aws

Panduan Developerr

Amazon Kinesis Data Streams

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon Kinesis Data Streams: Panduan Developerr

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Merek dagang dan tampilan dagang Amazon tidak boleh digunakan sehubungan dengan produk atau layanan apa pun yang bukan milik Amazon, dengan cara apa pun yang dapat menyebabkan kebingungan di antara pelanggan, atau dengan cara apa pun yang merendahkan atau mendiskreditkan Amazon. Semua merek dagang lain yang tidak dimiliki oleh Amazon merupakan hak milik masing-masing pemiliknya, yang mungkin atau mungkin tidak terafiliasi, terkait dengan, atau disponsori oleh Amazon.

Table of Contents

Apa itu Amazon Kinesis Data Streams? 1
Apa yang dapat saya lakukan dengan Kinesis Data Streams? 1
Manfaat menggunakan Kinesis Data Streams 2
Layanan terkait 3
Terminologi dan konsep 4
Tinjau arsitektur tingkat tinggi Kinesis Data Streams 4
Menjadi akrab dengan terminologi Kinesis Data Streams 5
Aliran Data Kinesis 5
Rekam Data 5
Mode Kapasitas 5
Periode Retensi 5
Produser 6
Konsumen 6
Aplikasi Amazon Kinesis Data Streams 6
Shard 6
Kunci partisi 7
Nomor Urutan 7
Perpustakaan Klien Kinesis 7
Nama Aplikasi 8
Enkripsi Sisi-Server 8
Kuota dan batas 9
Batas API 11
Batas API Pesawat Kontrol KDS 12
Batas API Pesawat Data KDS 17
Meningkatkan Kuota 20
Prasyarat lengkap untuk menyiapkan Amazon Kinesis Data Streams 21
Mendaftar untuk AWS 21
Unduh perpustakaan dan alat 21
Konfigurasikan lingkungan pengembangan Anda 22
Gunakan AWS CLI untuk menjalankan operasi Amazon Kinesis Data Streams 23
Tutorial: Instal dan konfigurasikan AWS CLI untuk Kinesis Data Streams 23
Instal AWS CLI 23
Konfigurasikan AWS CLI 25
Tutorial: Lakukan operasi Kinesis Data Streams dasar menggunakan AWS CLI 25

Langkah 1: Buat aliran 25
Langkah 2: Letakkan catatan 27
Langkah 3: Dapatkan catatan 28
Langkah 4: Membersihkan 31
Memulai tutorial 33
Tutorial: Memproses data stok real-time menggunakan KPL dan KCL 2.x 33
Prasyarat lengkap 34
Buat aliran data 35
Buat kebijakan dan pengguna IAM 36
Unduh dan buat kodenya 41
Implementasikan produsen 42
Implementasikan konsumen 47
(Opsional) Memperluas konsumen 51
Pembersihan sumber daya 53
Tutorial: Memproses data stok real-time menggunakan KPL dan KCL 1.x 54
Prasyarat lengkap 55
Buat aliran data 56
Buat kebijakan dan pengguna IAM 58
Unduh dan buat kode implementasi 63
Implementasikan produsen 64
Implementasikan konsumen 69
(Opsional) Memperluas konsumen 73
Pembersihan sumber daya 74
Tutorial: Menganalisis data stok real-time menggunakan Amazon Managed Service untuk Apache Flink 76
Prasyarat 77
Langkah 1: Menyiapkan akun 77
Langkah 2: Siapkan AWS CLI 81
Langkah 3: Buat aplikasi 82
Tutorial: Gunakan AWS Lambda dengan Amazon Kinesis Data Streams 99
Gunakan Solusi Data AWS Streaming untuk Amazon Kinesis 100
Membuat dan mengelola aliran data Kinesis 101
Pilih mode yang tepat untuk streaming 102
Apa saja mode yang berbeda dalam Kinesis Data Streams? 102
Fitur mode Standar sesuai permintaan dan kasus penggunaan 103
Fitur mode Advantage On-Demand dan kasus penggunaan 105

Fitur mode yang disediakan dan kasus penggunaan 106
Beralih antar mode 107
Buat aliran menggunakan Konsol Manajemen AWS 107
Buat aliran menggunakan APIs 108
Membangun klien Kinesis Data Streams 108
Buat aliran 109
Perbarui aliran 110
Gunakan konsol 111
Gunakan API 112
Gunakan AWS CLI 112
Daftar aliran 112
Daftar pecahan 114
Hapus aliran 117
Reshard aliran 117
Tentukan strategi untuk resharding 118
Pisahkan pecahan 119
Gabungkan dua pecahan 121
Selesaikan tindakan resharding 122
Ubah periode retensi data 125
Memberikan tag pada sumber daya Anda 126
Tinjau dasar-dasar tag 127
Lacak biaya menggunakan penandaan 128
Memahami batasan tag 128
Tag stream menggunakan konsol Kinesis Data Streams 129
Tag stream menggunakan AWS CLI 131
Tag stream menggunakan Kinesis Data Streams APIs 131
Tag konsumen menggunakan AWS CLI 132
Tag konsumen menggunakan Kinesis Data Streams APIs 132
Menangani catatan besar 133
Perbarui aliran Anda untuk menggunakan catatan besar 133
Optimalkan performa streaming Anda dengan catatan besar 134
Mengurangi pelambatan dengan catatan besar 134
Menangani rekaman besar menggunakan Kinesis Data Streams APIs 134
AWS komponen yang kompatibel dengan catatan besar 135
Wilayah di mana catatan besar didukung 139
Lakukan pengujian ketahanan dengan AWS Fault Injection Service 141

Kesalahan pengecualian throughput yang disediakan 142
Kesalahan pengecualian iterator kedaluwarsa 145
Menulis data ke Kinesis Data Streams 147
Kembangkan produsen menggunakan Amazon Kinesis Producer Library (KPL) 148
Tinjau peran KPL 149
Sadarilah keuntungan menggunakan KPL 149
Memahami kapan tidak menggunakan KPL 150
Instal KPL 151
Migrasi ke KPL 1.x 152
Transisi ke sertifikat Amazon Trust Services (ATS) untuk KPL 156
Platform yang didukung KPL 156
Konsep kunci KPL 157
Integrasikan KPL dengan kode produsen 159
Menulis ke aliran data Kinesis Anda menggunakan KPL 162
Konfigurasikan KPL 164
Menerapkan de-agregasi konsumen 165
Gunakan KPL dengan Amazon Data Firehose 168
Gunakan KPL dengan AWS Glue Schema Registry 168
Konfigurasikan konfigurasi proxy KPL 169
Kebijakan siklus hidup versi KPL 170
Mengembangkan produsen menggunakan Kinesis Data Streams API dengan AWS SDK untuk Java 171
Tambahkan data ke aliran 171
Berinteraksi dengan data menggunakan AWS Glue Schema Registry 178
Menulis ke Amazon Kinesis Data Streams menggunakan Agen Kinesis 178
Lengkapi prasyarat untuk Agen Kinesis 179
Unduh dan instal agen 180
Konfigurasikan dan mulai agen 181
Tentukan pengaturan konfigurasi agen 182
Pantau beberapa direktori file dan tulis ke beberapa aliran 185
Gunakan agen untuk melakukan pra-proses data 186
Gunakan perintah CLI agen 190
Pertanyaan yang Sering Diajukan 191
Menulis ke Kinesis Data AWS Streams menggunakan layanan lain 192
Menulis ke Kinesis Data Streams menggunakan AWS Amplify 193
Menulis ke Kinesis Data Streams menggunakan Amazon Aurora 193

Menulis ke Kinesis Data Streams menggunakan Amazon CloudFront 194
Menulis ke Kinesis Data Streams CloudWatch menggunakan Amazon Logs 194
Menulis ke Kinesis Data Streams menggunakan Amazon Connect 194
Menulis ke Kinesis Data Streams menggunakan AWS Database Migration Service 195
Menulis ke Kinesis Data Streams menggunakan Amazon DynamoDB 195
Menulis ke Kinesis Data Streams menggunakan Amazon EventBridge 195
Menulis ke Kinesis Data Streams menggunakan AWS IoT Core 195
Menulis ke Kinesis Data Streams menggunakan Amazon Relational Database Service 196
Menulis ke Kinesis Data Streams menggunakanAmazon Pinpoint 196
Menulis ke Kinesis Data Streams menggunakan Amazon Quantum Ledger Database (Amazon QLDB) 196
Menulis ke Kinesis Data Streams menggunakan integrasi pihak ketiga 197
Apache Flink 197
Lancar 197
Debezium 197
Oracle GoldenGate 198
Kafka Connect 198
Pengalaman Adobe 198
Striim 198
Memecahkan Masalah Produsen Kinesis Data Streams 198
Aplikasi produser saya menulis pada tingkat yang lebih lambat dari yang diharapkan 199
Saya menerima kesalahan izin kunci master KMS yang tidak sah 201
Memecahkan masalah umum lainnya untuk produsen 201
Optimalkan produsen Kinesis Data Streams 201
Kustomisasi percobaan ulang KPL dan perilaku batas nilai 202
Menerapkan praktik terbaik untuk agregasi KPL 203
Baca data dari Kinesis Data Streams 204
Kembangkan konsumen fan-out yang ditingkatkan dengan throughput khusus 205
Perbedaan antara konsumen throughput bersama dan konsumen fan-out yang ditingkatkan 207
Wilayah yang didukung hingga 50 konsumen fan-out yang ditingkatkan (On-Demand Advantage only) 139
Kelola konsumen fan-out yang ditingkatkan 210
Gunakan Penampil Data di konsol Kinesis 211
Kueri aliran data Anda di konsol Kinesis 212
Gunakan Perpustakaan Klien Kinesis 213

Apa itu Perpustakaan Klien Kinesis? ... 213
Fitur dan manfaat utama KCL ... 214
Konsep KCL ... 215
Tabel metadata DynamoDB dan load balancing di KCL ... 216
Kembangkan konsumen dengan KCL ... 220
Pemrosesan multi-aliran dengan KCL ... 231
Gunakan registri AWS Glue Skema dengan KCL ... 234
Izin IAM diperlukan untuk aplikasi konsumen KCL ... 235
Konfigurasi KCL ... 242
Kebijakan siklus hidup versi KCL ... 259
Migrasi dari versi KCL sebelumnya ... 260
Dokumentasi versi KCL sebelumnya ... 276
Mengembangkan konsumen dengan AWS SDK untuk Java ... 360
Mengembangkan konsumen throughput bersama dengan AWS SDK untuk Java ... 360
Mengembangkan konsumen fan-out yang ditingkatkan dengan AWS SDK untuk Java ... 366
Berinteraksi dengan data menggunakan AWS Glue Schema Registry ... 369
Mengembangkan konsumen menggunakan AWS Lambda ... 369
Kembangkan konsumen menggunakan Managed Service untuk Apache Flink ... 370
Kembangkan konsumen menggunakan Amazon Data Firehose ... 370
Menggunakan AWS layanan lain untuk membaca data dari Kinesis Data Streams ... 371
Baca data dari Kinesis Data Streams menggunakan Amazon EMR ... 371
Baca data dari Kinesis Data Streams EventBridge menggunakan Amazon Pipes ... 371
Membaca data dari Kinesis Data Streams menggunakan AWS Glue ... 372
Baca data dari Kinesis Data Streams menggunakan Amazon Redshift ... 372
Baca dari Kinesis Data Streams menggunakan integrasi pihak ketiga ... 372
Apache Flink ... 373
Platform Pengalaman Adobe ... 373
Apache Druid ... 373
Apache Spark ... 373
Databricks ... 374
Platform Konfluen Kafka ... 374
Kinesumer ... 374
Talend ... 374
Memecahkan Masalah Kinesis Data Streams konsumen ... 374
Kesalahan kompilasi dengan LeaseManagementConfig konstruktor ... 375

Beberapa catatan Kinesis Data Streams dilewati saat menggunakan Perpustakaan Klien Kinesis ... 376
Catatan milik pecahan yang sama diproses oleh prosesor rekaman yang berbeda pada saat yang sama ... 376
Aplikasi konsumen membaca pada tingkat yang lebih lambat dari yang diharapkan ... 377
GetRecords mengembalikan array catatan kosong bahkan ketika ada data dalam aliran ... 378
Iterator shard kedaluwarsa secara tak terduga ... 379
Pemrosesan catatan konsumen tertinggal ... 379
Kesalahan izin kunci KMS yang tidak sah ... 380
DynamoDbException: Jalur dokumen yang disediakan dalam ekspresi pembaruan tidak valid untuk pembaruan ... 380
Memecahkan masalah umum lainnya bagi konsumen ... 381
Optimalkan Kinesis Data Streams konsumen ... 381
Tingkatkan pemrosesan latensi rendah ... 382
Memproses data serial menggunakan Perpustakaan AWS Lambda Produser Amazon Kinesis ... 383
Gunakan resharding, scaling, dan parallel processing untuk mengubah jumlah pecahan ... 383
Menangani catatan duplikat ... 385
Menangani startup, shutdown, dan throttling ... 387
Pantau Kinesis Data Streams ... 390
Memantau layanan Kinesis Data Streams dengan CloudWatch ... 390
Dimensi dan metrik Amazon Kinesis Data Streams ... 391
Akses CloudWatch metrik Amazon untuk Kinesis Data Streams ... 408
Pantau kesehatan Agen Aliran Data Kinesis dengan CloudWatch ... 409
Monitor dengan CloudWatch ... 409
Log panggilan API Amazon Kinesis Data Streams dengan AWS CloudTrail ... 410
Informasi Kinesis Data Streams di CloudTrail ... 410
Contoh: Entri file log Kinesis Data Streams ... 412
Pantau KCL dengan CloudWatch ... 416
Metrik dan namespace ... 416
Tingkat dan dimensi metrik ... 416
Konfigurasi metrik ... 417
Daftar metrik ... 418
Pantau KPL dengan CloudWatch ... 435
Metrik, dimensi, dan ruang nama ... 436
Tingkat metrik dan granularitas ... 436

Akses lokal dan CloudWatch unggahan Amazon 437
Daftar metrik 438
Keamanan 442
Perlindungan data di Kinesis Data Streams 443
Apa itu enkripsi sisi server untuk Kinesis Data Streams? 443
Biaya, Wilayah, dan pertimbangan kinerja 445
Bagaimana cara memulai dengan enkripsi sisi server? 446
Membuat dan menggunakan kunci KMS buatan pengguna 447
Izin untuk menggunakan kunci KMS buatan pengguna 447
Verifikasi dan Memecahkan Masalah izin kunci KMS 450
Gunakan Kinesis Data Streams dengan titik akhir VPC antarmuka 450
Mengontrol akses ke sumber daya Kinesis Data Streams menggunakan IAM 453
Sintaksis kebijakan 455
Tindakan untuk Kinesis Data Streams 456
Nama Sumber Daya Amazon (ARNs) untuk Kinesis Data Streams 456
Contoh kebijakan untuk Kinesis Data Streams 456
Bagikan aliran data Anda dengan akun lain 459
Konfigurasikan AWS Lambda fungsi untuk membaca dari Kinesis Data Streams di akun lain 465
Berbagi akses menggunakan kebijakan berbasis sumber daya 466
Validasi kepatuhan untuk Kinesis Data Streams 468
Ketahanan dalam Kinesis Data Streams 469
Pemulihan bencana di Kinesis Data Streams 469
Keamanan infrastruktur di Kinesis Data Streams 470
Praktik keamanan terbaik untuk Kinesis Data Streams 471
Terapkan akses hak akses paling rendah 471
Gunakan IAM role 471
Menerapkan enkripsi sisi server dalam sumber daya dependen 472
Gunakan CloudTrail untuk memantau panggilan API 472
Bekerja dengan AWS SDKs 473
Contoh kode 475
Hal-hal mendasar 476
Pelajari dasar-dasarnya 476
Tindakan 480
Contoh nirserver 538
Memanggil fungsi Lambda dari pemicu Kinesis 538

Melaporkan kegagalan item batch untuk fungsi Lambda dengan pemicu Kinesis 549
Riwayat dokumen 563
..... dlxvi

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 Kinesis Client Library, dan mereka 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 saya lakukan dengan Kinesis Data Streams?
- Manfaat menggunakan Kinesis Data Streams
- Layanan terkait

Apa yang dapat saya lakukan 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 (DAGs) 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 Amazon Kinesis Data Streams

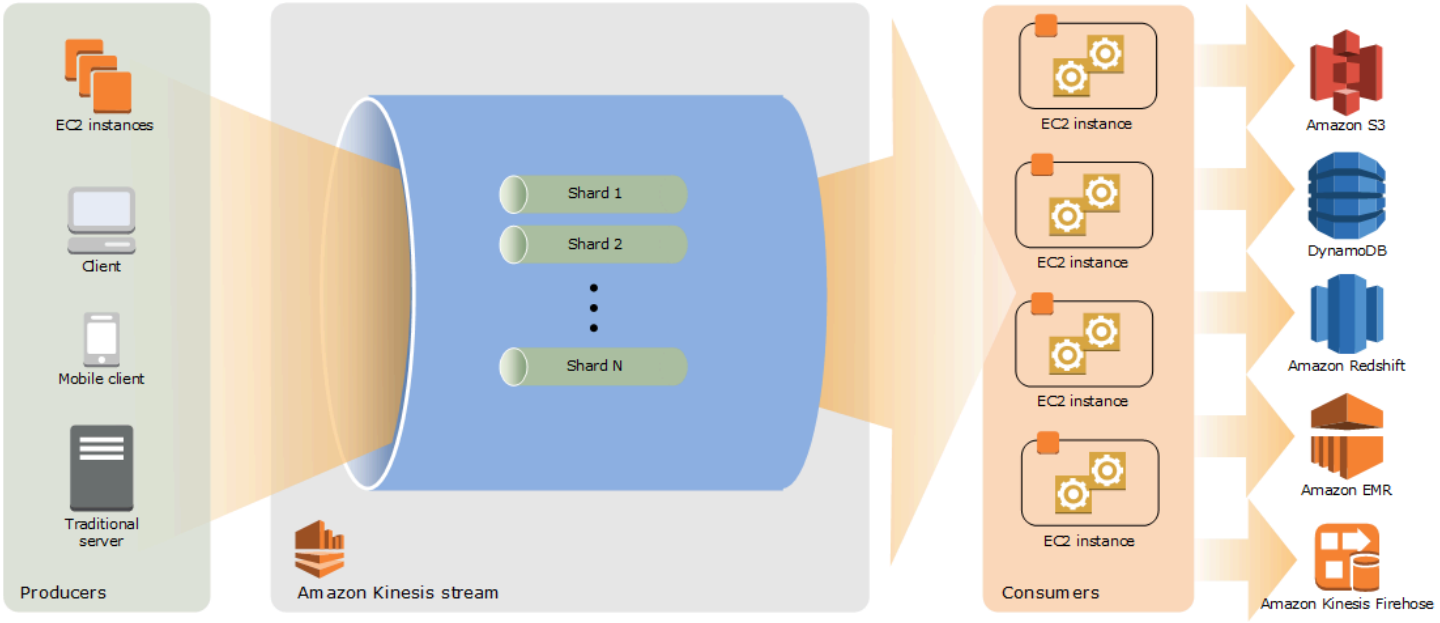
Sebelum Anda memulai Amazon Kinesis Data Streams, pelajari tentang arsitektur dan terminologinya.

Topik

- Tinjau arsitektur tingkat tinggi Kinesis Data Streams
- Menjadi akrab dengan terminologi Kinesis Data Streams

Tinjau 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.



Menjadi akrab dengan terminologi Kinesis Data Streams

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 Pilih mode yang tepat untuk streaming.

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 Fitur mode Standar sesuai permintaan dan kasus penggunaan.

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 Fitur mode yang disediakan dan kasus penggunaan.

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, lihatBaca 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 Reshard 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 MD5 hash 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 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 seiring 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

Kinesis Client Library 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 metadata yang terkait dengan konsumsi data. Ini menciptakan tiga tabel per aplikasi yang memproses data. Untuk informasi selengkapnya, lihat Gunakan Perpustakaan Klien Kinesis.

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 KMS buatan pengguna”

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

Kuota dan batas

Tabel berikut menjelaskan kuota dan batas aliran dan pecahan untuk Amazon Kinesis Data Streams.

Kuota	Mode sesuai permintaan	Mode yang disediakan
Jumlah aliran data	Tidak ada kuota atas jumlah stream dalam akun Anda AWS . Secara default, Anda dapat membuat hingga 50 aliran data dengan mode kapasitas sesuai permintaa n. Jika Anda memerlukan peningkatan kuota ini, naikkan tiket dukungan.	Tidak ada kuota atas jumlah stream dengan mode yang disediakan dalam akun.
Jumlah pecahan	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 shard default adalah 20.000 pecahan per Akun AWS untuk hal berikut: Wilayah AWS • AS Timur (Virginia N.) • AS Barat (Oregon) • Eropa (Irlandia) Untuk semua Wilayah lainnya, kuota pecahan default adalah 1.000 atau 6.000 pecahan per. Akun AWS Anda dapat melihat kuota dan pemanfaat an shard akun Anda melalui konsol Service Quotas di.

Kuota	Mode sesuai permintaan	Mode yang disediakan
		https://console.aws.amazon.com/servicequotas/ Untuk meminta peningkatan kuota pecahan, gunakan konsol Service Quotas atau. AWS CLI Untuk informasi selengkapnya, lihat Meminta peningkatan kuota.
Throughput aliran data	Secara default, aliran data baru yang dibuat dengan mode kapasitas sesuai permintaan memiliki 4 MB/s penulisan dan 8 throughput MB/s baca. Di AS Timur (Virginia N.), AS Barat (Oregon), dan Eropa (Irlandia) Wilayah AWS, aliran data dengan mode kapasitas sesuai permintaan menskalakan hingga 10 throughput tulis dan 20 GB/s baca. GB/s Untuk Wilayah lain, aliran data dengan mode kapasitas sesuai permintaan menskalakan hingga 200 throughput MB/s tulis dan 400 MB/s baca. Jika Anda memerlukan peningkatan hingga 10 kapasitas GB/s tulis dan 20 GB/s baca untuk Wilayah ini, kirimkan tiket dukungan.	Tidak ada batas atas. Throughput maksimum tergantung pada jumlah pecahan yang disediakan untuk aliran. Setiap pecahan dapat mendukung hingga 1 MB/sec atau 1.000 throughput records/sec tulis atau hingga 2 MB/sec atau 2.000 throughput records/sec baca. Jika Anda membutuhkan lebih banyak kapasitas konsumsi, Anda dapat dengan mudah meningkatkan jumlah pecahan dalam aliran menggunakan Konsol Manajemen AWS atau API. UpdateShardCount

Kuota	Mode sesuai permintaan	Mode yang disediakan
Ukuran payload data	Ukuran maksimum payload data rekaman sebelumnya `base64-encoding` adalah hingga 10 MiB. Kinesis dirancang untuk menangani rekaman besar intermiten (berukuran 1-10MiB) menggunakan kapasitas burst.	
`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 shard 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	Dengan mode Kinesis On-Demand Advantage, Anda dapat membuat hingga 50 konsumen terdaftar (Enhanced Fan-out). Dengan Kinesis On-Demand Standard dan mode Kinesis Provisioned, 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 APIs, operasi API Kinesis Data Streams dibatasi kecepatan. Batas berikut berlaku per AWS akun per Wilayah. Untuk informasi selengkapnya tentang Kinesis APIs Data Streams, lihat Referensi API Amazon Kinesis.

Batas API Pesawat Kontrol KDS

Bagian berikut menjelaskan batas untuk bidang APIs kontrol KDS. Bidang kontrol KDS APIs 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 Akun	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 yang diizinkan untuk akun Anda.

API	Batas panggilan API	Per Akun/Aliran	Deskripsi
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	
DescribeAccountSettings	5 TPS	Per akun	
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	
GetResourcePolicy	5 TPS	Per Akun	Jika Anda memerlukan peningkatan batas ini, naikkan tiket Support.

API	Batas panggilan API	Per Akun/Aliran	Deskripsi
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.
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 selengkapnya, 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.

API	Batas panggilan API	Per Akun/Aliran	Deskripsi
UpdateStr eamWarmThroughput	5 TPS	Per akun	Throughput hangat maksimum yang dapat dikonfigu rasi adalah batas throughput aliran data mode on-demand untuk akun dan Wilayah.
UpdateAccountSetti ngs	5 TPS	Per akun	Mengaktifkan atau menonakti fkan pengaturan akun, seperti mode Advantage On-Demand.

Batas API Pesawat Data KDS

Bagian berikut menjelaskan batas untuk bidang APIs data KDS. Pesawat data KDS APIs 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 berikutny a dilakukan dalam 5 detik berikutny `aProvision edThrough putExceed`

API	Batas panggilan API	Batas muatan	Detail tambahan
			`edException` . Jika throughput yang disediakan tidak mencukupi pada aliran, panggilan berikutnya dilakukan dalam lemparan 1 detik berikutny a. `Provision edThrough putExceed edException`
GetShardIterator	5 TPS		Sebuah iterator shard kedaluwarsa 5 menit setelah dikembali kan ke pemohon. Jika GetShardIterator permintaan dibuat terlalu sering, Anda menerima Provision edThroughputExceed edException.
PutRecord	1000 TPS	Setiap pecahan dapat mendukung penulisan hingga 1.000 catatan per detik, hingga total penulisan data maksimum 10MiB per detik.	Kinesis dirancang untuk menangani rekaman besar intermiten (berukuran 1-10MiB) menggunak an kapasitas burst.

