshift. Konsumsi streaming Amazon Redshift menghilangkan kebutuhan untuk mementaskan data di Amazon S3 sebelum menelan Amazon Redshift.

Untuk informasi selengkapnya, lihat Konsumsi streaming di Panduan Rilis Amazon Redshift.

Menggunakan integrasi pihak ketiga

Anda dapat membaca data dari aliran data Amazon Kinesis Data Streams menggunakan salah satu opsi pihak ketiga berikut yang terintegrasi dengan Kinesis Data Streams:

Topik

- Apache Flink
- Platform Pengalaman Adobe
- Apache Druid
- Apache Spark
- Databricks
- Platform Konfluen Kafka
- Kinesumer
- Talend

Apache Flink

Apache Flink adalah kerangka kerja dan mesin pemrosesan terdistribusi untuk perhitungan stateful melalui aliran data tak terbatas dan terbatas. Untuk informasi selengkapnya tentang penggunaan Kinesis Data Streams menggunakan Apache Flink, lihat Konektor Amazon Kinesis Data Streams.

Platform Pengalaman Adobe

Adobe Experience Platform memungkinkan organisasi untuk memusatkan dan menstandarisasi data pelanggan dari sistem apa pun. Ini kemudian menerapkan ilmu data dan pembelajaran mesin untuk secara dramatis meningkatkan desain dan penyampaian pengalaman yang kaya dan dipersonalisasi. Untuk informasi selengkapnya tentang penggunaan aliran data Kinesis menggunakan Adobe Experience Platform, lihat konektor Amazon Kinesis.

Apache Druid

Druid adalah database analitik real-time berkinerja tinggi yang memberikan kueri sub-detik pada streaming dan data batch pada skala dan di bawah beban. Untuk informasi lebih lanjut tentang menelan aliran data Kinesis menggunakan Apache Druid, lihat konsumsi Amazon Kinesis.

Apache Spark

Apache Spark adalah mesin analitik terpadu untuk pemrosesan data skala besar. Ini menyediakan API tingkat tinggi di Java, Scala, Python dan R, dan mesin yang dioptimalkan yang mendukung grafik eksekusi umum. Anda dapat menggunakan Apache Spark untuk membangun aplikasi pemrosesan aliran yang menggunakan data dalam aliran data Kinesis Anda.

Untuk menggunakan aliran data Kinesis menggunakan Apache Spark Structured Streaming, gunakan konektor Amazon Kinesis Data Streams. Konektor ini mendukung konsumsi dengan Enhanced Fan-Out, yang menyediakan aplikasi Anda dengan throughput baca khusus hingga 2 MB data per detik per pecahan. Untuk informasi selengkapnya, lihat Mengembangkan Konsumen Kustom dengan Throughput Khusus (Enhanced Fan-Out).

Untuk menggunakan aliran data Kinesis menggunakan Spark Streaming, lihat Spark Streaming + Integrasi Kinesis.

Databricks

Databricks adalah platform berbasis cloud yang menyediakan lingkungan kolaboratif untuk rekayasa data, ilmu data, dan pembelajaran mesin. Untuk informasi selengkapnya tentang penggunaan aliran data Kinesis menggunakan Databricks, lihat Connect to Amazon Kinesis.

Platform Konfluen Kafka

Platform Confluent dibangun di atas Kafka dan menyediakan fitur dan fungsionalitas tambahan yang membantu perusahaan membangun dan mengelola saluran data real-time dan aplikasi streaming. Untuk informasi selengkapnya tentang penggunaan aliran data Kinesis menggunakan Platform Confluent, lihat Konektor Sumber Amazon Kinesis untuk Platform Confluent.

Kinesumer

Kinesumer adalah klien Go yang menerapkan klien grup konsumen terdistribusi sisi klien untuk aliran data Kinesis. Untuk informasi selengkapnya, lihat repositori Kinesumer Github.

Talend

Talend adalah perangkat lunak integrasi dan manajemen data yang memungkinkan pengguna untuk mengumpulkan, mengubah, dan menghubungkan data dari berbagai sumber dengan cara yang terukur dan efisien. Untuk informasi selengkapnya tentang penggunaan aliran data Kinesis menggunakan Talend, lihat Connect talend ke aliran Amazon Kinesis.

Pemecahan Masalah Kinesis Data Streams Konsumen

Bagian berikut menawarkan solusi untuk beberapa masalah umum yang mungkin Anda temukan saat bekerja dengan konsumen Amazon Kinesis Data Streams.

Beberapa Catatan Aliran Data Kinesis Dilewati Saat Menggunakan Perpustakaan Klien Kinesis

Penyebab paling umum dari catatan yang dilewati adalah pengecualian yang tidak tertangani yang dilemparkan. `processRecords` Perpustakaan Klien Kinesis (KCL) bergantung pada `processRecords` kode Anda untuk menangani pengecualian apa pun yang muncul dari pemrosesan catatan data. Setiap pengecualian yang dilemparkan `processRecords` diserap oleh KCL. Untuk menghindari percobaan ulang tak terbatas pada kegagalan berulang, KCL tidak mengirim ulang batch catatan yang diproses pada saat pengecualian. KCL kemudian memanggil `processRecords` batch rekaman data berikutnya tanpa memulai ulang prosesor rekaman. Ini secara efektif menghasilkan aplikasi konsumen yang mengamati catatan yang dilewati. Untuk mencegah catatan yang dilewati, tangani semua pengecualian di dalamnya `processRecords` dengan tepat.

Catatan Milik Pecahan yang Sama Diproses oleh Prosesor Rekaman yang Berbeda pada Saat yang Sama

Untuk aplikasi Kinesis Client Library (KCL) yang berjalan, pecahan hanya memiliki satu pemilik. Namun, beberapa prosesor rekaman dapat memproses pecahan yang sama untuk sementara. Dalam kasus instance pekerja yang kehilangan konektivitas jaringan, KCL mengasumsikan bahwa

pekerja yang tidak dapat dijangkau tidak lagi memproses catatan, setelah waktu failover berakhir, dan mengarahkan instance pekerja lain untuk mengambil alih. Untuk waktu yang singkat, prosesor rekaman baru dan prosesor rekaman dari pekerja yang tidak terjangkau dapat memproses data dari pecahan yang sama.

Anda harus menetapkan waktu failover yang sesuai untuk aplikasi Anda. Untuk aplikasi latensi rendah, default 10 detik mungkin mewakili waktu maksimum yang ingin Anda tunggu. Namun, dalam kasus di mana Anda mengharapkan masalah konektivitas seperti melakukan panggilan di seluruh wilayah geografis di mana konektivitas dapat hilang lebih sering, jumlah ini mungkin terlalu rendah.

Aplikasi Anda harus mengantisipasi dan menangani skenario ini, terutama karena konektivitas jaringan biasanya dikembalikan ke pekerja yang sebelumnya tidak dapat dijangkau. Jika prosesor rekaman memiliki pecahan yang diambil oleh prosesor rekaman lain, ia harus menangani dua kasus berikut untuk melakukan shutdown yang anggun:

1. Setelah panggilan saat ini `processRecords` selesai, KCL memanggil metode shutdown pada prosesor rekaman dengan alasan shutdown 'ZOMBIE'. Prosesor rekaman Anda diharapkan untuk membersihkan sumber daya apa pun yang sesuai dan kemudian keluar.
2. Ketika Anda mencoba untuk memeriksa dari pekerja 'zombie', KCL melempar. `ShutdownException` Setelah menerima pengecualian ini, kode Anda diharapkan keluar dari metode saat ini dengan bersih.

Untuk informasi selengkapnya, lihat Menangani Catatan Duplikat.

Aplikasi Konsumen Membaca pada Tingkat Lebih Lambat Dari yang Diharapkan

Alasan paling umum untuk throughput baca lebih lambat dari yang diharapkan adalah sebagai berikut:

1. Beberapa aplikasi konsumen memiliki total pembacaan melebihi batas per pecahan. Untuk informasi selengkapnya, lihat Kuota dan Batas. Dalam hal ini, tingkatkan jumlah pecahan dalam aliran data Kinesis.
2. Batas yang menentukan jumlah maksimum GetRecords per panggilan mungkin telah dikonfigurasi dengan nilai rendah. Jika Anda menggunakan KCL, Anda mungkin telah mengonfigurasi pekerja dengan nilai rendah untuk `maxRecords` properti tersebut. Secara umum, kami sarankan menggunakan default sistem untuk properti ini.

3. Logika di dalam `processRecords` panggilan Anda mungkin memakan waktu lebih lama dari yang diharapkan karena sejumlah alasan yang mungkin; logikanya mungkin intensif CPU, pemblokiran I/O, atau macet pada sinkronisasi. Untuk menguji apakah ini benar, uji coba jalankan prosesor rekaman kosong dan bandingkan throughput baca. Untuk informasi tentang cara mengikuti data yang masuk, lihatResharding, Scaling, dan Pengolahan Paralel.

Jika Anda hanya memiliki satu aplikasi konsumen, selalu mungkin untuk membaca setidaknya dua kali lebih cepat dari tarif put. Itu karena Anda dapat menulis hingga 1.000 catatan per detik untuk menulis, hingga total tingkat penulisan data maksimum 1 MB per detik (termasuk kunci partisi). Setiap pecahan terbuka dapat mendukung hingga 5 transaksi per detik untuk pembacaan, hingga total kecepatan baca data maksimum 2 MB per detik. Perhatikan bahwa setiap pembacaan (GetRecordspanggilan) mendapat sekumpulan catatan. Ukuran data yang dikembalikan GetRecords bervariasi tergantung pada pemanfaatan pecahan. Ukuran maksimum data yang GetRecords dapat dikembalikan adalah 10 MB. Jika panggilan mengembalikan batas itu, panggilan berikutnya dilakukan dalam 5 detik berikutnya`ProvisionedThroughputExceededException`.

GetRecords Mengembalikan Array Catatan Kosong Bahkan Ketika Ada Data di Stream

Mengkonsumsi, atau mendapatkan catatan adalah model tarik. Pengembang diharapkan untuk memanggil GetRecordsdalam loop kontinu tanpa back-off. Setiap panggilan untuk GetRecords juga mengembalikan `ShardIterator` nilai, yang harus digunakan dalam iterasi berikutnya dari loop.

GetRecordsOperasi tidak memblokir. Sebaliknya, ia segera kembali; dengan catatan data yang relevan atau dengan `Records` elemen kosong. `Records`Elemen kosong dikembalikan dalam dua kondisi:

1. Tidak ada lagi data saat ini di pecahan.
2. Tidak ada data di dekat bagian pecahan yang ditunjukkan oleh. `ShardIterator`

Kondisi terakhir tidak kentara, tetapi merupakan tradeoff desain yang diperlukan untuk menghindari waktu pencarian (latensi) tak terbatas saat mengambil catatan. Dengan demikian, aplikasi yang memakan streaming harus mengulang dan memanggilGetRecords, menangani catatan kosong sebagai hal yang biasa.

Dalam skenario produksi, satu-satunya waktu loop kontinu harus keluar adalah ketika `NextShardIterator` nilainya. `NULL` Kapan`NULL`, `NextShardIterator` itu berarti bahwa pecahan

saat ini telah ditutup dan `ShardIterator` nilainya akan melewati rekor terakhir. Jika aplikasi yang mengkonsumsi tidak pernah memanggil SplitShard atauMergeShards, pecahan tetap terbuka dan panggilan untuk GetRecords tidak pernah mengembalikan `NextShardIterator` nilai yang ada`NULL`.

Jika Anda menggunakan Kinesis Client Library (KCL), pola konsumsi di atas diabstraksikan untuk Anda. Ini termasuk penanganan otomatis satu set pecahan yang berubah secara dinamis. Dengan KCL, pengembang hanya memasok logika untuk memproses catatan yang masuk. Ini dimungkinkan karena perpustakaan membuat panggilan terus menerus GetRecords untuk Anda.

Shard Iterator Kedaluwarsa Tanpa Diduga

Sebuah iterator shard baru dikembalikan oleh setiap GetRecords permintaan (as`NextShardIterator`), yang kemudian Anda gunakan dalam GetRecords permintaan berikutnya (as`ShardIterator`). Biasanya, iterator pecahan ini tidak kedaluwarsa sebelum Anda menggunakannya. Namun, Anda mungkin menemukan bahwa iterator shard kedaluwarsa karena Anda belum menelepon GetRecords lebih dari 5 menit, atau karena Anda telah melakukan restart aplikasi konsumen Anda.

Jika iterator shard segera kedaluwarsa, sebelum Anda dapat menggunakannya, ini mungkin menunjukkan bahwa tabel DynamoDB yang digunakan oleh Kinesis tidak memiliki kapasitas yang cukup untuk menyimpan data sewa. Situasi ini lebih mungkin terjadi jika Anda memiliki sejumlah besar pecahan. Untuk mengatasi masalah ini, tingkatkan kapasitas tulis yang ditetapkan ke tabel pecahan. Untuk informasi selengkapnya, lihat Menggunakan Meja Sewa untuk Melacak Pecahan yang Diproses oleh Aplikasi Konsumen KCL.

Pemrosesan Rekor Konsumen Tertinggal

Untuk sebagian besar kasus penggunaan, aplikasi konsumen membaca data terbaru dari aliran. Dalam keadaan tertentu, bacaan konsumen mungkin tertinggal, yang mungkin tidak diinginkan. Setelah Anda mengidentifikasi seberapa jauh di belakang konsumen Anda membaca, lihat alasan paling umum mengapa konsumen tertinggal.

Mulailah dengan `GetRecords.IteratorAgeMilliseconds` metrik, yang melacak posisi baca di semua pecahan dan konsumen di aliran. Perhatikan bahwa jika usia iterator melewati 50% dari periode retensi (secara default, 24 jam, dapat dikonfigurasi hingga 365 hari), ada risiko kehilangan data karena kedaluwarsa rekaman. Solusi sementara cepat adalah meningkatkan periode retensi. Ini menghentikan hilangnya data penting saat Anda memecahkan masalah lebih lanjut. Untuk

informasi selengkapnya, lihat Memantau Layanan Amazon Kinesis Data Streams dengan Amazon CloudWatch. Selanjutnya, identifikasi seberapa jauh di belakang aplikasi konsumen Anda membaca dari setiap pecahan menggunakan CloudWatch metrik khusus yang dipancarkan oleh Kinesis Client Library (KCL),. `MillisBehindLatest` Untuk informasi selengkapnya, lihat Memantau Perpustakaan Klien Kinesis dengan Amazon CloudWatch.

Berikut adalah alasan paling umum konsumen dapat tertinggal:

- Peningkatan besar yang tiba-tiba ke `GetRecords.IteratorAgeMilliseconds` atau `MillisBehindLatest` biasanya menunjukkan masalah sementara, seperti kegagalan operasi API ke aplikasi hilir. Anda harus menyelidiki peningkatan mendadak ini jika salah satu metrik secara konsisten menampilkan perilaku ini.
- Peningkatan bertahap pada metrik ini menunjukkan bahwa konsumen tidak mengikuti aliran karena tidak memproses catatan dengan cukup cepat. Akar penyebab paling umum untuk perilaku ini adalah sumber daya fisik yang tidak mencukupi atau logika pemrosesan catatan yang belum diskalakan dengan peningkatan throughput aliran. Anda dapat memverifikasi perilaku ini dengan melihat CloudWatch metrik kustom lain yang dipancarkan KCL terkait dengan `processTask` operasi, termasuk`RecordProcessor.processRecords.Time`,, dan. `Success` `RecordsProcessed`
  - Jika Anda melihat peningkatan `processRecords.Time` metrik yang berkorelasi dengan peningkatan throughput, Anda harus menganalisis logika pemrosesan catatan Anda untuk mengidentifikasi mengapa metrik tidak menskalakan dengan peningkatan throughput.
  - Jika Anda melihat peningkatan `processRecords.Time` nilai yang tidak berkorelasi dengan peningkatan throughput, periksa untuk melihat apakah Anda melakukan panggilan pemblokiran di jalur kritis, yang sering menjadi penyebab perlambatan dalam pemrosesan catatan. Pendekatan alternatif adalah meningkatkan paralelisme Anda dengan meningkatkan jumlah pecahan. Terakhir, konfirmasikan bahwa Anda memiliki jumlah sumber daya fisik yang memadai (memori, pemanfaatan CPU, dll.) Pada node pemrosesan yang mendasarinya selama permintaan puncak.

Kesalahan izin kunci master KMS yang tidak sah

Kesalahan ini terjadi ketika aplikasi konsumen membaca dari aliran terenkripsi tanpa izin pada kunci master KMS. Untuk menetapkan izin ke aplikasi untuk mengakses kunci KMS, lihat Menggunakan Kebijakan Utama di KMS dan Menggunakan Kebijakan IAM dengan AWS KMS. AWS

Masalah umum, pertanyaan, dan ide pemecahan masalah bagi konsumen

- Mengapa pemicu Kinesis Data Streams tidak dapat menjalankan fungsi Lambda saya?
- Bagaimana cara mendeteksi dan memecahkan masalah ReadProvisionedThroughputExceeded pengecualian di Kinesis Data Streams?
- Mengapa saya mengalami masalah latensi tinggi dengan Kinesis Data Streams?
- Mengapa aliran data Kinesis saya mengembalikan Kesalahan Server Internal 500?
- Bagaimana cara memecahkan masalah aplikasi KCL yang diblokir atau macet untuk Kinesis Data Streams?
- Dapatkah saya menggunakan aplikasi Amazon Kinesis Client Library yang berbeda dengan tabel Amazon DynamoDB yang sama?

Topik Lanjutan untuk Konsumen Amazon Kinesis Data Streams

Pelajari cara mengoptimalkan konsumen Amazon Kinesis Data Streams Anda.

Isi

- Pengolahan Latensi Rendah
- Menggunakan AWS Lambda dengan Kinesis Producer Library
- Resharding, Scaling, dan Pengolahan Paralel
- Menangani Catatan Duplikat
- Menangani Startup, Shutdown, dan Throttling

Pengolahan Latensi Rendah

Penundaan propagasididefinisikan sebagaiend-to-endlatensi dari saat catatan ditulis ke sungai sampai dibaca oleh aplikasi konsumen. Penundaan ini bervariasi tergantung pada sejumlah faktor, tetapi terutama dipengaruhi oleh interval pemungutan suara aplikasi konsumen.

Untuk sebagian besar aplikasi, kami merekomendasikan pemungutan suara setiap pecahan satu kali per detik per aplikasi. Hal ini memungkinkan Anda untuk memiliki beberapa aplikasi konsumen yang memproses stream secara bersamaan tanpa menekan batas Amazon Kinesis Data Streams 5`GetRecords`per detik. Selain itu, memproses batch data yang lebih besar cenderung lebih efisien dalam mengurangi jaringan dan latensi hilir lainnya dalam aplikasi Anda.

Default KCL diatur untuk mengikuti praktik terbaik pemungutan suara setiap 1 detik. Default ini menghasilkan penundaan propagasi rata-rata yang biasanya di bawah 1 detik.