API	Batas panggilan API	Batas muatan	Detail tambahan
PutRecords		Setiap PutRecords permintaan dapat mendukung hingga 500 catatan. Setiap catatan dalam permintaan dapat sebesar 10 MiB, hingga batas 10 MiB 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.	Kinesis dirancang untuk menangani rekaman besar intermiten (berukuran 1-10MiB) menggunakan kapasitas burst.
SubscribeToShard	Anda dapat melakukan satu panggilan SubscribeToShard 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 dukungan prioritas. 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.

Prasyarat lengkap untuk menyiapkan Amazon Kinesis Data Streams

Sebelum Anda menggunakan Amazon Kinesis Data Streams untuk pertama kalinya, selesaikan tugas-tugas berikut untuk menyiapkan lingkungan Anda.

Tugas

- Mendaftar untuk AWS
- Unduh perpustakaan dan alat
- Konfigurasikan lingkungan pengembangan Anda

Mendaftar untuk AWS

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

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

Untuk mendaftar AWS akun

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

   Bagian dari prosedur pendaftaran melibatkan menerima panggilan telepon atau pesan teks dan memasukkan kode verifikasi pada keypad telepon.

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

Unduh perpustakaan dan alat

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

- Referensi API Amazon Kinesis adalah kumpulan operasi dasar yang didukung oleh Kinesis Data Streams. Untuk informasi selengkapnya tentang melakukan operasi dasar menggunakan kode Java, lihat berikut ini:
  - Kembangkan produsen menggunakan API Amazon Kinesis Data Streams dengan AWS SDK untuk Java
  - Mengembangkan konsumen dengan AWS SDK untuk Java
  - Membuat dan mengelola aliran data Kinesis
- AWS SDKs Untuk Go, Java,, .NET JavaScript, PHP, Python, dan Ruby mencakup dukungan dan sampel Kinesis Data Streams. Jika versi Anda AWS SDK untuk Java tidak menyertakan sampel untuk Kinesis Data Streams, Anda juga dapat mengunduhnya. GitHub
- Perpustakaan Klien Kinesis (KCL) menyediakan model easy-to-use pemrograman untuk memproses data. KCL dapat membantu Anda memulai dengan cepat dengan Kinesis Data Streams di Java, Node.js, .NET, Python, dan Ruby. Untuk informasi selengkapnya lihat Membaca Data dari Aliran.
- AWS Command Line InterfaceMendukung Kinesis Data Streams. AWS CLI Ini memungkinkan Anda untuk mengontrol beberapa AWS layanan dari baris perintah dan mengotomatiskannya melalui skrip.

Konfigurasikan 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 web Oracle.
- Paket Apache Commons (Kode, Klien HTTP, dan Logging)
- Prosesor Jackson JSON

Perhatikan bahwa AWS SDK untuk 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.

Gunakan AWS CLI untuk menjalankan operasi Amazon Kinesis Data Streams

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

Jika Anda baru mengenal Kinesis Data Streams, mulailah dengan menjadi akrab dengan konsep dan terminologi yang disajikan dalam. Terminologi dan konsep Amazon Kinesis Data Streams

Topik

- Tutorial: Instal dan konfigurasikan AWS CLI untuk Kinesis Data Streams
- Tutorial: Lakukan operasi Kinesis Data Streams dasar menggunakan AWS CLI

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

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

Pada bagian ini, perintah spesifik yang dibahas diberikan kata demi kata, kecuali jika nilai spesifik tentu berbeda untuk setiap proses. Juga, contohnya menggunakan wilayah AS Barat (Oregon), tetapi langkah-langkah di bagian ini berfungsi di salah satu wilayah di mana Kinesis Data Streams didukung.

Tutorial: Instal dan konfigurasikan AWS CLI untuk Kinesis Data Streams

Instal AWS CLI

Untuk langkah-langkah rinci tentang cara menginstal untuk Windows dan AWS CLI untuk Linux, OS X, dan sistem operasi Unix, lihat Menginstal AWS CLI.

Gunakan perintah berikut untuk membuat daftar opsi dan layanan yang tersedia:

```
aws help
```

Anda akan menggunakan layanan Kinesis Data Streams, sehingga Anda dapat AWS CLI meninjau subperintah yang terkait dengan Kinesis Data Streams menggunakan perintah berikut:

```
aws kinesis help
```

Perintah ini menghasilkan output yang menyertakan 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 API Kinesis Data Streams yang didokumentasikan dalam Referensi API Amazon Kinesis Service. Misalnya, `create-stream` perintah sesuai dengan tindakan `CreateStream` API.

Sekarang berhasil diinstal, tetapi tidak dikonfigurasi. AWS CLI Ini ditunjukkan di bagian selanjutnya.

Konfigurasikan AWS CLI

Untuk penggunaan umum, `aws configure` perintah adalah cara tercepat untuk mengatur AWS CLI instalasi Anda. Untuk informasi selengkapnya, lihat Mengonfigurasi AWS CLI.

Tutorial: Lakukan operasi Kinesis Data Streams dasar menggunakan AWS CLI

Bagian ini menjelaskan penggunaan dasar aliran data Kinesis dari baris perintah menggunakan file. AWS CLI Pastikan Anda terbiasa dengan konsep yang dibahas diTerminologi dan konsep Amazon Kinesis Data Streams.

> **Note**
>
> Setelah Anda membuat stream, akun Anda akan dikenakan biaya nominal untuk penggunaan Kinesis Data Streams karena Kinesis Data Streams tidak memenuhi syarat untuk Tingkat Gratis. AWS Setelah Anda selesai dengan tutorial ini, hapus AWS sumber daya Anda untuk menghentikan biaya. Untuk informasi selengkapnya, lihat Langkah 4: Membersihkan.

Topik

- Langkah 1: Buat aliran
- Langkah 2: Letakkan catatan
- Langkah 3: Dapatkan catatan
- Langkah 4: Membersihkan

Langkah 1: Buat aliran

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

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

Selanjutnya, keluarkan 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 CREATING, yang berarti belum siap 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 Anda perlukan untuk tutorial ini. Informasi penting untuk saat ini adalah`"StreamStatus": "ACTIVE"`, yang memberi tahu Anda bahwa aliran siap digunakan, dan informasi tentang 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: Letakkan catatan

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

```
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 cara mengeluarkan data dari aliran.

Langkah 3: Dapatkan catatan

GetShardIterator

Sebelum Anda bisa mendapatkan data dari stream, Anda harus mendapatkan iterator shard untuk shard yang Anda minati. Sebuah iterator shard mewakili posisi aliran dan pecahan 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 belakangnya, jadi jika Anda ingin tahu tentang salah satu parameter yang ditampilkan, Anda dapat membacanya `GetShardIterator`di topik referensi API. Eksekusi yang berhasil akan menghasilkan output yang mirip dengan contoh berikut:

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

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

GetRecords

`get-records`Perintah mendapatkan data dari aliran, dan menyelesaikan panggilan ke `GetRecords`dalam Kinesis Data Streams API. Iterator shard menentukan posisi dalam pecahan dari mana Anda ingin mulai membaca catatan data secara berurutan. Jika tidak ada catatan yang tersedia di bagian pecahan yang ditunjuk iterator, `GetRecords` mengembalikan daftar kosong. Mungkin diperlukan beberapa panggilan untuk sampai ke sebagian pecahan yang berisi catatan.

Dalam contoh `get-records` perintah berikut:

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

Jika Anda menjalankan tutorial ini dari prosesor perintah tipe Unix seperti bash, Anda dapat mengotomatiskan akuisisi iterator shard menggunakan perintah bersarang, seperti ini:

```
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 mendukung PowerShell, Anda dapat mengotomatiskan akuisisi iterator shard menggunakan perintah seperti ini:

```
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 `get-records` perintah yang berhasil akan meminta catatan dari aliran Anda untuk pecahan yang Anda tentukan saat Anda memperoleh iterator shard, seperti pada contoh berikut:

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

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

Perhatikan bahwa `get-records` dijelaskan di atas sebagai permintaan, yang berarti Anda mungkin menerima nol atau lebih catatan bahkan jika ada catatan dalam aliran Anda. Catatan apa pun yang

dikembalikan mungkin tidak mewakili semua catatan yang saat ini ada di aliran Anda. Ini normal, dan kode produksi akan melakukan polling aliran untuk catatan pada interval yang sesuai. Kecepatan polling ini akan bervariasi tergantung pada persyaratan desain aplikasi spesifik Anda.

Dalam catatan Anda di bagian tutorial ini, Anda akan melihat bahwa data tampaknya sampah - dan itu bukan teks yang jelas yang `testdata` kami kirim. Ini karena cara `put-record` menggunakan pengkodean Base64 untuk memungkinkan Anda mengirim data biner. Namun, dukungan Kinesis Data Streams AWS CLI di tidak menyediakan decoding Base64 karena decoding Base64 ke konten biner mentah yang 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 memecahkan kode secara manual, `dGVzdGRhdGE=` Anda akan melihat bahwa itu sebenarnya. `testdata` Ini cukup untuk kepentingan tutorial ini karena, dalam praktiknya, jarang AWS CLI digunakan untuk mengkonsumsi data. Lebih sering, ini digunakan untuk memantau keadaan aliran dan mendapatkan informasi, seperti yang ditunjukkan sebelumnya (`describe-stream`dan`list-streams`). Untuk informasi selengkapnya tentang KCL, lihat Mengembangkan Konsumen Kustom dengan Throughput Bersama Menggunakan KCL.

`get-records`tidak selalu mengembalikan semua catatan dalam stream/shard yang ditentukan. Ketika itu terjadi, gunakan `NextShardIterator` dari hasil terakhir untuk mendapatkan set catatan berikutnya. Jika lebih banyak data dimasukkan ke dalam aliran, yang merupakan situasi normal dalam aplikasi produksi, Anda dapat terus melakukan polling untuk data yang digunakan `get-records` setiap kali. Namun, jika Anda tidak memanggil `get-records` menggunakan iterator shard berikutnya dalam masa pakai iterator shard 300 detik, Anda akan mendapatkan pesan kesalahan, dan Anda harus menggunakan `get-shard-iterator` perintah untuk mendapatkan iterator shard baru.

Juga disediakan dalam output ini adalah`MillisBehindLatest`, yang merupakan jumlah milidetik respons GetRecordsoperasi dari ujung aliran, menunjukkan seberapa jauh di belakang waktu konsumen saat ini. Nilai nol menunjukkan pemrosesan catatan tertangkap, 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. Secara default, catatan data tetap berada dalam aliran selama 24 jam menunggu Anda untuk mengambilnya. Kerangka waktu ini disebut periode retensi dan dapat dikonfigurasi hingga 365 hari.

`get-records`Hasil yang sukses akan selalu memiliki `NextShardIterator` bahkan jika tidak ada lagi catatan saat ini dalam aliran. Ini adalah model polling yang mengasumsikan produsen berpotensi menempatkan lebih banyak catatan ke dalam aliran pada waktu tertentu. Meskipun Anda dapat

menulis rutinitas polling Anda sendiri, jika Anda menggunakan KCL yang disebutkan sebelumnya untuk mengembangkan aplikasi konsumen, polling ini ditangani untuk Anda.

Jika Anda menelepon `get-records` sampai tidak ada lagi catatan dalam aliran dan pecahan yang Anda tarik, Anda akan melihat output dengan catatan kosong yang mirip dengan contoh berikut:

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

Langkah 4: Membersihkan

Hapus streaming Anda untuk mengosongkan sumber daya dan menghindari tagihan yang tidak diinginkan ke akun Anda. Lakukan ini setiap kali Anda telah membuat aliran dan tidak akan menggunakannya, karena biaya bertambah per aliran apakah Anda menempatkan dan mendapatkan data dengannya atau tidak. Perintah pembersihan adalah sebagai berikut:

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

Sukses menghasilkan tidak ada output. Gunakan `describe-stream` untuk memeriksa kemajuan penghapusan:

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

Jika Anda menjalankan perintah ini segera setelah perintah delete, Anda akan melihat 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.
```

Memulai tutorial untuk Amazon Kinesis Data Streams

Amazon Kinesis Data Streams menyediakan sejumlah solusi berbeda untuk menelan dan mengkonsumsi data dari aliran data Kinesis. Tutorial di bagian ini dirancang untuk lebih membantu Anda dalam memahami konsep dan fungsionalitas Amazon Kinesis Data Streams dan mengidentifikasi solusi yang memenuhi kebutuhan Anda.

Topik

- Tutorial: Memproses data stok real-time menggunakan KPL dan KCL 2.x
- Tutorial: Memproses data stok real-time menggunakan KPL dan KCL 1.x
- Tutorial: Menganalisis data stok real-time menggunakan Amazon Managed Service untuk Apache Flink
- Tutorial: Gunakan AWS Lambda dengan Amazon Kinesis Data Streams
- Gunakan Solusi Data AWS Streaming untuk Amazon Kinesis

Tutorial: Memproses data stok 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 dasar 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 dekat.

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

Kode tersebut tidak mengakses data pasar saham yang sebenarnya, melainkan 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 pada 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 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.

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

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

Tugas

- Prasyarat lengkap
- Buat aliran data
- Buat kebijakan dan pengguna IAM
- Unduh dan buat kodenya
- Implementasikan produsen
- Implementasikan konsumen
- (Opsional) Memperluas konsumen
- Pembersihan sumber daya

Prasyarat lengkap

Anda harus memenuhi persyaratan berikut untuk menyelesaikan tutorial ini:

Membuat dan menggunakan Akun Amazon Web Services

Sebelum Anda mulai, pastikan Anda terbiasa dengan konsep yang dibahasTerminologi dan konsep Amazon Kinesis Data Streams, terutama dengan aliran, pecahan, produsen, dan konsumen. Hal ini

juga membantu untuk menyelesaikan langkah-langkah dalam panduan berikut:Tutorial: Instal dan konfigurasikan AWS CLI untuk Kinesis Data Streams.

Anda harus memiliki AWS akun dan browser web untuk mengakses file Konsol Manajemen AWS.

Untuk akses konsol, gunakan nama pengguna dan kata sandi IAM Anda untuk masuk ke halaman masuk Konsol Manajemen AWSdari IAM. Untuk informasi tentang AWS kredensil keamanan, termasuk akses terprogram dan alternatif untuk kredensil jangka panjang, lihat kredensil AWS keamanan di Panduan Pengguna IAM. Untuk detail tentang masuk ke Anda Akun AWS, lihat Cara masuk AWS di Panduan AWS Sign-In Pengguna.

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

Memenuhi 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.

Anda membutuhkan AWS SDK untuk Javaversi terbaru.

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

Langkah selanjutnya

Buat aliran data

Buat aliran data

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

Untuk membuat aliran

1. Masuk ke Konsol Manajemen AWS dan buka konsol Kinesis di /kinesis. https://console.aws.amazon.com
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** untuk jumlah pecahan, tetapi tetap Perkirakan jumlah pecahan yang Anda perlukan untuk diciutkan.
7. Pilih Buat aliran Kinesis.

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

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

Langkah selanjutnya

Buat kebijakan dan pengguna IAM

Buat kebijakan dan pengguna IAM

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

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

Produser

Tindakan	Sumber daya	Tujuan
`DescribeStream` , `DescribeStreamSummary` , `DescribeStreamConsumer`	Aliran data Kinesis	Sebelum mencoba membaca catatan, konsumen memeriks ada, apakah aktif, dan apakah pecahan terkandung dalam
`SubscribeToShard` , `RegisterStreamConsumer`	Aliran data Kinesis	Berlangganan dan mendaftarkan konsumen ke pecahan.

Tindakan	Sumber daya	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 aktif, dan apakah pecahan terkandung dalam
`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), ia memerlukan izin ke tabel DynamoDB pemrosesan aplikasi.
`DeleteItem`	Tabel Amazon DynamoDB	Untuk saat konsumen melakukan split/merge operasi pada Kinesis.
`PutMetricData`	CloudWatch Log Amazon	KCL juga mengunggah metrik ke CloudWatch, yang bergun aplikasi.

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

Untuk membuat kebijakan IAM

1. Temukan Nama Sumber Daya Amazon (ARN) untuk aliran data baru yang Anda buat di langkah sebelumnya. Anda dapat menemukan ARN ini terdaftar sebagai Stream ARN di bagian atas tab Detail. Format ARN adalah sebagai berikut:

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

region

Kode AWS Wilayah; misalnya,`us-west-2`. Untuk informasi selengkapnya, lihat Konsep Wilayah dan Zona Ketersediaan.

akun

ID AWS akun, seperti yang ditunjukkan pada Pengaturan Akun.

name

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

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

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

Region 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 Policy Generator.
5. Pilih Amazon Kinesis sebagai layanannya. AWS
6. Pilih`DescribeStream`,`GetShardIterator`,`GetRecords`,`PutRecord`, dan `PutRecords` sebagai tindakan yang diizinkan.
7. Masukkan ARN dari aliran data yang Anda gunakan dalam tutorial ini.
8. Gunakan Tambah Pernyataan untuk masing-masing hal berikut:

AWS Layanan	Tindakan	ARN
Amazon DynamoDB	`CreateTable` , `DeleteItem` , `DescribeTable` ,	ARN dari tabel DynamoDB yang Anda

AWS Layanan	Tindakan	ARN
	`GetItem`, `PutItem`, `Scan`, `UpdateItem`	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:

JSON

```
{
    "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-2:111122223333:stream/StockTradeStream"
            ]
        },
        {
```

```
            "Sid": "Stmt234",
            "Effect": "Allow",
            "Action": [
                "kinesis:SubscribeToShard",
                "kinesis:DescribeStreamConsumer"
            ],
            "Resource": [
                "arn:aws:kinesis:us-west-2:111122223333:stream/StockTradeStream/
*"
            ]
        },
        {
            "Sid": "Stmt456",
            "Effect": "Allow",
            "Action": [
                "dynamodb:*"
            ],
            "Resource": [
                "arn:aws:dynamodb:us-west-2:111122223333: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 Access, 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 sebelumnya (. `StockTradeStreamPolicy` Pilih kotak di sebelah kiri nama kebijakan, lalu pilih Berikutnya: Tinjau.
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. Rekatkan akses dan kunci 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

Unduh dan buat kodenya

Unduh dan buat kodenya

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

Untuk mengunduh dan membangun kode

1. Unduh kode sumber dari https://github.com/aws-samples/amazon-kinesis-learning GitHub repo 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 Lang
- Pencatatan Apache Commons
- Jambu Biji (Perpustakaan Google Core Untuk Java)
- Anotasi Jackson
- Inti Jackson
- Databind Jackson
- Jackson Format Data: CBOR
- Waktu Joda

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

Jika Anda berhasil menyelesaikan langkah-langkah ini, Anda sekarang siap untuk pindah ke bagian berikutnya,the section called “Implementasikan produsen”.

Langkah selanjutnya

Implementasikan produsen

Tutorial ini menggunakan skenario dunia nyata 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 contoh StockTrade kelas. Contoh ini berisi atribut seperti simbol ticker, harga, jumlah saham, jenis perdagangan (beli atau jual), dan ID yang secara unik mengidentifikasi perdagangan. Kelas ini diterapkan untuk Anda.

Rekam aliran

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

```
{
  "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 diterapkan untuk Anda.

StockTradesWriter kelas

`main`Metode produsen, 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 input.
2. Menggunakan `KinesisAsyncClientBuilder` untuk mengatur Region, kredensial, dan konfigurasi klien.
3. Memeriksa apakah aliran ada dan aktif (jika tidak, ia keluar dengan kesalahan).
4. Dalam loop kontinu, panggil `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 harus 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 aliran, 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 ticker 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 aliran, lihatMenulis data ke 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 mencatat kondisi kesalahan:

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

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

```
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);
}
```

Ini karena operasi put Kinesis Data Streams dapat gagal karena kesalahan jaringan, atau karena aliran data mencapai batas throughputnya dan terhambat. Disarankan agar Anda

mempertimbangkan dengan cermat kebijakan coba ulang Anda untuk `put` operasi untuk menghindari kehilangan data, seperti menggunakan coba lagi.

- Pencatatan status sangat membantu tetapi opsional:

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

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

Untuk menjalankan produser

1. Verifikasi bahwa kunci akses dan key pair rahasia yang diambil Buat kebijakan dan 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
```

Perdagangan saham Anda sekarang sedang dicerna oleh Kinesis Data Streams.

Langkah selanjutnya

Implementasikan konsumen

Implementasikan konsumen

Aplikasi konsumen dalam tutorial ini terus memproses perdagangan saham dalam aliran data Anda. Ini kemudian menghasilkan saham paling populer yang dibeli dan dijual setiap menit. Aplikasi ini dibangun di atas Kinesis Client Library (KCL), yang melakukan banyak pekerjaan berat yang umum untuk aplikasi konsumen. Untuk informasi selengkapnya, lihat Informasi KCL 1.x dan 2.x.

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

StockTradeRecordProcessor kelas

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

`shutdownRequested`Metode `initialize` dan digunakan oleh KCL untuk memberi tahu prosesor rekaman kapan harus siap untuk mulai menerima catatan dan kapan harus berhenti menerima catatan, masing-masing, sehingga dapat melakukan tugas pengaturan dan penghentian khusus aplikasi apa pun. `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, kami 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 yang sebagian besar kosong untuk Anda terapkan 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 diterapkan untuk Anda, dan melakukan langkah-langkah berikut:

- Untuk setiap catatan yang masuk, 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, hubungi. `checkpoint()`
- Menetapkan waktu checkpointing berikutnya.

Metode ini menggunakan interval 60 detik untuk tingkat pelaporan dan pos pemeriksaan. Untuk informasi selengkapnya tentang checkpointing, lihat Menggunakan Perpustakaan Klien Kinesis.

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 yang sedang 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. Terapkan `processRecord` metode dengan membuat instance `StockTrade` objek berukuran benar dan menambahkan data catatan ke dalamnya, mencatat 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. Ubah format output 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` instance 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 key pair 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.

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

[(Opsional) Memperluas konsumen]

(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. 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

Pembersihan sumber daya

Pembersihan sumber daya

Karena Anda membayar untuk menggunakan aliran data Kinesis, pastikan Anda menghapusnya dan tabel Amazon DynamoDB yang sesuai ketika Anda selesai dengannya. Biaya nominal terjadi pada aliran aktif bahkan ketika Anda tidak mengirim dan mendapatkan catatan. Ini karena aliran aktif menggunakan sumber daya dengan terus “mendengarkan” 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 /kinesis. https://console.aws.amazon.com
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 dekat tidak memerlukan penulisan kode yang rumit atau mengembangkan infrastruktur yang besar. Ini sama dasarnya dengan menulis logika untuk memproses sejumlah kecil data (seperti menulis`processRecord(Record)`) tetapi menggunakan Kinesis Data Streams untuk skala sehingga berfungsi untuk sejumlah besar data streaming. Anda tidak perlu khawatir tentang bagaimana skala pemrosesan Anda 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 rekaman baru yang diterima.

Berikut adalah beberapa peningkatan potensial untuk aplikasi ini.

Agregat di semua pecahan

Saat ini, Anda mendapatkan statistik yang dihasilkan dari agregasi catatan data yang diterima oleh satu pekerja dari satu pecahan. (Pecahan tidak dapat diproses oleh lebih dari satu pekerja dalam satu aplikasi pada saat yang bersamaan.) Tentu saja, ketika Anda menskalakan dan memiliki lebih dari satu pecahan, Anda mungkin ingin menggabungkan 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 dengan mudah ditangani oleh satu pecahan.

Pemrosesan skala

Ketika aliran meningkat untuk memiliki banyak pecahan (karena banyak produsen mengirim data), cara untuk menskalakan pemrosesan adalah dengan menambahkan lebih banyak pekerja. Anda dapat menjalankan pekerja di instans Amazon EC2 dan menggunakan grup Auto Scaling.

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

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

Tutorial: Memproses data stok 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 pada aliran. Anda akan belajar cara mengirim aliran catatan ke Kinesis Data Streams dan mengimplementasikan aplikasi yang mengkonsumsi dan memproses catatan dalam waktu dekat.

Important

Setelah Anda membuat stream, akun Anda akan dikenakan biaya nominal untuk penggunaan Kinesis Data Streams karena Kinesis Data Streams tidak memenuhi syarat untuk Tingkat Gratis. AWS Setelah aplikasi konsumen dimulai, itu juga menimbulkan biaya nominal untuk penggunaan Amazon DynamoDB. Aplikasi konsumen menggunakan DynamoDB untuk melacak status pemrosesan. Setelah Anda selesai dengan aplikasi ini, hapus AWS sumber

daya Anda untuk menghentikan biaya. Untuk informasi selengkapnya, lihat Pembersihan sumber daya.

Kode tersebut tidak mengakses data pasar saham yang sebenarnya, melainkan 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 pada 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 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.

Anda dapat mengerjakan langkah-langkah dalam tutorial ini di komputer desktop atau laptop Anda dan menjalankan kode produsen 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 Barat AS (Oregon), tetapi mereka bekerja di salah satu AWS Wilayah yang mendukung Kinesis Data Streams.

Tugas

- Prasyarat lengkap
- Buat aliran data
- Buat kebijakan dan pengguna IAM
- Unduh dan buat kode implementasi
- Implementasikan produsen
- Implementasikan konsumen
- (Opsional) Memperluas konsumen
- Pembersihan sumber daya

Prasyarat lengkap

Berikut ini adalah persyaratan untuk menyelesaikanTutorial: Memproses data stok real-time menggunakan KPL dan KCL 1.x.

Membuat dan menggunakan Akun Amazon Web Services

Sebelum Anda mulai, pastikan Anda terbiasa dengan konsep yang dibahasTerminologi dan konsep Amazon Kinesis Data Streams, terutama aliran, pecahan, produsen, dan konsumen. Ini juga membantu untuk menyelesaikannyaTutorial: Instal dan konfigurasikan AWS CLI untuk Kinesis Data Streams.

Anda memerlukan AWS akun dan browser web untuk mengakses file Konsol Manajemen AWS.

Untuk akses konsol, gunakan nama pengguna dan kata sandi IAM Anda untuk masuk ke halaman masuk Konsol Manajemen AWSdari IAM. Untuk informasi tentang AWS kredensil keamanan, termasuk akses terprogram dan alternatif untuk kredensil jangka panjang, lihat kredensil AWS keamanan di Panduan Pengguna IAM. Untuk detail tentang masuk ke Anda Akun AWS, lihat Cara masuk AWS di Panduan AWS Sign-In Pengguna.

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

Memenuhi 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 membutuhkan AWS SDK untuk Javaversi terbaru. Jika Anda menggunakan Eclipse sebagai IDE Anda, Anda dapat menginstal AWS Toolkit for Eclipse sebagai gantinya.

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

Langkah Berikutnya

Buat aliran data

Buat aliran data

Pada langkah pertamaTutorial: Memproses data stok real-time menggunakan KPL dan KCL 1.x, Anda membuat aliran yang akan Anda gunakan dalam langkah selanjutnya.

Untuk membuat aliran

1. Masuk ke Konsol Manajemen AWS dan buka konsol Kinesis di /kinesis. https://console.aws.amazon.com
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 streaming Anda (misalnya,**`StockTradeStream`**).
6. Masukkan **1** untuk jumlah pecahan, tetapi tetap Perkirakan jumlah pecahan yang Anda perlukan untuk diciutkan.
7. Pilih Buat aliran Kinesis.

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

Informasi tambahan tentang pecahan

Ketika Anda mulai menggunakan Kinesis Data Streams di luar tutorial ini, Anda mungkin perlu merencanakan proses pembuatan aliran 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 pada siang hari (waktu Timur) dan perkiraan permintaan harus diambil sampelnya dari waktu itu. Anda kemudian memiliki pilihan untuk menyediakan permintaan maksimum yang diharapkan, atau meningkatkan aliran Anda ke atas dan ke bawah sebagai respons terhadap fluktuasi permintaan.

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

Catatan Max ditulis

Pertimbangkan 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 dikonsumsi

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 pecahan Anda saat ini, Anda mungkin perlu mengirimkan permintaan untuk meningkatkan batas tersebut sebelum Anda dapat membuat aliran dengan jumlah pecahan tersebut. Untuk meminta peningkatan batas pecahan, gunakan formulir Batas Aliran Data Kinesis. Untuk informasi lebih lanjut tentang aliran dan pecahan, lihat. Membuat dan mengelola aliran data Kinesis

Langkah selanjutnya

Buat kebijakan dan pengguna IAM

Buat kebijakan dan pengguna IAM

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

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

Produser

Tindakan	Sumber daya	Tujuan
`DescribeStream` , `DescribeStreamSumm`	Aliran data Kinesis	Sebelum mencoba menulis catatan, produsen memeriksa a aktif, dan apakah pecahan terkandung dalam aliran, dan ap konsumen.

Tindakan	Sumber daya	Tujuan
`ary` , `DescribeStreamConsumer`		
`SubscribeToShard` , `RegisterStreamConsumer`	Aliran data Kinesis	Berlangganan dan daftarkan konsumen ke pecahan Kinesis
`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 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 izin ke tabel DynamoDB untuk melacak status pemrosesan pertama mulai membuat tabel.
`DeleteItem`	Tabel Amazon DynamoDB	Untuk saat konsumen melakukan split/merge operasi pada Kinesis.
`PutMetricData`	CloudWatch Log Amazon	KCL juga mengunggah metrik ke CloudWatch, 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 Nama Sumber Daya Amazon (ARN) untuk aliran baru. Anda dapat menemukan ARN ini terdaftar sebagai Stream ARN di bagian atas tab Detail. Format ARN adalah sebagai berikut:

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

   region

   Kode Wilayah; misalnya,`us-west-2`. Untuk informasi selengkapnya, lihat Konsep Wilayah dan Zona Ketersediaan.

   akun

   ID AWS akun, seperti yang ditunjukkan pada Pengaturan Akun.

   name

   Nama aliran dariBuat aliran data, yaitu`StockTradeStream`.

2. Tentukan ARN untuk tabel DynamoDB yang akan digunakan oleh konsumen (dan dibuat oleh contoh 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 kali 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 Policy Generator.
5. Pilih Amazon Kinesis sebagai layanannya. AWS
6. Pilih`DescribeStream`,`GetShardIterator`,`GetRecords`,`PutRecord`, dan `PutRecords` sebagai tindakan yang diizinkan.
7. Masukkan ARN yang Anda buat di Langkah 1.
8. Gunakan Tambah Pernyataan untuk masing-masing hal berikut:

AWS Layanan	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 menjadi`StockTradeStreamPolicy`, tinjau kode, dan pilih Buat Kebijakan.

Dokumen kebijakan yang dihasilkan akan terlihat seperti berikut:

JSON