Catatan Kinesis Data Streams tersedia untuk dibaca segera setelah ditulis. Ada beberapa kasus penggunaan yang perlu mengambil keuntungan dari ini dan memerlukan mengkonsumsi data dari aliran segera setelah tersedia. Anda dapat secara signifikan mengurangi penundaan propagasi dengan mengesampingkan pengaturan default KCL untuk lebih sering, seperti yang ditunjukkan dalam contoh berikut.

Java KCL kode konfigurasi:

```
kinesisClientLibConfiguration = new
        KinesisClientLibConfiguration(applicationName,
        streamName,
        credentialsProvider,

 workerId).withInitialPositionInStream(initialPositionInStream).withIdleTimeBetweenReadsInMilli
```

Pengaturan file properti untuk Python dan Ruby KCL:

```
idleTimeBetweenReadsInMillis = 250
```

> **Note**
>
> Karena Kinesis Data Streams memiliki batas 5`GetRecords`panggilan per detik, per beling, pengaturan`idleTimeBetweenReadsInMillis`properti yang lebih rendah dari 200ms dapat mengakibatkan aplikasi Anda mengamati`ProvisionedThroughputExceededException`pengecualian. Terlalu banyak pengecualian ini dapat mengakibatkan back-off eksponensial dan dengan demikian menyebabkan latensi tak terduga yang signifikan dalam pemrosesan. Jika Anda mengatur properti ini berada di atau tepat di atas 200 ms dan memiliki lebih dari satu aplikasi pemrosesan, Anda akan mengalami throttling serupa.

Menggunakan AWS Lambda dengan Kinesis Producer Library

Kinesis Producer Library (KPL) menggabungkan catatan kecil yang diformat pengguna ke dalam catatan yang lebih besar hingga 1 MB untuk memanfaatkan throughput Amazon Kinesis Data Streams dengan lebih baik. Sementara KCL untuk Java mendukung deaggregating catatan ini,

Anda perlu menggunakan modul khusus untuk deaggregate record saat menggunakanAWS Lambdasebagai konsumen aliran Anda. Anda dapat memperoleh kode proyek yang diperlukan dan instruksi dariGitHubpadaModul Deaggregasi Perpustakaan Produser Kinesis untukAWSLambda. Komponen dalam proyek ini memberi Anda kemampuan untuk memproses data serial KPL dalamAWS Lambda, di Jawa, Node.js dan Python. Komponen-komponen ini juga dapat digunakan sebagai bagian dariaplikasi KCL multi-lang.

Resharding, Scaling, dan Pengolahan Paralel

Reshardingmemungkinkan Anda untuk menambah atau mengurangi jumlah pecahan dalam aliran untuk beradaptasi dengan perubahan laju data yang mengalir melalui aliran. Resharding biasanya dilakukan oleh aplikasi administratif yang memonitor metrik penanganan data shard. Meskipun KCL sendiri tidak memulai operasi resharding, itu dirancang untuk beradaptasi dengan perubahan jumlah pecahan yang dihasilkan dari resharding.

Seperti dicatat dalamMenggunakan Meja Sewa untuk Melacak Pecahan yang Diproses oleh Aplikasi Konsumen KCL, KCL melacak pecahan dalam aliran menggunakan tabel Amazon DynamoDB. Ketika pecahan baru dibuat sebagai hasil dari resharding, KCL menemukan pecahan baru dan mengisi baris baru dalam tabel. Para pekerja secara otomatis menemukan pecahan baru dan membuat prosesor untuk menangani data dari mereka. KCL juga mendistribusikan pecahan di sungai di semua pekerja yang tersedia dan prosesor rekaman.

KCL memastikan bahwa setiap data yang ada di pecahan sebelum resharding diproses terlebih dahulu. Setelah data yang telah diproses, data dari pecahan baru dikirim ke prosesor rekaman. Dengan cara ini, KCL mempertahankan urutan di mana catatan data ditambahkan ke aliran untuk kunci partisi tertentu.

Contoh: Resharding, Scaling, dan Pengolahan Paralel

Contoh berikut menggambarkan cara KCL membantu Anda menangani penskalaan dan resharding:

- Misalnya, jika aplikasi Anda berjalan pada satu instans EC2, dan sedang memproses satu aliran data Kinesis yang memiliki empat pecahan. Contoh yang satu ini memiliki satu pekerja KCL dan empat prosesor rekaman (satu prosesor rekaman untuk setiap pecahan). Keempat prosesor rekaman ini berjalan secara paralel dalam proses yang sama.
- Selanjutnya, jika Anda menskalakan aplikasi untuk menggunakan instance lain, Anda memiliki dua instance yang memproses satu aliran yang memiliki empat pecahan. Ketika pekerja KCL dimulai pada instance kedua, itu load-balance dengan instance pertama, sehingga setiap instance sekarang memproses dua pecahan.

- Jika Anda kemudian memutuskan untuk membagi empat pecahan menjadi lima pecahan. KCL kembali mengkoordinasikan pemrosesan lintas instance: satu instans memproses tiga pecahan, dan yang lainnya memproses dua pecahan. Koordinasi serupa terjadi ketika Anda menggabungkan pecahan.

Biasanya, ketika Anda menggunakan KCL, Anda harus memastikan bahwa jumlah instance tidak melebihi jumlah pecahan (kecuali untuk tujuan siaga kegagalan). Setiap pecahan diproses oleh salah satu pekerja KCL dan memiliki satu prosesor rekaman yang sesuai, sehingga Anda tidak perlu beberapa contoh untuk memproses satu pecahan. Namun, satu pekerja dapat memproses sejumlah pecahan, jadi tidak apa-apa jika jumlah pecahan melebihi jumlah instance.

Untuk meningkatkan pemrosesan dalam aplikasi Anda, Anda harus menguji kombinasi dari pendekatan ini:

- Meningkatkan ukuran instance (karena semua prosesor rekaman berjalan secara paralel dalam suatu proses)
- Meningkatkan jumlah instance hingga jumlah maksimum pecahan terbuka (karena pecahan dapat diproses secara independen)
- Meningkatkan jumlah pecahan (yang meningkatkan tingkat paralelisme)

Perhatikan bahwa Anda dapat menggunakan Auto Scaling untuk secara otomatis menskalakan instans berdasarkan metrik yang sesuai. Untuk informasi lebih lanjut, lihat Panduan Pengguna Amazon EC2 Auto Scaling.

Ketika resharding meningkatkan jumlah pecahan di sungai, peningkatan yang sesuai dalam jumlah prosesor rekaman meningkatkan beban pada instans EC2 yang hosting mereka. Jika instance adalah bagian dari grup Auto Scaling, dan beban meningkat cukup, grup Auto Scaling menambahkan lebih banyak instance untuk menangani peningkatan beban. Anda harus mengkonfigurasi instans Anda untuk meluncurkan aplikasi Amazon Kinesis Data Streams saat startup, sehingga pekerja tambahan dan prosesor rekaman menjadi aktif pada instans baru segera.

Untuk informasi selengkapnya tentang resharding, lihatMembagikan Ulang Aliran.

Menangani Catatan Duplikat

Ada dua alasan utama mengapa catatan dapat dikirimkan lebih dari satu kali ke aplikasi Amazon Kinesis Data Streams Anda: percobaan ulang produsen dan percobaan ulang konsumen. Aplikasi

Anda harus mengantisipasi dan menangani pemrosesan catatan individual dengan tepat beberapa kali.

Coba Ulang Produsen

Pertimbangkan produser yang mengalami batas waktu yang berhubungan dengan jaringan setelah melakukan panggilan`PutRecord`, tapi sebelum dapat menerima pengakuan dari Amazon Kinesis Data Streams. Produser tidak dapat memastikan apakah catatan dikirim ke Kinesis Data Streams. Dengan asumsi bahwa setiap catatan penting untuk aplikasi, produser akan ditulis untuk mencoba kembali panggilan dengan data yang sama. Jika keduanya`PutRecord`panggilan pada data yang sama berhasil berkomitmen untuk Kinesis Data Streams, maka akan ada dua data Kinesis Data Streams. Meskipun dua catatan memiliki data yang identik, mereka juga memiliki nomor urut yang unik. Aplikasi yang membutuhkan jaminan ketat harus menanamkan kunci utama dalam catatan untuk menghapus duplikat nanti saat memproses. Perhatikan bahwa jumlah duplikat karena percobaan ulang produsen biasanya rendah dibandingkan dengan jumlah duplikat karena retries konsumen.

> **Note**
>
> Jika Anda menggunakanAWSSDK`PutRecord`, defaultkonfigurasi ulang gagal`PutRecord`panggilan hingga tiga kali.

Coba Ulang Konsumen

Konsumen (aplikasi pengolahan data) retries terjadi ketika prosesor rekaman restart. Rekam prosesor untuk restart pecahan yang sama dalam kasus berikut:

1. Seorang pekerja berakhir secara tak terduga
2. Instans pekerja ditambahkan atau dihapus
3. Pecahan digabungkan atau dibagi
4. Aplikasi ini dikerahkan

Dalam semua kasus ini,shards-to-worker-to-record-processorpemetaan terus diperbarui untuk memuat pengolahan keseimbangan. Prosesor Shard yang dimigrasi ke instance lain me-restart catatan pemrosesan dari pos pemeriksaan terakhir. Ini menghasilkan pemrosesan rekaman duplikasi seperti yang ditunjukkan pada contoh di bawah ini. Untuk informasi selengkapnya tentang penyeimbang beban, lihatResharding, Scaling, dan Pengolahan Paralel.

Contoh: Percobaan Ulang Konsumen Menghasilkan Records Record

Dalam contoh ini, Anda memiliki aplikasi yang terus membaca catatan dari aliran, menggabungkan rekaman ke file lokal, dan mengunggah file ke Amazon S3. Untuk kesederhanaan, asumsikan hanya ada 1 beling dan 1 pekerja memproses beling. Pertimbangkan contoh berikut urutan peristiwa, dengan asumsi bahwa pos pemeriksaan terakhir adalah pada catatan nomor 10000:

1. Seorang pekerja membaca batch berikutnya catatan dari pecahan, mencatat 10001 untuk 20000.
2. Pekerja kemudian melewati batch catatan ke prosesor rekaman terkait.
3. Prosesor rekaman menggabungkan data, membuat file Amazon S3, dan mengunggah file ke Amazon S3 dengan sukses.
4. Pekerja berakhir secara tak terduga sebelum pos pemeriksaan baru dapat terjadi.
5. Aplikasi, pekerja, dan merekam prosesor restart.
6. Pekerja sekarang mulai membaca dari pos pemeriksaan terakhir yang berhasil, dalam hal ini 10001.

Dengan demikian, catatan 10001-20000 dikonsumsi lebih dari satu kali.

Menjadi Tangguh untuk Pengulangan Konsumen

Meskipun catatan dapat diproses lebih dari satu kali, aplikasi Anda mungkin ingin menyajikan efek samping seolah-olah catatan diproses hanya satu kali (pengolahan idempoten). Solusi untuk masalah ini bervariasi dalam kompleksitas dan akurasi. Jika tujuan data akhir dapat menangani duplikat dengan baik, sebaiknya mengandalkan tujuan akhir untuk mencapai pemrosesan idempoten. Misalnya, dengan OpenSearch Anda dapat menggunakan kombinasi versi dan ID unik untuk mencegah pemrosesan duplikat.

Pada contoh aplikasi di bagian sebelumnya, aplikasi ini terus membaca catatan dari stream, menggabungkan rekaman ke file lokal, dan mengunggah file ke Amazon S3. Seperti yang diilustrasikan, catatan 10001 -20000 dikonsumsi lebih dari satu kali menghasilkan beberapa file Amazon S3 dengan data yang sama. Salah satu cara untuk mengurangi duplikat dari contoh ini adalah untuk memastikan bahwa langkah 3 menggunakan skema berikut:

1. Rekam Prosesor menggunakan jumlah rekaman tetap per file Amazon S3, seperti 5000.
2. Nama file menggunakan skema ini: Awalan Amazon S3, shard-id, dan `First-Sequence-Num`. Dalam hal ini, bisa jadi sesuatu seperti `sample-shard000001-10001`.

3. Setelah Anda mengunggah file Amazon S3, pos pemeriksaan dengan menentukan`Last-Sequence-Num`. Dalam hal ini, Anda akan memeriksa nomor catatan 15000.

Dengan skema ini, bahkan jika catatan diproses lebih dari satu kali, file Amazon S3 yang dihasilkan memiliki nama yang sama dan memiliki data yang sama. Coba ulang hanya menghasilkan penulisan data yang sama ke file yang sama lebih dari satu kali.

Dalam kasus operasi reshard, jumlah catatan yang tersisa di beling mungkin kurang dari jumlah tetap yang Anda inginkan yang diperlukan. Dalam hal ini,`shutdown()`Metode harus menyiram file ke Amazon S3 dan pos pemeriksaan pada nomor urutan terakhir. Skema di atas kompatibel dengan operasi reshard juga.

Menangani Startup, Shutdown, dan Throttling

Berikut adalah beberapa pertimbangan tambahan untuk dimasukkan ke dalam desain aplikasi Amazon Kinesis Data Streams Anda.

Isi

Memulai Produsen Data dan Konsumen Data

Secara default, KCL mulai membaca catatan dari ujung sungai, yang merupakan catatan yang paling baru ditambahkan. Dalam konfigurasi ini, jika aplikasi penghasil data menambahkan catatan ke sungai sebelum prosesor rekaman menerima berjalan, catatan tidak dibaca oleh prosesor rekaman setelah mereka memulai.

Untuk mengubah perilaku prosesor rekaman sehingga selalu membaca data dari awal aliran, tetapkan nilai berikut dalam file properti untuk aplikasi Amazon Kinesis Data Streams Anda:

```
initialPositionInStream = TRIM_HORIZON
```

Secara default, Amazon Kinesis Data Streams menyimpan semua data selama 24 jam. Ini juga mendukung retensi yang diperpanjang hingga 7 hari dan retensi jangka panjang hingga 365 hari. Kerangka waktu ini disebutperiode retensi. Mengatur posisi awal`TRIM_HORIZON`akan memulai

prosesor rekaman dengan data tertua di sungai, seperti yang didefinisikan oleh periode retensi. Bahkan dengan`TRIM_HORIZON`pengaturan, jika prosesor rekaman dimulai setelah waktu yang lebih besar telah berlalu daripada periode retensi, maka beberapa catatan di sungai tidak akan lagi tersedia. Untuk alasan ini, Anda harus selalu memiliki aplikasi konsumen yang membaca dari aliran dan menggunakanCloudWatchmetris`GetRecords.IteratorAgeMilliseconds`untuk memantau bahwa aplikasi menjaga dengan data yang masuk.

Dalam beberapa skenario, mungkin baik-baik saja untuk prosesor rekaman untuk melewatkan beberapa catatan pertama di sungai. Misalnya, Anda mungkin menjalankan beberapa catatan awal melalui aliran untuk menguji bahwa aliran bekerjaend-to-endsesuai dengan yang diharapkan. Setelah melakukan verifikasi awal ini, Anda kemudian akan memulai pekerja Anda dan mulai memasukkan data produksi ke dalam aliran.

Untuk informasi selengkapnya tentang pengaturan `TRIM_HORIZON`, lihat Menggunakan Iterator Shard.

Mematikan Aplikasi Amazon Kinesis Data Streams

Ketika aplikasi Amazon Kinesis Data Streams Anda telah menyelesaikan tugas yang dimaksudkan, Anda harus mematikannya dengan menghentikan instans EC2 yang sedang berjalan. Anda dapat mengakhiri instance menggunakanAWS Management ConsoleatauAWS CLI.

Setelah mematikan aplikasi Amazon Kinesis Data Streams, Anda harus menghapus tabel Amazon DynamoDB yang digunakan KCL untuk melacak status aplikasi.

Baca Throttling

Throughput aliran disediakan pada tingkat pecahan. Setiap pecahan memiliki throughput baca hingga 5 transaksi per detik untuk dibaca, hingga total total data membaca rate maksimum 2 MB per detik. Jika aplikasi (atau grup aplikasi yang beroperasi pada aliran yang sama) mencoba untuk mendapatkan data dari serpihan pada tingkat yang lebih cepat, Kinesis Data Streams throttles sesuai operasi Get.

Dalam aplikasi Amazon Kinesis Data Streams, jika prosesor rekaman memproses data lebih cepat daripada batasnya — seperti dalam kasus failover — throttling terjadi. Karena KCL mengelola interaksi antara aplikasi dan Kinesis Data Streams, pengecualian throttling terjadi pada kode KCL daripada dalam kode aplikasi. Namun, karena KCL log pengecualian ini, Anda melihat mereka di log.

Jika Anda menemukan bahwa aplikasi Anda throttled secara konsisten, Anda harus mempertimbangkan untuk meningkatkan jumlah pecahan untuk aliran.

Memantau Aliran Data Amazon Kinesis

Anda dapat memantau aliran data di Amazon Kinesis Data Streams menggunakan fitur berikut:

- CloudWatch metrik — Kinesis Data Streams mengirimkan metrik khusus CloudWatch Amazon dengan pemantauan terperinci untuk setiap aliran.
- Agen Kinesis — Agen Kinesis menerbitkan CloudWatch metrik khusus untuk membantu menilai apakah agen bekerja seperti yang diharapkan.
- Pencatatan API — Kinesis Data AWS CloudTrail Streams digunakan untuk mencatat panggilan API dan menyimpan data dalam bucket Amazon S3.
- Perpustakaan Klien Kinesis — Perpustakaan Klien Kinesis (KCL) menyediakan metrik per shard, pekerja, dan aplikasi KCL.
- Kinesis Producer Library — Kinesis Producer Library (KPL) menyediakan metrik per shard, pekerja, dan aplikasi KPL.

Untuk informasi selengkapnya tentang masalah pemantauan umum, pertanyaan, dan pemecahan masalah, lihat berikut ini:

- Metrik mana yang harus saya gunakan untuk memantau dan memecahkan masalah Kinesis Data Streams?
- Mengapa IteratorAgeMilliseconds nilai dalam Kinesis Data Streams terus meningkat?

Memantau Layanan Amazon Kinesis Data Streams dengan Amazon CloudWatch

Amazon Kinesis Data Streams CloudWatch dan Amazon terintegrasi sehingga Anda dapat mengumpulkan, melihat CloudWatch, dan menganalisis metrik untuk aliran data Kinesis Anda. Misalnya, untuk melacak penggunaan shard, Anda dapat memantau `IncomingBytes` dan `OutgoingBytes` metrik dan membandingkannya dengan jumlah pecahan dalam aliran.

Metrik yang Anda konfigurasikan untuk aliran Anda secara otomatis dikumpulkan dan didorong ke CloudWatch setiap menit. Metrik diarsipkan selama dua minggu; setelah periode itu, data akan dibuang.

Tabel berikut menjelaskan tingkat aliran dasar dan pemantauan tingkat shard yang ditingkatkan untuk aliran data Kinesis.

Tipe	Deskripsi
Dasar (tingkat aliran)	Data tingkat aliran dikirim secara otomatis setiap menit tanpa biaya.
Ditingkatkan (shard-level)	Data tingkat Shard dikirim setiap menit dengan biaya tambahan. Untuk mendapatkan tingkat data ini, Anda harus mengaktifkannya secara khusus untuk streaming menggunakan EnableEnhancedMonitoringoperasi. Untuk informasi tentang harga, lihat halaman CloudWatch produk Amazon.

Dimensi dan Metrik Aliran Data Kinesis Amazon

Kinesis Data Streams mengirimkan CloudWatch metrik ke dua tingkat: tingkat aliran dan, secara opsional, tingkat pecahan. Metrik tingkat aliran adalah untuk kasus penggunaan pemantauan yang paling umum dalam kondisi normal. Metrik tingkat Shard adalah untuk tugas pemantauan tertentu, biasanya terkait dengan pemecahan masalah, dan diaktifkan menggunakan operasi. EnableEnhancedMonitoring

Untuk penjelasan tentang statistik yang dikumpulkan dari CloudWatch metrik, lihat CloudWatch Statistik di Panduan CloudWatch Pengguna Amazon.

Topik

- Metrik Tingkat Aliran Dasar
- Metrik Tingkat Shard yang Ditingkatkan
- Dimensi untuk Metrik Aliran Data Kinesis Amazon
- Metrik Aliran Data Kinesis Amazon yang Direkomendasikan

Metrik Tingkat Aliran Dasar

`AWS/Kinesis`Namespace menyertakan metrik tingkat aliran berikut.

Kinesis Data Streams mengirimkan metrik tingkat aliran CloudWatch ini ke setiap menit. Metrik ini selalu tersedia.

Metrik	Deskripsi
`GetRecords.Bytes`	Jumlah byte yang diambil dari aliran Kinesis, diukur selama periode waktu yang ditentukan. Statistik Minimum, Maksimum, dan Rata-rata mewakili byte dalam satu `GetRecords` operasi untuk aliran dalam periode waktu yang ditentukan. Nama metrik tingkat shard: `OutgoingBytes` Dimensi: StreamName Statistik: Minimum, Maksimum, Rata-rata, Jumlah, Sampel Unit: Byte
`GetRecords.IteratorAge`	Metrik ini tidak digunakan lagi. Gunakan `GetRecords.IteratorAgeMilliseconds` .
`GetRecords.IteratorAgeMilliseconds`	Usia catatan terakhir dalam semua `GetRecords` panggilan yang dilakukan terhadap aliran Kinesis, diukur selama periode waktu yang ditentukan. Usia adalah perbedaan antara waktu saat ini dan kapan catatan terakhir `GetRecords` panggilan ditulis ke aliran. Statistik Minimum dan Maksimum dapat digunakan untuk melacak kemajuan aplikasi konsumen Kinesis. Nilai nol menunjukkan bahwa catatan yang dibaca benar-benar terjebak dengan aliran. Nama metrik tingkat shard: `IteratorAgeMilliseconds` Dimensi: StreamName Statistik: Minimum, Maksimum, Rata-rata, Sampel

Metrik	Deskripsi
	Satuan: Milidetik
GetRecords.Latency	Waktu yang dibutuhkan per operasi GetRecords , diukur selama periode waktu yang ditentukan. Dimensi: StreamName Statistik: Minimum, Maksimum, Rata-rata Satuan: Milidetik
GetRecords.Records	Jumlah catatan yang diambil dari pecahan, diukur selama periode waktu yang ditentukan. Statistik Minimum, Maksimum, dan Rata-rata mewakili catatan dalam satu GetRecords operasi untuk aliran dalam periode waktu yang ditentukan. Nama metrik tingkat shard: OutgoingRecords Dimensi: StreamName Statistik: Minimum, Maksimum, Rata-rata, Jumlah, Sampel Unit: Jumlah
GetRecords.Success	Jumlah GetRecords operasi yang berhasil per aliran, diukur selama periode waktu yang ditentukan. Dimensi: StreamName Statistik: Rata-rata, Jumlah, Sampel Unit: Jumlah

Metrik	Deskripsi
IncomingBytes	Jumlah byte berhasil dimasukkan ke aliran Kinesis selama periode waktu yang ditentukan. Metrik ini mencakup byte dari PutRecord dan PutRecords operasi. Statistik Minimum, Maksimum, dan Rata-rata mewakili byte dalam operasi put tunggal untuk aliran dalam periode waktu yang ditentukan. Nama metrik tingkat shard: IncomingBytes Dimensi: StreamName Statistik: Minimum, Maksimum, Rata-rata, Jumlah, Sampel Unit: Byte
IncomingRecords	Jumlah catatan yang berhasil dimasukkan ke aliran Kinesis selama periode waktu yang ditentukan. Metrik ini mencakup jumlah rekor dari PutRecord dan PutRecords operasi. Statistik Minimum, Maksimum, dan Rata-rata mewakili catatan dalam satu operasi put untuk aliran dalam periode waktu yang ditentukan. Nama metrik tingkat shard: IncomingRecords Dimensi: StreamName Statistik: Minimum, Maksimum, Rata-rata, Jumlah, Sampel Unit: Jumlah

Metrik	Deskripsi
PutRecord.Bytes	Jumlah byte yang dimasukkan ke aliran Kinesis menggunakan operasi selama PutRecord periode waktu yang ditentukan. Dimensi: StreamName Statistik: Minimum, Maksimum, Rata-rata, Jumlah, Sampel Unit: Byte
PutRecord.Latency	Waktu yang dibutuhkan per operasi PutRecord , diukur selama periode waktu yang ditentukan. Dimensi: StreamName Statistik: Minimum, Maksimum, Rata-rata Satuan: Milidetik
PutRecord.Success	Jumlah PutRecord operasi yang berhasil per aliran Kinesis, diukur selama periode waktu yang ditentuka n. Rata-rata mencerminkan persentase penulisan yang berhasil ke aliran. Dimensi: StreamName Statistik: Rata-rata, Jumlah, Sampel Unit: Jumlah

Metrik	Deskripsi
PutRecords.Bytes	Jumlah byte yang dimasukkan ke aliran Kinesis menggunakan operasi selama PutRecords periode waktu yang ditentukan. Dimensi: StreamName Statistik: Minimum, Maksimum, Rata-rata, Jumlah, Sampel Unit: Byte
PutRecords.Latency	Waktu yang dibutuhkan per operasi PutRecords , diukur selama periode waktu yang ditentukan. Dimensi: StreamName Statistik: Minimum, Maksimum, Rata-rata Satuan: Milidetik
PutRecords.Records	Metrik ini tidak digunakan lagi. Gunakan PutRecords.SuccessfulRecords . Dimensi: StreamName Statistik: Minimum, Maksimum, Rata-rata, Jumlah, Sampel Unit: Jumlah
PutRecords.Success	Jumlah PutRecords operasi di mana setidaknya satu catatan berhasil, per aliran Kinesis, diukur selama periode waktu yang ditentukan. Dimensi: StreamName Statistik: Rata-rata, Jumlah, Sampel Unit: Jumlah

Metrik	Deskripsi
PutRecords.TotalRecords	Jumlah total catatan yang dikirim dalam PutRecords operasi per aliran data Kinesis, diukur selama periode waktu yang ditentukan. Dimensi: StreamName Statistik: Minimum, Maksimum, Rata-rata, Jumlah, Sampel Unit: Jumlah
PutRecords.SuccessfulRecords	Jumlah catatan yang berhasil dalam PutRecords operasi per aliran data Kinesis, diukur selama periode waktu yang ditentukan. Dimensi: StreamName Statistik: Minimum, Maksimum, Rata-rata, Jumlah, Sampel Unit: Jumlah
PutRecords.FailedRecords	Jumlah catatan yang ditolak karena kegagalan internal dalam PutRecords operasi per aliran data Kinesis, diukur selama periode waktu yang ditentukan. Kegagalan internal sesekali diharapkan dan harus dicoba lagi. Dimensi: StreamName Statistik: Minimum, Maksimum, Rata-rata, Jumlah, Sampel Unit: Jumlah

Metrik	Deskripsi
PutRecords.ThrottledRecords	Jumlah catatan yang ditolak karena pelambatan dalam PutRecords operasi per aliran data Kinesis, diukur selama periode waktu yang ditentukan. Dimensi: StreamName Statistik: Minimum, Maksimum, Rata-rata, Jumlah, Sampel Unit: Jumlah
ReadProvisionedThroughputExceeded	Jumlah GetRecords panggilan dibatasi untuk aliran selama periode waktu yang ditentukan. Statistik yang paling umum digunakan untuk metrik ini adalah Rata-rata. Ketika statistik Minimum memiliki nilai 1, semua catatan dibatasi untuk aliran selama periode waktu yang ditentukan. Ketika statistik Maksimum memiliki nilai 0 (nol), tidak ada catatan yang dibatasi untuk aliran selama periode waktu yang ditentukan. Nama metrik tingkat shard: ReadProvisionedThroughputExceeded Dimensi: StreamName Statistik: Minimum, Maksimum, Rata-rata, Jumlah, Sampel Unit: Jumlah
SubscribeToShard.RateExceeded	Metrik ini dipancarkan ketika upaya berlangganan baru gagal karena sudah ada langganan aktif oleh konsumen yang sama atau jika Anda melebihi jumlah panggilan per detik yang diizinkan untuk operasi ini. Dimensi: StreamName, ConsumerName

Metrik	Deskripsi
`SubscribeToShard.Success`	Metrik ini mencatat apakah SubscribeToShard langganan berhasil dibuat. Langganan hanya hidup paling lama 5 menit. Oleh karena itu, metrik ini dipancarkan setidaknya sekali setiap 5 menit. Dimensi: StreamName, ConsumerName
`SubscribeToShardEvent.Bytes`	Jumlah byte yang diterima dari pecahan, diukur selama periode waktu yang ditentukan. Statistik Minimum, Maksimum, dan Rata-rata mewakili byte yang diterbitkan dalam satu peristiwa untuk periode waktu yang ditentukan. Nama metrik tingkat shard: `OutgoingBytes` Dimensi: StreamName, ConsumerName Statistik: Minimum, Maksimum, Rata-rata, Jumlah, Sampel Unit: Byte
`SubscribeToShardEvent.MillisBehindLatest`	Perbedaan antara waktu saat ini dan kapan catatan terakhir dari SubscribeToShard acara tersebut ditulis ke aliran. Dimensi: StreamName, ConsumerName Statistik: Minimum, Maksimum, Rata-rata, Sampel Satuan: Milidetik

Metrik	Deskripsi
`SubscribeToShardEvent.Records`	Jumlah catatan yang diterima dari pecahan, diukur selama periode waktu yang ditentukan. Statistik Minimum, Maksimum, dan Rata-rata mewakili catatan dalam satu peristiwa untuk periode waktu yang ditentukan. Nama metrik tingkat shard: `OutgoingRecords` Dimensi: StreamName, ConsumerName Statistik: Minimum, Maksimum, Rata-rata, Jumlah, Sampel Unit: Jumlah
`SubscribeToShardEvent.Success`	Metrik ini dipancarkan setiap kali suatu peristiwa berhasil diterbitkan. Itu hanya dipancarkan ketika ada langganan aktif. Dimensi: StreamName, ConsumerName Statistik: Minimum, Maksimum, Rata-rata, Jumlah, Sampel Unit: Jumlah

Metrik	Deskripsi
`WriteProvisionedThroughputExceeded`	Jumlah catatan yang ditolak karena pelambatan untuk aliran selama periode waktu yang ditentukan. Metrik ini mencakup pembatasan dari `PutRecord` dan `PutRecords` operasi. Statistik yang paling umum digunakan untuk metrik ini adalah Rata-rata. Ketika statistik Minimum memiliki nilai bukan nol, catatan sedang dibatasi untuk aliran selama periode waktu yang ditentukan. Ketika statistik Maksimum memiliki nilai 0 (nol), tidak ada catatan yang dibatasi untuk aliran selama periode waktu yang ditentukan. Nama metrik tingkat shard: `WriteProvisionedThroughputExceeded` Dimensi: StreamName Statistik: Minimum, Maksimum, Rata-rata, Jumlah, Sampel Unit: Jumlah

Metrik Tingkat Shard yang Ditingkatkan

`AWS/Kinesis`Namespace menyertakan metrik tingkat shard berikut.

Kinesis mengirimkan metrik level shard berikut ke setiap menit. CloudWatch Setiap dimensi metrik membuat 1 CloudWatch metrik dan membuat sekitar 43.200 panggilan `PutMetricData` API per bulan. Metrik ini tidak diaktifkan secara default. Ada biaya untuk metrik yang ditingkatkan yang dipancarkan dari Kinesis. Untuk informasi selengkapnya, lihat CloudWatch Harga Amazon di bawah judul Metrik CloudWatch Kustom Amazon. Biaya diberikan per pecahan per metrik per bulan.

Metrik	Deskripsi
IncomingBytes	Jumlah byte berhasil dimasukkan ke pecahan selama periode waktu yang ditentukan. Metrik ini mencakup byte dari PutRecord dan PutRecords operasi. Statistik Minimum, Maksimum, dan Rata-rata mewakili byte dalam operasi put tunggal untuk pecahan dalam periode waktu yang ditentukan. Nama metrik tingkat aliran: IncomingBytes Dimensi: StreamName, ShardId Statistik: Minimum, Maksimum, Rata-rata, Jumlah, Sampel Unit: Byte
IncomingRecords	Jumlah catatan yang berhasil dimasukkan ke pecahan selama periode waktu yang ditentukan. Metrik ini mencakup jumlah rekor dari PutRecord dan PutRecords operasi. Statistik Minimum, Maksimum, dan Rata-rata mewakili catatan dalam operasi put tunggal untuk pecahan dalam periode waktu yang ditentukan. Nama metrik tingkat aliran: IncomingRecords Dimensi: StreamName, ShardId Statistik: Minimum, Maksimum, Rata-rata, Jumlah, Sampel Unit: Jumlah
IteratorAgeMilliseconds	Usia catatan terakhir dalam semua GetRecords panggilan yang dilakukan terhadap pecahan, diukur selama periode waktu yang ditentukan. Usia adalah perbedaan antara waktu saat ini dan kapan catatan terakhir GetRecords panggilan ditulis ke aliran.

Metrik	Deskripsi
	Statistik Minimum dan Maksimum dapat digunakan untuk melacak kemajuan aplikasi konsumen Kinesis. Nilai 0 (nol) menunjukkan bahwa catatan yang dibaca benar-ben ar terjebak dengan aliran. Nama metrik tingkat aliran: `GetRecords.Iterato rAgeMilliseconds` Dimensi: StreamName, ShardId Statistik: Minimum, Maksimum, Rata-rata, Sampel Satuan: Milidetik
`OutgoingBytes`	Jumlah byte yang diambil dari pecahan, diukur selama periode waktu yang ditentukan. Statistik Minimum, Maksimum, dan Rata-rata mewakili byte yang dikembali kan dalam satu `GetRecords` operasi atau diterbitk an dalam satu `SubscribeToShard` peristiwa untuk pecahan dalam periode waktu yang ditentukan. Nama metrik tingkat aliran: `GetRecords.Bytes` Dimensi: StreamName, ShardId Statistik: Minimum, Maksimum, Rata-rata, Jumlah, Sampel Unit: Byte

Metrik	Deskripsi
`OutgoingRecords`	Jumlah catatan yang diambil dari pecahan, diukur selama periode waktu yang ditentukan. Statistik Minimum, Maksimum, dan Rata-rata mewakili catatan yang dikembalikan dalam satu `GetRecords` operasi atau diterbitkan dalam satu `SubscribeToShard` peristiwa untuk pecahan dalam periode waktu yang ditentukan. Nama metrik tingkat aliran: `GetRecords.Records` Dimensi: StreamName, ShardId Statistik: Minimum, Maksimum, Rata-rata, Jumlah, Sampel Unit: Jumlah

Metrik	Deskripsi
`ReadProvisionedThroughputExceeded`	Jumlah `GetRecords` panggilan dibatasi untuk pecahan selama periode waktu yang ditentukan. Jumlah pengecualian ini mencakup semua dimensi dari batas berikut: 5 pembacaan per pecahan per detik atau 2 MB per detik per pecahan. Statistik yang paling umum digunakan untuk metrik ini adalah Rata-rata. Ketika statistik Minimum memiliki nilai 1, semua catatan dibatasi untuk pecahan selama periode waktu yang ditentukan. Ketika statistik Maksimum memiliki nilai 0 (nol), tidak ada catatan yang dibatasi untuk pecahan selama periode waktu yang ditentukan. Nama metrik tingkat aliran: `ReadProvisionedThroughputExceeded` Dimensi: StreamName, ShardId Statistik: Minimum, Maksimum, Rata-rata, Jumlah, Sampel Unit: Jumlah

Metrik	Deskripsi
`WriteProvisionedThroughputExceeded`	Jumlah catatan yang ditolak karena pelambatan untuk pecahan selama periode waktu yang ditentukan. Metrik ini mencakup pembatasan dari `PutRecord` dan `PutRecords` operasi dan mencakup semua dimensi dari batas berikut: 1.000 catatan per detik per pecahan atau 1 MB per detik per pecahan. Statistik yang paling umum digunakan untuk metrik ini adalah Rata-rata. Ketika statistik Minimum memiliki nilai bukan nol, catatan sedang dibatasi untuk pecahan selama periode waktu yang ditentukan. Ketika statistik Maksimum memiliki nilai 0 (nol), tidak ada catatan yang dibatasi untuk pecahan selama periode waktu yang ditentukan. Nama metrik tingkat aliran: `WriteProvisionedThroughputExceeded` Dimensi: StreamName, ShardId Statistik: Minimum, Maksimum, Rata-rata, Jumlah, Sampel Unit: Jumlah

Dimensi untuk Metrik Aliran Data Kinesis Amazon

Dimensi	Deskripsi
`StreamName`	Nama pengaliran Kinesis. Semua statistik yang tersedia disaring oleh`StreamName` .

Metrik Aliran Data Kinesis Amazon yang Direkomendasikan

Beberapa metrik Amazon Kinesis Data Streams mungkin menarik bagi pelanggan Kinesis Data Streams. Daftar berikut menyediakan metrik yang direkomendasikan dan penggunaannya.

Metrik	Catatan Penggunaan
`GetRecords.IteratorAgeMilliseconds`	Melacak posisi baca di semua pecahan dan konsumen di sungai. Jika usia iterator melewati 50% dari periode retensi (secara default, 24 jam, dapat dikonfigurasi hingga 7 hari), ada risiko kehilangan data karena kedaluwarsa rekaman. Kami menyarankan Anda menggunakan CloudWatch alarm pada statistik Maksimum untuk mengingatkan Anda sebelum kerugian ini berisiko. Untuk contoh skenario yang menggunakan metrik ini, lihatPemrosesan Rekor Konsumen Tertinggal.
`ReadProvisionedThroughputExceeded`	Ketika pemrosesan catatan sisi konsumen Anda tertinggal, terkadang sulit untuk mengetahui di mana kemacetannya. Gunakan metrik ini untuk menentukan apakah pembacaan Anda sedang dibatasi karena melebihi batas throughput baca Anda. Statistik yang paling umum digunakan untuk metrik ini adalah Rata-rata.
`WriteProvisionedThroughputExceeded`	Ini untuk tujuan yang sama dengan `ReadProvisionedThroughputExceeded` metrik, tetapi untuk sisi produser (put) aliran. Statistik yang paling umum digunakan untuk metrik ini adalah Rata-rata.
`PutRecord.Success` , `PutRecords.Success`	Kami merekomendasikan penggunaan CloudWatch alarm pada statistik Rata-rata untuk menunjukkan kapan catatan gagal ke aliran. Pilih salah satu atau kedua jenis put tergantung pada apa yang digunakan produsen Anda. Jika menggunakan Kinesis Producer Library (KPL), gunakan. `PutRecords.Success`
`GetRecords.Success`	Sebaiknya gunakan CloudWatch alarm pada statistik Rata-rata untuk menunjukkan kapan catatan gagal dari aliran.