```
{
    "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-2:111122223333:stream/StockTradeStream"
            ]
        },
        {
            "Sid": "Stmt234",
            "Effect": "Allow",
            "Action": [
                "kinesis:SubscribeToShard",
                "kinesis:DescribeStreamConsumer"
            ],
            "Resource": [
                "arn:aws:kinesis:us-west-2:111122223333:stream/StockTradeStream/
*"
            ]
        },
        {
            "Sid": "Stmt456",
            "Effect": "Allow",
            "Action": [
                "dynamodb:*"
            ],
            "Resource": [
                "arn:aws:dynamodb:us-west-2:111122223333: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 Access, 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: Tinjau.
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. Rekatkan akses dan kunci 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 Berikutnya

Unduh dan buat kode implementasi

Unduh dan buat kode implementasi

Kode kerangka disediakan untuk. the section called “Tutorial: Memproses data stok 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 membangun 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 Lang
   - Pencatatan Apache Commons
   - Jambu Biji (Perpustakaan Google Core Untuk Java)
   - Anotasi Jackson
   - Inti Jackson
   - Databind Jackson
   - Jackson Format Data: CBOR
   - Waktu Joda
4. Tergantung pada IDE Anda, proyek mungkin dibangun secara otomatis. Jika tidak, bangun proyek menggunakan langkah-langkah yang sesuai untuk IDE Anda.

Jika Anda berhasil menyelesaikan langkah-langkah ini, Anda sekarang siap untuk pindah ke bagian berikutnya,the section called “Implementasikan produsen”. Jika build Anda menghasilkan kesalahan pada tahap apa pun, selidiki dan perbaiki sebelum melanjutkan.

Langkah selanjutnya

Implementasikan produsen

Aplikasi dalam Tutorial: Memproses data stok real-time menggunakan KPL dan KCL 1.x menggunakan skenario dunia nyata 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 contoh `StockTrade` kelas. Contoh ini berisi atribut seperti simbol ticker, harga, jumlah saham, jenis perdagangan (beli atau jual), dan ID yang secara unik mengidentifikasi perdagangan. Kelas ini diterapkan untuk Anda.

Rekam aliran

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

```
{
  "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 diterapkan untuk Anda.

StockTradesWriter kelas

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

1. Membaca nama aliran dan nama Wilayah sebagai masukan.
2. Menciptakan sebuah`AmazonKinesisClientBuilder`.
3. Menggunakan pembuat klien untuk mengatur Region, kredensial, dan konfigurasi klien.
4. Membangun `AmazonKinesis` klien menggunakan pembuat klien.
5. Memeriksa apakah aliran ada dan aktif (jika tidak, ia keluar dengan kesalahan).
6. Dalam loop kontinu, panggil `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 harus mengonversi `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 (dipanggil `putRecord` dalam kasus ini):

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

  Setiap `PutRecord` panggilan memerlukan nama aliran, kunci partisi, dan gumpalan data. Kode berikut mengisi bidang-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 aliran, lihatTambahkan data ke aliran.

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

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

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

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

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

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

Ini karena operasi Kinesis `put` Data Streams dapat gagal karena kesalahan jaringan, atau karena aliran mencapai batas throughputnya dan terhambat. Sebaiknya pertimbangkan dengan cermat kebijakan coba ulang Anda untuk `put` operasi untuk menghindari kehilangan data, seperti menggunakan coba lagi.

- Pencatatan status sangat membantu tetapi opsional:

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

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

Untuk menjalankan produser

1. Verifikasi bahwa kunci akses dan key pair 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

Implementasikan konsumen

Implementasikan konsumen

Aplikasi konsumen secara Tutorial: Memproses data stok real-time menggunakan KPL dan KCL 1.x terus menerus memproses aliran perdagangan saham yang Anda buat. Ini kemudian menghasilkan saham paling populer yang dibeli dan dijual setiap menit. Aplikasi ini dibangun di atas Kinesis Client Library (KCL), yang melakukan banyak pekerjaan berat yang umum untuk aplikasi konsumen. Untuk informasi selengkapnya, lihat Kembangkan 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 nama aplikasi, aliran, dan Wilayah, diteruskan sebagai argumen.
- Membaca kredensil dari. `~/.aws/credentials`
- Menciptakan sebuah `RecordProcessorFactory` instance yang melayani instance dari`RecordProcessor`, diimplementasikan oleh sebuah `StockTradeRecordProcessor` instance.
- Membuat pekerja KCL dengan `RecordProcessorFactory` instance dan konfigurasi standar termasuk nama aliran, kredensi, dan nama aplikasi.
- Pekerja membuat utas baru untuk setiap pecahan (ditetapkan ke instance konsumen ini), yang terus menerus melakukan loop untuk membaca catatan dari Kinesis Data Streams. Kemudian memanggil `RecordProcessor` instance untuk memproses setiap batch catatan yang diterima.

StockTradeRecordProcessor kelas

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

Seperti namanya, `initialize` dan `shutdown` digunakan oleh Perpustakaan Klien Kinesis 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 sebagian besar kode kerangka kosong untuk Anda terapkan 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 diterapkan untuk Anda, dan melakukan langkah-langkah berikut:

- Untuk setiap catatan yang diteruskan, panggil `processRecord` saja.
- 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, hubungi. `checkpoint()`
- Menetapkan waktu checkpointing berikutnya.

Metode ini menggunakan interval 60 detik untuk tingkat pelaporan dan pos pemeriksaan. Untuk informasi selengkapnya tentang pos pemeriksaan, 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 yang sedang 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. Terapkan `processRecord` metode dengan membuat instance `StockTrade` objek berukuran benar dan menambahkan data catatan ke dalamnya, mencatat 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, implementasikan `resetStats` metode, yang menciptakan `stockStats` instance 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 key pair 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.

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 keunggulan Perpustakaan Klien Kinesis, dibahas di dalam Kembangkan konsumen KCL 1.x dan di tempat lain, Anda mungkin bertanya-tanya mengapa Anda harus menggunakannya di sini. Meskipun Anda hanya menggunakan satu aliran pecahan dan satu contoh 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 contoh konsumen yang ada. Jika aliran ditingkatkan, Anda tidak perlu menulis ulang aplikasi Anda untuk menangani lebih dari satu pecahan atau satu instance konsumen.

Istilah checkpointing berarti mencatat titik dalam aliran hingga catatan data yang telah dikonsumsi dan diproses sejauh ini. Jika aplikasi macet, aliran dibaca dari titik itu dan bukan dari awal aliran. Subjek checkpointing dan berbagai pola desain dan praktik terbaik untuknya berada di luar cakupan pasal ini. Namun, itu adalah sesuatu yang mungkin Anda temui di lingkungan produksi.

Seperti yang Anda pelajari, `put` operasi di Kinesis Data Streams API 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 merutekan 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 mengumpulkan semua catatan dengan kunci partisi yang sama tanpa khawatir bahwa itu mungkin kehilangan data yang diperlukan.

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 untuk meningkatkan jumlah pecahan. Dalam beberapa kasus, Anda mungkin ingin mengalihkan pemrosesan ke utas yang berbeda, atau menggunakan kumpulan utas jika pemrosesan rekaman Anda diharapkan intensif. Dengan cara ini, KCL dapat mengambil catatan baru lebih cepat sementara utas lainnya dapat memproses catatan secara paralel. Desain multithreaded tidak sepele dan harus didekati dengan teknik canggih, jadi meningkatkan jumlah pecahan Anda biasanya merupakan cara paling efektif untuk meningkatkan.

Langkah selanjutnya

(Opsional) Memperluas konsumen

(Opsional) Memperluas konsumen

Aplikasi di Tutorial: Memproses data stok 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

Pembersihan sumber daya

Pembersihan sumber daya

Karena Anda membayar untuk menggunakan aliran data Kinesis, pastikan Anda menghapusnya dan tabel Amazon DynamoDB yang sesuai ketika Anda selesai dengannya. Biaya nominal terjadi pada aliran aktif bahkan ketika Anda tidak mengirim dan mendapatkan catatan. Ini karena aliran aktif menggunakan sumber daya dengan terus “mendengarkan” 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 /kinesis. https://console.aws.amazon.com
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 dekat tidak memerlukan penulisan kode yang rumit atau mengembangkan infrastruktur yang besar. Ini sama dasarnya dengan menulis logika untuk memproses sejumlah kecil data (seperti menulis`processRecord(Record)`) tetapi menggunakan Kinesis Data Streams untuk skala sehingga berfungsi untuk sejumlah besar data streaming. Anda tidak perlu khawatir tentang bagaimana skala pemrosesan Anda 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 rekaman baru yang diterima.

Berikut adalah beberapa peningkatan potensial untuk aplikasi ini.

Agregat di semua pecahan

Saat ini, Anda mendapatkan statistik yang dihasilkan dari agregasi catatan data yang diterima oleh satu pekerja dari satu pecahan. (Pecahan tidak dapat diproses oleh lebih dari satu pekerja dalam satu aplikasi pada saat yang bersamaan.) Tentu saja, ketika Anda menskalakan dan memiliki lebih dari satu pecahan, Anda mungkin ingin menggabungkan 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 dengan mudah ditangani oleh satu pecahan.

Pemrosesan skala

Ketika aliran meningkat untuk memiliki banyak pecahan (karena banyak produsen mengirim data), cara untuk menskalakan pemrosesan adalah dengan menambahkan lebih banyak pekerja. Anda dapat menjalankan pekerja di instans Amazon EC2 dan menggunakan grup Auto Scaling.

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

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

Langkah selanjutnya

- Untuk informasi selengkapnya tentang penggunaan operasi API Kinesis Data StreamsKembangkan produsen menggunakan API Amazon Kinesis Data Streams dengan AWS SDK untuk Java, Mengembangkan konsumen throughput bersama dengan AWS SDK untuk Java lihat, dan. Membuat dan mengelola aliran data Kinesis

- Untuk informasi lebih lanjut tentang Perpustakaan Klien Kinesis, lihat. Kembangkan konsumen KCL 1.x
- Untuk informasi selengkapnya tentang cara mengoptimalkan aplikasi Anda, lihatOptimalkan konsumen Amazon Kinesis Data Streams.

Tutorial: Menganalisis data stok real-time menggunakan Amazon Managed Service untuk Apache 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 dekat.

Dengan Amazon Managed Service untuk Apache Flink, Anda dapat menggunakan Java atau Scala untuk memproses dan menganalisis data streaming. Layanan ini memungkinkan Anda 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 AWS Wilayah yang mendukung Layanan Terkelola untuk Apache Flink.

Tugas

- Prasyarat untuk menyelesaikan latihan
- Siapkan AWS akun dan buat pengguna administrator
- Mengatur AWS Command Line Interface (AWS CLI)
- Membuat dan menjalankan Managed Service untuk aplikasi Apache 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 Siapkan AWS akun dan buat pengguna administrator.

Siapkan AWS akun dan buat pengguna administrator

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

1. Mendaftar untuk AWS
2. Mmebuat pengguna IAM

Mendaftar untuk AWS

Saat Anda mendaftar ke Amazon Web Services (AWS), AWS akun Anda secara otomatis mendaftar untuk semua layanan AWS, 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 AWS akun, lompat ke tugas berikutnya. Jika Anda tidak memiliki akun AWS , buat dengan mengikuti langkah-langkah ini.

Untuk membuat AWS akun

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

   Bagian dari prosedur pendaftaran melibatkan menerima panggilan telepon atau pesan teks dan memasukkan kode verifikasi pada keypad telepon.

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

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

Mmebuat pengguna IAM

Layanan di AWS, 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. Yang Konsol Manajemen AWS mengharuskan Anda memasukkan kata sandi Anda.

Anda dapat membuat kunci akses untuk AWS akun Anda untuk mengakses AWS Command Line Interface (AWS CLI) atau API. Namun, kami tidak menyarankan Anda mengakses AWS menggunakan kredensil untuk akun Anda AWS . Sebagai gantinya, kami menyarankan Anda menggunakan 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 Anda belum membuat pengguna IAM untuk diri 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 Konsol Manajemen AWS 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 Berikutnya: 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 Konsol Manajemen AWS.

2. Gunakan format URL berikut untuk masuk ke konsol:

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

   *aws_account_number*Ini adalah ID AWS akun Anda tanpa tanda hubung. Misalnya, jika ID AWS akun Anda adalah 1234-5678-9012, ganti dengan. *aws_account_number* **123456789012** Untuk informasi tentang cara menemukan nomor akun Anda, lihat ID AWS Akun Anda dan Aliasnya 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 login Anda berisi ID AWS akun 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 menggunakan ID AWS akun 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 berikut ini:

- AWS Identity and Access Management (IAM)

- Memulai dengan IAM
- Panduan Pengguna IAM

Langkah berikutnya

Mengatur AWS Command Line Interface (AWS CLI)

Mengatur AWS Command Line Interface (AWS CLI)

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

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

> **Note**
>
> Jika Anda sudah AWS CLI menginstal, 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 AWS CLI versi berikut atau yang lebih baru:
>
> ```
> aws-cli/1.16.63
> ```

Untuk mengatur AWS CLI

1. Unduh dan konfigurasikan AWS CLI. Untuk instruksi, lihat topik berikut di AWS Command Line Interface Panduan Pengguna:
   - Menginstal AWS Command Line Interface

- Mengonfigurasi AWS CLI

2. Tambahkan profil bernama untuk pengguna administrator di file AWS CLI konfigurasi. Anda menggunakan profil ini saat menjalankan AWS CLI perintah. 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 AWS Wilayah 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 dan AWS CLI, Anda dapat mencoba latihan berikutnya, di mana Anda mengkonfigurasi aplikasi sampel dan menguji end-to-end pengaturan.

Langkah berikutnya

Membuat dan menjalankan Managed Service untuk aplikasi Apache Flink

Membuat dan menjalankan Managed Service untuk aplikasi Apache Flink

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

Bagian ini berisi langkah-langkah berikut.

- Buat dua aliran data Amazon Kinesis
- 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 aplikasi Apache Flink

Buat dua aliran data Amazon Kinesis

Sebelum Anda membuat Amazon Managed Service untuk Apache 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 stream (`ExampleInputStream`) pertama, gunakan perintah Amazon Kinesis `create-stream` AWS CLI 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 untuk 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 Amazon Managed Service untuk pustaka Apache Flink.
- `main`Metode yang berisi logika aplikasi.

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

Untuk membuat dan mengkompilasi kode aplikasi

1. Buat Java/Maven aplikasi 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 menggunakan AWS CLI, Anda menentukan jenis 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 ***<username>*** bucket ka-app-code-, dan pilih Unggah.
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 aplikasi Apache Flink

Anda dapat membuat dan menjalankan Layanan Terkelola untuk aplikasi Apache 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 /kinesis. https://console.aws.amazon.com
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 Amazon Managed Service untuk aplikasi Apache Flink menggunakan konsol, Anda memiliki opsi untuk membuat peran dan kebijakan IAM 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 akun sampel IDs (*012345678901*) dengan ID akun Anda.

   JSON

```
{
    "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:

Key	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.

Memperbarui 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 untuk Apache Flink 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 (ARNs) (*`012345678901`*) dengan ID akun Anda.

JSON

```
{
    "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 menggunakan AWS SDK untuk Java. Layanan Terkelola untuk Apache Flink secara otomatis menetapkan kredensil 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 Managed Service for Apache 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 Berikutnya: 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 KAReadInputStreamWriteOutputStreamkebijakan, lalu 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: Gunakan AWS Lambda dengan Amazon Kinesis Data Streams

Dalam tutorial ini, Anda membuat fungsi Lambda untuk mengkonsumsi peristiwa dari aliran data Kinesis. Dalam skenario contoh ini, aplikasi kustom menulis catatan ke aliran data Kinesis. AWS Lambda kemudian polling aliran data ini dan, ketika mendeteksi catatan data baru, memanggil fungsi Lambda Anda. AWS Lambda kemudian menjalankan fungsi Lambda dengan mengasumsikan peran eksekusi yang Anda tentukan saat Anda membuat fungsi Lambda.

Untuk petunjuk langkah demi langkah yang terperinci, lihat Tutorial: Menggunakan AWS Lambda dengan Amazon Kinesis.

> **Note**
>
> Tutorial ini mengasumsikan bahwa Anda memiliki pengetahuan tentang operasi Lambda dasar dan AWS Lambda konsol. Jika Anda belum melakukannya, ikuti petunjuk di Memulai dengan AWS Lambda untuk membuat fungsi Lambda pertama Anda.

Gunakan Solusi Data AWS Streaming untuk Amazon Kinesis

Solusi Data AWS Streaming untuk Amazon Kinesis secara otomatis mengonfigurasi AWS layanan yang diperlukan untuk menangkap, menyimpan, memproses, dan mengirimkan data streaming dengan mudah. 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:

- CloudFormation Paket untuk menyebarkan 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 aliran data Kinesis

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. Pecahan mendukung 1 MB/s dan 1000 catatan per detik untuk penulisan dan 2 MB/s untuk pembacaan dalam mode kapasitas sesuai permintaan dan yang disediakan. Batas pecahan memastikan kinerja yang dapat diprediksi, membuatnya lebih mudah untuk merancang dan mengoperasikan alur kerja streaming data yang sangat andal.

Di bagian ini, Anda mempelajari cara mengatur mode kapasitas untuk aliran dan cara membuat aliran menggunakan Konsol Manajemen AWS atau APIs. Kemudian Anda dapat mengambil tindakan tambahan di streaming.

Topik

- Pilih mode yang tepat untuk streaming
- Buat aliran menggunakan Konsol Manajemen AWS
- Buat aliran menggunakan APIs
- Perbarui aliran
- Daftar aliran
- Daftar pecahan
- Hapus aliran
- Reshard aliran
- Ubah periode retensi data
- Tandai sumber daya Amazon Kinesis Data Streams Anda
- Menangani catatan besar
- Lakukan pengujian ketahanan dengan AWS Fault Injection Service

Pilih mode yang tepat untuk streaming

Topik berikut menjelaskan cara memilih mode terbaik untuk aplikasi Anda dan cara beralih antar mode, jika diperlukan.

Topik

- Apa saja mode yang berbeda dalam Kinesis Data Streams?
- Fitur mode Standar sesuai permintaan dan kasus penggunaan
- Fitur mode Advantage On-Demand dan kasus penggunaan
- Fitur mode yang disediakan dan kasus penggunaan
- Beralih antar mode

Apa saja mode yang berbeda dalam Kinesis Data Streams?

Mode menentukan bagaimana kapasitas aliran data dikelola dan bagaimana Anda dikenakan biaya untuk penggunaan aliran data Anda. Di Amazon Kinesis Data Streams, Anda dapat memilih Standar Sesuai Permintaan, Keuntungan Sesuai Permintaan, dan disediakan sebagai mode untuk aliran data Anda.

- Standar 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.
- On-Demand Advantage - Mode tingkat akun yang memungkinkan lebih banyak kemampuan dan menyediakan struktur harga yang lebih sederhana untuk aliran sesuai permintaan. Dalam mode ini, Anda dapat secara proaktif menghangatkan kapasitas throughput tulis aliran kapan saja. Untuk penetapan harga, tidak ada lagi biaya tingkat per aliran tetap, dan konsumsi data, pengambilan data, dan penggunaan retensi yang diperpanjang di semua aliran sesuai permintaan setidaknya 60% lebih rendah daripada Standar Sesuai Permintaan.
- 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` APIs Streams dan untuk menulis `PutRecord` data ke dalam aliran data Anda dalam mode apa pun. Untuk mengambil data, ketiga mode

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.

Fitur mode Standar sesuai permintaan dan kasus penggunaan

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.

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. Lihat informasi yang lebih lengkap di struktur harga Amazon Kinesis Data Streams.

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, then Kinesis Data Streams ensures that you can easily burst to double your previous peak throughput, or 80 MB/s. If the same data stream sustains a new peak throughput of 50 MB/s, Kinesis Data Streams ensures that there is enough capacity to ingest 100 MB/s 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 on-demand (sama seperti 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 yang masuk melebihi 500 KB/s per pecahan, 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 kunci partisi tunggal melebihi 1 MB/s throughput pecahan dan 1000 record per detik batas.

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.

Fitur mode Advantage On-Demand dan kasus penggunaan

On-Demand Advantage adalah pengaturan tingkat akun yang membuka lebih banyak kemampuan dan menyediakan struktur harga yang berbeda untuk semua aliran sesuai permintaan di Wilayah. Dalam mode ini, aliran sesuai permintaan mempertahankan fungsionalitasnya dan terus menskalakan kapasitas secara otomatis berdasarkan penggunaan data aktual. Jika Anda ingin secara proaktif menghangatkan kapasitas throughput tulis aliran, Anda dapat mengonfigurasi throughput hangat. Misalnya, jika aliran data Anda memiliki throughput tulis antara 10 MB/s dan 40 MB/s, you can expect it to handle up to 80MB/s of instant throughput increases without throttling. However, if you forecast an upcoming event to peak around 200MB/s of traffic, you can configure the stream with a warm throughput of 200MB/s untuk memastikan bahwa kapasitas tersedia saat throughput data tiba. Menggunakan throughput hangat tidak dikenakan biaya tambahan.

Manfaat lain dari mode On-Demand Advantage adalah bahwa aliran on-demand bertransisi ke struktur harga yang lebih sederhana. Dengan mode diaktifkan, akun tidak akan lagi melihat biaya tetap per aliran, dan Anda hanya akan berurusan dengan penyerapan data, pengambilan data, dan biaya retensi tambahan opsional. Setiap dimensi harga juga memiliki diskon yang signifikan dibandingkan dengan dimensi yang sesuai dalam Standar On-Demand. Lihat informasi yang lebih lengkap di struktur harga Amazon Kinesis Data Streams.

Pengambilan data fan-out yang disempurnakan juga tidak memiliki premi harga dibandingkan dengan pengambilan data standar dalam mode ini. Selain itu, dengan mode On-Demand Advantage, Anda dapat mendaftarkan hingga 50 konsumen per streaming untuk menggunakan fan-out yang ditingkatkan. Mengaktifkan On-Demand Advantage mengikat akun ke setidaknya 25 MiB/s of data ingest and 25MiB/s pengambilan data di semua aliran sesuai permintaan. Untuk akun yang memenuhi persyaratan penggunaan minimum, konsol Kinesis Data Streams memiliki pemeriksaan untuk melihat apakah pola penggunaan akun Anda cocok untuk menggunakan mode Advantage Sesuai Permintaan.

Jika penggunaan data akun Anda di bawah persyaratan, Anda akan dikenakan selisihnya sebagai kekurangan, tetapi masih akan memiliki tarif diskon yang sama. Mengaktifkan On-Demand Advantage juga memiliki periode minimum 24 jam sebelum Anda dapat menonaktifkan mode. Secara keseluruhan, On-Demand Advantage adalah cara terbaik untuk melakukan streaming dengan Kinesis Data Streams jika Anda memiliki penggunaan throughput yang konsisten mendekati atau di atas komitmen minimum, membutuhkan banyak konsumen penggemar, atau beroperasi dengan ratusan aliran data.

Fitur mode yang disediakan dan kasus penggunaan

Dengan mode yang disediakan, setelah Anda membuat aliran data, Anda dapat menskalakan kapasitas pecahan secara dinamis ke atas atau ke bawah menggunakan atau API. Konsol Manajemen AWS 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 input 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 = ceiling(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

Untuk setiap aliran data di Anda Akun AWS, Anda dapat beralih antara mode sesuai permintaan dan yang disediakan dua kali dalam 24 jam. Beralih antar mode tidak menyebabkan gangguan pada aplikasi Anda yang menggunakan aliran data ini. Anda dapat terus menulis dan membaca dari aliran data ini. Saat Anda beralih antar mode, 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.

Anda dapat beralih dari mode On-Demand Standard ke On-Demand Advantage dengan mengaktifkan pengaturan level akun. Saat diaktifkan, akun berkomitmen untuk setidaknya 25 MiB/s of data ingest and 25MiB/s penggunaan pengambilan data di semua aliran sesuai permintaan di Wilayah. Setelah diaktifkan, Anda harus menunggu setidaknya 24 jam sebelum Anda dapat menonaktifkan On-Demand Advantage, tetapi Anda dapat meminta perubahan kapan saja. Jika Anda ingin beralih dari On-Demand Advantage ke On-Demand Standard, Anda harus terlebih dahulu menghapus throughput hangat yang dikonfigurasi dengan aliran permintaan.

Buat aliran menggunakan Konsol Manajemen AWS

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 Konsol Manajemen AWS dan buka konsol Kinesis di /kinesis. https://console.aws.amazon.com
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 Pilih mode yang tepat untuk streaming.

   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. Buat aliran menggunakan APIs

Untuk membuat aliran menggunakan AWS CLI

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

Buat aliran menggunakan APIs

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, kredensyal, 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 aliran

Sekarang setelah Anda membuat klien Kinesis Data Streams, Anda dapat membuat aliran menggunakan konsol atau secara terprogram. Untuk membuat aliran secara terprogram, buat instance `CreateStreamRequest` objek dan tentukan nama untuk aliran. Jika Anda ingin menggunakan mode yang disediakan, tentukan 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. Untuk throughput yang disediakan lebih besar, Anda membutuhkan lebih banyak pecahan. Lebih banyak pecahan juga meningkatkan biaya yang AWS dikenakan untuk aliran. Untuk informasi selengkapnya tentang menghitung jumlah pecahan yang sesuai untuk aplikasi Anda, lihatPilih mode yang tepat untuk streaming.

Setelah Anda mengkonfigurasi `createStreamRequest` objek, 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" );
}
```