Mengakses CloudWatch Metrik Amazon untuk Kinesis Data Streams

Anda dapat memantau metrik untuk Kinesis Data Streams menggunakan CloudWatch konsol, baris perintah, atau API. CloudWatch Prosedur berikut menunjukkan cara mengakses metrik menggunakan berbagai metode ini.

Untuk mengakses metrik menggunakan konsol CloudWatch

1. Buka CloudWatch konsol di https://console.aws.amazon.com/cloudwatch/.
2. Pada bilah navigasi, pilih Wilayah.
3. Di panel navigasi, pilih Metrik.
4. Di panel CloudWatch Metrik menurut Kategori, pilih Metrik Kinesis.
5. Klik baris yang relevan untuk melihat statistik untuk yang ditentukan MetricNamedan StreamName.

   Catatan: Sebagian besar nama statistik konsol cocok dengan nama CloudWatch metrik terkait yang tercantum di atas, kecuali untuk Throughput Baca dan Tulis Throughput. Statistik ini dihitung selama interval 5 menit: Write Throughput memantau `IncomingBytes` CloudWatch metrik, dan monitor Read Throughput. `GetRecords.Bytes`
6. (Opsional) Di panel grafik, pilih statistik dan periode waktu, lalu buat CloudWatch alarm menggunakan pengaturan ini.

Untuk mengakses metrik menggunakan AWS CLI

Gunakan daftar-metrik dan perintah. get-metric-statistics

Untuk mengakses metrik menggunakan CLI CloudWatch

Gunakan mon-list-metricsdan mon-get-statsperintah.

Untuk mengakses metrik menggunakan API CloudWatch

Gunakan ListMetricsdan GetMetricStatisticsoperasi.

Memantau Kinesis Data Streams Agen Kesehatan dengan Amazon CloudWatch

Agen menerbitkan CloudWatch metrik kustom dengan namespace dari. AWSKinesisAgent Metrik ini membantu Anda menilai apakah agen mengirimkan data ke Kinesis Data Streams seperti yang ditentukan, dan apakah itu sehat dan mengkonsumsi jumlah CPU dan sumber daya memori yang sesuai pada produsen data. Metrik seperti jumlah catatan dan byte yang dikirim berguna untuk memahami tingkat di mana agen mengirimkan data ke aliran. Bila metrik ini turun di bawah ambang batas yang diharapkan sejumlah sekian persen atau turun ke nol, hal ini dapat menunjukkan masalah konfigurasi, kesalahan jaringan, atau masalah kondisi agen. Metrik seperti konsumsi CPU dan memori on-host serta penghitung kesalahan agen menunjukkan penggunaan sumber daya produsen data dan memberikan gambaran mengenai kemungkinan kesalahan konfigurasi atau host. Pada akhirnya, agen juga mencatat pengecualian layanan untuk membantu menyelidiki masalah agen. Metrik ini dilaporkan di Wilayah yang ditentukan dalam `cloudwatch.endpoint` setelan konfigurasi agen. Metrik Cloudwatch yang diterbitkan dari beberapa agen Kinesis dikumpulkan atau digabungkan. Untuk informasi selengkapnya tentang konfigurasi agen, silakan lihat Pengaturan Konfigurasi Agen.

Pemantauan CloudWatch dengan

Agen Kinesis Data Streams mengirimkan metrik berikut ke. CloudWatch

Metrik	Deskripsi
`BytesSent`	Jumlah byte yang dikirim ke Kinesis Data Streams selama periode waktu yang ditentukan. Unit: Byte
`RecordSendAttempts`	Jumlah catatan yang dicoba (baik pertama kali, atau percobaan ulang) dalam panggilan ke `PutRecords` selama periode waktu yang ditentukan. Unit: Jumlah
`RecordSendErrors`	Jumlah catatan yang menghasilkan status gagal dalam panggilan ke `PutRecords`, termasuk percobaan ulang, selama periode waktu yang ditentukan.

Metrik	Deskripsi
	Unit: Jumlah
`ServiceErrors`	Jumlah panggilan ke `PutRecords` yang mengakibatkan kesalahan layanan (selain kesalahan throttling) selama periode waktu yang ditentukan. Unit: Jumlah

Mencatat Panggilan API Amazon Kinesis Data Streams dengan AWS CloudTrail

Amazon Kinesis Data Streams AWS CloudTrail terintegrasi dengan, layanan yang menyediakan catatan tindakan yang diambil oleh pengguna, peran, AWS atau layanan di Kinesis Data Streams. CloudTrail menangkap semua panggilan API untuk Kinesis Data Streams sebagai peristiwa. Panggilan yang diambil mencakup panggilan dari konsol Kinesis Data Streams dan panggilan kode ke operasi API Kinesis Data Streams. Jika Anda membuat jejak, Anda dapat mengaktifkan pengiriman CloudTrail peristiwa secara terus menerus ke bucket Amazon S3, termasuk peristiwa untuk Kinesis Data Streams. Jika Anda tidak mengonfigurasi jejak, Anda masih dapat melihat peristiwa terbaru di CloudTrail konsol dalam Riwayat acara. Dengan menggunakan informasi yang dikumpulkan oleh CloudTrail, Anda dapat menentukan permintaan yang dibuat untuk Kinesis Data Streams, alamat IP dari mana permintaan dibuat, siapa yang membuat permintaan, kapan dibuat, dan detail tambahan.

Untuk mempelajari selengkapnya CloudTrail, termasuk cara mengonfigurasi dan mengaktifkannya, lihat Panduan AWS CloudTrail Pengguna.

Informasi Aliran Data Kinesis di CloudTrail

CloudTrail diaktifkan di AWS akun Anda saat Anda membuat akun. Ketika aktivitas peristiwa yang didukung terjadi di Kinesis Data Streams, aktivitas tersebut direkam CloudTrail dalam suatu peristiwa bersama AWS dengan peristiwa layanan lainnya dalam riwayat Peristiwa. Anda dapat melihat, mencari, dan mengunduh peristiwa terbaru di akun AWS. Untuk informasi lain, lihat Melihat Peristiwa dengan Riwayat Peristiwa CloudTrail.

Untuk catatan peristiwa yang sedang berlangsung di AWS akun Anda, termasuk peristiwa untuk Kinesis Data Streams, buat jejak. Jejak memungkinkan CloudTrail untuk mengirimkan file log

ke bucket Amazon S3. Secara default, saat Anda membuat jejak di dalam konsol tersebut, jejak diterapkan ke semua Wilayah AWS. Jejak mencatat peristiwa dari semua Wilayah di partisi AWS dan mengirimkan file log ke bucket Amazon S3 yang Anda tentukan. Selain itu, Anda dapat mengonfigurasi AWS layanan lain untuk menganalisis lebih lanjut dan menindaklanjuti data peristiwa yang dikumpulkan dalam CloudTrail log. Untuk informasi selengkapnya, lihat berikut:

- Gambaran Umum untuk Membuat Jejak
- CloudTrail Layanan dan Integrasi yang Didukung
- Mengkonfigurasi Notifikasi Amazon SNS untuk CloudTrail
- Menerima File CloudTrail Log dari Beberapa Wilayah dan Menerima File CloudTrail Log dari Beberapa Akun

Kinesis Data Streams mendukung pencatatan tindakan berikut sebagai CloudTrail peristiwa dalam file log:

- AddTagsToStream
- CreateStream
- DecreaseStreamRetentionPeriod
- DeleteStream
- DeregisterStreamConsumer
- DescribeStream
- DescribeStreamConsumer
- DisableEnhancedMonitoring
- EnableEnhancedMonitoring
- IncreaseStreamRetentionPeriod
- ListStreamConsumers
- ListStreams
- ListTagsForStream
- MergeShards
- RegisterStreamConsumer
- RemoveTagsFromStream
- SplitShard
- StartStreamEncryption

- StopStreamEncryption
- UpdateShardCount
- UpdateStreamMode

Setiap entri peristiwa atau log berisi informasi tentang siapa yang membuat permintaan tersebut. Informasi identitas membantu Anda menentukan berikut ini:

- Apakah permintaan tersebut dibuat dengan kredensial root atau pengguna AWS Identity and Access Management IAM.
- Apakah permintaan tersebut dibuat dengan kredensial keamanan sementara untuk satu peran atau pengguna gabungan.
- Apakah permintaan tersebut dibuat oleh layanan AWS lain.

Untuk informasi lain, lihat Elemen userIdentity CloudTrail.

Contoh: Entri File Log Aliran Data Kinesis

Trail adalah konfigurasi yang memungkinkan pengiriman peristiwa sebagai file log ke bucket Amazon S3 yang Anda tentukan. CloudTrail file log berisi satu atau lebih entri log. Peristiwa merepresentasikan satu permintaan dari sumber apa pun dan menyertakan informasi tentang tindakan yang diminta, tanggal dan waktu tindakan, parameter permintaan, dan sebagainya. CloudTrail file log bukanlah jejak tumpukan yang diurutkan dari panggilan API publik, jadi file tersebut tidak muncul dalam urutan tertentu.

Contoh berikut menunjukkan entri CloudTrail log yang menunjukkan`CreateStream`,,, `DescribeStream`,`ListStreams`, `DeleteStream`,`SplitShard`, dan `MergeShards` tindakan.

```
{
    "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"
        }
    ]
}
```

Memantau Perpustakaan Klien Kinesis dengan Amazon CloudWatch

Perpustakaan Klien Kinesis (KCL) untuk Amazon Kinesis Data Streams menerbitkan metrik CloudWatch Amazon kustom atas nama Anda, menggunakan nama aplikasi KCL Anda sebagai namespace. Anda dapat melihat metrik ini dengan menavigasi ke CloudWatch konsol dan memilih Metrik Kustom. Untuk informasi selengkapnya tentang metrik kustom, lihat Menerbitkan Metrik Kustom di CloudWatch Panduan Pengguna Amazon.

Ada biaya nominal untuk metrik yang diunggah CloudWatch oleh KCL; khususnya, biaya Amazon CloudWatch Custom Metrics dan CloudWatch Amazon API Requests berlaku. Untuk informasi selengkapnya, lihat CloudWatch Harga Amazon.

Topik

- Metrik dan Namespace
- Tingkat dan Dimensi Metrik
- Konfigurasi Metrik
- Daftar Metrik

Metrik dan Namespace

Namespace yang digunakan untuk mengunggah metrik adalah nama aplikasi yang Anda tentukan saat meluncurkan KCL.

Tingkat dan Dimensi Metrik

Ada dua opsi untuk mengontrol metrik mana yang diunggah: CloudWatch

tingkat metrik

Setiap metrik diberikan tingkat individu. Saat Anda menetapkan tingkat pelaporan metrik, metrik dengan tingkat individual di bawah tingkat pelaporan tidak akan dikirim. CloudWatch Levelnya adalah:`NONE`,`SUMMARY`, dan`DETAILED`. Pengaturan default adalah`DETAILED`; yaitu, semua metrik

dikirim ke CloudWatch. Tingkat pelaporan `NONE` berarti tidak ada metrik yang dikirim sama sekali. Untuk informasi tentang level mana yang ditetapkan ke metrik apa, lihatDaftar Metrik.

dimensi yang diaktifkan

Setiap metrik KCL memiliki dimensi terkait yang juga dikirim ke CloudWatch. Di KCL 2.x, jika KCL dikonfigurasi untuk memproses aliran data tunggal, semua dimensi metrik (`Operation`,`ShardId`, dan`WorkerIdentifier`) diaktifkan secara default. Juga, di KCL 2.x, jika KCL dikonfigurasi untuk memproses aliran data tunggal, `Operation` dimensi tidak dapat dinonaktifkan. Di KCL 2.x, jika KCL dikonfigurasi untuk memproses beberapa aliran data, semua dimensi metrik (,`Operation`, `ShardIdStreamId`, dan`WorkerIdentifier`) diaktifkan secara default. Juga, di KCL 2.x, jika KCL dikonfigurasi untuk memproses beberapa aliran data, `Operation` dan `StreamId` dimensi tidak dapat dinonaktifkan. `StreamId`dimensi hanya tersedia untuk metrik per-shard.

Di KCL 1.x, hanya `Operation` dan `ShardId` dimensi yang diaktifkan secara default, dan `WorkerIdentifier` dimensi dinonaktifkan. Di KCL 1.x, `Operation` dimensi tidak dapat dinonaktifkan.

Untuk informasi selengkapnya tentang dimensi CloudWatch metrik, lihat bagian Dimensi dalam topik CloudWatch Konsep Amazon, di Panduan CloudWatch Pengguna Amazon.

Ketika `WorkerIdentifier` dimensi diaktifkan, jika nilai yang berbeda digunakan untuk properti ID pekerja setiap kali pekerja KCL tertentu memulai ulang, set metrik baru dengan nilai `WorkerIdentifier` dimensi baru akan dikirim ke. CloudWatch Jika Anda memerlukan nilai `WorkerIdentifier` dimensi yang sama di seluruh restart pekerja KCL tertentu, Anda harus secara eksplisit menentukan nilai ID pekerja yang sama selama inisialisasi untuk setiap pekerja. Perhatikan bahwa nilai ID pekerja untuk setiap pekerja KCL yang aktif harus unik di semua pekerja KCL.

Konfigurasi Metrik

Level metrik dan dimensi yang diaktifkan dapat dikonfigurasi menggunakan KinesisClientLibConfiguration instance, yang diteruskan ke Worker saat meluncurkan aplikasi KCL. MultiLangDaemon Dalam kasus ini, `metricsEnabledDimensions` properti `metricsLevel` and dapat ditentukan dalam file.properties yang digunakan untuk meluncurkan aplikasi MultiLangDaemon KCL.

Level metrik dapat diberikan salah satu dari tiga nilai: TIDAK ADA, RINGKASAN, atau DETAIL. Nilai dimensi yang diaktifkan harus berupa string yang dipisahkan koma dengan daftar

dimensi yang diizinkan untuk metrik. CloudWatch Dimensi yang digunakan oleh aplikasi KCL adalah`Operation`,`ShardId`, dan`WorkerIdentifier`.

Daftar Metrik

Tabel berikut mencantumkan metrik KCL, dikelompokkan berdasarkan ruang lingkup dan operasi.

Topik

Metrik Per-KCL-Aplikasi

Metrik ini digabungkan di semua pekerja KCL dalam lingkup aplikasi, seperti yang didefinisikan oleh namespace Amazon. CloudWatch

Topik

InitializeTask

`InitializeTask`Operasi ini bertanggung jawab untuk menginisialisasi prosesor rekaman untuk aplikasi KCL. Logika untuk operasi ini termasuk mendapatkan iterator pecahan dari Kinesis Data Streams dan menginisialisasi prosesor rekaman.

Metrik	Deskripsi
KinesisDataFetcher .getIterator.Success	Jumlah `GetShardIterator` operasi yang berhasil per aplikasi KCL. Tingkat metrik: Terperinci

Metrik	Deskripsi
	Unit: Jumlah
KinesisDataFetcher .getIterator.waktu	Waktu yang dibutuhkan per `GetShardIterator` operasi untuk aplikasi KCL yang diberikan. Tingkat metrik: Terperinci Satuan: Milidetik
RecordProcessor.me nginisialize.waktu	Waktu yang dibutuhkan oleh metode inisialisasi prosesor rekaman. Tingkat metrik: Ringkasan Satuan: Milidetik
Berhasil	Jumlah inisialisasi prosesor rekaman yang berhasil. Tingkat metrik: Ringkasan Unit: Jumlah
Waktu	Waktu yang dibutuhkan oleh pekerja KCL untuk inisialisasi prosesor rekaman. Tingkat metrik: Ringkasan Satuan: Milidetik

ShutdownTask

`ShutdownTask`Operasi memulai urutan shutdown untuk pemrosesan shard. Hal ini dapat terjadi karena pecahan dipecah atau digabung, atau ketika sewa shard hilang dari pekerja. Dalam kedua kasus, `shutdown()` fungsi prosesor rekaman dipanggil. Pecahan baru juga ditemukan dalam kasus di mana pecahan dipecah atau digabungkan, menghasilkan penciptaan satu atau dua pecahan baru.

Metrik	Deskripsi
CreateLease.Sukses	Berapa kali pecahan anak baru berhasil ditambahkan ke dalam tabel DynamoDB aplikasi KCL setelah shutdown shard induk.

Metrik	Deskripsi
	Tingkat metrik: Terperinci Unit: Jumlah
CreateLease.Waktu	Waktu yang dibutuhkan untuk menambahkan informasi pecahan anak baru dalam tabel DynamoDB aplikasi KCL. Tingkat metrik: Terperinci Satuan: Milidetik
UpdateLease.Sukses	Jumlah pos pemeriksaan akhir yang berhasil selama shutdown prosesor rekaman. Tingkat metrik: Terperinci Unit: Jumlah
UpdateLease.Waktu	Waktu yang dibutuhkan oleh operasi pos pemeriksaan selama shutdown prosesor rekaman. Tingkat metrik: Terperinci Satuan: Milidetik
RecordProcessor.shutdown.waktu	Waktu yang dibutuhkan oleh metode shutdown prosesor rekaman. Tingkat metrik: Ringkasan Satuan: Milidetik
Berhasil	Jumlah tugas shutdown yang berhasil. Tingkat metrik: Ringkasan Unit: Jumlah

Metrik	Deskripsi
Waktu	Waktu yang dibutuhkan oleh pekerja KCL untuk tugas shutdown. Tingkat metrik: Ringkasan Satuan: Milidetik

ShardSyncTask

`ShardSyncTask`Operasi menemukan perubahan informasi pecahan untuk aliran data Kinesis, sehingga pecahan baru dapat diproses oleh aplikasi KCL.

Metrik	Deskripsi
CreateLease.Sukses	Jumlah upaya yang berhasil untuk menambahkan informasi pecahan baru ke dalam tabel DynamoDB aplikasi KCL. Tingkat metrik: Terperinci Unit: Jumlah
CreateLease.Waktu	Waktu yang dibutuhkan untuk menambahkan informasi pecahan baru dalam tabel DynamoDB aplikasi KCL. Tingkat metrik: Terperinci Satuan: Milidetik
Berhasil	Jumlah operasi sinkronisasi shard yang berhasil. Tingkat metrik: Ringkasan Unit: Jumlah
Waktu	Waktu yang dibutuhkan untuk operasi sinkronisasi shard. Tingkat metrik: Ringkasan Satuan: Milidetik

BlockOnParentTask

Jika pecahan dipecah atau digabung dengan pecahan lain, maka pecahan anak baru dibuat. `BlockOnParentTask`Operasi memastikan bahwa pemrosesan rekaman untuk pecahan baru tidak dimulai sampai pecahan induk benar-benar diproses oleh KCL.

Metrik	Deskripsi
Berhasil	Jumlah pemeriksaan yang berhasil untuk penyelesaian pecahan induk. Tingkat metrik: Ringkasan Unit: Jumlah
Waktu	Waktu yang dibutuhkan untuk penyelesaian pecahan induk. Tingkat metrik: Ringkasan Satuan: Milidetik

PeriodicShardSyncManager

Bertanggung `PeriodicShardSyncManager` jawab untuk memeriksa aliran data yang sedang diproses oleh aplikasi konsumen KCL, mengidentifikasi aliran data dengan sewa sebagian dan menyerahkannya untuk sinkronisasi.

Metrik berikut tersedia ketika KCL dikonfigurasi untuk memproses aliran data tunggal (kemudian nilai NumStreamsToSync dan NumStreamsWithPartialLeases diatur ke 1) dan juga ketika KCL dikonfigurasi untuk memproses beberapa aliran data.

Metrik	Deskripsi
NumStreamsToSync	Jumlah aliran data (per AWS akun) yang diproses oleh aplikasi konsumen yang berisi sewa sebagian dan yang harus diserahkan untuk sinkronisasi. Tingkat metrik: Ringkasan Unit: Jumlah

Metrik	Deskripsi
NumStreamsWithPartialLeases	Jumlah aliran data (per AWS akun) yang diproses aplikasi konsumen yang berisi sewa sebagian. Tingkat metrik: Ringkasan Unit: Jumlah
Berhasil	Berapa kali `PeriodicShardSyncManager` berhasil mengidentifikasi sewa sebagian dalam aliran data yang diproses oleh aplikasi konsumen. Tingkat metrik: Ringkasan Unit: Jumlah
Waktu	Jumlah waktu (dalam milidetik) yang `PeriodicShardSyncManager` diperlukan untuk memeriksa aliran data yang diproses aplikasi konsumen, untuk menentukan aliran data mana yang memerlukan sinkronisasi pecahan. Tingkat metrik: Ringkasan Satuan: Milidetik

MultistreamTracker

`MultistreamTracker`Antarmuka memungkinkan Anda untuk membangun aplikasi konsumen KCL yang dapat memproses beberapa aliran data secara bersamaan.

Metrik	Deskripsi
DeletedStreams.Hitung	Jumlah aliran data yang dihapus pada periode waktu ini. Tingkat metrik: Ringkasan Unit: Jumlah
ActiveStreams.Hitung	Jumlah aliran data aktif yang sedang diproses.

Metrik	Deskripsi
	Tingkat metrik: Ringkasan Unit: Jumlah
StreamsPendingDeletion.Hitung	Jumlah aliran data yang tertunda penghapusan berdasarkan. `FormerStreamsLeasesDeletionStrategy` Tingkat metrik: Ringkasan Unit: Jumlah

Metrik Per-Pekerja

Metrik ini dikumpulkan di semua prosesor rekaman yang menggunakan data dari aliran data Kinesis, seperti instans Amazon EC2.

Topik

- RenewAllLeases
- TakeLeases

RenewAllLeases

`RenewAllLeases`Operasi secara berkala memperbarui sewa pecahan yang dimiliki oleh contoh pekerja tertentu.

Metrik	Deskripsi
RenewLease.Sukses	Jumlah perpanjangan sewa yang berhasil oleh pekerja. Tingkat metrik: Terperinci Unit: Jumlah
RenewLease.Waktu	Waktu yang dibutuhkan oleh operasi perpanjangan sewa. Tingkat metrik: Terperinci

Metrik	Deskripsi
	Satuan: Milidetik
CurrentLeases	Jumlah sewa pecahan yang dimiliki oleh pekerja setelah semua sewa diperbarui. Tingkat metrik: Ringkasan Unit: Jumlah
LostLeases	Jumlah sewa pecahan yang hilang setelah upaya untuk memperbarui semua sewa yang dimiliki oleh pekerja. Tingkat metrik: Ringkasan Unit: Jumlah
Berhasil	Berapa kali operasi pembaruan sewa berhasil bagi pekerja. Tingkat metrik: Ringkasan Unit: Jumlah
Waktu	Waktu yang dibutuhkan untuk memperbarui semua sewa untuk pekerja. Tingkat metrik: Ringkasan Satuan: Milidetik

TakeLeases

`TakeLeases`Operasi menyeimbangkan pemrosesan catatan antara semua pekerja KCL. Jika pekerja KCL saat ini memiliki sewa pecahan yang lebih sedikit daripada yang dibutuhkan, dibutuhkan sewa pecahan dari pekerja lain yang kelebihan beban.

Metrik	Deskripsi
ListLeases.Sukses	Berapa kali semua sewa shard berhasil diambil dari tabel DynamoDB aplikasi KCL.

Metrik	Deskripsi
	Tingkat metrik: Terperinci Unit: Jumlah
ListLeases.Waktu	Waktu yang dibutuhkan untuk mengambil semua sewa shard dari tabel DynamoDB aplikasi KCL. Tingkat metrik: Terperinci Satuan: Milidetik
TakeLease.Sukses	Berapa kali pekerja berhasil mengambil sewa pecahan dari pekerja KCL lainnya. Tingkat metrik: Terperinci Unit: Jumlah
TakeLease.Waktu	Waktu yang dibutuhkan untuk memperbarui tabel sewa dengan sewa yang diambil oleh pekerja. Tingkat metrik: Terperinci Satuan: Milidetik
NumWorkers	Jumlah total pekerja, sebagaimana diidentifikasi oleh pekerja tertentu. Tingkat metrik: Ringkasan Unit: Jumlah
NeededLeases	Jumlah sewa pecahan yang dibutuhkan pekerja saat ini untuk beban pemrosesan pecahan yang seimbang. Tingkat metrik: Terperinci Unit: Jumlah

Metrik	Deskripsi
LeasesToTake	Jumlah sewa yang akan coba diambil oleh pekerja. Tingkat metrik: Terperinci Unit: Jumlah
TakenLeases	Jumlah sewa yang berhasil diambil oleh pekerja. Tingkat metrik: Ringkasan Unit: Jumlah
TotalLeases	Jumlah pecahan yang diproses aplikasi KCL. Tingkat metrik: Terperinci Unit: Jumlah
ExpiredLeases	Jumlah total pecahan yang tidak diproses oleh pekerja mana pun, sebagaimana diidentifikasi oleh pekerja tertentu. Tingkat metrik: Ringkasan Unit: Jumlah
Berhasil	Berapa kali `TakeLeases` operasi berhasil diselesaikan. Tingkat metrik: Ringkasan Unit: Jumlah
Waktu	Waktu yang dibutuhkan oleh `TakeLeases` operasi untuk seorang pekerja. Tingkat metrik: Ringkasan Satuan: Milidetik

Metrik Per-Shard

Metrik ini dikumpulkan di satu prosesor rekaman.

ProcessTask

`ProcessTask`Operasi memanggil GetRecordsdengan posisi iterator saat ini untuk mengambil catatan dari aliran dan memanggil fungsi prosesor rekaman. `processRecords`

Metrik	Deskripsi
KinesisDataFetcher .getRecords.sukses	Jumlah `GetRecords` operasi yang berhasil per pecahan aliran data Kinesis. Tingkat metrik: Terperinci Unit: Jumlah
KinesisDataFetcher .getRecords.waktu	Waktu yang dibutuhkan per `GetRecords` operasi untuk pecahan aliran data Kinesis. Tingkat metrik: Terperinci Satuan: Milidetik
UpdateLease.Sukses	Jumlah pos pemeriksaan yang berhasil dibuat oleh prosesor rekaman untuk pecahan yang diberikan. Tingkat metrik: Terperinci Unit: Jumlah
UpdateLease.Waktu	Waktu yang dibutuhkan untuk setiap operasi pos pemeriksaan untuk pecahan yang diberikan. Tingkat metrik: Terperinci Satuan: Milidetik
DataBytesProcessed	Ukuran total catatan yang diproses dalam byte pada setiap `ProcessTask` pemanggilan.

Metrik	Deskripsi
	Tingkat metrik: Ringkasan Unit: Byte
RecordsProcessed	Jumlah catatan yang diproses pada setiap `ProcessTask` doa. Tingkat metrik: Ringkasan Unit: Jumlah
ExpiredIterator	Jumlah yang ExpiredIteratorException diterima saat menelepon `GetRecords` . Tingkat metrik: Ringkasan Unit: Jumlah
MillisBehindLatest	Waktu iterator saat ini tertinggal dari catatan terbaru (tip) di pecahan. Nilai ini kurang dari atau sama dengan perbedaan waktu antara catatan terbaru dalam respons dan waktu saat ini. Ini adalah refleksi yang lebih akurat tentang seberapa jauh pecahan dari ujung daripada membandin gkan stempel waktu dalam catatan respons terakhir. Nilai ini berlaku untuk kumpulan catatan terbaru, bukan rata-rata dari semua stempel waktu di setiap catatan. Tingkat metrik: Ringkasan Satuan: Milidetik
RecordProcessor.Pr ocessRecords.time	Waktu yang dibutuhkan oleh `processRecords` metode prosesor rekaman. Tingkat metrik: Ringkasan Satuan: Milidetik

Metrik	Deskripsi
Berhasil	Jumlah operasi tugas proses yang berhasil. Tingkat metrik: Ringkasan Unit: Jumlah
Waktu	Waktu yang dibutuhkan untuk operasi tugas proses. Tingkat metrik: Ringkasan Satuan: Milidetik

Memantau Perpustakaan Produser Kinesis dengan Amazon CloudWatch

Perpustakaan Produser Kinesis (KPL) untuk Amazon Kinesis Data Streams menerbitkan metrik Amazon khusus atas nama Anda. CloudWatch Anda dapat melihat metrik ini dengan menavigasi ke CloudWatch konsol dan memilih Metrik Kustom. Untuk informasi selengkapnya tentang metrik kustom, lihat Menerbitkan Metrik Kustom di CloudWatch Panduan Pengguna Amazon.

Ada biaya nominal untuk metrik yang diunggah CloudWatch oleh KPL; khususnya, biaya Amazon CloudWatch Custom Metrics dan CloudWatch Amazon API Requests berlaku. Untuk informasi selengkapnya, lihat CloudWatch Harga Amazon. Pengumpulan metrik lokal tidak dikenakan biaya CloudWatch.

Topik

- Metrik, Dimensi, dan Ruang Nama
- Tingkat Metrik dan Granularitas
- Akses Lokal dan CloudWatch Unggah Amazon
- Daftar Metrik

Metrik, Dimensi, dan Ruang Nama

Anda dapat menentukan nama aplikasi saat meluncurkan KPL, yang kemudian digunakan sebagai bagian dari namespace saat mengunggah metrik. Ini opsional; KPL memberikan nilai default jika nama aplikasi tidak disetel.

Anda juga dapat mengonfigurasi KPL untuk menambahkan dimensi tambahan yang sewenang-wenang ke metrik. Ini berguna jika Anda menginginkan data berbutir halus dalam metrik Anda. CloudWatch Misalnya, Anda dapat menambahkan nama host sebagai dimensi, yang kemudian memungkinkan Anda mengidentifikasi distribusi beban yang tidak merata di seluruh armada Anda. Semua pengaturan konfigurasi KPL tidak dapat diubah, sehingga Anda tidak dapat mengubah dimensi tambahan ini setelah instance KPL diinisialisasi.

Tingkat Metrik dan Granularitas

Ada dua opsi untuk mengontrol jumlah metrik yang diunggah ke: CloudWatch

tingkat metrik

Ini adalah ukuran kasar tentang seberapa penting metrik. Setiap metrik diberi level. Saat Anda menetapkan level, metrik dengan level di bawah ini yang tidak dikirim ke CloudWatch. Levelnya adalah`NONE`,`SUMMARY`, dan`DETAILED`. Pengaturan default adalah`DETAILED`; yaitu, semua metrik. `NONE`berarti tidak ada metrik sama sekali, jadi tidak ada metrik yang benar-benar ditetapkan ke level itu.

granularitas

Ini mengontrol apakah metrik yang sama dipancarkan pada tingkat granularitas tambahan. Levelnya adalah`GLOBAL`,`STREAM`, dan`SHARD`. Pengaturan defaultnya adalah`SHARD`, yang berisi metrik paling granular.

Kapan `SHARD` dipilih, metrik dipancarkan dengan nama aliran dan ID pecahan sebagai dimensi. Selain itu, metrik yang sama juga dipancarkan hanya dengan dimensi nama aliran, dan metrik tanpa nama aliran. Ini berarti bahwa, untuk metrik tertentu, dua aliran dengan dua pecahan masing-masing akan menghasilkan tujuh CloudWatch metrik: satu untuk setiap pecahan, satu untuk setiap aliran, dan satu keseluruhan; semua menggambarkan statistik yang sama tetapi pada tingkat granularitas yang berbeda. Untuk ilustrasi, lihat diagram berikut.

Tingkat granularitas yang berbeda membentuk hierarki, dan semua metrik dalam sistem membentuk pohon, berakar pada nama metrik:

```
MetricName (GLOBAL):           Metric X                    Metric Y
                                  |                           |
                            -----------------            ------------
                            |               |            |          |
StreamName (STREAM):    Stream A        Stream B      Stream A   Stream B
                            |               |
                         --------        ---------
                         |      |        |       |
ShardID (SHARD):     Shard 0 Shard 1  Shard 0 Shard 1
```

Tidak semua metrik tersedia di tingkat pecahan; beberapa di antaranya adalah tingkat aliran atau global secara alami. Ini tidak diproduksi pada tingkat pecahan, bahkan jika Anda telah mengaktifkan metrik tingkat shard (`Metric Y`dalam diagram sebelumnya).

Saat Anda menentukan dimensi tambahan, Anda perlu memberikan nilai untuk`tuple:<DimensionName, DimensionValue, Granularity>`. Granularitas digunakan untuk menentukan di mana dimensi kustom dimasukkan dalam hierarki: `GLOBAL` berarti bahwa dimensi tambahan disisipkan setelah nama metrik, `STREAM` berarti itu dimasukkan setelah nama aliran, dan `SHARD` berarti itu dimasukkan setelah ID pecahan. Jika beberapa dimensi tambahan diberikan per tingkat granularitas, mereka dimasukkan dalam urutan yang diberikan.

Akses Lokal dan CloudWatch Unggah Amazon

Metrik untuk instans KPL saat ini tersedia secara lokal secara real time; Anda dapat menanyakan KPL kapan saja untuk mendapatkannya. KPL secara lokal menghitung jumlah, rata-rata, minimum, maksimum, dan hitungan setiap metrik, seperti dalam. CloudWatch

Anda bisa mendapatkan statistik yang kumulatif dari awal program hingga saat ini dalam waktu, atau menggunakan jendela bergulir selama N detik terakhir, di mana N adalah bilangan bulat antara 1 dan 60.

Semua metrik tersedia untuk diunggah. CloudWatch Ini sangat berguna untuk mengumpulkan data di beberapa host, pemantauan, dan mengkhawatirkan. Fungsi ini tidak tersedia secara lokal.

Seperti yang dijelaskan sebelumnya, Anda dapat memilih metrik mana yang akan diunggah dengan pengaturan tingkat metrik dan granularitas. Metrik yang tidak diunggah tersedia secara lokal.

Mengunggah titik data secara individual tidak dapat dipertahankan karena dapat menghasilkan jutaan unggahan per detik, jika lalu lintas tinggi. Untuk alasan ini, KPL menggabungkan metrik secara lokal

ke dalam bucket 1 menit dan mengunggah objek statistik ke CloudWatch satu kali per menit, per metrik yang diaktifkan.

Daftar Metrik

Metrik	Deskripsi
`UserRecordsReceived`	Hitung berapa banyak catatan pengguna logis yang diterima oleh inti KPL untuk operasi put. Tidak tersedia pada tingkat pecahan. Tingkat metrik: Terperinci Unit: Jumlah
`UserRecordsPending`	Sampel periodik berapa banyak catatan pengguna yang saat ini tertunda. Catatan tertunda jika saat ini di-buffer dan menunggu untuk dikirim, atau dikirim dan dalam penerbangan ke layanan backend. Tidak tersedia pada tingkat pecahan. KPL menyediakan metode khusus untuk mengambil metrik ini di tingkat global bagi pelanggan untuk mengelola tarif put mereka. Tingkat metrik: Terperinci Unit: Jumlah
`UserRecordsPut`	Hitung berapa banyak catatan pengguna logis yang berhasil dimasukkan. KPL tidak menghitung catatan gagal untuk metrik ini. Hal ini memungkinkan rata-rata untuk memberikan tingkat keberhasilan, hitungan untuk memberikan total upaya, dan perbedaan antara hitungan dan jumlah untuk memberikan hitungan kegagalan. Tingkat metrik: Ringkasan Unit: Jumlah
`UserRecordsDataPut`	Byte dalam catatan pengguna logis berhasil dimasukkan.

Metrik	Deskripsi
	Tingkat metrik: Terperinci Unit: Bita
`KinesisRecordsPut`	Hitungan berapa banyak catatan Kinesis Data Streams berhasil dimasukkan (setiap catatan Kinesis Data Streams dapat berisi beberapa catatan pengguna). KPL menghasilkan nol untuk catatan yang gagal. Hal ini memungkinkan rata-rata untuk memberikan tingkat keberhasilan, hitungan untuk memberikan total upaya, dan perbedaan antara hitungan dan jumlah untuk memberikan hitungan kegagalan. Tingkat metrik: Ringkasan Unit: Jumlah
`KinesisRecordsDataPut`	Byte dalam catatan Kinesis Data Streams. Tingkat metrik: Terperinci Unit: Bita
`ErrorsByCode`	Hitung setiap jenis kode kesalahan. Ini memperkenalkan dimensi tambahan`ErrorCode` , selain dimensi normal seperti `StreamName` dan`ShardId`. Tidak setiap kesalahan dapat ditelusuri ke pecahan. Kesalahan yang tidak dapat dilacak hanya dipancarkan pada tingkat aliran atau global. Metrik ini menangkap informasi tentang hal-hal seperti pelambatan, perubahan peta pecahan, kegagalan internal, layanan tidak tersedia, batas waktu, dan sebagainya. Kesalahan API Kinesis Data Streams dihitung satu kali per catatan Kinesis Data Streams. Beberapa catatan pengguna dalam catatan Kinesis Data Streams tidak menghasilkan beberapa hitungan. Tingkat metrik: Ringkasan Unit: Jumlah

Metrik	Deskripsi
`AllErrors`	Ini dipicu oleh kesalahan yang sama seperti Errors by Code, tetapi tidak membedakan antara tipe. Ini berguna sebagai monitor umum tingkat kesalahan tanpa memerlukan jumlah manual dari hitungan dari semua jenis kesalahan yang berbeda. Tingkat metrik: Ringkasan Unit: Jumlah
`RetriesPerRecord`	Jumlah percobaan ulang yang dilakukan per catatan pengguna. Nol dipancarkan untuk catatan yang berhasil dalam satu percobaan. Data dipancarkan pada saat catatan pengguna selesai (ketika berhasil atau tidak dapat lagi dicoba lagi). Jika catatan time-to-live adalah nilai yang besar, metrik ini mungkin tertunda secara signifikan. Tingkat metrik: Terperinci Unit: Jumlah
`BufferingTime`	Waktu antara catatan pengguna tiba di KPL dan berangkat ke backend. Informasi ini dikirimkan kembali ke pengguna berdasarkan per catatan, tetapi juga tersedia sebagai statistik agregat. Tingkat metrik: Ringkasan Satuan: Milidetik
`Request Time`	Waktu yang dibutuhkan untuk tampil`PutRecordsRequests` . Tingkat metrik: Terperinci Satuan: Milidetik

Metrik	Deskripsi
`User Records per Kinesis Record`	Jumlah catatan pengguna logis yang dikumpulkan ke dalam catatan Kinesis Data Streams tunggal. Tingkat metrik: Terperinci Unit: Jumlah
`Amazon Kinesis Records per PutRecord sRequest`	Jumlah catatan Kinesis Data Streams dikumpulkan menjadi satu. `PutRecordsRequest` Tidak tersedia pada tingkat pecahan. Tingkat metrik: Terperinci Unit: Jumlah
`User Records per PutRecord sRequest`	Jumlah total catatan pengguna yang terkandung dalam file`PutRecord sRequest` . Ini kira-kira setara dengan produk dari dua metrik sebelumnya. Tidak tersedia pada tingkat pecahan. Tingkat metrik: Terperinci Unit: Jumlah

Keamanan di Amazon Kinesis Data Streams

Keamanan cloud di AWS adalah prioritas tertinggi. Sebagai AWS pelanggan, Anda akan mendapat manfaat dari pusat data dan arsitektur jaringan yang dibangun untuk memenuhi persyaratan organisasi yang paling sensitif terhadap keamanan.