Perbarui aliran

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.

Gunakan konsol

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 Pilih mode yang tepat untuk streaming.

   > **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 pengguna AWS KMS , lihat. the section called “Izin untuk menggunakan kunci 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 Kinesis Data Streams dengan CloudWatch”.

Gunakan API

Untuk memperbarui detail streaming menggunakan API, lihat metode berikut:

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

Gunakan AWS CLI

Untuk informasi tentang memperbarui aliran menggunakan AWS CLI, lihat referensi Kinesis CLI.

Daftar aliran

Stream 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 `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 ke`listStreams`. Jika demikian, kode memanggil `setExclusiveStartStreamName` metode dengan nama aliran terakhir yang dikembalikan dalam panggilan sebelumnya ke`listStreams`. `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,Buat aliran menggunakan APIs.

Daftar pecahan

Aliran data dapat memiliki satu atau lebih pecahan. 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` APIs 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 Anda AWS . 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 Anda yang memanggil DescribeStream API untuk memanggil DescribeStreamSummary dan. ListShard APIs 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`

Hapus aliran

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.

Reshard 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 yang APIs 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

- Tentukan strategi untuk resharding
- Pisahkan pecahan
- Gabungkan dua pecahan
- Selesaikan tindakan resharding

Tentukan 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 MD5Streams digunakan 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 pecahan tempat catatan data Anda ditetapkan. IDs 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 shard IDs 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.

Pisahkan 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 MD5hash 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 diTunggu streaming menjadi aktif kembali.

Gabungkan 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` Pertimbangkan perutean data, persistensi data, dan status pecahan 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 diTunggu streaming menjadi aktif kembali.

Selesaikan tindakan 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

- Tunggu streaming menjadi aktif kembali

- Pertimbangkan perutean data, persistensi data, dan status pecahan setelah reshard

Tunggu streaming menjadi aktif kembali

Setelah Anda memanggil operasi resharding, salah satu `splitShard` ataumergeShards, Anda harus menunggu streaming menjadi aktif kembali. Kode yang 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" );
}
```

Pertimbangkan perutean data, persistensi data, dan status pecahan setelah reshard

Kinesis Data Streams adalah layanan streaming data real-time. Aplikasi Anda harus berasumsi 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. Pecahan induk tidak hilang saat 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, lihatUbah 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, lihatUbah 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 mungkin masih 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.

Ubah 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 menggunakan AWS 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.

Tandai sumber daya Amazon Kinesis Data Streams Anda

Anda dapat menetapkan metadata Anda sendiri ke streaming dan konsumen penggemar yang disempurnakan yang Anda buat di Amazon Kinesis Data Streams dalam bentuk tag. Tanda adalah pasangan nilai-kunci yang Anda tetapkan untuk sebuah aliran. Menggunakan tag adalah cara sederhana namun ampuh untuk mengelola AWS sumber daya dan mengatur data, termasuk data penagihan.

Daftar Isi

- Tinjau dasar-dasar tag
- Lacak biaya menggunakan penandaan
- Memahami batasan tag
- Tag stream menggunakan konsol Kinesis Data Streams
- Tag stream menggunakan AWS CLI
- Tag stream menggunakan Kinesis Data Streams APIs
- Tag konsumen menggunakan AWS CLI
- Tag konsumen menggunakan Kinesis Data Streams APIs

Tinjau dasar-dasar tag

Sumber daya Kinesis Data Streams yang dapat Anda beri tag termasuk aliran data dan konsumen penggemar yang disempurnakan. Anda menggunakan AWS CLI konsol Kinesis Data Streams, atau Kinesis Data Streams API untuk menyelesaikan tugas-tugas berikut:

- Buat sumber daya dengan tag
- Tambahkan tag ke sumber daya
- Buat daftar tag untuk sumber daya Anda
- Hapus tag dari sumber daya

> **Note**
>
> Anda tidak dapat menerapkan tag ke konsumen penggemar yang disempurnakan menggunakan konsol Kinesis Data Streams. Untuk menerapkan tag ke konsumen, gunakan AWS CLI atau Kinesis Data Streams API.

Anda dapat menggunakan tag untuk mengkategorikan sumber daya Anda. Misalnya, Anda dapat mengkategorikan sumber daya berdasarkan tujuan, 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 tag yang membantu Anda melacak sumber daya oleh pemilik dan aplikasi terkait. Berikut adalah beberapa contoh tanda:

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

> **Important**
>
> - Untuk menambahkan tag saat membuat aliran, Anda harus menyertakan `kinesis:AddTagsToStream` izin `kinesis:CreateStream` dan untuk aliran tersebut.

Anda tidak dapat menggunakan `kinesis:TagResource` izin untuk menandai aliran saat membuatnya.

- Untuk menambahkan tag selama pendaftaran konsumen, Anda harus menyertakan `kinesis:TagResource` dan `kinesis:RegisterStreamConsumer` izin.

Lacak biaya menggunakan penandaan

Anda dapat menggunakan tag untuk mengkategorikan dan melacak biaya Anda AWS . Saat Anda menerapkan tag ke sumber daya Kinesis Data Streams AWS , laporan 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 beberapa layanan. Untuk informasi selengkapnya, lihat Menggunakan Tanda Alokasi Biaya untuk Laporan Penagihan Khusus dalam Panduan Pengguna AWS Billing .

Memahami batasan tag

Batasan berikut ini berlaku untuk tag:

Batasan dasar

- Jumlah maksimum tag untuk setiap sumber daya adalah 50.
- Kunci dan nilai tag peka terhadap huruf besar dan kecil.
- Anda tidak dapat mengubah atau mengedit tag untuk sumber daya 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 tag `aws:` karena awalan ini dicadangkan untuk digunakan oleh AWS. AWS membuat tag yang dimulai dengan awalan ini atas nama Anda, tetapi Anda tidak dapat mengedit atau menghapusnya.
- Kunci tanda harus memiliki panjang antara 1 dan 128 karakter Unicode.
- Kunci tanda harus terdiri dari karakter berikut: huruf Unicode, digit, spasi, dan karakter khusus berikut: `_ . / = + - @`.

Batasan nilai tanda

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

Tag stream menggunakan konsol Kinesis Data Streams

Anda dapat menambahkan, memperbarui, mencantumkan, dan menghapus tag di aliran menggunakan konsol Kinesis Data Streams.

Untuk melihat tag untuk aliran

1. Masuk ke Konsol Manajemen AWS dan buka konsol Kinesis di /kinesis. https://console.aws.amazon.com
2. Di panel navigasi kiri, pilih Aliran data.
3. Pada halaman Data stream, pilih stream yang ingin Anda tag.
4. Pada halaman Detail Stream, pilih Konfigurasi.
5. Di bagian Tag, lihat tag yang diterapkan ke aliran.

Untuk membuat aliran data dengan tag

1. Buka konsol Kinesis Data Streams.
2. Di panel navigasi kiri, pilih Aliran data.
3. Pilih Create data stream (Buat aliran data).
4. Pada halaman Buat aliran data, masukkan nama untuk aliran data Anda.
5. Untuk kapasitas aliran Data, pilih mode On-Demand atau Provisioned capacity.

   Untuk informasi selengkapnya tentang mode kapasitas, lihatPilih mode yang tepat untuk streaming.
6. Di bagian Tag, lakukan hal berikut:

   a. Pilih Tambahkan tag baru.

   b. Untuk Kunci, masukkan tag dan secara opsional tentukan nilai di bidang Nilai.

Jika Anda melihat kesalahan, kunci tag atau nilai yang Anda tentukan tidak memenuhi batasan tag. Untuk informasi selengkapnya, lihat Memahami batasan tag.

7. Pilih Create data stream (Buat aliran data).

Untuk menambah atau memperbarui tag pada aliran

1. Buka konsol Kinesis Data Streams.
2. Di panel navigasi kiri, pilih Aliran data.
3. Pada halaman Aliran data, pilih aliran yang ingin Anda tambahkan atau perbarui tag.
4. Pada halaman Detail Stream, pilih Konfigurasi.
5. Di bagian Tanda, pilih Kelola tanda.
6. Di bawah Tag, lakukan salah satu hal berikut:

   - Untuk menambahkan tag baru, pilih Tambahkan tag baru, lalu masukkan data Kunci dan Nilai tag. Ulangi langkah ini seperlunya.

     Jumlah maksimum tag yang dapat Anda tambahkan untuk setiap aliran adalah 50.
   - Untuk memperbarui tag yang ada, masukkan nilai tag baru di bidang Nilai Kunci tag tersebut.

   Jika Anda melihat kesalahan, kunci tag atau nilai yang Anda tentukan tidak memenuhi batasan tag. Untuk informasi selengkapnya, lihat Memahami batasan tag.
7. Pilih Simpan perubahan.

Untuk menghapus tag dari aliran

1. Buka konsol Kinesis Data Streams.
2. Di panel navigasi kiri, pilih Aliran data.
3. Pada halaman Aliran data, pilih aliran dari mana Anda ingin menghapus tag.
4. Pada halaman Detail Stream, pilih Konfigurasi.
5. Di bagian Tanda, pilih Kelola tanda.
6. Temukan tag Key dan Value pair yang ingin Anda hapus. Kemudian, pilih Hapus.
7. Pilih Simpan perubahan.

Tag stream menggunakan AWS CLI

Anda dapat menambahkan, membuat daftar, dan menghapus tag pada aliran Anda menggunakan. AWS CLI Sebagai contoh, lihat dokumentasi berikut.

buat-aliran

Membuat aliran dengan tag.

add-tags-to-stream

Menambahkan atau memperbarui tag untuk aliran yang ditentukan.

list-tags-for-stream

Daftar tag untuk aliran yang ditentukan.

remove-tags-from-stream

Menghapus tag dari aliran yang ditentukan.

Tag stream menggunakan Kinesis Data Streams APIs

Anda dapat menambahkan, membuat daftar, dan menghapus tag pada aliran Anda menggunakan Kinesis Data Streams. APIs Untuk contoh, lihat dokumentasi berikut:

CreateStream

Membuat aliran dengan tag.

AddTagsToStream

Menambahkan atau memperbarui tag untuk aliran yang ditentukan.

ListTagsForStream

Daftar tag untuk aliran yang ditentukan.

RemoveTagsFromStream

Menghapus tag dari aliran yang ditentukan.

Tag konsumen menggunakan AWS CLI

Anda dapat menambahkan, membuat daftar, dan menghapus tag pada konsumen Anda menggunakan AWS CLI. Untuk contoh, lihat dokumentasi berikut:

register-stream-consumer

Mendaftarkan konsumen untuk aliran data Kinesis dengan tag.

tag-sumber daya

Menambahkan atau memperbarui tag untuk sumber Kinesis tertentu.

list-tags-for-resource

Daftar tag untuk sumber daya Kinesis yang ditentukan.

untag-sumber daya

Menghapus tag dari sumber Kinesis yang ditentukan.

Tag konsumen menggunakan Kinesis Data Streams APIs

Anda dapat menambahkan, membuat daftar, dan menghapus tag pada konsumen Anda menggunakan Kinesis APIs Data Streams. Untuk contoh, lihat dokumentasi berikut:

RegisterStreamConsumer

Mendaftarkan konsumen untuk aliran data Kinesis dengan tag.

TagResource

Menambahkan atau memperbarui tag untuk sumber Kinesis tertentu.

ListTagsForResource

Daftar tag untuk sumber daya Kinesis yang ditentukan.

UntagResource

Menghapus tag dari sumber Kinesis yang ditentukan.

Menangani catatan besar

Amazon Kinesis Data Streams mendukung rekaman hingga 10 MiBs mebibytes (). Kemampuan ini direkomendasikan untuk memproses muatan data intermiten yang melebihi batas ukuran catatan 1 MiB default. Ukuran rekaman maksimum default untuk aliran yang ada, dan yang baru dibuat disetel ke 1 MiB.

Fitur ini menguntungkan aplikasi Internet of Things (IoT), mengubah jalur pengambilan data (CDC), dan alur kerja pembelajaran mesin yang memerlukan pemrosesan muatan data yang sesekali lebih besar. Untuk mulai menggunakan rekaman besar di streaming Anda, perbarui batas ukuran rekaman maksimum aliran Anda.

> **Important**
>
> Batas throughput pecahan individu 1 MB/s untuk penulisan, dan 2 untuk pembacaan tetap tidak berubah dengan dukungan MB/s untuk ukuran rekaman yang lebih besar. Kinesis Data Streams dirancang untuk mengakomodasi rekaman besar intermiten di samping lalu lintas dasar catatan kurang dari, atau sama dengan 1 MiB. Ini tidak dirancang untuk mengakomodasi konsumsi rekor besar volume tinggi yang berkelanjutan.

Perbarui aliran Anda untuk menggunakan catatan besar

Untuk memproses rekaman yang lebih besar dengan Kinesis Data Streams

1. Arahkan ke konsol Kinesis Data Streams.
2. Pilih aliran Anda, dan buka tab Konfigurasi.
3. Klik Edit, yang terletak di sebelah Ukuran rekaman maksimum.
4. Tetapkan ukuran rekaman maksimum Anda (hingga 10 MiB).
5. Simpan perubahan Anda.

Pengaturan ini hanya menyesuaikan ukuran rekaman maksimum untuk aliran data Kinesis ini. Sebelum meningkatkan batas ini, verifikasi bahwa semua aplikasi hilir dapat menangani catatan yang lebih besar.

Anda juga dapat memperbarui pengaturan ini menggunakan AWS CLI:

```
aws kinesis update-max-record-size \ --stream-arn  \
        --max-record-size-in-ki-b 5000
```

Optimalkan performa streaming Anda dengan catatan besar

Disarankan untuk menyimpan catatan besar hingga kurang dari 2% dari keseluruhan lalu lintas Anda. Dalam aliran, setiap pecahan memiliki kapasitas throughput 1 MiB per detik. Untuk mengakomodasi rekaman besar, aliran Data Kinesis meledak hingga 10 MiBs, sementara rata-rata hingga 1 MiB per detik. Kapasitas untuk mendukung catatan besar ini terus diisi ulang ke dalam aliran. Tingkat pengisian ulang tergantung pada ukuran catatan besar dan ukuran catatan dasar. Untuk hasil terbaik, gunakan kunci partisi yang didistribusikan secara seragam. Untuk informasi selengkapnya tentang bagaimana Kinesis sesuai permintaan skala, lihat Fitur mode sesuai permintaan dan kasus penggunaan.

Mengurangi pelambatan dengan catatan besar

Untuk mengurangi pelambatan

1. Terapkan logika coba lagi dengan back-off eksponensial dalam aplikasi produser Anda.
2. Gunakan kunci partisi acak untuk mendistribusikan catatan besar di seluruh pecahan yang tersedia.
3. Simpan muatan di Amazon S3 dan kirim hanya referensi metadata ke aliran untuk aliran rekaman besar yang berkelanjutan. Untuk informasi selengkapnya, lihat Memproses rekaman besar dengan Amazon Kinesis Data Streams.

Menangani rekaman besar menggunakan Kinesis Data Streams APIs

Dukungan rekaman besar memperkenalkan satu API baru, dan memperbarui dua bidang kontrol yang ada APIs untuk menangani catatan hingga 10 MiBs.

API untuk memodifikasi ukuran rekaman:

- `UpdateMaxRecordSize`: Mengonfigurasi batas ukuran rekaman maksimum untuk aliran yang ada hingga 10. MiBs

Pembaruan untuk yang ada APIs:

- `CreateStream`: Menambahkan `MaxRecordSizeInKiB` parameter opsional untuk menetapkan batas ukuran rekaman selama pembuatan aliran.
- `DescribeStreamSummary`: Mengembalikan `MaxRecordSizeInKiB` bidang untuk menunjukkan konfigurasi aliran saat ini.

Semua APIs yang terdaftar mempertahankan kompatibilitas mundur untuk aliran yang ada. Untuk dokumentasi API selengkapnya, lihat Referensi API Layanan Amazon Kinesis Data Streams.

AWS komponen yang kompatibel dengan catatan besar

AWS Komponen-komponen berikut kompatibel dengan catatan besar:

Komponen	Deskripsi
AWS SDK	AWS SDK mendukung penanganan catatan besar. Anda dapat memperbarui ukuran rekaman maksimum streaming hingga 10 MiB menggunakan metode yang tersedia di file. AWS SDKs Untuk informasi selengkapnya, lihat Menggunakan layanan ini dengan AWS SDK.
Perpustakaan Konsumen Kinesis (KCL)	Dimulai dengan versi 2.x, KCL mendukung penanganan catatan besar. Untuk menggunak an dukungan rekaman besar, perbarui `maxRecordSize` aliran Anda, dan gunakan KCL. Untuk informasi selengkapnya, lihat Menggunakan Perpustakaan Klien Kinesis.
Perpustakaan Produsen Kinesis (KPL)	Dimulai dengan versi 1.0.5, KPL mendukung penanganan catatan besar. Untuk menggunak an dukungan rekaman besar, perbarui `maxRecordSize` aliran Anda, dan gunakan KPL. Untuk informasi selengkapnya, lihat Mengembangkan produsen menggunakan Amazon Kinesis Producer Library (KPL).
Amazon EMR	Amazon EMR dengan Apache Spark mendukung penanganan rekaman besar

Komponen	Deskripsi
	hingga batas Kinesis Data Streams (10). MiBs Untuk menggunakan dukungan rekaman besar, gunakan `readStream` fungsi. Untuk informasi selengkapnya, lihat Amazon EMR dan integrasi Amazon Kinesis.

Komponen	Deskripsi
Amazon Data Firehose	Saat digunakan dengan Kinesis Data Streams, perilaku Amazon Data Firehose dengan catatan besar bergantung pada tujuan pengiriman: • Amazon S3: Pengiriman catatan besar didukung tanpa konfigurasi tambahan. Saat Anda menggunakan konversi format data, pengiriman catatan besar didukung dengan Firehose. Bila Anda menggunakan partisi dinamis, pengiriman rekaman besar tidak didukung dengan Firehose. • Lambda: Kami tidak menyarankan menggunakan catatan besar dengan Firehose saat memicu fungsi Lambda di hilir. Ini dapat menyebabkan kegagalan intermite n. • HTTP: Pengiriman catatan besar tidak didukung dengan Firehose. • Snowflake: Pengiriman catatan besar tidak didukung dengan Firehose. • Amazon Redshift: Pengiriman rekaman besar tidak didukung dengan Firehose. Untuk aplikasi yang memerlukan pengiriman ke Snowflake atau Redshift dengan catatan besar, kirimkan data ke Amazon S3 terlebih dahulu. Setelah itu, gunakan proses Extract, Transform , Load (ETL) untuk memuat data. Untuk semua tujuan lainnya, uji perilaku dengan catatan besar di proof-of-concept lingkungan sebelum menskalakan ke penggunaan produksi. Penanganan catatan besar bervariasi menurut tujuan.

Komponen	Deskripsi
AWS Lambda	AWS Lambda mendukung muatan hingga 6 MiBs. Batas ini mencakup payload Kinesis yang dikonversi ke pengkodean basis-64, dan metadata yang terkait dengan Pemetaan Sumber Peristiwa (ESM). Untuk catatan kurang dari 6 MiBs, Lambda memproses nya menggunakan ESM tanpa konfigurasi tambahan yang diperlukan. Untuk catatan yang lebih besar dari 6 MiBs, Lambda memproses nya menggunakan tujuan yang gagal. Anda harus mengonfigurasi tujuan saat gagal menggunakan ESM untuk menangani catatan yang melebihi batas pemrosesan Lambda. Setiap peristiwa yang dikirim ke tujuan kegagalan adalah dokumen JSON yang berisi metadata mengenai pemanggilan yang gagal. Disarankan untuk membuat tujuan on-failur e di ESM, terlepas dari ukuran rekaman. Ini memastikan bahwa tidak ada catatan yang dibuang. Untuk informasi selengkapnya, lihat Mengonfigurasi tujuan untuk pemanggilan yang gagal.
Amazon Redshift	Amazon Redshift hanya mendukung ukuran rekaman kurang dari 1 MiB saat streaming data dari Kinesis Data Streams. Catatan yang melebihi batas ini tidak diproses. Catatan yang tidak diproses dicatat sebagai`sys_strea m_scan_errors` . Untuk informasi selengkapnya, lihat SYS_STREAM_SCAN_ER RORS.

Komponen	Deskripsi
Konektor Flink untuk Kinesis Data Streams	Ada dua pendekatan untuk mengkonsumsi data dari Kinesis Data Streams: konektor sumber Kinesis, dan konektor sink Kinesis. Konektor sumber mendukung penanganan catatan kurang dari 1 MiB, dan hingga 10. MiBs Jangan gunakan konektor wastafel untuk catatan yang lebih besar dari 1 MiB. Untuk informasi selengkapnya, lihat Menggunakan konektor untuk memindahkan data di Amazon Managed Service untuk Apache Flink dengan API. DataStream

Wilayah di mana catatan besar didukung

Fitur Amazon Kinesis Data Streams ini hanya tersedia di AWS Wilayah berikut:

AWS Wilayah	Nama wilayah
eu-north-1	Eropa (Stockholm)
me-south-1	Timur Tengah (Bahrain)
ap-south-1	Asia Pasifik (Mumbai)
eu-west-3	Eropa (Paris)
ap-southeast-3	Asia Pasifik (Jakarta)
us-east-2	AS Timur (Ohio)
af-south-1	Africa (Cape Town)
eu-west-1	Eropa (Irlandia)
me-central-1	Timur Tengah (UAE)

AWS Wilayah	Nama wilayah
eu-central-1	Eropa (Frankfurt)
sa-east-1	Amerika Selatan (Sao Paulo)
ap-east-1	Asia Pasifik (Hong Kong)
ap-south-2	Asia Pasifik (Hyderabad)
us-east-1	AS Timur (Virginia Utara)
ap-northeast-2	Asia Pasifik (Seoul)
ap-northeast-3	Asia Pasifik (Osaka)
eu-west-2	Eropa (London)
ap-southeast-4	Asia Pacific (Melbourne)
ap-northeast-1	Asia Pasifik (Tokyo)
us-west-2	AS Barat (Oregon)
us-west-1	AS Barat (California Utara)
ap-southeast-1	Asia Pasifik (Singapura)
ap-southeast-2	Asia Pasifik (Sydney)
il-central-1	Israel (Tel Aviv)
ca-central-1	Kanada (Pusat)
ca-west-1	Kanada Barat (Calgary)
eu-south-2	Eropa (Spanyol)
cn-northwest-1	Tiongkok (Ningxia)
eu-central-2	Europe (Zurich)

AWS Wilayah	Nama wilayah
us-gov-east-1	AWS GovCloud (AS-Timur)
us-gov-west-1	AWS GovCloud (AS-Barat)

Lakukan pengujian ketahanan dengan AWS Fault Injection Service

AWS Fault Injection Service adalah layanan yang dikelola sepenuhnya yang membantu Anda melakukan eksperimen injeksi kesalahan pada AWS beban kerja Anda. AWS FIS integrasi dengan Amazon Kinesis Data Streams memungkinkan Anda menguji ketahanan aplikasi terhadap kesalahan API Amazon Kinesis Data Streams yang umum di lingkungan terkontrol. Kemampuan ini memungkinkan Anda untuk memvalidasi penanganan kesalahan, mencoba kembali logika, dan memantau sistem sebelum mengalami kegagalan. Untuk informasi lebih lanjut, lihat Apa itu AWS Fault Injection Service? .

Tindakan

- Kesalahan internal API: Ini menyuntikkan kesalahan internal ke dalam permintaan yang dibuat oleh peran IAM target. Respons spesifik tergantung pada setiap layanan dan API. Tindakan `aws:fis:inject-api-internal-error` menciptakan `InternalFailure` kesalahan (HTTP 500).
- Kesalahan throttle API: Ini menyuntikkan kesalahan internal ke dalam permintaan yang dibuat oleh peran IAM target. Respons spesifik tergantung pada setiap layanan dan API. Tindakan `aws:fis:inject-api-throttle-error` menciptakan `ThrottlingException` kesalahan (HTTP 400).
- Kesalahan API tidak tersedia: Ini menyuntikkan kesalahan internal ke dalam permintaan yang dibuat oleh peran IAM target. Respons spesifik tergantung pada setiap layanan dan API. Tindakan `aws:fis:inject-api-unavailable-error` menciptakan `ServiceUnavailable` kesalahan (HTTP 503).
- Pengecualian throughput yang disediakan API: Ini menyuntikkan kesalahan internal ke dalam permintaan yang dibuat oleh peran IAM target. Respons spesifik tergantung pada setiap layanan dan API. Tindakan `aws:kinesis:inject-api-provisioned-throughput-exception` menciptakan `ProvisionedThroughputExceededException` kesalahan (HTTP 400).
- Pengecualian iterator kedaluwarsa API: Ini menyuntikkan kesalahan internal ke dalam permintaan yang dibuat oleh peran IAM target. Respons spesifik tergantung pada setiap layanan dan API.

Tindakan `aws:kinesis:inject-api-expired-iterator-exception` menciptakan `ExpiredIteratorException` kesalahan (HTTP 400).

Untuk informasi selengkapnya, lihat tindakan Amazon Kinesis Data Streams.

Pertimbangan-pertimbangan

- Anda dapat menggunakan tindakan di atas dengan penawaran yang disediakan dan sesuai permintaan untuk Amazon Kinesis Data Streams.
- Streaming Anda dilanjutkan setelah percobaan selesai berdasarkan durasi yang dipilih. Anda juga dapat menghentikan eksperimen yang sedang berjalan sebelum selesai. Atau, Anda dapat menentukan kondisi berhenti untuk menghentikan eksperimen berdasarkan alarm yang menentukan kesehatan aplikasi di Amazon CloudWatch Application Insights.
- Anda dapat menguji hingga 280 aliran.

Untuk informasi lebih lanjut tentang dukungan regional, lihat AWS Fault Injection Service titik akhir dan kuota.

Kesalahan pengecualian throughput yang disediakan

Throughput yang disediakan melebihi kesalahan pengecualian (HTTP 400) terjadi ketika tingkat permintaan untuk aliran Kinesis melampaui batas throughput dari satu atau lebih pecahan. Setiap pecahan memiliki batas kapasitas baca, dan tulis tertentu, dan melebihi batas tersebut memicu pengecualian ini. Skenario yang mengarah ke pengecualian ini meliputi: lonjakan mendadak dalam konsumsi atau konsumsi data, kapasitas pecahan yang tidak mencukupi untuk volume data yang sedang diproses, atau distribusi kunci partisi yang tidak merata.

Rekomendasi untuk menangani pengecualian

- Menerapkan mekanisme back-off dan coba ulang eksponensial.
- Tingkatkan jumlah pecahan untuk mengakomodasi throughput yang lebih tinggi.
- Pastikan bahwa ada distribusi kunci partisi yang tepat.
- Pantau metrik aliran.

Selain itu, menggunakan mode kapasitas Kinesis sesuai permintaan membantu menyesuaikan beban kerja secara otomatis, dan meminimalkan terjadinya pengecualian ini. Untuk informasi selengkapnya, lihat Apa itu AWS Fault Injection Service?

> **Note**
>
> Masalah distribusi yang tidak tepat berada di luar kemampuan mode on-demand dari penskalaan otomatis.

Untuk melakukan eksperimen dasar

1. Gunakan metrik dasar: catat pola throughput normal sebelum pengujian.
2. Buat eksperimen: gunakan `aws:kinesis:inject-api-provisioned-throughput-exception` tindakan.
3. Konfigurasikan intensitas: mulai dengan pelambatan permintaan 25%.
4. Pantau tanggapan: verifikasi logika coba ulang dengan back-off eksponensial.
5. Validasi penskalaan: konfirmasikan bahwa auto-scaling memicu aktivasi.
6. Periksa alarm: pastikan `CloudWatch` alarm berjalan seperti yang diharapkan.

Aplikasi harus menerapkan strategi, monitor, dan `ReadProvisionedThroughputExceeded` metrik back-off yang tepat`WriteProvisionedThroughputExceeded`, dan memicu penskalaan shard bila sesuai.

Detail tindakan

- Jenis Sumber Daya: IAM Peran ARN
- Operasi Target:`PutRecord`,`PutRecords`, `GetRecords`
- Kode Kesalahan: `ProvisionedThroughputExceededException` (HTTP 400)
- Deskripsi: mensimulasikan skenario di mana tingkat permintaan melebihi batas kapasitas pecahan, menguji pelambatan aplikasi, dan respons penskalaan.

Parameter

- IAM Role ARN: peran yang digunakan aplikasi Anda untuk operasi Kinesis Data Streams.
- Operasi: operasi target:`PutRecord`,`PutRecords`,`GetRecords`.
- Daftar Sumber Daya: nama aliran tertentu atau pengidentifikasi pecahan.

- Durasi: durasi percobaan, yaitu durasi dari satu menit hingga 12 jam. Di AWS FIS API, nilainya adalah string dalam format ISO 8601. Misalnya, PT1 M mewakili satu menit. Di AWS FIS konsol, Anda memasukkan jumlah detik, menit, atau jam.
- Intensitas: persentase permintaan untuk throttle.

Izin yang diperlukan

- `kinesis:InjectApiError`

Contoh template percobaan

Contoh berikut menunjukkan pengecualian throughput yang disediakan untuk semua permintaan hingga 5 aliran Data Kinesis dengan tag yang ditentukan. AWS FIS memilih aliran untuk mempengaruhi secara acak. Setelah 5 menit kesalahan dihapus.

```
{
    "description": "Kinesis stream experiment",
    "targets": {
        "KinesisStreams-Target-1": {
            "resourceType": "aws:kinesis:stream",
            "resourceTags": {
                   "tag-key": "tag-value"
            },
            "selectionMode": "COUNT(5)"
        }
    },
    "actions": {
         "kinesis": {
              "actionId": "aws:kinesis:stream-provisioned-throughput-exception",
              "description": "my-stream",
              "parameters": {
                   "duration": "PT5M",
                   "percentage": "100",
                   "service": "kinesis"
              },
              "targets": {
                    "KinesisStreams": "KinesisStreams-Target-1"
              }
         }
   },
   "stopConditions": [
```

```
        {
            "source": "none"
        }
    ],
    "roleArn": "arn:aws:iam::111122223333:role/role-name",
    "tags": {},
    "experimentOptions": {
        "accountTargeting": "single-account",
        "emptyTargetResolutionMode": "fail"
    }
}
```

Contoh izin peran percobaan

Izin berikut memungkinkan Anda menjalankan `aws:kinesis:stream-provisioned-throughput-exception` dan `aws:kinesis:stream-expired-iterator-exception` tindakan pada aliran tertentu yang memengaruhi 50% permintaan.

Kesalahan pengecualian iterator kedaluwarsa

Kesalahan pengecualian iterator kedaluwarsa (HTTP 400) terjadi ketika iterator shard kedaluwarsa, dan tidak lagi digunakan untuk mengambil catatan aliran saat memanggil. `GetRecords` Ini terjadi ketika ada penundaan antara operasi baca, yang disebabkan oleh tugas pemrosesan data yang berjalan lama, masalah jaringan, atau waktu henti aplikasi.

> **Note**
>
> Sebuah iterator shard berlaku selama 5 menit setelah waktu dikeluarkan.

Rekomendasi untuk menangani pengecualian

- Menyegarkan iterator pecahan sebelum kedaluwarsa.
- Memasukkan penanganan kesalahan untuk mendapatkan iterator baru.
- Memanfaatkan Kinesis Kinesis Client Library (KCL) yang secara otomatis mengelola kedaluwarsa iterator shard.

Untuk informasi lebih lanjut, lihat Apa itu AWS Fault Injection Service?

Untuk melakukan eksperimen dasar

1. Buat templat eksperimen: gunakan AWS FIS konsol.
2. Pilih tindakan: gunakan `aws:kinesis:inject-api-expired-iterator-exception` tindakan.
3. Konfigurasikan target: tentukan peran IAM dan Streamsoperasi Data Kinesis.
4. Atur durasinya: mulai dengan 5-10 menit untuk pengujian awal.
5. Tambahkan kondisi berhenti: kondisi berhenti untuk AWS FIS.
6. Jalankan eksperimen: pantau perilaku aplikasi.

Detail tindakan

- Jenis Sumber Daya: IAM Peran ARN
- Operasi Target: `GetRecords`
- Kode Kesalahan: `ExpiredIteratorException` (HTTP 400)
- Deskripsi: iterator yang disediakan melebihi usia maksimum yang diizinkan, mensimulasikan skenario di mana pemrosesan rekaman terlalu lambat atau logika pos pemeriksaan gagal.

Parameter

- IAM Role ARN: peran yang digunakan aplikasi Anda untuk operasi Kinesis Data Streams.
- Operasi: operasi target: `GetRecords`
- Daftar Sumber Daya: nama aliran tertentu atau ARNs.
- Durasi: durasi percobaan. Ini dapat dikonfigurasi.
- Intensitas: persentase permintaan untuk throttle.

Izin yang diperlukan

- `kinesis:InjectApiError`

Menulis data ke Amazon Kinesis Data Streams

Produser adalah aplikasi yang menulis data ke Amazon Kinesis Data Streams. Anda dapat membangun produsen untuk Kinesis Data Streams AWS SDK untuk Java menggunakan dan Kinesis Producer Library (KPL).

Jika Anda baru mengenal Kinesis Data Streams, mulailah dengan menjadi akrab dengan konsep dan terminologi Apa itu Amazon Kinesis Data Streams? yang disajikan dalam dan. Gunakan AWS CLI untuk menjalankan operasi Amazon Kinesis Data Streams

> **Important**
>
> Kinesis Data Streams mendukung perubahan pada periode retensi rekaman data aliran data Anda. Untuk informasi selengkapnya, lihat Ubah 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 mana dalam aliran catatan data ditambahkan.

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 cara memetakan catatan data ke pecahan tertentu. Jika Anda memiliki cukup kunci partisi, data dapat didistribusikan secara merata di seluruh pecahan dalam aliran.

Topik

- Kembangkan produsen menggunakan Amazon Kinesis Producer Library (KPL)
- Kembangkan produsen menggunakan API Amazon Kinesis Data Streams dengan AWS SDK untuk Java
- Menulis ke Amazon Kinesis Data Streams menggunakan Agen Kinesis
- Menulis ke Kinesis Data AWS Streams menggunakan layanan lain
- Menulis ke Kinesis Data Streams menggunakan integrasi pihak ketiga
- Memecahkan masalah produsen Amazon Kinesis Data Streams
- Optimalkan produsen Kinesis Data Streams

Kembangkan produsen menggunakan Amazon Kinesis Producer Library (KPL)

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

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

Topik

- Tinjau peran KPL
- Sadarilah keuntungan menggunakan KPL
- Memahami kapan tidak menggunakan KPL
- Instal KPL
- Migrasi dari KPL 0.x ke KPL 1.x
- Transisi ke sertifikat Amazon Trust Services (ATS) untuk KPL
- Platform yang didukung KPL
- Konsep kunci KPL
- Integrasikan KPL dengan kode produsen
- Menulis ke aliran data Kinesis Anda menggunakan KPL
- Konfigurasikan Perpustakaan Produsen Amazon Kinesis
- Menerapkan de-agregasi konsumen
- Gunakan KPL dengan Amazon Data Firehose
- Gunakan KPL dengan AWS Glue Schema Registry
- Konfigurasikan konfigurasi proxy KPL
- Kebijakan siklus hidup versi KPL

> **Note**
>
> Disarankan agar Anda meningkatkan ke versi KPL terbaru. KPL diperbarui secara berkala dengan rilis yang lebih baru yang mencakup patch dependensi dan keamanan terbaru,

perbaikan bug, dan fitur baru yang kompatibel ke belakang. Untuk informasi selengkapnya, lihat https://github.com/awslabs/amazon-kinesis-producer/releases/.

Tinjau 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. AWS SDKs 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.

Sadarilah 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 runtime. 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

Memahami 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. Kembangkan produsen menggunakan API Amazon Kinesis Data Streams dengan AWS SDK untuk Java Untuk informasi selengkapnya tentang `RecordMaxBufferedTime` dan properti KPL yang dapat dikonfigurasi pengguna lainnya, lihat. Konfigurasikan Perpustakaan Produsen Amazon Kinesis

Instal KPL

Amazon menyediakan binari pra-bangun dari C++ Amazon Kinesis Producer Library (KPL) 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 sebelumnya dapat membangun KPL menggunakan instruksi build yang disediakan bersama dengan sumbernya. GitHub Untuk mengunduh KPL dari GitHub, lihat Perpustakaan Produser Amazon Kinesis.

> **Important**
>
> Amazon Kinesis Producer Library (KPL) 0.x akan mencapai end-of-support pada 30 Januari 2026. Kami sangat menyarankan agar Anda memigrasikan aplikasi KPL Anda menggunakan versi 0.x ke versi KPL terbaru sebelum 30 Januari 2026. Untuk menemukan versi KPL terbaru, lihat halaman KPL di Github. Untuk informasi tentang migrasi dari KPL 0.x ke KPL 1.x, lihat. Migrasi dari KPL 0.x ke KPL 1.x

Migrasi dari KPL 0.x ke KPL 1.x

Topik ini memberikan step-by-step petunjuk untuk memigrasikan konsumen Anda dari KPL 0.x ke KPL 1.x. KPL 1.x memperkenalkan dukungan untuk AWS SDK untuk Java 2.x sambil mempertahankan kompatibilitas antarmuka dengan versi sebelumnya. Anda tidak perlu memperbarui logika pemrosesan data inti Anda untuk bermigrasi ke KPL 1.x.

1. Pastikan Anda memiliki prasyarat berikut:

   - Java Development Kit (JDK) 8 atau lebih baru
   - AWS SDK untuk Java 2.x
   - Maven atau Gradle untuk manajemen ketergantungan

2. Tambahkan dependensi

   Jika Anda menggunakan Maven, tambahkan dependensi berikut ke file pom.xml Anda. Pastikan Anda memperbarui GroupId dari `com.amazonaws` ke `software.amazon.kinesis` dan versi `1.x.x` ke versi KPL terbaru.

   ```
   <dependency>
       <groupId>software.amazon.kinesis</groupId>
       <artifactId>amazon-kinesis-producer</artifactId>
       <version>1.x.x</version> <!-- Use the latest version -->
   </dependency>
   ```

   Jika Anda menggunakan Gradle, tambahkan berikut ini ke `build.gradle` file Anda. Pastikan untuk mengganti `1.x.x` dengan versi KPL terbaru.

   ```
   implementation 'software.amazon.kinesis:amazon-kinesis-producer:1.x.x'
   ```

   Anda dapat memeriksa versi terbaru KPL di Repositori Pusat Maven.

3. Perbarui pernyataan impor untuk KPL

   KPL 1.x menggunakan AWS SDK untuk Java 2.x dan menggunakan nama paket yang diperbarui yang dimulai dengan`software.amazon.kinesis`, dibandingkan dengan nama paket di KPL sebelumnya yang dimulai dengan. `com.amazonaws.services.kinesis`

Ganti impor untuk `com.amazonaws.services.kinesis` dengan`software.amazon.kinesis`. Tabel berikut mencantumkan impor yang harus Anda ganti.

Penggantian impor

Ganti:	Dengan:
impor com.amazonaws.services.kinesis.producer.attempt;	impor software.amazon.kinesis.producer.Attry;
impor com.amazonaws.services.kinesis.producer. BinaryToHexConverter;	impor software.amazon.kinesis.producer. BinaryToHexConverter;
impor com.amazonaws.services.kinesis.producer. CertificateExtractor;	impor software.amazon.kinesis.producer. CertificateExtractor;
impor com.amazonaws.services.kinesis.producer.daemon;	impor perangkat lunak.amazon.kinesis.producer.daemon;
impor com.amazonaws.services.kinesis.producer. DaemonException;	impor software.amazon.kinesis.producer. DaemonException;
impor com.amazonaws.services.kinesis.producer. FileAgeManager;	impor software.amazon.kinesis.producer. FileAgeManager;
impor com.amazonaws.services.kinesis.producer. FutureTimedOutException;	impor software.amazon.kinesis.producer. FutureTimedOutException;
impor com.amazonaws.services.kinesis.producer. GlueSchemaRegistrySerializerInstance;	impor software.amazon.kinesis.producer. GlueSchemaRegistrySerializerInstance;
impor com.amazonaws.services.kinesis.producer. HashedFileCopier;	impor software.amazon.kinesis.producer. HashedFileCopier;
impor com.amazonaws.services.kinesis.producer. IKinesisProduser;	impor software.amazon.kinesis.producer. IKinesisProduser;

Ganti:	Dengan:
impor com.amazonaws.services.kinesis.producer. IrrecoverableError;	impor software.amazon.kinesis.producer. IrrecoverableError;
impor com.amazonaws.services.kinesis.producer. KinesisProducer;	impor software.amazon.kinesis.producer. KinesisProducer;
impor com.amazonaws.services.kinesis.producer. KinesisProducerConfiguration;	impor software.amazon.kinesis.producer. KinesisProducerConfiguration;
impor com.amazonaws.services.kinesis.producer. LogInputStreamReader;	impor software.amazon.kinesis.producer. LogInputStreamReader;
impor com.amazonaws.services.kinesis.producer.metric;	impor software.amazon.kinesis.producer.metric;
impor com.amazonaws.services.kinesis.producer. ProcessFailureBehavior;	impor software.amazon.kinesis.producer. ProcessFailureBehavior;
impor com.amazonaws.services.kinesis.producer. UnexpectedMessageException;	impor software.amazon.kinesis.producer. UnexpectedMessageException;
impor com.amazonaws.services.kinesis.producer. UserRecord;	impor software.amazon.kinesis.producer. UserRecord;
impor com.amazonaws.services.kinesis.producer. UserRecordFailedException;	impor software.amazon.kinesis.producer. UserRecordFailedException;
impor com.amazonaws.services.kinesis.producer. UserRecordResult;	impor software.amazon.kinesis.producer. UserRecordResult;
impor com.amazonaws.services.kinesis.producer.protobuf.messages;	impor software.amazon.kinesis.producer.protobuf.messages;
impor com.amazonaws.services.kinesis.producer.protobuf.config;	impor software.amazon.kinesis.producer.protobuf.config;

4. Perbarui pernyataan impor untuk kelas AWS penyedia kredensional

Saat bermigrasi ke KPL 1.x, Anda harus memperbarui paket dan kelas dalam impor Anda dalam kode aplikasi KPL Anda yang didasarkan pada AWS SDK untuk Java 1.x ke yang sesuai berdasarkan 2.x. AWS SDK untuk Java Impor umum dalam aplikasi KPL adalah kelas penyedia kredensial. Lihat Perubahan penyedia kredensial dalam dokumentasi panduan migrasi AWS SDK untuk Java 2.x untuk mengetahui daftar lengkap perubahan penyedia kredensial. Berikut adalah perubahan impor umum yang mungkin perlu Anda lakukan di aplikasi KPL Anda.

Impor di KPL 0.x

```
import com.amazonaws.auth.DefaultAWSCredentialsProviderChain;
```

Impor di KPL 1.x

```
import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
```

Jika Anda mengimpor penyedia kredensi lain berdasarkan AWS SDK untuk Java 1.x, Anda harus memperbaruinya ke yang setara AWS SDK untuk Java 2.x. Jika Anda tidak mengimpor apa pun classes/packages dari AWS SDK untuk Java 1.x, Anda dapat mengabaikan langkah ini.

5. Perbarui konfigurasi penyedia kredensional dalam konfigurasi KPL

   Konfigurasi penyedia kredensional di KPL 1.x memerlukan penyedia kredensi 2.x. AWS SDK untuk Java Jika Anda meneruskan penyedia kredensi untuk AWS SDK untuk Java 1.x di `KinesisProducerConfiguration` dengan mengganti penyedia kredensil default, Anda harus memperbaruinya dengan penyedia kredensi 2.x. AWS SDK untuk Java Lihat Perubahan penyedia kredensial dalam dokumentasi panduan migrasi AWS SDK untuk Java 2.x untuk mengetahui daftar lengkap perubahan penyedia kredensial. Jika Anda tidak mengganti penyedia kredensi default dalam konfigurasi KPL, Anda dapat mengabaikan langkah ini.

   Misalnya, jika Anda mengganti penyedia kredensi default untuk KPL dengan kode berikut:

   ```
   KinesisProducerConfiguration config = new KinesisProducerConfiguration();
   // SDK v1 default credentials provider
   config.setCredentialsProvider(new DefaultAWSCredentialsProviderChain());
   ```

   Anda harus memperbaruinya dengan kode berikut untuk menggunakan penyedia kredensi AWS SDK untuk Java 2.x:

```
KinesisProducerConfiguration config = new KinesisProducerConfiguration();
// New SDK v2 default credentials provider
config.setCredentialsProvider(DefaultCredentialsProvider.create());
```

Transisi ke sertifikat Amazon Trust Services (ATS) untuk KPL

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 Amazon 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, lihat Cara AWS Mempersiapkan Pindah ke Otoritas Sertifikat Sendiri.

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

Platform yang didukung KPL

Amazon Kinesis Producer Library (KPL) ditulis dalam C++ dan berjalan sebagai proses turunan 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 iOS 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 Produsen Amazon 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 Amazon Kinesis Producer Library (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-east-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:

Integrasikan KPL dengan kode produsen

Amazon 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 mencantumkan 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 Java	*	*	*	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 Java	Menggunakan SDK	N/A	Ya	Harus menonakti

Bahasa sampingan produser	Bahasa sisi konsumen	Versi KCL	Logika titik pemeriksaan	Bisakah Anda menggunakan KPL?	Peringatan
		secara langsung			fkan agregasi.
Java	Java	1.3.x	N/A	Ya	Harus menonaktifkan agregasi.
Java	Java	1.4.x	Panggilan pos pemeriksaan tanpa argumen	Ya	Tidak ada
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 Java	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 paling dasar ke kode asinkron sepenuhnya.

Kode produsen Barebones

Kode berikut adalah semua yang diperlukan untuk menulis produsen kerja minimal. Catatan pengguna Amazon 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 runtime. Jika Anda tidak ingin memblokir runtime, 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);
}
```

Konfigurasikan Perpustakaan Produsen Amazon 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. Contoh:

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

Untuk informasi selengkapnya tentang aturan penggunaan parameter konfigurasi dan batas nilai, lihat contoh berkas 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.

Menerapkan 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.

Opsi

- Migrasi dari versi KCL sebelumnya
- Gunakan ekstensi KCL untuk de-agregasi KPL
- Gunakan GetRecords secara langsung

Migrasi dari versi KCL sebelumnya

Anda tidak diharuskan mengubah panggilan yang ada untuk melakukan checkpointing 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, dijelaskan berikut.