Keamanan adalah tanggung jawab bersama antara Anda AWS dan Anda. Model tanggung jawab bersama menjelaskan hal ini sebagai keamanan dari cloud dan keamanan di cloud:

- Keamanan cloud — AWS bertanggung jawab untuk melindungi infrastruktur yang menjalankan AWS layanan di AWS Cloud. AWS juga memberi Anda layanan yang dapat Anda gunakan dengan aman. Efektivitas keamanan kami diuji dan diverifikasi secara rutin oleh auditor pihak ketiga sebagai bagian dari program kepatuhanAWS. Untuk mempelajari tentang program kepatuhan yang berlaku untuk Kinesis Data Streams AWS , lihat Layanan dalam Lingkup berdasarkan Program Kepatuhan.
- Keamanan di cloud — Tanggung jawab Anda ditentukan oleh AWS layanan yang Anda gunakan. Anda juga bertanggung jawab atas faktor-faktor lain termasuk sensitivitas data Anda, persyaratan organisasi Anda, serta undang-undang dan peraturan yang berlaku.

Dokumentasi ini membantu Anda memahami cara menerapkan model tanggung jawab bersama saat menggunakan Kinesis Data Streams. Topik berikut menunjukkan cara mengonfigurasi Kinesis Data Streams untuk memenuhi tujuan keamanan dan kepatuhan Anda. Anda juga akan mempelajari cara menggunakan AWS layanan lain yang dapat membantu Anda memantau dan mengamankan sumber daya Kinesis Data Streams Anda.

Topik

Perlindungan Data di Amazon Kinesis Data Streams

Enkripsi sisi server menggunakan kunci AWS Key Management Service (AWS KMS) memudahkan Anda memenuhi persyaratan manajemen data yang ketat dengan mengenkripsi data Anda saat istirahat dalam Amazon Kinesis Data Streams.

> **Note**
>
> Jika Anda memerlukan modul kriptografi tervalidasi FIPS 140-2 saat mengakses AWS melalui antarmuka baris perintah atau API, gunakan titik akhir FIPS. Lihat informasi selengkapnya tentang titik akhir FIPS yang tersedia di Standar Pemrosesan Informasi Federal (FIPS) 140-2.

Topik

Apa itu Enkripsi Sisi Server untuk Kinesis Data Streams?

Enkripsi sisi server adalah fitur di Amazon Kinesis Data Streams yang secara otomatis mengenkripsi data sebelum diam dengan AWS KMS menggunakan kunci master pelanggan (CMK) yang Anda tentukan. Data dienkripsi sebelum ditulis ke lapisan penyimpanan aliran Kinesis, dan didekripsi setelah diambil dari penyimpanan. Akibatnya, data Anda dienkripsi saat istirahat dalam layanan Kinesis Data Streams. Ini memungkinkan Anda untuk memenuhi persyaratan peraturan yang ketat dan meningkatkan keamanan data Anda.

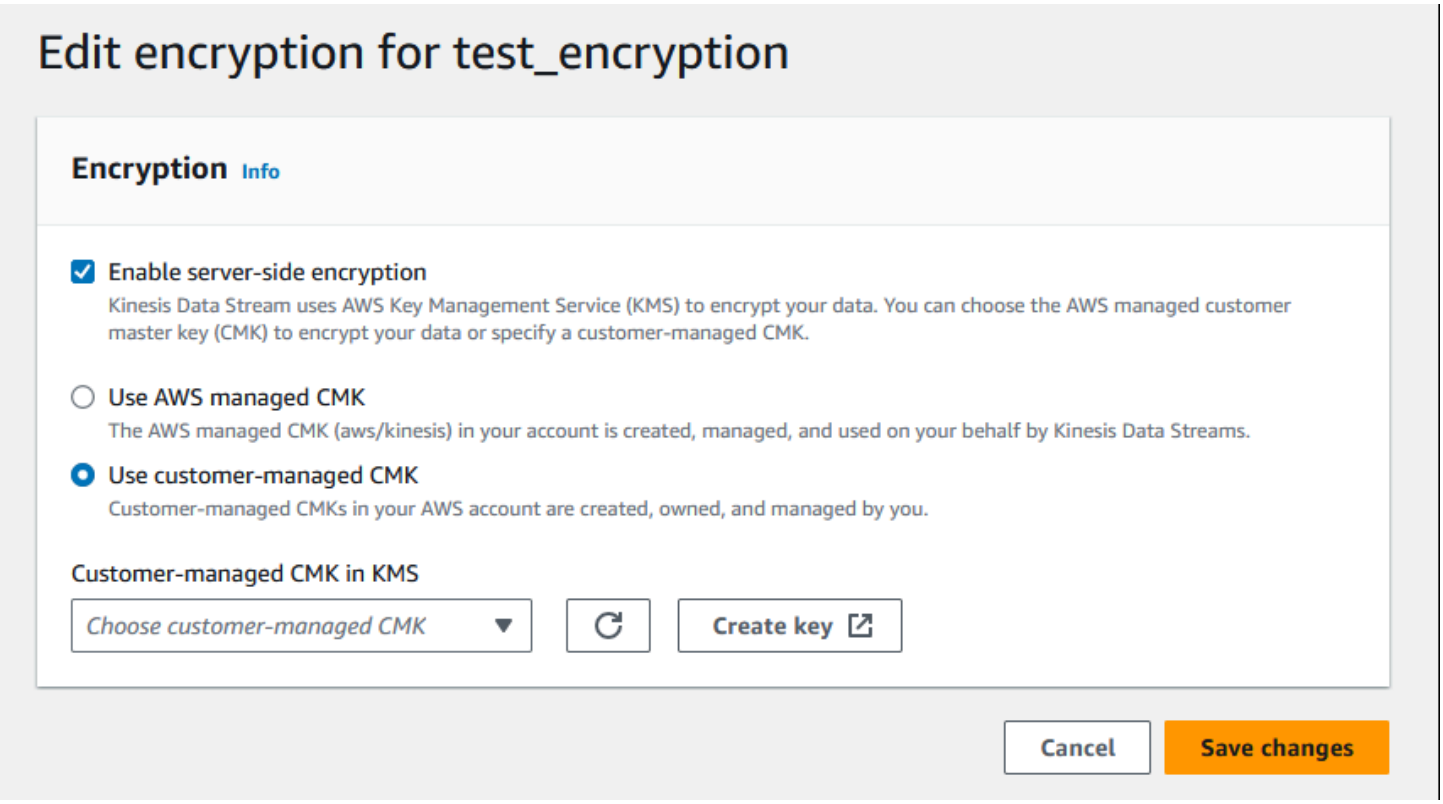
Dengan enkripsi sisi server, produsen dan konsumen Kinesis stream Anda tidak perlu mengelola kunci utama atau operasi kriptografi. Data Anda secara otomatis dienkripsi saat masuk dan meninggalkan layanan Kinesis Data Streams, sehingga data Anda saat istirahat dienkripsi. AWS KMS menyediakan semua kunci master yang digunakan oleh fitur enkripsi sisi server. AWS KMS

memudahkan penggunaan CMK untuk Kinesis yang dikelola AWSoleh, CMK yang AWS KMS ditentukan pengguna, atau kunci master yang diimpor ke layanan. AWS KMS

> **Note**
>
> Enkripsi sisi server mengenkripsi data yang masuk hanya setelah enkripsi diaktifkan. Data yang sudah ada sebelumnya dalam aliran yang tidak terenkripsi tidak dienkripsi setelah enkripsi sisi server diaktifkan.

Saat mengenkripsi aliran data dan berbagi akses ke prinsipal lain, Anda harus memberikan izin dalam kebijakan kunci untuk AWS KMS kunci dan kebijakan IAM di akun eksternal. Untuk informasi selengkapnya, lihat Mengizinkan pengguna di akun lain untuk menggunakan kunci KMS.

Jika Anda telah mengaktifkan enkripsi sisi server untuk aliran data dengan kunci KMS AWS terkelola dan ingin berbagi akses melalui kebijakan sumber daya, Anda harus beralih menggunakan kunci yang dikelola pelanggan (CMK), seperti yang ditunjukkan berikut:



Selain itu, Anda harus mengizinkan entitas utama berbagi Anda untuk memiliki akses ke CMK Anda, menggunakan kemampuan berbagi lintas akun KMS. Pastikan juga untuk membuat perubahan

dalam kebijakan IAM untuk entitas utama berbagi. Untuk informasi selengkapnya, lihat Mengizinkan pengguna di akun lain untuk menggunakan kunci KMS.

Biaya, Wilayah, dan Pertimbangan Kinerja

Ketika Anda menerapkan enkripsi sisi server, Anda tunduk pada penggunaan AWS KMS API dan biaya utama. Tidak seperti kunci master KMS khusus, kunci master `(Default) aws/kinesis` pelanggan (CMK) ditawarkan secara gratis. Namun, Anda tetap harus membayar biaya penggunaan API yang dikeluarkan Amazon Kinesis Data Streams atas nama Anda.

Biaya penggunaan API berlaku untuk setiap CMK, termasuk yang khusus. Kinesis Data AWS KMS Streams memanggil kira-kira setiap lima menit ketika memutar kunci data. Dalam 30 hari, total biaya panggilan AWS KMS API yang diprakarsai oleh aliran Kinesis harus kurang dari beberapa dolar. Biaya ini disesuaikan dengan jumlah kredensil pengguna yang Anda gunakan pada produsen data dan konsumen Anda karena setiap kredensi pengguna memerlukan panggilan API yang unik. AWS KMSBila Anda menggunakan peran IAM untuk autentikasi, masing-masing menganggap panggilan peran menghasilkan kredensi pengguna yang unik. Untuk menghemat biaya KMS, Anda mungkin ingin menyimpan kredenal pengguna cache yang dikembalikan oleh panggilan peran asumsi.

Berikut ini menjelaskan biaya berdasarkan sumber daya:

Kunci

- CMK untuk Kinesis yang dikelola AWS oleh (alias `aws/kinesis` =) gratis.
- Kunci KMS buatan pengguna dikenakan biaya utama KMS. Untuk informasi selengkapnya, lihat Harga Layanan ManajemenAWS Utama.

Biaya penggunaan API berlaku untuk setiap CMK, termasuk yang khusus. Kinesis Data Streams memanggil KMS kira-kira setiap 5 menit ketika memutar kunci data. Dalam bulan 30 hari, total biaya panggilan API KMS yang diprakarsai oleh aliran data Kinesis harus kurang dari beberapa dolar. Harap dicatat bahwa biaya ini disesuaikan dengan jumlah kredensional pengguna yang Anda gunakan pada produsen dan konsumen data Anda karena setiap kredensi pengguna memerlukan panggilan API unik ke KMS. AWS Bila Anda menggunakan peran IAM untuk otentikasi, masing-masing assume-role-call akan menghasilkan kredensi pengguna yang unik dan Anda mungkin ingin menyimpan kredensi pengguna cache yang dikembalikan oleh untuk menghemat biaya KMS. assume-role-call

Penggunaan API KMS

Untuk setiap aliran terenkripsi, ketika membaca dari TIP dan menggunakan satu akun IAM/kunci akses pengguna di seluruh pembaca dan penulis, layanan Kinesis memanggil layanan sekitar 12 kali setiap 5 menit AWS KMS . Tidak membaca dari TIP dapat menyebabkan panggilan yang lebih tinggi ke AWS KMS layanan. Permintaan API untuk menghasilkan kunci enkripsi data baru dikenakan biaya AWS KMS penggunaan. Untuk informasi selengkapnya, lihat Harga Layanan ManajemenAWS Utama: Penggunaan.

Ketersediaan Enkripsi Sisi Server menurut Wilayah

Saat ini, enkripsi sisi server dari aliran Kinesis tersedia di semua wilayah yang didukung untuk Kinesis Data Streams, termasuk (AS-Barat), dan wilayah China. AWS GovCloud Untuk informasi selengkapnya tentang wilayah yang didukung untuk Kinesis Data Streams, lihat https://docs.aws.amazon.com/general/latest/gr/ak.html.

Pertimbangan Kinerja

Karena overhead layanan penerapan enkripsi, menerapkan enkripsi sisi server meningkatkan latensi tipikal`PutRecord`,`PutRecords`, dan `GetRecords` kurang dari 100µs.

Bagaimana Saya Memulai Enkripsi Sisi Server?

Cara termudah untuk memulai dengan enkripsi sisi server adalah dengan menggunakan AWS Management Console dan Kunci Layanan Amazon Kinesis KMS,. `aws/kinesis`

Prosedur berikut menunjukkan cara mengaktifkan enkripsi sisi server untuk aliran Kinesis.

Untuk mengaktifkan enkripsi sisi server untuk aliran Kinesis

1. Masuk ke AWS Management Console dan buka konsol Amazon Kinesis Data Streams.
2. Buat atau pilih aliran Kinesis di file. AWS Management Console
3. Pilih tab detail.
4. Di enkripsi sisi server, pilih edit.
5. Kecuali Anda ingin menggunakan kunci master KMS buatan pengguna, pastikan kunci master KMS aws/kinesis (Default) dipilih. Ini adalah kunci master KMS yang dihasilkan oleh layanan Kinesis. Pilih Diaktifkan, lalu pilih Simpan.

> **Note**
>
> Kunci master layanan Kinesis default gratis, namun panggilan API yang dilakukan oleh Kinesis ke AWS KMS layanan dikenakan biaya penggunaan KMS.

6. Aliran transisi melalui status tertunda. Setelah aliran kembali ke status aktif dengan enkripsi diaktifkan, semua data masuk yang ditulis ke aliran dienkripsi menggunakan kunci master KMS yang Anda pilih.
7. Untuk menonaktifkan enkripsi sisi server, pilih Dinonaktifkan di enkripsi sisi server di, lalu pilih Simpan. AWS Management Console

Membuat dan Menggunakan Kunci Master KMS Buatan Pengguna

Bagian ini menjelaskan cara membuat dan menggunakan kunci master KMS Anda sendiri, alih-alih menggunakan kunci master yang dikelola oleh Amazon Kinesis.

Membuat Kunci Master KMS Buatan Pengguna

Untuk petunjuk cara membuat kunci master Anda sendiri, lihat Membuat Kunci di PanduanAWS Key Management Service Pengembang. Setelah Anda membuat kunci untuk akun Anda, layanan Kinesis Data Streams mengembalikan kunci ini dalam daftar kunci master KMS.

Menggunakan Kunci Master KMS Buatan Pengguna

Setelah izin yang benar diterapkan ke konsumen, produsen, dan administrator Anda, Anda dapat menggunakan kunci master KMS khusus di akun Anda sendiri atau AWS akun lain. AWS Semua kunci master KMS di akun Anda muncul di daftar Kunci Master KMS di dalam. AWS Management Console

Untuk menggunakan kunci master KMS khusus yang terletak di akun lain, Anda memerlukan izin untuk menggunakan kunci tersebut. Anda juga harus menentukan ARN dari kunci master KMS di kotak input ARN di. AWS Management Console

Izin untuk Menggunakan Kunci Master KMS Buatan Pengguna

Sebelum Anda dapat menggunakan enkripsi sisi server dengan kunci master KMS buatan pengguna, Anda harus mengonfigurasi kebijakan AWS KMS kunci untuk mengizinkan enkripsi aliran dan

enkripsi serta dekripsi catatan aliran. Untuk contoh dan informasi selengkapnya tentang AWS KMS izin, lihat Izin APIAWS KMS: Referensi Tindakan dan Sumber Daya.

> **Note**
>
> Penggunaan kunci layanan default untuk enkripsi tidak memerlukan penerapan izin IAM khusus.

Sebelum Anda menggunakan kunci master KMS buatan pengguna, pastikan bahwa produsen dan konsumen Kinesis stream Anda (prinsip IAM) adalah pengguna dalam kebijakan kunci utama KMS. Jika tidak, menulis dan membaca dari aliran akan gagal, yang pada akhirnya dapat mengakibatkan kehilangan data, pemrosesan tertunda, atau aplikasi yang macet. Anda dapat mengelola izin untuk kunci KMS menggunakan kebijakan IAM. Untuk informasi selengkapnya, lihat Menggunakan Kebijakan IAM dengan AWS KMS.

Contoh Izin Produser

Produsen aliran Kinesis Anda harus memiliki izin. `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"
    }
  ]
}
```

Contoh Izin Konsumen

Konsumen Kinesis stream Anda harus memiliki izin. `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"
    }
  ]
}
```

Amazon Managed Service untuk Apache Flink dan AWS Lambda menggunakan peran untuk menggunakan aliran Kinesis. Pastikan untuk menambahkan `kms:Decrypt` izin ke peran yang digunakan konsumen ini.