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 harus 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.

Gunakan ekstensi KCL untuk de-agregasi KPL

De-agregasi KPL dapat melibatkan checkpointing berikutnya. Untuk memfasilitasi penggunaan checkpointing berikutnya, 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` `UserRecord`Kelas 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 ke`IRecordProcessorCheckpointer`:

```
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 menyimpan`subSequenceNumber`, 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`. 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.

Gunakan GetRecords secara 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.

Gunakan KPL dengan Amazon Data 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 Lambda Untuk informasi selengkapnya, lihat Menulis ke Amazon Firehose Menggunakan Kinesis Data Streams.

Gunakan KPL dengan AWS Glue Schema Registry

Anda dapat mengintegrasikan aliran data Kinesis Anda dengan Registri Skema. AWS Glue Registri AWS 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 AWS Glue Skema 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, Kinesis Data AWS Glue Streams dan integrasi registri skema 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 Pustaka” di Kasus KPL/KCL Penggunaan: Mengintegrasikan Amazon Kinesis Data Streams dengan Glue Schema Registry. AWS

Konfigurasikan 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);
```

Kebijakan siklus hidup versi KPL

Topik ini menguraikan kebijakan siklus hidup versi untuk Amazon Kinesis Producer Library (KPL). AWS secara teratur menyediakan rilis baru untuk versi KPL untuk mendukung fitur dan penyempurnaan baru, perbaikan bug, patch keamanan, dan pembaruan ketergantungan. Kami menyarankan Anda tetap up-to-date menggunakan versi KPL untuk mengikuti fitur-fitur terbaru, pembaruan keamanan, dan dependensi yang mendasarinya. Kami tidak menyarankan untuk terus menggunakan versi KPL yang tidak didukung.

Siklus hidup untuk versi KPL utama terdiri dari tiga fase berikut:

- Ketersediaan umum (GA) — Selama fase ini, versi utama didukung sepenuhnya. AWS menyediakan rilis versi minor dan patch reguler yang mencakup dukungan untuk fitur baru atau pembaruan API untuk Kinesis Data Streams, serta perbaikan bug dan keamanan.
- Mode pemeliharaan — AWS membatasi rilis versi patch untuk mengatasi perbaikan bug kritis dan masalah keamanan saja. Versi utama tidak akan menerima pembaruan untuk fitur baru atau APIs Kinesis Data Streams.
- E nd-of-support — Versi utama tidak akan lagi menerima pembaruan atau rilis. Rilis yang diterbitkan sebelumnya akan terus tersedia melalui manajer paket publik dan kode akan tetap aktif GitHub. Penggunaan versi yang telah end-of-support dicapai dilakukan atas kebijaksanaan pengguna. Kami menyarankan Anda meningkatkan ke versi utama terbaru.

Versi utama	Fase saat ini	Tanggal rilis	Tanggal mode pemeliharaan	End-of-support tanggal
KPL 0.x	Mode pemeliharaan	2015-06-02	2025-04-17	2026-01-30
KPL 1.x	Ketersediaan umum	2024-12-15	--	--

Kembangkan produsen menggunakan API Amazon Kinesis Data Streams dengan AWS SDK untuk Java

Anda dapat mengembangkan produsen menggunakan Amazon Kinesis Data Streams API AWS dengan SDK for Java. Jika Anda baru mengenal Kinesis Data Streams, mulailah dengan menjadi akrab dengan konsep dan terminologi Apa itu Amazon Kinesis Data Streams? yang disajikan dalam dan. Gunakan AWS CLI untuk menjalankan operasi Amazon Kinesis Data Streams

Contoh-contoh ini membahas Kinesis Data Streams API dan AWS menggunakan SDK for Java untuk menambahkan (menempatkan) data ke stream. Namun, untuk sebagian besar kasus penggunaan, Anda harus memilih pustaka KPL Kinesis Data Streams. Untuk informasi selengkapnya, lihat Kembangkan produsen menggunakan Amazon Kinesis Producer Library (KPL).

Contoh kode Java dalam chapter ini menunjukkan bagaimana melakukan operasi API Kinesis Data Streams dasar, dan dibagi secara logis berdasarkan 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. Selain itu, Anda dapat memanggil Kinesis Data Streams API menggunakan bahasa pemrograman lainnya. Untuk informasi selengkapnya tentang semua yang tersedia AWS SDKs, lihat Mulai Mengembangkan dengan Amazon Web Services.

Setiap tugas memiliki prasyarat; misalnya, Anda tidak dapat menambahkan data ke aliran sampai Anda membuat aliran, yang mengharuskan Anda membuat klien. Untuk informasi selengkapnya, lihat Membuat dan mengelola aliran data Kinesis.

Topik

- Tambahkan data ke aliran
- Berinteraksi dengan data menggunakan AWS Glue Schema Registry

Tambahkan data ke aliran

Setelah aliran 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 terkait.

Ada dua operasi berbeda di Kinesis Data Streams API yang menambahkan data ke aliran, dan. PutRecordsPutRecord PutRecordsOperasi mengirimkan beberapa catatan ke aliran Anda per permintaan HTTP, dan PutRecord operasi tunggal mengirimkan catatan ke aliran Anda satu per satu (permintaan HTTP terpisah diperlukan untuk setiap rekaman). Anda sebaiknya lebih suka 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

- Tambahkan beberapa catatan dengan PutRecords
- Tambahkan satu catatan dengan PutRecord

Selalu ingat bahwa, karena aplikasi sumber Anda menambahkan data ke aliran 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, lihat. Dapatkan data dari aliran

> **Important**
>
> Ubah periode retensi data

Tambahkan beberapa catatan dengan PutRecords

PutRecordsOperasi 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. 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. Seperti PutRecord operasi tunggal yang dijelaskan di bawah ini, PutRecords menggunakan nomor urut dan kunci partisi. Namun, PutRecord parameter SequenceNumberForOrdering tidak termasuk dalam PutRecords panggilan. PutRecordsOperasi mencoba untuk memproses semua catatan dalam urutan alami permintaan.

Setiap catatan data memiliki nomor urut yang unik. Nomor urutan ditetapkan oleh Kinesis Data Streams setelah client.putRecords Anda memanggil untuk menambahkan catatan data ke

aliran. Nomor urutan untuk kunci partisi yang sama umumnya meningkat dari waktu ke waktu; semakin lama periode waktu antara `PutRecords` permintaan, 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.

`PutRecords`Permintaan dapat menyertakan 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 untuk streaming 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 daripada jumlah pecahan untuk mengurangi latensi dan memaksimalkan throughput.

PutRecords contoh

Kode berikut membuat 100 catatan data dengan kunci partisi berurutan dan menempatkannya 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);
```

`PutRecords`Responsnya mencakup serangkaian respons`Records`. Setiap catatan dalam larik respons berkorelasi langsung dengan catatan dalam array permintaan menggunakan urutan alami, dari atas ke bawah permintaan dan respons. `Records`Array respons selalu menyertakan jumlah catatan yang sama dengan array permintaan.

Menangani kegagalan saat menggunakan PutRecords

Secara default, kegagalan catatan individu dalam permintaan tidak menghentikan pemrosesan catatan berikutnya dalam `PutRecords` permintaan. Ini berarti bahwa `Records` array respons mencakup catatan yang berhasil dan tidak berhasil diproses. Anda harus mendeteksi catatan yang tidak berhasil diproses dan memasukkannya ke dalam panggilan berikutnya.

Catatan yang berhasil termasuk `SequenceNumber` dan `ShardID` nilai, dan catatan yang gagal termasuk `ErrorCode` dan `ErrorMessage` nilai. `ErrorCode`Parameter mencerminkan jenis kesalahan dan dapat menjadi salah satu dari nilai berikut: `ProvisionedThroughputExceededException` atau`InternalFailure`. `ErrorMessage`memberikan informasi lebih rinci tentang `ProvisionedThroughputExceededException` pengecualian termasuk ID akun, nama aliran, dan ID pecahan dari catatan yang dibatasi. Contoh di bawah ini memiliki tiga catatan dalam satu `PutRecords` permintaan. Catatan kedua gagal dan tercermin dalam respons.

Example PutRecords Permintaan Sintaks

```
{
    "Records": [
        {
     "Data": "XzxkYXRhPl8w",
     "PartitionKey": "partitionKey1"
        },
        {
     "Data": "AbceddeRFfg12asd",
     "PartitionKey": "partitionKey1"
        },
```

```
        {
     "Data": "KFpcd98*7nd1",
     "PartitionKey": "partitionKey3"
        }
    ],
    "StreamName": "myStream"
}
```

Example PutRecords Sintaks Respon

```
{
    "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 dimasukkan dalam `PutRecords` permintaan berikutnya. Pertama, periksa parameter `FailedRecordCount` dalam `putRecordsResult` untuk mengonfirmasi apakah ada catatan kegagalan dalam permintaan. Jika demikian, masing-masing `putRecordsEntry` yang memiliki `ErrorCode` yang tidak `null` harus ditambahkan ke permintaan berikutnya. Untuk contoh jenis handler ini, lihat kode berikut.

Example PutRecords penangan 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);
}
```

Tambahkan satu catatan dengan PutRecord

Setiap panggilan untuk `PutRecord`beroperasi pada satu catatan. Lebih suka `PutRecords` operasi yang dijelaskan Tambahkan beberapa catatan dengan PutRecords kecuali aplikasi Anda secara khusus perlu selalu mengirim catatan tunggal per permintaan, atau alasan lain tidak `PutRecords` dapat digunakan.

Setiap catatan data memiliki nomor urut yang unik. Nomor urutan ditetapkan oleh Kinesis Data Streams setelah `client.putRecord` Anda memanggil untuk menambahkan catatan data ke aliran. Nomor urutan untuk kunci partisi yang sama umumnya meningkat dari waktu ke waktu; semakin lama periode waktu antara `PutRecord` permintaan, semakin besar nomor urut menjadi.

Ketika put terjadi secara berurutan, nomor urut yang dikembalikan tidak dijamin akan meningkat karena operasi put pada dasarnya muncul bersamaan dengan Kinesis Data Streams. Untuk

menjamin peningkatan nomor urut secara ketat untuk kunci partisi yang sama, gunakan `SequenceNumberForOrdering` parameter, seperti yang ditunjukkan pada sampel PutRecord contoh kode.

Apakah Anda menggunakan atau tidak`SequenceNumberForOrdering`, catatan yang diterima Kinesis Data Streams `GetRecords` melalui panggilan diurutkan secara ketat berdasarkan nomor urut.

> **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.

Kunci partisi digunakan untuk mengelompokkan data dalam aliran. Catatan data ditugaskan ke pecahan dalam aliran berdasarkan kunci partisi. Secara khusus, Kinesis Data Streams menggunakan kunci partisi sebagai input ke fungsi hash yang memetakan kunci partisi (dan data terkait) ke pecahan tertentu.

Sebagai hasil dari mekanisme hashing ini, semua data mencatat dengan kunci partisi yang sama memetakan 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 digunakan dengan baik, jumlah pecahan (ditentukan oleh `setShardCount` metode`CreateStreamRequest`) harus jauh lebih sedikit daripada jumlah kunci partisi unik, dan jumlah data yang mengalir ke kunci partisi tunggal harus jauh lebih kecil dari kapasitas pecahan.

PutRecord contoh

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

```
for (int j = 0; j < 10; j++)
{
  PutRecordRequest putRecordRequest = new PutRecordRequest();
  putRecordRequest.setStreamName( myStreamName );
  putRecordRequest.setData(ByteBuffer.wrap( String.format( "testData-%d",
 j ).getBytes() ));
  putRecordRequest.setPartitionKey( String.format( "partitionKey-%d", j/5 ));
```

```
  putRecordRequest.setSequenceNumberForOrdering( sequenceNumberOfPreviousRecord );
  PutRecordResult putRecordResult = client.putRecord( putRecordRequest );
  sequenceNumberOfPreviousRecord = putRecordResult.getSequenceNumber();
}
```

Contoh kode sebelumnya digunakan `setSequenceNumberForOrdering` untuk menjamin peningkatan pemesanan secara ketat dalam setiap kunci partisi. Untuk menggunakan parameter ini secara efektif, atur catatan saat ini (record n) ke nomor urut catatan sebelumnya (record n-1). `SequenceNumberForOrdering` Untuk mendapatkan nomor urut rekaman yang telah ditambahkan ke aliran, panggil `getSequenceNumber` hasil dari`putRecord`.

`SequenceNumberForOrdering`Parameter memastikan secara ketat meningkatkan nomor urut untuk kunci partisi yang sama. `SequenceNumberForOrdering`tidak menyediakan urutan catatan di beberapa kunci partisi.

Berinteraksi dengan data menggunakan AWS Glue Schema Registry

Anda dapat mengintegrasikan aliran data Kinesis Anda dengan Registri Skema. AWS Glue Registri AWS 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 AWS Glue Skema memungkinkan Anda 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 `PutRecords` dan `PutRecord` Kinesis APIs Data Streams yang tersedia AWS di Java SDK.

Untuk petunjuk terperinci tentang cara mengatur integrasi Kinesis Data Streams dengan registri skema menggunakan dan Kinesis Data Streams, lihat bagian “Berinteraksi dengan Data PutRecords Menggunakan PutRecord Aliran Data APIs Kinesis” di Kasus Penggunaan: Mengintegrasikan Amazon Kinesis Data APIs Streams dengan Registri Skema Glue. AWS

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 Tentukan 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

- Lengkapi prasyarat untuk Agen Kinesis
- Unduh dan instal agen
- Konfigurasikan dan mulai agen
- Tentukan pengaturan konfigurasi agen
- Pantau beberapa direktori file dan tulis ke beberapa aliran
- Gunakan agen untuk melakukan pra-proses data
- Gunakan perintah CLI agen
- Pertanyaan yang Sering Diajukan

Lengkapi prasyarat untuk Agen Kinesis

- 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, lihatPantau kesehatan Agen Aliran Data Kinesis dengan Amazon CloudWatch, dan Kontrol CloudWatch Akses.

Unduh dan instal agen

Pertama-tama, hubungkan ke instans Anda. Untuk informasi selengkapnya, lihat Connect to Your Instance di Panduan Pengguna Amazon EC2. Jika Anda mengalami masalah saat menyambung, lihat Pemecahan Masalah Menyambung ke Instans Anda di Panduan Pengguna Amazon EC2.

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"]
```

Konfigurasikan dan mulai 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`.

Tentukan 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.

Pengaturan Konfigurasi	Deskripsi
`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
`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 melakukan pra-proses 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` .

Pengaturan Konfigurasi	Deskripsi
	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) 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.

Pengaturan Konfigurasi	Deskripsi
`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`
`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)

Pantau beberapa direktori file dan tulis 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 melakukan pra-proses data

Agen dapat melakukan pra-proses rekaman 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 RFC3164 Syslog. 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"}
```

Gunakan perintah CLI agen

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 meningkat? and/or **`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

AWS Layanan berikut dapat diintegrasikan langsung dengan Amazon Kinesis Data Streams untuk menulis data ke aliran data Kinesis. Tinjau informasi untuk setiap layanan yang Anda minati dan lihat referensi yang disediakan.

Topik

- Menulis ke Kinesis Data Streams menggunakan AWS Amplify
- Menulis ke Kinesis Data Streams menggunakan Amazon Aurora
- Menulis ke Kinesis Data Streams menggunakan Amazon CloudFront
- Menulis ke Kinesis Data Streams CloudWatch menggunakan Amazon Logs
- Menulis ke Kinesis Data Streams menggunakan Amazon Connect
- Menulis ke Kinesis Data Streams menggunakan AWS Database Migration Service
- Menulis ke Kinesis Data Streams menggunakan Amazon DynamoDB
- Menulis ke Kinesis Data Streams menggunakan Amazon EventBridge
- Menulis ke Kinesis Data Streams menggunakan AWS IoT Core
- Menulis ke Kinesis Data Streams menggunakan Amazon Relational Database Service
- Menulis ke Kinesis Data Streams menggunakanAmazon Pinpoint
- Menulis ke Kinesis Data Streams menggunakan Amazon Quantum Ledger Database (Amazon QLDB)

Menulis ke Kinesis Data Streams menggunakan AWS Amplify

Anda dapat menggunakan Amazon Kinesis Data Streams untuk mengalirkan data 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 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 AWS Amplify.

Menulis ke Kinesis Data Streams menggunakan 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.

Menulis ke Kinesis Data Streams menggunakan 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 Data Firehose untuk mengirim data log ke Amazon S3, Amazon Redshift, Amazon Service, atau layanan pemrosesan log pihak OpenSearch ketiga.

Untuk informasi selengkapnya, lihat Log waktu nyata di Panduan CloudFront Pengembang Amazon.

Menulis ke Kinesis Data Streams CloudWatch menggunakan Amazon Logs

Anda dapat menggunakan CloudWatch langganan untuk mendapatkan akses ke umpan real-time peristiwa log dari Amazon CloudWatch Logs dan mengirimkannya ke aliran data Kinesis 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.

Menulis ke Kinesis Data Streams menggunakan 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 CRMs 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.

Menulis ke Kinesis Data Streams menggunakan AWS Database Migration Service

Anda dapat menggunakan AWS Database Migration Service untuk memigrasikan data ke aliran data Kinesis. Anda dapat membangun aplikasi konsumen yang memproses catatan data secara real time. Anda juga dapat dengan mudah mengirim data 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 Service di Panduan Pengguna.

Menulis ke Kinesis Data Streams menggunakan Amazon DynamoDB

Gunakan Amazon Kinesis Data Streams untuk menangkap perubahan pada Amazon DynamoDB. Kinesis Data Streams menangkap modifikasi tingkat item di tabel DynamoDB apa pun dan mereplikasikannya 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.

Menulis ke Kinesis Data Streams menggunakan 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.

Menulis ke Kinesis Data Streams menggunakan 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 AWS Pengembang Inti IoT.

Menulis ke Kinesis Data Streams menggunakan Amazon Relational Database Service

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 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 Data Firehose untuk menyimpan data.

Untuk informasi selengkapnya, lihat Aliran Aktivitas Database di Panduan Pengembang Amazon RDS.

Menulis ke Kinesis Data Streams menggunakanAmazon 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.

Menulis ke Kinesis Data Streams menggunakan Amazon Quantum Ledger Database (Amazon QLDB)

Anda dapat membuat aliran di Amazon 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 aliran 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.

Menulis ke Kinesis Data Streams menggunakan integrasi pihak ketiga

Anda dapat menulis data ke Kinesis Data Streams menggunakan salah satu opsi pihak ketiga berikut yang terintegrasi dengan Kinesis Data Streams. Pilih opsi yang ingin Anda pelajari lebih lanjut dan temukan sumber daya serta tautan ke dokumentasi yang relevan.

Topik

- Apache Flink
- Lancar
- Debezium
- Oracle GoldenGate
- Kafka Connect
- Pengalaman Adobe
- Striim

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 menulis ke Kinesis Data Streams dari Apache Flink, lihat Konektor Amazon Kinesis Data Streams.

Lancar

Fluentd adalah pengumpul data open source untuk lapisan logging terpadu. Untuk informasi lebih lanjut tentang menulis ke Kinesis Data Streams dari Fluentd. Untuk informasi lebih lanjut, 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, memfilter, dan mengubah data dari satu database ke database lain. Untuk informasi lebih lanjut tentang menulis ke Kinesis Data Streams GoldenGate dari Oracle, lihat Replikasi data ke Kinesis Data Stream menggunakan Oracle. GoldenGate

Kafka Connect

Kafka Connect adalah alat untuk streaming data yang terukur dan andal antara Apache Kafka dan sistem lainnya. Untuk informasi lebih lanjut tentang penulisan data dari Apache Kafka ke Kinesis Data Streams, lihat konektor Kinesis kafka.

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 penulisan data dari Adobe Experience Platform ke Kinesis Data Stream. lihat cara membuat sambungan 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 Penulis Kinesis.

Memecahkan masalah produsen Amazon Kinesis Data Streams

Topik berikut menawarkan solusi untuk masalah umum dengan produsen Amazon Kinesis Data Streams:

- Aplikasi produser saya menulis pada tingkat yang lebih lambat dari yang diharapkan
- Saya menerima kesalahan izin kunci master KMS yang tidak sah
- Memecahkan masalah umum lainnya untuk produsen

Aplikasi produser saya menulis pada tingkat yang lebih lambat dari yang diharapkan

Alasan paling umum untuk throughput tulis menjadi lebih lambat dari yang diharapkan adalah:

- Batas layanan terlampaui
- Saya ingin mengoptimalkan produser saya
- Penyalahgunaan operasi flushSync()

Batas layanan terlampaui

Untuk mengetahui apakah batas layanan terlampaui, periksa apakah produsen Anda mengeluarkan pengecualian throughput dari layanan, dan validasi operasi API apa yang sedang dibatasi. Perlu diingat bahwa ada batasan berbeda berdasarkan panggilan, lihatKuota dan batas. Misalnya, selain batas tingkat shard untuk menulis dan membaca yang paling umum dikenal, ada batas tingkat aliran berikut:

- CreateStream
- DeleteStream
- ListStreams
- GetShardIterator
- MergeShards
- DescribeStream
- DescribeStreamSummary

Operasi`CreateStream`,,`DeleteStream`,`ListStreams`,`GetShardIterator`, dan `MergeShards` dibatasi hingga 5 panggilan per detik. `DescribeStream`Operasi dibatasi hingga 10 panggilan per detik. `DescribeStreamSummary`Operasi dibatasi hingga 20 panggilan per detik.

Jika panggilan ini bukan masalahnya, pastikan Anda telah memilih kunci partisi yang memungkinkan Anda mendistribusikan operasi put secara merata di semua pecahan, dan Anda tidak memiliki kunci partisi tertentu yang menabrak batas layanan saat sisanya tidak. Ini mengharuskan Anda mengukur throughput puncak dan memperhitungkan jumlah pecahan dalam aliran Anda. Untuk informasi selengkapnya tentang mengelola aliran, lihatMembuat dan mengelola aliran data Kinesis.

> **Tip**
>
> Ingatlah untuk membulatkan ke kilobyte terdekat untuk perhitungan throttling throughput saat menggunakan operasi rekam tunggal PutRecord, sedangkan operasi multi-rekaman PutRecordsberputar pada jumlah kumulatif catatan di setiap panggilan. Misalnya, `PutRecords` permintaan dengan 600 catatan yang berukuran 1,1 KB tidak akan dibatasi.

Saya ingin mengoptimalkan produser saya

Sebelum Anda mulai mengoptimalkan produser Anda, selesaikan tugas-tugas utama berikut. Pertama, identifikasi throughput puncak yang Anda inginkan dalam hal ukuran rekaman dan catatan per detik. Selanjutnya, singkirkan kapasitas aliran sebagai faktor pembatas (Batas layanan terlampaui). Jika Anda telah mengesampingkan kapasitas streaming, gunakan tips pemecahan masalah berikut dan pedoman pengoptimalan untuk dua jenis produsen umum.

Produser Besar

Produsen besar biasanya berjalan dari server lokal atau instans Amazon EC2. Pelanggan yang membutuhkan throughput lebih tinggi dari produsen besar biasanya peduli dengan latensi per rekaman. Strategi untuk menangani latensi meliputi hal berikut: Jika pelanggan dapat melakukan micro-batch/buffer record, gunakan Amazon Kinesis Producer Library (yang memiliki logika agregasi lanjutan), operasi multi-record PutRecords, atau agregat record ke dalam file yang lebih besar sebelum menggunakan operasi rekaman tunggal. PutRecord Jika Anda tidak dapat melakukan batch atau buffer, gunakan beberapa thread untuk menulis ke layanan Kinesis Data Streams secara bersamaan. Yang AWS SDK untuk Java dan lainnya SDKs termasuk klien async yang dapat melakukan ini dengan kode yang sangat sedikit.

Produser Kecil

Produsen kecil biasanya adalah aplikasi seluler, perangkat IoT, atau klien web. Jika ini adalah aplikasi seluler, kami sarankan untuk menggunakan `PutRecords` operasi atau Perekam Kinesis di Ponsel. AWS SDKs Untuk informasi selengkapnya, lihat Panduan AWS Mobile SDK untuk Android Memulai dan Panduan AWS Mobile SDK for iOS Memulai. Aplikasi seluler harus menangani koneksi intermiten secara inheren dan memerlukan semacam batch put, seperti. `PutRecords` Jika Anda tidak dapat melakukan batch karena alasan tertentu, lihat informasi Produsen Besar di atas. Jika produsen Anda adalah browser, jumlah data yang dihasilkan biasanya sangat kecil. Namun, Anda menempatkan operasi put di jalur kritis aplikasi, yang tidak kami rekomendasikan.

Penyalahgunaan operasi `flushSync()`

Menggunakan `flushSync()` secara tidak benar dapat secara signifikan memengaruhi kinerja penulisan. `flushSync()`Operasi ini dirancang untuk skenario shutdown untuk memastikan bahwa semua catatan buffer dikirim sebelum aplikasi KPL berakhir. Jika Anda menerapkan operasi ini setelah setiap operasi penulisan, ini dapat menambahkan latensi ekstra yang substansif, sekitar 500 ms per penulisan. Pastikan bahwa Anda telah menerapkan `flushSync()` hanya untuk shutdown aplikasi untuk menghindari penundaan ekstra yang tidak perlu dalam kinerja tulis.

Saya menerima kesalahan izin kunci master KMS yang tidak sah

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, lihat Menggunakan Kebijakan Utama di KMS dan Menggunakan Kebijakan IAM dengan AWS KMS. AWS

Memecahkan masalah umum lainnya untuk produsen

- Mengapa aliran data Kinesis saya mengembalikan Kesalahan Server Internal 500?
- Bagaimana cara mengatasi kesalahan batas waktu saat menulis dari Flink ke Kinesis Data Streams?
- Bagaimana cara memecahkan masalah kesalahan pelambatan di Kinesis Data Streams?
- Mengapa aliran data Kinesis saya melambat?
- Bagaimana saya bisa memasukkan catatan data ke dalam aliran data Kinesis menggunakan KPL?

Optimalkan produsen Kinesis Data Streams

Anda dapat mengoptimalkan lebih lanjut produsen Amazon Kinesis Data Streams tergantung pada perilaku tertentu yang Anda lihat. Tinjau topik berikut untuk mengidentifikasi solusi.

Topik

- Kustomisasi percobaan ulang KPL dan perilaku batas nilai
- Menerapkan praktik terbaik untuk agregasi KPL

Kustomisasi percobaan ulang KPL dan perilaku batas nilai

Saat Anda menambahkan data pengguna Amazon Kinesis Producer Library (KPL) menggunakan `addUserRecord()` operasi KPL, catatan akan diberikan stempel waktu dan ditambahkan ke buffer dengan tenggat waktu yang ditetapkan oleh parameter konfigurasi. `RecordMaxBufferedTime` stamp/deadline Kombinasi kali ini menetapkan prioritas buffer. Rekaman dikeluarkan dari buffer berdasarkan kriteria berikut:

- Prioritas penyangga
- Konfigurasi agregasi
- Konfigurasi koleksi

Parameter konfigurasi agregasi dan koleksi yang memengaruhi perilaku buffer adalah sebagai berikut:

- `AggregationMaxCount`
- `AggregationMaxSize`
- `CollectionMaxCount`
- `CollectionMaxSize`

Rekaman yang di-flush kemudian dikirim ke aliran data Kinesis Anda saat Amazon Kinesis Data Streams merekam menggunakan panggilan ke operasi API Kinesis Data Streams. `PutRecords` `PutRecords`Operasi mengirimkan permintaan ke aliran Anda yang terkadang menunjukkan kegagalan penuh atau sebagian. Catatan yang gagal secara otomatis ditambahkan kembali ke buffer KPL. Batas waktu baru ditetapkan berdasarkan minimum dari dua nilai ini:

- Setengah dari `RecordMaxBufferedTime` konfigurasi saat ini
- time-to-liveNilai catatan

Strategi ini memungkinkan data pengguna KPL yang dicoba ulang untuk dimasukkan dalam panggilan API Kinesis Data Streams berikutnya, untuk meningkatkan throughput dan mengurangi kompleksitas sambil menerapkan nilai rekaman Kinesis Data Streams. time-to-live Tidak ada algoritma backoff, menjadikannya strategi coba lagi yang relatif agresif. Spamming karena percobaan ulang yang berlebihan dicegah dengan pembatasan tarif, dibahas di bagian selanjutnya.

Pembatasan tarif

KPL mencakup fitur pembatas laju, yang membatasi throughput per shard yang dikirim dari satu produsen. Pembatasan laju diimplementasikan menggunakan algoritma token bucket dengan bucket terpisah untuk catatan Kinesis Data Streams dan byte. Setiap penulisan yang berhasil 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 spam karena percobaan ulang yang berlebihan. Namun, praktik terbaik adalah bagi setiap produsen untuk mencoba lagi untuk throughput maksimum secara agresif dan untuk menangani setiap pelambatan yang dihasilkan ditentukan sebagai berlebihan dengan memperluas kapasitas aliran dan menerapkan strategi kunci partisi yang sesuai.

Menerapkan praktik terbaik untuk agregasi KPL

Meskipun skema nomor urut dari catatan Amazon Kinesis Data Streams yang dihasilkan tetap sama, agregasi menyebabkan pengindeksan data pengguna Amazon Kinesis Producer Library (KPL) yang terkandung dalam rekaman Aliran Data Kinesis agregat dimulai dari 0 (nol); namun, selama Anda tidak bergantung pada nomor urut untuk mengidentifikasi catatan pengguna KPL Anda secara unik, catatan pengguna KPL Anda kode dapat mengabaikan ini, sebagai agregasi (catatan pengguna KPL Anda ke dalam catatan Kinesis Data Streams) dan de-agregasi berikutnya (dari catatan Kinesis Data Streams ke dalam catatan pengguna KPL Anda) secara otomatis mengurus ini untuk Anda. Ini berlaku apakah konsumen Anda menggunakan KCL atau AWS SDK. Untuk menggunakan fungsionalitas agregasi ini, Anda harus menarik bagian Java dari KPL ke dalam build jika konsumen Anda ditulis menggunakan API yang disediakan di SDK. AWS

Jika Anda bermaksud menggunakan nomor urut sebagai pengidentifikasi unik untuk catatan pengguna KPL Anda, kami sarankan Anda menggunakan patuh kontrak `public int hashCode()` dan `public boolean equals(Object obj)` operasi yang disediakan dalam `Record` dan `UserRecord` untuk memungkinkan perbandingan catatan pengguna KPL Anda. Selain itu, jika Anda ingin memeriksa nomor urutan catatan pengguna KPL Anda, Anda dapat mentransmisikannya ke sebuah `UserRecord` instance dan mengambil nomor urutannya.

Lihat informasi yang lebih lengkap di Menerapkan de-agregasi konsumen.

Baca 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 2 MB/sec jatah throughput baca sendiri, memungkinkan beberapa 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. Kembangkan konsumen fan-out yang ditingkatkan dengan throughput khusus

Anda dapat membangun konsumen untuk Kinesis Data Streams menggunakan Kinesis Client Library (KCL) atau. AWS SDK untuk Java Anda juga dapat mengembangkan konsumen menggunakan AWS layanan lain seperti AWS Lambda, Amazon Managed Service untuk Apache Flink, dan Amazon Data Firehose. Kinesis Data Streams mendukung integrasi AWS dengan layanan lain seperti Amazon EMR, Amazon, dan Amazon Redshift Ini juga mendukung integrasi pihak ketiga termasuk Apache Flink EventBridge AWS Glue, Adobe Experience Platform, Apache Druid, Apache Spark, Databricks, Confluent Platform, Kinesumer, dan Talend.

Topik

- Kembangkan konsumen fan-out yang ditingkatkan dengan throughput khusus
- Gunakan Penampil Data di konsol Kinesis
- Kueri aliran data Anda di konsol Kinesis
- Gunakan Perpustakaan Klien Kinesis
- Mengembangkan konsumen dengan AWS SDK untuk Java
- Mengembangkan konsumen menggunakan AWS Lambda
- Kembangkan konsumen menggunakan Amazon Managed Service untuk Apache Flink
- Kembangkan konsumen menggunakan Amazon Data Firehose
- Membaca data dari Kinesis Data AWS Streams menggunakan layanan lain
- Baca dari Kinesis Data Streams menggunakan integrasi pihak ketiga
- Memecahkan Masalah Kinesis Data Streams konsumen
- Optimalkan konsumen Amazon Kinesis Data Streams

Kembangkan konsumen fan-out yang ditingkatkan dengan throughput khusus

Di Amazon Kinesis Data Streams, Anda dapat membangun konsumen yang menggunakan fitur yang disebut fan-out yang disempurnakan. Fitur ini memungkinkan konsumen 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 disempurnakan. Oleh karena itu, konsumen ini tidak perlu melakukan polling untuk data.

> **Important**
>
> Dengan mode On-Demand Advantage, Anda dapat mendaftarkan hingga 50 konsumen per streaming untuk menggunakan fan-out yang ditingkatkan. Dengan streaming Standar dan Penyediaan Sesuai Permintaan, Anda dapat mendaftarkan hingga 20 konsumen per aliran untuk menggunakan fan-out yang disempurnakan.

Diagram berikut menunjukkan arsitektur fan-out yang ditingkatkan. Jika Anda menggunakan Amazon Kinesis Client Library (KCL) versi 2.0 atau yang lebih baru untuk membangun konsumen, KCL akan menyiapkan 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, Anda dapat berlangganan pecahan individual.

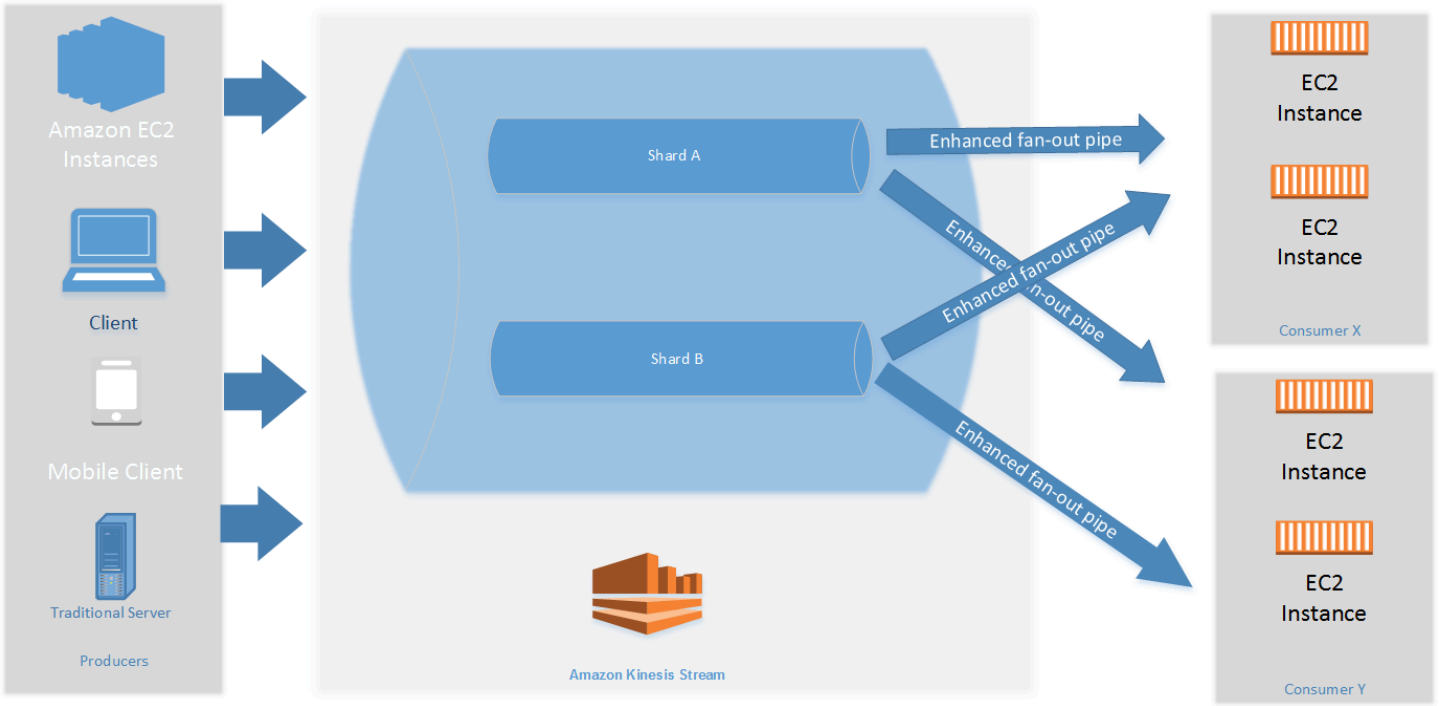


Diagram menunjukkan yang berikut:

- Aliran dengan dua pecahan.
- Dua konsumen yang menggunakan fan-out yang ditingkatkan untuk menerima data dari aliran: Konsumen X dan Konsumen Y. Masing-masing dari dua konsumen berlangganan semua pecahan dan semua catatan aliran. 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 individual.
- Panah mewakili pipa kipas yang disempurnakan yang digunakan konsumen untuk menerima data dari aliran. Pipa kipas yang disempurnakan menyediakan hingga 2 MB/sec data per pecahan, terlepas dari pipa lain atau dari jumlah total konsumen.

Topik

- Perbedaan antara konsumen throughput bersama dan konsumen fan-out yang ditingkatkan
- Wilayah yang didukung hingga 50 konsumen fan-out yang ditingkatkan (On-Demand Advantage only)
- Kelola konsumen fan-out yang ditingkatkan dengan atau AWS CLI APIs

Perbedaan antara konsumen throughput bersama dan konsumen fan-out yang ditingkatkan

Tabel berikut membandingkan konsumen throughput bersama default dengan konsumen fan-out yang ditingkatkan. Penundaan propagasi pesan didefinisikan sebagai waktu yang dibutuhkan dalam milidetik untuk payload yang dikirim menggunakan payload-dispatching APIs (like `PutRecord` dan`PutRecords`) untuk menjangkau aplikasi konsumen melalui payload-consuming (like dan). APIs `GetRecords` `SubscribeToShard`

Tabel ini membandingkan konsumen throughput bersama dengan konsumen fan-out yang ditingkatkan

Karakteristik	Konsumen throughput bersama tanpa peningkatan fan-out	Konsumen fan-out yang ditingkatkan
Throughput Baca	Diperbaiki pada total 2 MB/sec 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 menggunak an fan-out yang ditingkat kan. Setiap konsumen yang terdaftar untuk menggunak an fan-out yang ditingkatkan menerima throughput 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 naik menjadi sekitar 1000 ms jika Anda memiliki lima konsumen.	Biasanya rata-rata 70 ms apakah Anda memiliki satu konsumen atau lima konsumen.
Biaya	Tidak berlaku	Ada biaya pengambilan data dan biaya jam pecahan konsumen. Untuk informasi selengkapnya, lihat Harga

Karakteristik	Konsumen throughput bersama tanpa peningkatan fan-out	Konsumen fan-out yang ditingkatkan
		Amazon Kinesis Data Streams.
Model pengiriman catatan	Tarik model melalui HTTP menggunakan GetRecords.	Kinesis Data Streams mendorong catatan kepada Anda melalui HTTP/2 menggunakan. Subscribe ToShard

Wilayah yang didukung hingga 50 konsumen fan-out yang ditingkatkan (On-Demand Advantage only)

Support untuk hingga 50 konsumen fan-out yang ditingkatkan dalam mode On-Demand Advantage hanya tersedia di Wilayah berikut: AWS

AWS Wilayah	Nama wilayah
eu-north-1	Eropa (Stockholm)
me-south-1	Timur Tengah (Bahrain)
ap-south-1	Asia Pasifik (Mumbai)
eu-west-3	Eropa (Paris)
ap-southeast-3	Asia Pasifik (Jakarta)
us-east-2	AS Timur (Ohio)
af-south-1	Africa (Cape Town)
eu-west-1	Eropa (Irlandia)
me-central-1	Timur Tengah (UAE)

AWS Wilayah	Nama wilayah
eu-central-1	Eropa (Frankfurt)
sa-east-1	Amerika Selatan (Sao Paulo)
ap-east-1	Asia Pasifik (Hong Kong)
ap-south-2	Asia Pasifik (Hyderabad)
us-east-1	AS Timur (Virginia Utara)
ap-northeast-2	Asia Pasifik (Seoul)
ap-northeast-3	Asia Pasifik (Osaka)
eu-west-2	Eropa (London)
ap-southeast-4	Asia Pacific (Melbourne)
ap-northeast-1	Asia Pasifik (Tokyo)
us-west-2	AS Barat (Oregon)
us-west-1	AS Barat (California Utara)
ap-southeast-1	Asia Pasifik (Singapura)
ap-southeast-2	Asia Pasifik (Sydney)
il-central-1	Israel (Tel Aviv)
ca-central-1	Kanada (Pusat)
ca-west-1	Kanada Barat (Calgary)
eu-south-2	Eropa (Spanyol)
cn-northwest-1	Tiongkok (Ningxia)
eu-central-2	Europe (Zurich)

AWS Wilayah	Nama wilayah
us-gov-east-1	AWS GovCloud (AS-Timur)
us-gov-west-1	AWS GovCloud (AS-Barat)

Kelola konsumen fan-out yang ditingkatkan dengan atau AWS CLI APIs

Konsumen yang menggunakan fan-out yang disempurnakan di Amazon Kinesis Data Streams dapat menerima catatan dari aliran data dengan throughput khusus hingga 2 MB data per detik per pecahan. Untuk informasi selengkapnya, lihat Kembangkan konsumen fan-out yang ditingkatkan dengan throughput khusus.

Anda dapat menggunakan AWS CLI atau Kinesis APIs Data Streams untuk mendaftarkan, mendeskripsikan, mencantumkan, dan membatalkan pendaftaran konsumen yang menggunakan fan-out yang disempurnakan di Kinesis Data Streams.

Mengelola konsumen menggunakan AWS CLI

Anda dapat mendaftarkan, mendeskripsikan, membuat daftar, dan membatalkan pendaftaran konsumen penggemar yang ditingkatkan menggunakan. AWS CLI Sebagai contoh, lihat dokumentasi berikut.

register-stream-consumer

Mendaftarkan konsumen untuk aliran data Kinesis. Anda dapat menerapkan tag saat mendaftarkan konsumen.

describe-stream-consumer

Mendapat deskripsi konsumen terdaftar dengan ARN konsumen atau nama konsumen dan aliran ARN kombinasi.

list-stream-consumers

Daftar konsumen yang terdaftar untuk menerima data dari aliran menggunakan fan-out yang disempurnakan.

deregister-stream-consumer

Deregister konsumen dengan ARN konsumen atau nama konsumen dan streaming ARN kombinasi.

Mengelola konsumen menggunakan Kinesis Data Streams APIs

Anda dapat mendaftarkan, mendeskripsikan, membuat daftar, dan membatalkan pendaftaran konsumen penggemar yang disempurnakan menggunakan Kinesis Data Streams. APIs Sebagai contoh, lihat dokumentasi berikut.

RegisterStreamConsumer

Mendaftarkan konsumen untuk aliran data Kinesis dengan tag. Anda dapat menerapkan tag saat mendaftarkan konsumen.

DescribeStreamConsumer

Mendapat deskripsi konsumen terdaftar dengan ARN konsumen atau nama konsumen dan aliran ARN kombinasi.

ListStreamConsumers

Daftar konsumen yang terdaftar untuk menerima data dari aliran menggunakan fan-out yang disempurnakan.

DeregisterStreamConsumer

Deregister konsumen dengan ARN konsumen atau nama konsumen dan streaming ARN kombinasi.

Menandai konsumen

Anda dapat menetapkan metadata Anda sendiri ke streaming dan konsumen penggemar yang disempurnakan yang Anda buat di Kinesis Data Streams dalam bentuk tag. Anda dapat menggunakan tag untuk mengkategorikan dan melacak biaya konsumen Anda. Anda juga dapat mengontrol akses ke konsumen menggunakan tag dengan kontrol akses berbasis atribut (ABAC). Lihat informasi yang lebih lengkap di Tandai sumber daya Amazon Kinesis Data Streams Anda.

Gunakan Penampil Data di konsol Kinesis

Penampil Data di Konsol Manajemen Kinesis memungkinkan Anda 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 Konsol Manajemen AWS dan buka konsol Kinesis di /kinesis. https://console.aws.amazon.com

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.

Kueri 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 Konsol Manajemen AWS dan buka konsol Kinesis di /kinesis. https://console.aws.amazon.com
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.

Gunakan Perpustakaan Klien Kinesis

Apa itu Perpustakaan Klien Kinesis?

Kinesis Client Library (KCL) adalah pustaka perangkat lunak Java mandiri yang dirancang untuk menyederhanakan proses konsumsi dan pemrosesan data dari Amazon Kinesis Data Streams. KCL menangani banyak tugas kompleks yang terkait dengan komputasi terdistribusi, memungkinkan pengembang fokus pada penerapan logika bisnis mereka untuk memproses data. Ini mengelola aktivitas seperti penyeimbangan beban di beberapa pekerja, menanggapi kegagalan pekerja, memeriksa catatan yang diproses, dan menanggapi perubahan jumlah pecahan dalam aliran.

KCL sering diperbarui untuk menggabungkan versi yang lebih baru dari pustaka yang mendasari, peningkatan keamanan, dan perbaikan bug. Kami menyarankan Anda menggunakan versi terbaru KCL untuk menghindari masalah yang diketahui dan mendapatkan manfaat dari semua peningkatan terbaru. Untuk menemukan versi KCL terbaru, lihat KCL Github.

Important

- Kami menyarankan Anda menggunakan versi KCL terbaru untuk menghindari bug dan masalah yang diketahui. Jika Anda menggunakan KCL 2.6.0 atau sebelumnya, tingkatkan ke KCL 2.6.1 atau yang lebih baru untuk menghindari kondisi langka yang dapat memblokir pemrosesan pecahan saat kapasitas aliran berubah.

- KCL adalah perpustakaan Java. Support untuk bahasa selain Java disediakan menggunakan daemon berbasis Java yang disebut. MultiLangDaemon MultiLangDaemonberinteraksi dengan aplikasi KCL melalui STDIN dan STDOUT. Untuk informasi lebih lanjut tentang MultiLangDaemon on GitHub, lihatMengembangkan konsumen dengan KCL dalam bahasa non-Jawa.
- Jangan gunakan AWS SDK untuk Java versi 2.27.19 hingga 2.27.23 dengan KCL 3.x. Versi ini menyertakan masalah yang menyebabkan kesalahan pengecualian terkait dengan penggunaan DynamoDB KCL. Kami menyarankan Anda menggunakan AWS SDK untuk Java versi 2.28.0 atau yang lebih baru untuk menghindari masalah ini.

Fitur dan manfaat utama KCL

Berikut ini adalah fitur utama dan manfaat terkait dari KCL:

- Skalabilitas: KCL memungkinkan aplikasi untuk menskalakan secara dinamis dengan mendistribusikan beban pemrosesan ke beberapa pekerja. Anda dapat menskalakan aplikasi masuk atau keluar, secara manual atau dengan auto-scaling, tanpa khawatir tentang redistribusi beban.
- Load balancing: KCL secara otomatis menyeimbangkan beban pemrosesan di seluruh pekerja yang tersedia, menghasilkan distribusi pekerjaan yang merata di seluruh pekerja.
- Checkpointing: KCL mengelola checkpointing catatan yang diproses, memungkinkan aplikasi untuk melanjutkan pemrosesan dari posisi terakhir yang berhasil diproses.
- Toleransi kesalahan: KCL menyediakan mekanisme toleransi kesalahan bawaan, memastikan bahwa pemrosesan data berlanjut bahkan jika pekerja individu gagal. KCL juga menyediakan at-least-once pengiriman.
- Menangani perubahan tingkat aliran: KCL beradaptasi dengan pecahan pecahan dan penggabungan yang mungkin terjadi karena perubahan volume data. Ini mempertahankan pemesanan dengan memastikan bahwa pecahan anak diproses hanya setelah pecahan induknya selesai dan diperiksa.
- Pemantauan: KCL terintegrasi dengan Amazon CloudWatch untuk pemantauan tingkat konsumen.
- Dukungan multi-bahasa: KCL secara native mendukung Java dan memungkinkan beberapa bahasa pemrograman non-Java melalui. MultiLangDaemon

Konsep KCL

Bagian ini menjelaskan konsep inti dan interaksi Kinesis Client Library (KCL). Konsep-konsep ini sangat penting untuk mengembangkan dan mengelola aplikasi konsumen KCL.

- Aplikasi konsumen KCL — aplikasi yang dibuat khusus yang dirancang untuk membaca dan memproses catatan dari aliran data Kinesis menggunakan Perpustakaan Klien Kinesis.
- Worker — Aplikasi konsumen KCL biasanya didistribusikan, dengan satu atau lebih pekerja berjalan secara bersamaan. KCL mengkoordinasikan pekerja untuk mengkonsumsi data dari aliran secara terdistribusi dan menyeimbangkan beban secara merata di beberapa pekerja.
- Scheduler — kelas tingkat tinggi yang digunakan pekerja KCL untuk mulai memproses data. Setiap pekerja KCL memiliki satu penjadwal. Penjadwal menginisialisasi dan mengawasi berbagai tugas, termasuk menyinkronkan informasi pecahan dari aliran data Kinesis, melacak tugas pecahan di antara pekerja, dan memproses data dari aliran berdasarkan pecahan yang ditetapkan ke pekerja. Scheduler dapat mengambil berbagai konfigurasi yang memengaruhi perilaku penjadwal, seperti nama aliran untuk diproses dan kredensyal. AWS Scheduler memulai pengiriman catatan data dari aliran ke prosesor rekaman.
- Record processor — mendefinisikan logika bagaimana aplikasi konsumen KCL Anda memproses data yang diterimanya dari aliran data. Anda harus menerapkan logika pemrosesan data kustom Anda sendiri di prosesor rekaman. Seorang pekerja KCL membuat instance penjadwal. Penjadwal kemudian membuat instance satu prosesor rekaman untuk setiap pecahan yang disewakan. Seorang pekerja dapat menjalankan beberapa prosesor rekaman.
- Sewa - mendefinisikan penugasan antara pekerja dan pecahan. Aplikasi konsumen KCL menggunakan sewa untuk mendistribusikan pemrosesan catatan data di beberapa pekerja. Setiap pecahan terikat hanya untuk satu pekerja dengan sewa pada waktu tertentu dan setiap pekerja dapat memegang satu atau lebih sewa secara bersamaan. Ketika seorang pekerja berhenti memegang sewa karena berhenti atau gagal, KCL menugaskan pekerja lain untuk mengambil sewa. Untuk mempelajari lebih lanjut tentang sewa, lihat Dokumentasi Github: Siklus Hidup Sewa.
- Tabel sewa - adalah tabel Amazon DynamoDB unik yang digunakan untuk melacak semua sewa untuk aplikasi konsumen KCL. Setiap aplikasi konsumen KCL membuat tabel sewa sendiri. Tabel sewa digunakan untuk mempertahankan status di semua pekerja untuk mengoordinasikan pemrosesan data. Untuk informasi selengkapnya, lihat Tabel metadata DynamoDB dan load balancing di KCL.
- Checkpointing — adalah proses terus-menerus menyimpan posisi catatan terakhir yang berhasil diproses dalam pecahan. KCL mengelola checkpointing untuk memastikan bahwa pemrosesan dapat dilanjutkan dari posisi checkpoint terakhir jika pekerja gagal atau aplikasi restart. Pos

pemeriksaan disimpan dalam tabel sewa DynamoDB sebagai bagian dari metadata sewa. Hal ini memungkinkan pekerja untuk melanjutkan pemrosesan dari tempat pekerja sebelumnya berhenti.

Tabel metadata DynamoDB dan load balancing di KCL

KCL mengelola metadata seperti sewa dan metrik pemanfaatan CPU dari pekerja. KCL melacak metadata ini menggunakan tabel DynamoDB. Untuk setiap aplikasi Amazon Kinesis Data Streams, KCL membuat tiga tabel DynamoDB untuk mengelola metadata: tabel sewa, tabel metrik pekerja, dan tabel status koordinator.

> **Note**
>
> KCL 3.x memperkenalkan dua tabel metadata baru: metrik pekerja dan tabel status koordinator.

> **Important**
>
> Anda harus menambahkan izin yang tepat untuk aplikasi KCL untuk membuat dan mengelola tabel metadata di DynamoDB. Lihat perinciannya di Izin IAM diperlukan untuk aplikasi konsumen KCL.
> Aplikasi konsumen KCL tidak secara otomatis menghapus ketiga tabel metadata DynamoDB ini. Pastikan Anda menghapus tabel metadata DynamoDB ini yang dibuat oleh aplikasi konsumen KCL saat Anda menonaktifkan aplikasi konsumen Anda untuk mencegah biaya yang tidak perlu.

Meja sewa

Tabel sewa adalah tabel Amazon DynamoDB unik yang digunakan untuk melacak pecahan yang disewa dan diproses oleh penjadwal aplikasi konsumen KCL. Setiap aplikasi konsumen KCL membuat tabel sewa sendiri. KCL menggunakan nama aplikasi konsumen untuk nama tabel sewa secara default. Anda dapat mengatur nama tabel kustom menggunakan konfigurasi. KCL juga menciptakan indeks sekunder global pada tabel sewa dengan kunci partisi LeaseOwner untuk penemuan sewa yang efisien. Indeks sekunder global mencerminkan atribut LeaseKey dari tabel sewa dasar. Jika tabel sewa untuk aplikasi konsumen KCL Anda tidak ada saat aplikasi dimulai, salah satu pekerja membuat tabel sewa untuk aplikasi Anda.

Anda dapat melihat tabel sewa menggunakan konsol Amazon DynamoDB saat aplikasi konsumen sedang berjalan.

> **Important**
>
> - Setiap nama aplikasi konsumen KCL harus unik untuk mencegah duplikasi nama tabel sewa.
> - 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 penjadwal aplikasi konsumen Anda. Bidang utama meliputi yang berikut:

- LeaseKey: Untuk pemrosesan aliran tunggal, ini adalah ID pecahan. Untuk pemrosesan multi-stream dengan KCL, ini terstruktur sebagai. `account-id:StreamName:streamCreationTimestamp:ShardId` leaseKey adalah kunci partisi dari tabel sewa. Untuk informasi selengkapnya tentang pemrosesan multi-aliran, lihatPemrosesan multi-aliran dengan KCL.
- pos pemeriksaan: Nomor urutan pos pemeriksaan terbaru untuk pecahan.
- 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 memeriksa apakah seorang pekerja saat ini memproses sewa secara aktif. LeaseCounter meningkat jika kepemilikan sewa ditransfer ke pekerja lain.
- LeaseOwner: Pekerja saat ini yang memegang sewa ini.
- ownerSwitchesSincePos pemeriksaan: Berapa kali sewa ini telah berganti pekerja sejak pos pemeriksaan terakhir.
- parentShardId: ID dari induk pecahan ini. Pastikan pecahan induk diproses sepenuhnya sebelum pemrosesan dimulai pada pecahan anak, mempertahankan urutan pemrosesan catatan yang benar.
- childShardId: Daftar pecahan anak IDs yang dihasilkan dari pecahan atau penggabungan pecahan ini. Digunakan untuk melacak garis keturunan pecahan dan mengelola urutan pemrosesan selama operasi resharding.
- startingHashKey: Batas bawah kisaran kunci hash untuk pecahan ini.

- endingHashKey: Batas atas rentang kunci hash untuk pecahan ini.

Jika Anda menggunakan pemrosesan multi-aliran dengan KCL, Anda melihat dua bidang tambahan berikut dalam tabel sewa. Untuk informasi selengkapnya, lihat Pemrosesan multi-aliran dengan KCL.

- ShardID: ID pecahan.
- StreamName: Pengidentifikasi aliran data dalam format berikut:. `account-id:StreamName:streamCreationTimestamp`

Tabel metrik pekerja

Tabel metrik Worker adalah tabel Amazon DynamoDB unik untuk setiap aplikasi KCL dan digunakan untuk merekam metrik pemanfaatan CPU dari setiap pekerja. Metrik ini akan digunakan oleh KCL untuk melakukan penugasan sewa yang efisien untuk menghasilkan pemanfaatan sumber daya yang seimbang di seluruh pekerja. KCL menggunakan `KCLApplicationName-WorkerMetricStats` nama tabel metrik pekerja secara default.

Tabel negara koordinator

Tabel status koordinator adalah tabel Amazon DynamoDB unik untuk setiap aplikasi KCL dan digunakan untuk menyimpan informasi status internal untuk pekerja. Misalnya, tabel negara koordinator menyimpan data mengenai pemilihan pemimpin atau metadata yang terkait dengan migrasi di tempat dari KCL 2.x ke KCL 3.x. KCL menggunakan `KCLApplicationName-CoordinatorState` nama tabel status koordinator secara default.

Mode kapasitas DynamoDB untuk tabel metadata yang dibuat oleh KCL

Secara default, Kinesis Client Library (KCL) membuat tabel metadata DynamoDB seperti tabel sewa, tabel metrik pekerja, dan tabel status koordinator menggunakan mode kapasitas sesuai permintaan. Mode ini secara otomatis menskalakan kapasitas baca dan tulis untuk mengakomodasi lalu lintas tanpa memerlukan perencanaan kapasitas. Kami sangat menyarankan Anda untuk menjaga mode kapasitas sebagai mode sesuai permintaan untuk pengoperasian tabel metadata ini dengan lebih efisien.

Jika Anda memutuskan untuk mengganti tabel sewa ke mode kapasitas yang disediakan, ikuti praktik terbaik berikut:

- Menganalisis pola penggunaan:

  - Pantau pola dan penggunaan baca dan tulis aplikasi Anda (RCU, WCU) menggunakan metrik Amazon. CloudWatch
  - Memahami persyaratan throughput puncak dan rata-rata.
- Hitung kapasitas yang dibutuhkan:
  - Perkirakan unit kapasitas baca (RCUs) dan tulis unit kapasitas (WCUs) berdasarkan analisis Anda.
  - Pertimbangkan faktor-faktor seperti jumlah pecahan, frekuensi pos pemeriksaan, dan jumlah pekerja.
- Menerapkan penskalaan otomatis:
  - Gunakan penskalaan otomatis DynamoDB untuk menyesuaikan kapasitas yang disediakan secara otomatis dan menetapkan batas kapasitas minimum dan maksimum yang sesuai.
  - Penskalaan otomatis DynamoDB akan membantu menghindari tabel metadata KCL Anda mencapai batas kapasitas dan terhambat.
- Pemantauan dan pengoptimalan rutin:
  - Terus memantau CloudWatch metrik untuk`ThrottledRequests`.
  - Sesuaikan kapasitas saat beban kerja Anda berubah seiring waktu.

Jika Anda mengalami tabel DynamoDB `ProvisionedThroughputExceededException` dalam metadata untuk aplikasi konsumen KCL Anda, Anda harus meningkatkan kapasitas throughput yang disediakan dari tabel DynamoDB. Jika Anda menetapkan tingkat tertentu unit kapasitas baca (RCU) dan unit kapasitas tulis (WCU) saat pertama kali membuat aplikasi konsumen Anda, itu mungkin tidak cukup seiring dengan bertambahnya penggunaan Anda. Misalnya, jika aplikasi konsumen KCL Anda sering melakukan pemeriksaan atau beroperasi pada aliran dengan banyak pecahan, Anda mungkin memerlukan lebih banyak unit kapasitas. Untuk informasi tentang throughput yang disediakan di DynamoDB, lihat kapasitas throughput DynamoDB dan memperbarui tabel di Panduan Pengembang Amazon DynamoDB.

Bagaimana KCL memberikan sewa kepada pekerja dan menyeimbangkan beban

KCL terus mengumpulkan dan memantau metrik pemanfaatan CPU dari host komputasi yang menjalankan pekerja untuk memastikan distribusi beban kerja yang merata. Metrik pemanfaatan CPU ini disimpan dalam tabel metrik pekerja di DynamoDB. Jika KCL mendeteksi bahwa beberapa pekerja menunjukkan tingkat pemanfaatan CPU yang lebih tinggi dibandingkan dengan yang lain, itu akan menetapkan kembali sewa di antara pekerja untuk menurunkan beban pada pekerja yang sangat digunakan. Tujuannya adalah untuk menyeimbangkan beban kerja secara lebih

merata di seluruh armada aplikasi konsumen, mencegah setiap pekerja menjadi kelebihan beban. Saat KCL mendistribusikan pemanfaatan CPU di seluruh armada aplikasi konsumen, Anda dapat mengukur kapasitas armada aplikasi konsumen Anda dengan memilih jumlah pekerja yang tepat atau menggunakan penskalaan otomatis untuk mengelola kapasitas komputasi secara efisien guna mencapai biaya yang lebih rendah.

> **Important**
>
> KCL dapat mengumpulkan metrik pemanfaatan CPU dari pekerja hanya jika prasyarat tertentu terpenuhi. Lihat perinciannya di Prasyarat. Jika KCL tidak dapat mengumpulkan metrik pemanfaatan CPU dari pekerja, KCL akan kembali menggunakan throughput per pekerja untuk menetapkan sewa dan menyeimbangkan beban di seluruh pekerja di armada. KCL akan memantau throughput yang diterima setiap pekerja pada waktu tertentu dan menetapkan kembali sewa untuk memastikan bahwa setiap pekerja mendapatkan tingkat throughput total yang sama dari sewa yang ditugaskan.

Kembangkan konsumen dengan KCL

Anda dapat menggunakan Kinesis Client Library (KCL) untuk membangun aplikasi konsumen yang memproses data dari aliran data Kinesis Anda.

KCL tersedia dalam berbagai bahasa. Topik ini mencakup bagaimana mengembangkan konsumen KCL dalam bahasa Jawa dan non-Jawa.

- Untuk melihat referensi Javadoc Perpustakaan Klien Kinesis, lihat Javadoc Perpustakaan Klien Amazon Kinesis.
- Untuk mengunduh KCL untuk Java dari GitHub, lihat Perpustakaan Klien Amazon Kinesis untuk Java.
- Untuk menemukan KCL untuk Java di Apache Maven, lihat KCL Maven Central Repository.

Topik

- Kembangkan konsumen dengan KCL di Jawa
- Mengembangkan konsumen dengan KCL dalam bahasa non-Jawa

Kembangkan konsumen dengan KCL di Jawa

Prasyarat

Sebelum Anda mulai menggunakan KCL 3.x, pastikan Anda memiliki yang berikut:

- Java Development Kit (JDK) 8 atau lebih baru
- AWS SDK untuk Java 2.x
- Maven atau Gradle untuk manajemen ketergantungan

KCL mengumpulkan metrik pemanfaatan CPU seperti pemanfaatan CPU dari host komputasi yang dijalankan pekerja untuk menyeimbangkan beban untuk mencapai tingkat pemanfaatan sumber daya yang merata di seluruh pekerja. Untuk mengaktifkan KCL mengumpulkan metrik pemanfaatan CPU dari pekerja, Anda harus memenuhi prasyarat berikut:

Amazon Elastic Compute Cloud(Amazon EC2)

- Sistem operasi Anda harus OS Linux.
- Anda harus mengaktifkan IMDSv2di instans EC2 Anda.

Amazon Elastic Container Service (Amazon ECS) di Amazon EC2

- Sistem operasi Anda harus OS Linux.
- Anda harus mengaktifkan titik akhir metadata tugas ECS versi 4.
- Versi agen penampung Amazon ECS Anda harus 1.39.0 atau yang lebih baru.

Amazon ECS aktif AWS Fargate

- Anda harus mengaktifkan titik akhir metadata tugas Fargate versi 4. Jika Anda menggunakan platform Fargate versi 1.4.0 atau yang lebih baru, ini diaktifkan secara default.
- Platform Fargate versi 1.4.0 atau yang lebih baru.

Layanan Amazon Elastic Kubernetes (Amazon EKS) di Amazon EC2

- Sistem operasi Anda harus OS Linux.

Amazon EKS di AWS Fargate

- Platform Fargate 1.3.0 atau yang lebih baru.

> **Important**
>
> Jika KCL tidak dapat mengumpulkan metrik pemanfaatan CPU dari pekerja, KCL akan kembali menggunakan throughput per pekerja untuk menetapkan sewa dan menyeimbangkan beban di seluruh pekerja di armada. Untuk informasi selengkapnya, lihat Bagaimana KCL memberikan sewa kepada pekerja dan menyeimbangkan beban.

Instal dan tambahkan dependensi

Jika Anda menggunakan Maven, tambahkan dependensi berikut ke file Anda. `pom.xml` Pastikan Anda mengganti 3.xx ke versi KCL terbaru.