Izin Administrator Stream

Administrator aliran Kinesis harus memiliki otorisasi untuk menelepon dan. `kms:List*` `kms:DescribeKey*`

Memverifikasi dan Memecahkan Masalah Izin Kunci KMS

Setelah mengaktifkan enkripsi pada aliran Kinesis, sebaiknya Anda memantau keberhasilan`putRecords`, `getRecords` dan panggilan menggunakan metrik Amazon CloudWatch berikut: `putRecord`

- `PutRecord.Success`

- `PutRecords.Success`
- `GetRecords.Success`

Lihat informasi yang lebih lengkap di Memantau Aliran Data Amazon Kinesis

Menggunakan Amazon Kinesis Data Streams dengan Titik Akhir VPC Antarmuka

Anda dapat menggunakan titik akhir VPC antarmuka untuk menjaga lalu lintas antara Amazon VPC dan Kinesis Data Streams agar tidak meninggalkan jaringan Amazon. Endpoint VPC antarmuka tidak memerlukan gateway internet, perangkat NAT, koneksi VPN, atau koneksi. AWS Direct Connect Endpoint VPC antarmuka didukung oleh AWS PrivateLink, sebuah AWS teknologi yang memungkinkan komunikasi pribadi antar AWS layanan menggunakan antarmuka network elastis dengan IP pribadi di Amazon VPC Anda. Untuk informasi selengkapnya, lihat Amazon Virtual Private Cloud dan Interface VPC Endpoints ().AWS PrivateLink

Topik

- Menggunakan Endpoint VPC Antarmuka untuk Kinesis Data Streams
- Mengontrol Akses ke Titik Akhir VPCE untuk Kinesis Data Streams
- Ketersediaan Kebijakan Titik Akhir VPC untuk Kinesis Data Streams

Menggunakan Endpoint VPC Antarmuka untuk Kinesis Data Streams

Untuk memulai, Anda tidak perlu mengubah pengaturan untuk aliran, produsen, atau konsumen Anda. Cukup buat titik akhir VPC antarmuka agar lalu lintas Kinesis Data Streams Anda dari dan ke sumber daya VPC Amazon Anda mulai mengalir melalui titik akhir VPC antarmuka. Untuk informasi selengkapnya, lihat Membuat Titik Akhir Antarmuka.

Layanan panggilan Kinesis Producer Library (KPL) dan Kinesis Consumer Library (KCL) seperti AWS Amazon dan Amazon CloudWatch DynamoDB menggunakan titik akhir publik atau titik akhir VPC antarmuka pribadi, mana saja yang digunakan. Misalnya, jika aplikasi KCL Anda berjalan di VPC dengan antarmuka DynamoDB dengan titik akhir VPC diaktifkan, panggilan antara DynamoDB dan aplikasi KCL Anda mengalir melalui titik akhir VPC antarmuka.

Mengontrol Akses ke Titik Akhir VPCE untuk Kinesis Data Streams

Kebijakan titik akhir VPC memungkinkan Anda mengontrol akses dengan melampirkan kebijakan ke titik akhir VPC atau dengan menggunakan bidang tambahan dalam kebijakan yang dilampirkan ke pengguna, grup, atau peran IAM untuk membatasi akses hanya terjadi melalui titik akhir VPC yang ditentukan. Kebijakan ini dapat digunakan untuk membatasi akses ke aliran tertentu ke titik akhir VPC tertentu saat digunakan bersama dengan kebijakan IAM untuk hanya memberikan akses ke tindakan aliran data Kinesis melalui titik akhir VPC yang ditentukan.

Berikut ini adalah contoh kebijakan endpoint untuk mengakses aliran data Kinesis.

- Contoh kebijakan VPC: akses hanya-baca - kebijakan sampel ini dapat dilampirkan ke titik akhir VPC. (Untuk informasi selengkapnya, lihat Mengontrol Akses ke Sumber Daya VPC Amazon). Ini membatasi tindakan untuk hanya mencantumkan dan menggambarkan aliran data Kinesis melalui titik akhir VPC yang dilampirkan.

```
{
  "Statement": [
    {
      "Sid": "ReadOnly",
      "Principal": "*",
      "Action": [
        "kinesis:List*",
        "kinesis:Describe*"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

- Contoh kebijakan VPC: batasi akses ke aliran data Kinesis tertentu - kebijakan sampel ini dapat dilampirkan ke titik akhir VPC. Ini membatasi akses ke aliran data tertentu melalui titik akhir VPC yang dilampirkan.

```
{
  "Statement": [
    {
      "Sid": "AccessToSpecificDataStream",
```

```
      "Principal": "*",
      "Action": "kinesis:*",
      "Effect": "Allow",
      "Resource": "arn:aws:kinesis:us-east-1:123456789012:stream/MyStream"
    }
  ]
}
```

- Contoh kebijakan IAM: batasi akses ke Stream tertentu dari titik akhir VPC tertentu saja - kebijakan contoh ini dapat dilampirkan ke pengguna, peran, atau grup IAM. Ini membatasi akses ke aliran data Kinesis tertentu hanya terjadi dari titik akhir VPC tertentu.

```
{
   "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" } }
      }
   ]
}
```

Ketersediaan Kebijakan Titik Akhir VPC untuk Kinesis Data Streams

Titik akhir VPC antarmuka Kinesis Data Streams dengan kebijakan didukung di wilayah berikut:

- Eropa (Paris)
- Eropa (Irlandia)
- AS Timur (Virginia Utara)
- Eropa (Stockholm)
- AS Timur (Ohio)
- Eropa (Frankfurt)
- Amerika Selatan (Sao Paulo)

- Eropa (London)
- Asia Pasifik (Tokyo)
- AS Barat (California Utara)
- Asia Pasifik (Singapura)
- Asia Pasifik (Sydney)
- China (Beijing)
- China (Ningxia)
- Asia Pasifik (Hong Kong)
- Middle East (Bahrain) (Middle East (Bahrain))
- Middle East (UAE)
- Eropa (Milan)
- Afrika (Cape Town)
- Asia Pasifik (Mumbai)
- Asia Pasifik (Seoul)
- Kanada (Pusat)
- AS Barat (Oregon) kecuali usw2-az4
- AWS GovCloud (AS-Timur)
- AWS GovCloud (AS-Barat)
- Asia Pasifik (Osaka)
- Eropa (Zurich)
- Asia Pasifik (Hyderabad)

Mengontrol Akses ke Sumber Daya Amazon Kinesis Data Streams Menggunakan IAM

AWS Identity and Access Management (IAM) memungkinkan Anda untuk melakukan hal berikut:

- Buat pengguna dan grup di bawah AWS akun Anda
- Tetapkan kredensi keamanan unik untuk setiap pengguna di bawah akun Anda AWS
- Kontrol izin setiap pengguna untuk melakukan tugas menggunakan sumber daya AWS
- Izinkan pengguna di AWS akun lain untuk membagikan AWS sumber daya Anda

- Buat peran untuk AWS akun Anda dan tentukan pengguna atau layanan yang dapat mengasumsikan mereka
- Gunakan identitas yang ada untuk perusahaan Anda untuk memberikan izin untuk melakukan tugas menggunakan sumber daya AWS

Dengan menggunakan IAM dengan Kinesis Data Streams, Anda dapat mengontrol apakah pengguna di organisasi dapat melakukan tugas menggunakan tindakan API Kinesis Data Streams tertentu dan apakah mereka dapat menggunakan sumber daya tertentu. AWS

Jika Anda mengembangkan aplikasi menggunakan Kinesis Client Library (KCL), kebijakan Anda harus menyertakan izin untuk Amazon DynamoDB dan Amazon CloudWatch; KCL menggunakan DynamoDB untuk melacak informasi status aplikasi, dan mengirim metrik KCL ke atas nama Anda. CloudWatch CloudWatch Untuk informasi lebih lanjut tentang KCL, lihatMengembangkan Konsumen KCL 1.x.

Untuk informasi selengkapnya tentang IAM, lihat berikut ini:

- AWS Identity and Access Management (IAM)
- Memulai
- Panduan Pengguna IAM

Untuk informasi selengkapnya tentang IAM dan Amazon DynamoDB, lihat Menggunakan IAM untuk Mengontrol Akses ke Sumber Daya Amazon DynamoDB di Panduan Pengembang Amazon DynamoDB.

Untuk informasi selengkapnya tentang IAM dan Amazon CloudWatch, lihat Mengontrol Akses Pengguna ke AWS Akun Anda di Panduan CloudWatch Pengguna Amazon.

Daftar Isi

Sintaks Kebijakan

kebijakan IAM adalah dokumen JSON yang terdiri dari satu atau beberapa pernyataan. Setiap pernyataan memiliki struktur sebagai berikut:

```
{
  "Statement":[{
    "Effect":"effect",
    "Action":"action",
    "Resource":"arn",
    "Condition":{
      "condition":{
        "key":"value"
        }
      }
    }
  ]
}
```

Ada berbagai elemen yang membentuk pernyataan:

- Efek: Efek bisa berupa `Allow` atau `Deny`. Secara default, para pengguna IAM tidak memiliki izin untuk menggunakan sumber daya dan tindakan API, jadi semua permintaan akan ditolak. izin eksplisit akan menggantikan izin default. penolakan eksplisit akan menggantikan izin apa pun.
- Tindakan: Tindakan adalah tindakan API tertentu yang Anda izinkan atau tolak.
- Sumber Daya: Sumber daya yang dipengaruhi oleh tindakan. Untuk menentukan sumber daya dalam sebuah pernyataan kebijakan IAM, gunakan Amazon Resource Name (ARN).
- Syarat: Syarat bersifat opsional. Ketentuan ini dapat digunakan untuk mengontrol kapan kebijakan Anda akan berlaku.

Saat Anda membuat dan mengelola kebijakan IAM, Anda mungkin ingin menggunakan IAM Policy Generator dan IAM Policy Simulator.

Tindakan untuk Kinesis Data Streams

Dalam pernyataan kebijakan IAM, Anda dapat menentukan tindakan API apa pun dari layanan apa pun yang mendukung IAM. Untuk Kinesis Data Streams, gunakan awalan berikut dengan nama tindakan API:. `kinesis:` Misalnya:`kinesis:CreateStream`,`kinesis:ListStreams`, dan`kinesis:DescribeStreamSummary`.

Untuk menetapkan beberapa tindakan dalam satu pernyataan, pisahkan tindakan-tindakan tersebut menggunakan koma seperti berikut:

```
"Action": ["kinesis:action1", "kinesis:action2"]
```

Anda juga dapat menentukan beberapa tindakan menggunakan wildcard. Misalnya, Anda dapat menentukan semua tindakan yang namanya dimulai dengan kata “Dapatkan” sebagai berikut:

```
"Action": "kinesis:Get*"
```

Untuk menentukan semua operasi Kinesis Data Streams, gunakan wildcard * sebagai berikut:

```
"Action": "kinesis:*"
```

Untuk daftar lengkap tindakan API Kinesis Data Streams, lihat Referensi API Amazon Kinesis.

Nama Sumber Daya Amazon (ARN) untuk Kinesis Data Streams

Setiap pernyataan kebijakan IAM berlaku untuk sumber daya yang Anda tentukan menggunakan ARN.

Gunakan format sumber daya ARN berikut untuk aliran data Kinesis:

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

Sebagai contoh:

```
"Resource": arn:aws:kinesis:*:111122223333:stream/my-stream
```

Contoh Kebijakan untuk Kinesis Data Streams

Contoh kebijakan berikut menunjukkan bagaimana Anda dapat mengontrol akses pengguna ke aliran data Kinesis Anda.

Example 1: Allow users to get data from a stream

Example

Kebijakan ini memungkinkan pengguna atau grup untuk melakukan`DescribeStreamSummary`,`GetShardIterator`, dan `GetRecords` operasi pada

aliran tertentu dan `ListStreams` pada aliran apa pun. Kebijakan ini dapat diterapkan pada pengguna yang seharusnya bisa mendapatkan data dari aliran tertentu.

```
{
    "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

Kebijakan ini memungkinkan pengguna atau grup untuk menggunakan `PutRecord` operasi dengan salah satu aliran akun. Kebijakan ini dapat diterapkan pada pengguna yang harus dapat menambahkan catatan data ke semua aliran di akun.

```
{
    "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

Kebijakan ini memungkinkan pengguna atau grup untuk menggunakan operasi Kinesis Data Streams apa pun pada aliran yang ditentukan. Kebijakan ini dapat diterapkan pada pengguna yang harus memiliki kontrol administratif atas aliran tertentu.

```
{
    "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

Kebijakan ini memungkinkan pengguna atau grup untuk menggunakan operasi Kinesis Data Streams apa pun pada aliran apa pun di akun. Karena kebijakan ini memberikan akses penuh ke semua aliran Anda, Anda harus membatasinya hanya untuk administrator.

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": [
                "arn:aws:kinesis:*:111122223333:stream/*"
            ]
        }
    ]
}
```

Berbagi aliran data Anda dengan akun lain

Lampirkan kebijakan berbasis sumber daya ke aliran data Anda untuk memberikan akses ke akun lain, pengguna IAM, atau peran IAM. Kebijakan berbasis sumber daya adalah dokumen kebijakan JSON yang Anda lampirkan ke sumber daya seperti aliran data. Kebijakan ini memberikan izin pokok yang ditentukan untuk melakukan tindakan spesifik pada sumber daya tersebut dan menentukan dalam kondisi apa hal ini berlaku. Kebijakan dapat memiliki beberapa pernyataan. Anda harus menentukan pengguna utama dalam kebijakan berbasis sumber daya. Prinsipal dapat mencakup akun, pengguna, peran, pengguna federasi, atau layanan. AWS Anda dapat mengonfigurasi kebijakan di konsol Kinesis Data Streams, API, atau SDK.

Perhatikan bahwa berbagi akses ke konsumen terdaftar seperti Enhanced Fan Out memerlukan kebijakan tentang ARN aliran data dan ARN konsumen.

Mengaktifkan akses lintas akun

Untuk mengaktifkan akses lintas akun, Anda dapat menentukan seluruh akun atau entitas IAM di akun lain sebagai pengguna utama dalam kebijakan berbasis sumber daya. Menambahkan pengguna utama lintas akun ke kebijakan berbasis sumber daya bagian dari membangun hubungan kepercayaan. Ketika prinsipal dan sumber daya berada di AWS akun terpisah, Anda juga harus menggunakan kebijakan berbasis identitas untuk memberikan akses utama ke sumber daya. Namun, jika kebijakan berbasis sumber daya memberikan akses kepada pengguna utama dalam akun yang sama, kebijakan berbasis identitas lainnya tidak diperlukan.

Untuk informasi selengkapnya tentang penggunaan kebijakan berbasis sumber daya untuk akses lintas akun, lihat Akses sumber daya lintas akun di IAM.

Administrator aliran data dapat menggunakan AWS Identity and Access Management kebijakan untuk menentukan siapa yang memiliki akses ke apa. Yaitu, pengguna utama mana yang dapat

melakukan tindakan pada sumber daya apa, dan dalam kondisi apa. Elemen `Action` dari kebijakan JSON menjelaskan tindakan yang dapat Anda gunakan untuk mengizinkan atau menolak akses dalam sebuah kebijakan. Tindakan kebijakan biasanya memiliki nama yang sama dengan operasi AWS API terkait.

Tindakan Kinesis Data Streams yang dapat dibagikan:

Tindakan	Tingkat akses
DescribeStreamConsumer	Konsumen
DescribeStreamSummary	Aliran data
GetRecords	Aliran data
GetShardIterator	Aliran data
ListShards	Aliran data
PutRecord	Aliran data
PutRecords	Aliran data
SubscribeToShard	Konsumen

Berikut ini adalah contoh penggunaan kebijakan berbasis sumber daya untuk memberikan akses lintas akun ke aliran data Anda atau konsumen terdaftar.

Untuk melakukan tindakan lintas akun, Anda harus menentukan ARN aliran untuk akses aliran data dan ARN konsumen untuk akses konsumen terdaftar.

Contoh kebijakan berbasis sumber daya untuk Kinesis Data Streams

Berbagi konsumen terdaftar melibatkan kebijakan aliran data dan kebijakan konsumen karena tindakan yang diperlukan.

> **Note**
>
> Berikut ini adalah contoh nilai yang valid untuk`Principal`:

- {"AWS": "123456789012"}
- Pengguna IAM - {"AWS": "arn:aws:iam::123456789012:user/user-name"}
- Peran IAM - {"AWS":["arn:aws:iam::123456789012:role/role-name"]}
- Beberapa Prinsipal (dapat berupa kombinasi akun, pengguna, peran) - {"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

Pernyataan kebijakan aliran data:

```
{
    "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"
        }
    ]
```

```
}
```

Pernyataan kebijakan konsumen:

```
{
    "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 (*) tidak didukung untuk tindakan atau bidang utama untuk mempertahankan prinsip hak istimewa yang paling rendah..

Mengelola kebijakan untuk aliran data Anda secara terprogram

Di luar AWS Management Console, Kinesis Data Streams memiliki tiga API untuk mengelola kebijakan aliran data Anda:

- PutResourcePolicy
- GetResourcePolicy
- DeleteResourcePolicy

Gunakan `PutResourePolicy` untuk melampirkan atau menimpa kebijakan untuk aliran data atau konsumen. Gunakan `GetResourcePolicy` untuk memeriksa dan melihat kebijakan untuk aliran

data atau konsumen yang ditentukan. Gunakan `DeleteResourcePolicy` untuk menghapus kebijakan untuk aliran data atau konsumen yang ditentukan.

Batas kebijakan

Kebijakan sumber daya Kinesis Data Streams memiliki batasan sebagai berikut:

- Wildcard (*) tidak didukung untuk tindakan atau bagian utama untuk mempertahankan prinsip hak istimewa paling sedikit.
- AWS Kepala Pelayanan tidak didukung oleh kepala sekolah untuk mencegah calon deputi yang bingung.
- Prinsipal federasi tidak didukung.
- ID pengguna kanonik tidak didukung.
- Ukuran polis tidak boleh melebihi 20KB.

Berbagi akses ke data terenkripsi

Jika Anda telah mengaktifkan enkripsi sisi server untuk aliran data dengan kunci KMS AWS terkelola dan ingin berbagi akses melalui kebijakan sumber daya, Anda harus beralih menggunakan kunci yang dikelola pelanggan (CMK). Untuk informasi selengkapnya, lihat Apa itu Enkripsi Sisi Server untuk Kinesis Data Streams?. Selain itu, Anda harus mengizinkan entitas utama berbagi Anda untuk memiliki akses ke CMK Anda, menggunakan kemampuan berbagi lintas akun KMS. Pastikan juga untuk membuat perubahan dalam kebijakan IAM untuk entitas utama berbagi. Untuk informasi selengkapnya, lihat Mengizinkan pengguna di akun lain untuk menggunakan kunci KMS.

Konfigurasikan AWS Lambda fungsi untuk membaca dari Kinesis Data Streams di akun lain

Untuk contoh cara mengonfigurasi fungsi Lambda untuk membaca dari Kinesis Data Streams di akun lain, lihat. Berbagi akses dengan fungsi lintas akun AWS Lambda

Berbagi akses menggunakan kebijakan berbasis sumber daya

> **Note**
>
> Memperbarui kebijakan berbasis sumber daya yang ada berarti mengganti yang sudah ada, jadi pastikan untuk memasukkan semua informasi yang diperlukan dalam kebijakan baru Anda.

Berbagi akses dengan fungsi lintas akun AWS Lambda

Operator Lambda

1. Buka konsol IAM untuk membuat peran IAM yang akan digunakan sebagai peran eksekusi Lambda untuk fungsi Anda. AWS Lambda Tambahkan kebijakan IAM terkelola `AWSLambdaKinesisExecutionRole` yang memiliki Kinesis Data Streams dan izin pemanggilan Lambda yang diperlukan. Kebijakan ini juga memberikan akses ke semua sumber daya Kinesis Data Streams potensial yang mungkin dapat Anda akses.
2. Di AWS Lambda konsol, buat AWS Lambda fungsi untuk memproses rekaman dalam aliran data Kinesis Data Streams dan selama penyiapan untuk peran eksekusi, pilih peran yang Anda buat di langkah sebelumnya.
3. Berikan peran eksekusi kepada pemilik sumber daya Kinesis Data Streams untuk mengonfigurasi kebijakan sumber daya.
4. Selesai mengatur fungsi Lambda.

Pemilik sumber daya Kinesis Data Streams

1. Dapatkan peran eksekusi Lambda lintas akun yang akan memanggil fungsi Lambda.
2. Di konsol Amazon Kinesis Data Streams, pilih aliran data. Pilih tab Berbagi aliran data dan kemudian tombol Buat kebijakan berbagi untuk memulai editor kebijakan visual. Untuk berbagi konsumen terdaftar dalam aliran data, pilih konsumen, lalu pilih Buat kebijakan berbagi. Anda juga dapat menulis kebijakan JSON secara langsung.
3. Tentukan peran eksekusi Lambda lintas akun sebagai tindakan utama dan Kinesis Data Streams yang tepat yang Anda bagikan aksesnya. Pastikan untuk menyertakan tindakan`kinesis:DescribeStream`. Untuk informasi selengkapnya tentang contoh kebijakan

sumber daya untuk Kinesis Data Contoh kebijakan berbasis sumber daya untuk Kinesis Data Streams Streams, lihat.

4. Pilih Buat kebijakan atau gunakan PutResourcePolicyuntuk melampirkan kebijakan ke sumber daya Anda.

Berbagi akses dengan konsumen KCL lintas akun

- Jika Anda menggunakan KCL 1.x, pastikan Anda menggunakan KCL 1.15.0 atau lebih tinggi.
- Jika Anda menggunakan KCL 2.x, pastikan Anda menggunakan KCL 2.5.3 atau lebih tinggi.

Operator KCL

1. Berikan peran pengguna IAM atau IAM Anda yang akan menjalankan aplikasi KCL ke pemilik sumber daya.
2. Mintalah pemilik sumber daya untuk aliran data atau ARN konsumen.
3. Pastikan Anda menentukan ARN aliran yang disediakan sebagai bagian dari konfigurasi KCL Anda.
   - Untuk KCL 1.x: gunakan KinesisClientLibConfigurationkonstruktor dan berikan arus ARN.
   - Untuk KCL 2.x: Anda dapat menyediakan ARN streaming saja atau ke Perpustakaan Klien StreamTrackerKinesis. ConfigsBuilder Untuk StreamTracker, sediakan aliran ARN dan Epoch pembuatan dari DynamoDB Lease Table yang dihasilkan oleh perpustakaan. Jika Anda ingin membaca dari konsumen terdaftar bersama seperti Enhanced Fan-Out, gunakan StreamTracker dan juga berikan ARN kepada konsumen.

Pemilik sumber daya Kinesis Data Streams

1. Dapatkan peran pengguna IAM lintas akun atau IAM yang akan menjalankan aplikasi KCL.
2. Di konsol Amazon Kinesis Data Streams, pilih aliran data. Pilih tab Berbagi aliran data dan kemudian tombol Buat kebijakan berbagi untuk memulai editor kebijakan visual. Untuk berbagi konsumen terdaftar dalam aliran data, pilih konsumen, lalu pilih Buat kebijakan berbagi. Anda juga dapat menulis kebijakan JSON secara langsung.
3. Tentukan peran IAM atau IAM aplikasi KCL lintas akun sebagai tindakan utama dan tindakan Kinesis Data Streams yang tepat yang Anda bagikan aksesnya. Untuk informasi selengkapnya

tentang contoh kebijakan sumber daya untuk Kinesis Data Contoh kebijakan berbasis sumber daya untuk Kinesis Data Streams Streams, lihat.

4. Pilih Buat kebijakan atau gunakan PutResourcePolicyuntuk melampirkan kebijakan ke sumber daya Anda.

Berbagi akses ke data terenkripsi

Jika Anda telah mengaktifkan enkripsi sisi server untuk aliran data dengan kunci KMS AWS terkelola dan ingin berbagi akses melalui kebijakan sumber daya, Anda harus beralih menggunakan kunci yang dikelola pelanggan (CMK). Untuk informasi selengkapnya, lihat Apa itu Enkripsi Sisi Server untuk Kinesis Data Streams?. Selain itu, Anda harus mengizinkan entitas utama berbagi Anda untuk memiliki akses ke CMK Anda, menggunakan kemampuan berbagi lintas akun KMS. Pastikan juga untuk membuat perubahan dalam kebijakan IAM untuk entitas utama berbagi. Untuk informasi selengkapnya, lihat Mengizinkan pengguna di akun lain untuk menggunakan kunci KMS.

Validasi Kepatuhan untuk Amazon Kinesis Data Streams

Auditor pihak ketiga menilai keamanan dan kepatuhan Amazon Kinesis Data Streams sebagai bagian dari beberapa program kepatuhan. AWS Program ini mencakup SOC, PCI, FedRAMP, HIPAA, dan lainnya.

Untuk daftar AWS layanan dalam lingkup program kepatuhan tertentu, lihat AWS Layanan dalam Lingkup berdasarkan Program Kepatuhan. Untuk informasi umum, lihat Program KepatuhanAWS.

Anda dapat mengunduh laporan audit pihak ketiga menggunakan AWS Artifact. Untuk informasi selengkapnya, lihat Mengunduh Laporan di AWS Artifak.

Tanggung jawab kepatuhan Anda saat menggunakan Kinesis Data Streams ditentukan oleh sensitivitas data Anda, tujuan kepatuhan perusahaan Anda, serta undang-undang dan peraturan yang berlaku. Jika penggunaan Kinesis Data Streams oleh Anda tunduk pada kepatuhan terhadap standar seperti HIPAA, PCI, atau AWS FedRAMP, menyediakan sumber daya untuk membantu:

- Panduan Memulai Cepat Keamanan dan Kepatuhan — Panduan penerapan ini membahas pertimbangan arsitektur dan memberikan langkah-langkah untuk menerapkan lingkungan dasar yang berfokus pada keamanan dan kepatuhan. AWS
- Arsitektur untuk Whitepaper Keamanan dan Kepatuhan HIPAA — Whitepaper ini menjelaskan bagaimana perusahaan dapat menggunakan untuk membuat aplikasi yang sesuai dengan HIPAA. AWS

- AWS Sumber Daya Kepatuhan - Kumpulan buku kerja dan panduan ini yang mungkin berlaku untuk industri dan lokasi Anda
- AWS Config AWS Layanan ini yang menilai seberapa baik konfigurasi sumber daya Anda mematuhi praktik internal, pedoman industri, dan peraturan.
- AWS Security Hub— AWS Layanan ini memberikan pandangan komprehensif tentang keadaan keamanan Anda di dalamnya AWS yang membantu Anda memeriksa kepatuhan Anda terhadap standar industri keamanan dan praktik terbaik.

Ketahanan dalam Aliran Data Amazon Kinesis

Infrastruktur AWS global dibangun di sekitar AWS Wilayah dan Zona Ketersediaan. AWS Wilayah menyediakan beberapa Availability Zone yang terpisah secara fisik dan terisolasi, yang terhubung dengan latensi rendah, throughput tinggi, dan jaringan yang sangat redundan. Dengan Zona Ketersediaan, Anda dapat merancang dan mengoperasikan aplikasi dan basis data yang melakukan secara otomatis pinda saat gagal/failover di antara zona-zona tanpa terputus. Zona Ketersediaan lebih sangat tersedia, lebih toleran kesalahan, dan lebih dapat diskalakan daripadai infrastruktur pusat data tunggal atau multi tradisional.

Untuk informasi selengkapnya tentang AWS Wilayah dan Availability Zone, lihat InfrastrukturAWS Global.

Selain infrastruktur AWS global, Kinesis Data Streams menawarkan beberapa fitur untuk membantu mendukung ketahanan data dan kebutuhan pencadangan Anda.

Pemulihan Bencana di Amazon Kinesis Data Streams

Kegagalan dapat terjadi pada level berikut saat Anda menggunakan aplikasi Amazon Kinesis Data Streams untuk memproses data dari aliran:

- Prosesor rekaman bisa gagal
- Seorang pekerja bisa gagal, atau instance aplikasi yang membuat instance pekerja bisa gagal
- Instans EC2 yang menghosting satu atau lebih instance aplikasi bisa gagal

Rekam Kegagalan Prosesor

Pekerja memanggil metode prosesor rekaman menggunakan ExecutorServicetugas Java. Jika tugas gagal, pekerja mempertahankan kendali pecahan yang diproses oleh prosesor rekaman. Pekerja

memulai tugas prosesor rekaman baru untuk memproses pecahan itu. Untuk informasi selengkapnya, lihat Baca Throttling.

Kegagalan Pekerja atau Aplikasi

Jika pekerja—atau instance aplikasi Amazon Kinesis Data Streams' gagal, Anda harus mendeteksi dan menangani situasi tersebut. Misalnya, jika `Worker.run` metode melempar pengecualian, Anda harus menangkap dan menanganinya.

Jika aplikasi itu sendiri gagal, Anda harus mendeteksi ini dan memulai ulang. Ketika aplikasi dijalankan, itu membuat instance pekerja baru, yang pada gilirannya membuat instance prosesor rekaman baru yang secara otomatis diberi pecahan untuk diproses. Ini bisa berupa pecahan yang sama yang diproses oleh prosesor rekaman ini sebelum kegagalan, atau pecahan yang baru bagi prosesor ini.

Dalam situasi di mana pekerja atau aplikasi gagal, kegagalan tidak terdeteksi, dan ada contoh lain dari aplikasi yang berjalan pada instans EC2 lainnya, pekerja pada instance lain ini menangani kegagalan. Mereka membuat prosesor rekaman tambahan untuk memproses pecahan yang tidak lagi diproses oleh pekerja yang gagal. Beban pada instans EC2 lainnya ini meningkat.

Skenario yang dijelaskan di sini mengasumsikan bahwa meskipun pekerja atau aplikasi telah gagal, instans EC2 hosting masih berjalan dan oleh karena itu tidak dimulai ulang oleh grup Auto Scaling.

Kegagalan Instans Amazon EC2

Kami menyarankan Anda menjalankan instans EC2 untuk aplikasi Anda dalam grup Auto Scaling. Dengan cara ini, jika salah satu instans EC2 gagal, grup Auto Scaling secara otomatis meluncurkan instance baru untuk menggantikannya. Anda harus mengonfigurasi instance untuk meluncurkan aplikasi Amazon Kinesis Data Streams saat startup.

Keamanan Infrastruktur di Kinesis Data Streams

Sebagai layanan terkelola, Amazon Kinesis Data Streams dilindungi AWS oleh prosedur keamanan jaringan global yang dijelaskan dalam whitepaper Amazon Web Services: Tinjauan Proses Keamanan.

Anda menggunakan panggilan API yang AWS dipublikasikan untuk mengakses Kinesis Data Streams melalui jaringan. Klien harus mendukung Keamanan Lapisan Pengangkutan (TLS) 1.2 atau versi yang lebih baru. Klien juga harus mendukung cipher suite dengan perfect forward secrecy (PFS)

seperti Ephemeral Diffie-Hellman (DHE) atau Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). Sebagian besar sistem modern seperti Java 7 dan sistem yang lebih baru mendukung mode ini.

Selain itu, permintaan harus ditandatangani dengan menggunakan ID kunci akses dan kunci akses rahasia yang terkait dengan pengguna utama IAM. Atau Anda dapat menggunakan AWS Security Token Service (AWS STS) untuk menghasilkan kredensial keamanan sementara untuk menandatangani permintaan.

Praktik Terbaik Keamanan untuk Kinesis Data Streams

Amazon Kinesis Data Streams menyediakan sejumlah fitur keamanan yang perlu dipertimbangkan saat Anda mengembangkan dan menerapkan kebijakan keamanan Anda sendiri. Praktik terbaik berikut adalah pedoman umum dan tidak mewakili solusi keamanan yang lengkap. Karena praktik terbaik ini mungkin tidak sesuai atau tidak memadai untuk lingkungan Anda, perlakukan itu sebagai pertimbangan yang bermanfaat, bukan sebagai resep.

Terapkan akses hak akses paling rendah

Saat memberikan izin, Anda memutuskan siapa yang mendapatkan izin apa untuk sumber daya Kinesis Data Streams mana. Anda memungkinkan tindakan tertentu yang ingin Anda lakukan di sumber daya tersebut. Oleh karena itu, Anda harus memberikan hanya izin yang diperlukan untuk melaksanakan tugas. Menerapkan akses hak istimewa yang terkecil adalah hal mendasar dalam mengurangi risiko keamanan dan dampak yang dapat diakibatkan oleh kesalahan atau niat jahat.

Gunakan IAM role

Aplikasi produser dan klien harus memiliki kredensil yang valid untuk mengakses aliran data Kinesis. Anda tidak boleh menyimpan AWS kredensyal secara langsung di aplikasi klien atau di bucket Amazon S3. Ini adalah kredensial jangka panjang yang tidak dirotasi secara otomatis dan dapat menimbulkan dampak bisnis yang signifikan jika dibobol.

Sebagai gantinya, Anda harus menggunakan peran IAM untuk mengelola kredensi sementara untuk aplikasi produsen dan klien Anda untuk mengakses aliran data Kinesis. Saat Anda menggunakan peran, Anda tidak perlu menggunakan kredensial jangka panjang (seperti nama pengguna dan kata sandi atau access key) untuk mengakses sumber daya lainnya.

Untuk informasi selengkapnya, lihat topik berikut di Panduan Pengguna IAM:

- Peran IAM

- Skenario Umum untuk Peran: Pengguna, Aplikasi, dan Layanan

Terapkan Enkripsi Sisi Server di Sumber Daya Dependen

Data saat istirahat dan data dalam perjalanan dapat dienkripsi dalam Kinesis Data Streams. Untuk informasi selengkapnya, lihat Perlindungan Data di Amazon Kinesis Data Streams.

Gunakan CloudTrail untuk Memantau Panggilan API

Kinesis Data Streams AWS CloudTrailterintegrasi dengan, layanan yang menyediakan catatan tindakan yang diambil oleh pengguna, peran, AWS atau layanan di Kinesis Data Streams.

Dengan menggunakan informasi yang dikumpulkan oleh CloudTrail, Anda dapat menentukan permintaan yang dibuat untuk Kinesis Data Streams, alamat IP dari mana permintaan dibuat, siapa yang membuat permintaan, kapan dibuat, dan detail tambahan.

Lihat informasi yang lebih lengkap di the section called “Mencatat Panggilan API Amazon Kinesis Data Streams dengan AWS CloudTrail”.

Riwayat Dokumen

Tabel berikut menjelaskan perubahan penting pada dokumentasi Amazon Kinesis Data Streams.

Perubahan	Deskripsi	Tanggal yang Diubah
Menambahkan dukungan untuk berbagi aliran data di seluruh akun.	Ditambahkan Berbagi aliran data Anda dengan akun lain.	November 22, 2023
Menambahkan dukungan untuk mode kapasitas aliran data sesuai permintaan dan disediakan.	Ditambahkan Memilih Mode Kapasitas Aliran Data.	November 29, 2021
Konten baru untuk enkripsi sisi server.	Ditambahkan Perlindungan Data di Amazon Kinesis Data Streams.	7 Juli 2017
Konten baru untuk CloudWatch metrik yang disempurn akan.	DiperbaruiMemantau Aliran Data Amazon Kinesis.	19 April 2016
Konten baru untuk agen Kinesis yang disempurnakan.	DiperbaruiMenulis ke Amazon Kinesis Data Streams Menggunakan Agen Kinesis.	11 April 2016
Konten baru untuk menggunakan agen Kinesis.	Menambahkan Menulis ke Amazon Kinesis Data Streams Menggunakan Agen Kinesis.	2 Oktober 2015

Perubahan	Deskripsi	Tanggal yang Diubah
Perbarui konten KPL untuk rilis 0.10.0.	Ditambahkan Mengembangkan Produsen Menggunakan Perpustakaan Produsen Amazon Kinesis.	15 Juli 2015
Perbarui topik metrik KCL untuk metrik yang dapat dikonfigurasi.	Ditambahkan Memantau Perpustakaan Klien Kinesis dengan Amazon CloudWatch.	9 Juli 2015
Konten yang diatur ulang.	Topik konten yang diatur ulang secara signifikan untuk tampilan pohon yang lebih ringkas dan pengelompokan yang lebih logis.	01 Juli 2015
Topik panduan pengembang KPL baru.	Ditambahkan Mengembangkan Produsen Menggunakan Perpustakaan Produsen Amazon Kinesis.	02 Juni 2015
Topik metrik KCL baru.	Ditambahkan Memantau Perpustakaan Klien Kinesis dengan Amazon CloudWatch.	19 Mei 2015
Support untuk KCL.NET	Ditambahkan Mengembangkan Konsumen Perpustakaan Klien Kinesis di .NET.	1 Mei 2015
Support untuk KCL Node.js	Ditambahkan Mengembangkan Konsumen Perpustakaan Klien Kinesis di Node.js.	26 Maret 2015
Support untuk KCL Ruby	Menambahkan tautan ke perpustakaan KCL Ruby.	12 Januari 2015
API baru PutRecords	Menambahkan informasi tentang PutRecords API baru kethe section called “Menambahkan Beberapa Rekaman denganPutRecords”.	15 Desember 2014
Dukungan untuk penandaan	Ditambahkan Menandai Aliran di Amazon Kinesis Data Streams.	11 September 2014

Perubahan	Deskripsi	Tanggal yang Diubah
CloudWatch Metrik baru	Menambahkan metrik `GetRecords.IteratorAgeMilliseconds` keDimensi dan Metrik Aliran Data Kinesis Amazon.	3 September 2014
Bab pemantauan baru	Ditambahkan Memantau Aliran Data Amazon Kinesis danMemantau Layanan Amazon Kinesis Data Streams dengan Amazon CloudWatch.	Juli 30, 2014
Batas pecahan default	MemperbaruiKuota dan Batas: batas pecahan default telah dinaikkan dari 5 menjadi 10.	Februari 25, 2014
Batas pecahan default	MemperbaruiKuota dan Batas: batas pecahan default telah dinaikkan dari 2 menjadi 5.	Januari 28, 2014
Pembaruan versi API	Pembaruan untuk versi 2013-12-02 dari Kinesis Data Streams API.	12 Desember 2013
Rilis pertama	Rilis awal Panduan Pengembang Amazon Kinesis.	14 November 2013

AWSGlosarium

Untuk AWS terminologi terbaru, lihat AWSglosarium di Referensi. Glosarium AWS

Terjemahan disediakan oleh mesin penerjemah. Jika konten terjemahan yang diberikan bertentangan dengan versi bahasa Inggris aslinya, utamakan versi bahasa Inggris.