```
<dependency>
    <groupId>software.amazon.kinesis</groupId>
    <artifactId>amazon-kinesis-client</artifactId>
    <version>3.x.x</version> <!-- Use the latest version -->
</dependency>
```

Jika Anda menggunakan Gradle, tambahkan berikut ini ke `build.gradle` file Anda. Pastikan Anda mengganti 3.xx ke versi KCL terbaru.

```
implementation 'software.amazon.kinesis:amazon-kinesis-client:3.x.x'
```

Anda dapat memeriksa versi terbaru KCL di Repositori Pusat Maven.

Menerapkan konsumen

Aplikasi konsumen KCL terdiri dari komponen-komponen kunci berikut:

Komponen kunci

- RecordProcessor
- RecordProcessorFactory
- Penjadwal
- Aplikasi Konsumen Utama

RecordProcessor

RecordProcessor adalah komponen inti di mana logika bisnis Anda untuk memproses catatan aliran data Kinesis berada. Ini mendefinisikan bagaimana aplikasi Anda memproses data yang diterimanya dari aliran Kinesis.

Tanggung jawab utama:

- Inisialisasi pemrosesan untuk pecahan
- Memproses kumpulan catatan dari aliran Kinesis
- Pemrosesan shutdown untuk pecahan (misalnya, ketika pecahan pecah atau bergabung, atau sewa diserahkan ke host lain)
- Menangani checkpointing untuk melacak kemajuan

Berikut ini menunjukkan contoh implementasi:

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.slf4j.MDC;
import software.amazon.kinesis.exceptions.InvalidStateException;
import software.amazon.kinesis.exceptions.ShutdownException;
import software.amazon.kinesis.lifecycle.events.*;
import software.amazon.kinesis.processor.ShardRecordProcessor;

public class SampleRecordProcessor implements ShardRecordProcessor {
    private static final String SHARD_ID_MDC_KEY = "ShardId";
    private static final Logger log =
 LoggerFactory.getLogger(SampleRecordProcessor.class);
    private String shardId;

    @Override
    public void initialize(InitializationInput initializationInput) {
        shardId = initializationInput.shardId();
        MDC.put(SHARD_ID_MDC_KEY, shardId);
        try {
            log.info("Initializing @ Sequence: {}",
 initializationInput.extendedSequenceNumber());
        } finally {
            MDC.remove(SHARD_ID_MDC_KEY);
        }
    }
```

```
    @Override
    public void processRecords(ProcessRecordsInput processRecordsInput) {
        MDC.put(SHARD_ID_MDC_KEY, shardId);
        try {
            log.info("Processing {} record(s)", processRecordsInput.records().size());
            processRecordsInput.records().forEach(r ->
                log.info("Processing record pk: {} -- Seq: {}", r.partitionKey(),
r.sequenceNumber())
            );

            // Checkpoint periodically
            processRecordsInput.checkpointer().checkpoint();
        } catch (Throwable t) {
            log.error("Caught throwable while processing records. Aborting.", t);
        } finally {
            MDC.remove(SHARD_ID_MDC_KEY);
        }
    }

    @Override
    public void leaseLost(LeaseLostInput leaseLostInput) {
        MDC.put(SHARD_ID_MDC_KEY, shardId);
        try {
            log.info("Lost lease, so terminating.");
        } finally {
            MDC.remove(SHARD_ID_MDC_KEY);
        }
    }

    @Override
    public void shardEnded(ShardEndedInput shardEndedInput) {
        MDC.put(SHARD_ID_MDC_KEY, shardId);
        try {
            log.info("Reached shard end checkpointing.");
            shardEndedInput.checkpointer().checkpoint();
        } catch (ShutdownException | InvalidStateException e) {
            log.error("Exception while checkpointing at shard end. Giving up.", e);
        } finally {
            MDC.remove(SHARD_ID_MDC_KEY);
        }
    }

    @Override
```

```
    public void shutdownRequested(ShutdownRequestedInput shutdownRequestedInput) {
        MDC.put(SHARD_ID_MDC_KEY, shardId);
        try {
            log.info("Scheduler is shutting down, checkpointing.");
            shutdownRequestedInput.checkpointer().checkpoint();
        } catch (ShutdownException | InvalidStateException e) {
            log.error("Exception while checkpointing at requested shutdown. Giving
 up.", e);
        } finally {
            MDC.remove(SHARD_ID_MDC_KEY);
        }
    }
}
```

Berikut ini adalah penjelasan rinci dari setiap metode yang digunakan dalam contoh:

inisialisasi (Inisialisasi InitializationInput Input)

- Tujuan: Siapkan sumber daya atau status yang diperlukan untuk memproses catatan.
- Ketika disebut: Sekali, ketika KCL menetapkan pecahan ke prosesor rekaman ini.
- Poin kunci:
  - `initializationInput.shardId()`: ID pecahan yang akan ditangani prosesor ini.
  - `initializationInput.extendedSequenceNumber()`: Nomor urut untuk mulai memproses dari.

ProcessRecords () ProcessRecordsInput processRecordsInput

- Tujuan: Memproses catatan yang masuk dan kemajuan pos pemeriksaan opsional.
- Ketika disebut: Berulang kali, selama prosesor rekaman memegang sewa untuk pecahan.
- Poin kunci:
  - `processRecordsInput.records()`: Daftar catatan untuk diproses.
  - `processRecordsInput.checkpointer()`: Digunakan untuk memeriksa kemajuan.
  - Pastikan Anda menangani pengecualian apa pun selama pemrosesan untuk mencegah KCL gagal.
  - Metode ini harus idempoten, karena catatan yang sama dapat diproses lebih dari sekali dalam beberapa skenario, seperti data yang belum diperiksa sebelum pekerja yang tidak terduga mogok atau restart.

- Selalu siram data buffer sebelum checkpointing untuk memastikan konsistensi data.

LeaseLost () LeaseLostInput leaseLostInput

- Tujuan: Bersihkan sumber daya khusus untuk memproses pecahan ini.
- Ketika itu disebut: Ketika Scheduler lain mengambil alih sewa untuk pecahan ini.
- Poin kunci:
  - Checkpointing tidak diperbolehkan dalam metode ini.

shardEnded () ShardEndedInput shardEndedInput

- Tujuan: Selesaikan pemrosesan untuk pecahan dan pos pemeriksaan ini.
- Saat disebut: Ketika pecahan terbelah atau bergabung, menunjukkan semua data untuk pecahan ini telah diproses.
- Poin kunci:
  - `shardEndedInput.checkpointer()`: Digunakan untuk melakukan pemeriksaan akhir.
  - Checkpointing dalam metode ini adalah wajib untuk menyelesaikan pemrosesan.
  - Gagal menyiram data dan pos pemeriksaan di sini dapat mengakibatkan hilangnya data atau pemrosesan duplikat saat pecahan dibuka kembali.

ShutdownRequested () ShutdownRequestedInput shutdownRequestedInput

- Tujuan: Pos pemeriksaan dan bersihkan sumber daya saat KCL dimatikan.
- Ketika dipanggil: Ketika KCL dimatikan, misalnya, ketika aplikasi dihentikan).
- Poin kunci:
  - `shutdownRequestedInput.checkpointer()`: Digunakan untuk melakukan checkpointing sebelum shutdown.
  - Pastikan Anda menerapkan checkpointing dalam metode sehingga kemajuan disimpan sebelum aplikasi berhenti.
  - Kegagalan untuk menyiram data dan pos pemeriksaan di sini dapat mengakibatkan hilangnya data atau pemrosesan ulang catatan saat aplikasi dimulai ulang.

> **Important**
>
> KCL 3.x memastikan lebih sedikit pemrosesan ulang data saat sewa diserahkan dari satu pekerja ke pekerja lain dengan pos pemeriksaan sebelum pekerja sebelumnya dimatikan. Jika Anda tidak menerapkan logika checkpointing dalam `shutdownRequested()` metode, Anda tidak akan melihat manfaat ini. Pastikan Anda telah menerapkan logika checkpointing di dalam metode. `shutdownRequested()`

RecordProcessorFactory

RecordProcessorFactory bertanggung jawab untuk membuat RecordProcessor instance baru. KCL menggunakan pabrik ini untuk membuat yang baru RecordProcessor untuk setiap pecahan yang perlu diproses aplikasi.

Tanggung jawab utama:

- Buat RecordProcessor instance baru sesuai permintaan
- Pastikan masing-masing RecordProcessor diinisialisasi dengan benar

Berikut ini adalah contoh implementasi:

```
import software.amazon.kinesis.processor.ShardRecordProcessor;
import software.amazon.kinesis.processor.ShardRecordProcessorFactory;

public class SampleRecordProcessorFactory implements ShardRecordProcessorFactory {
    @Override
    public ShardRecordProcessor shardRecordProcessor() {
        return new SampleRecordProcessor();
    }
}
```

Dalam contoh ini, pabrik membuat yang baru SampleRecordProcessor setiap kali shardRecordProcessor () dipanggil. Anda dapat memperluas ini untuk memasukkan logika inisialisasi yang diperlukan.

Penjadwal

Scheduler adalah komponen tingkat tinggi yang mengoordinasikan semua aktivitas aplikasi KCL. Ini bertanggung jawab atas orkestrasi keseluruhan pemrosesan data.

Tanggung jawab utama:

- Mengelola siklus hidup RecordProcessors
- Menangani manajemen sewa untuk pecahan
- Mengkoordinasikan pos pemeriksaan
- Menyeimbangkan beban pemrosesan pecahan di beberapa pekerja aplikasi Anda
- Menangani sinyal shutdown dan penghentian aplikasi yang anggun

Scheduler biasanya dibuat dan dimulai di Aplikasi Utama. Anda dapat memeriksa contoh implementasi Scheduler di bagian berikut, Aplikasi Konsumen Utama.

Aplikasi Konsumen Utama

Aplikasi Konsumen Utama mengikat semua komponen bersama-sama. Ini bertanggung jawab untuk menyiapkan konsumen KCL, menciptakan klien yang diperlukan, mengonfigurasi Scheduler, dan mengelola siklus hidup aplikasi.

Tanggung jawab utama:

- Mengatur klien AWS layanan (Kinesis, DynamoDB,) CloudWatch
- Konfigurasikan aplikasi KCL
- Buat dan mulai Scheduler
- Menangani shutdown aplikasi

Berikut ini adalah contoh implementasi:

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cloudwatch.CloudWatchAsyncClient;
import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;
import software.amazon.awssdk.services.kinesis.KinesisAsyncClient;
import software.amazon.kinesis.common.ConfigsBuilder;
import software.amazon.kinesis.common.KinesisClientUtil;
import software.amazon.kinesis.coordinator.Scheduler;
import java.util.UUID;

public class SampleConsumer {
    private final String streamName;
    private final Region region;
    private final KinesisAsyncClient kinesisClient;
```

```
    public SampleConsumer(String streamName, Region region) {
        this.streamName = streamName;
        this.region = region;
        this.kinesisClient =
KinesisClientUtil.createKinesisAsyncClient(KinesisAsyncClient.builder().region(this.region));
    }

    public void run() {
        DynamoDbAsyncClient dynamoDbAsyncClient =
DynamoDbAsyncClient.builder().region(region).build();
        CloudWatchAsyncClient cloudWatchClient =
CloudWatchAsyncClient.builder().region(region).build();

        ConfigsBuilder configsBuilder = new ConfigsBuilder(
            streamName,
            streamName,
            kinesisClient,
            dynamoDbAsyncClient,
            cloudWatchClient,
            UUID.randomUUID().toString(),
            new SampleRecordProcessorFactory()
        );

        Scheduler scheduler = new Scheduler(
            configsBuilder.checkpointConfig(),
            configsBuilder.coordinatorConfig(),
            configsBuilder.leaseManagementConfig(),
            configsBuilder.lifecycleConfig(),
            configsBuilder.metricsConfig(),
            configsBuilder.processorConfig(),
            configsBuilder.retrievalConfig()
        );

        Thread schedulerThread = new Thread(scheduler);
        schedulerThread.setDaemon(true);
        schedulerThread.start();
    }

    public static void main(String[] args) {
        String streamName = "your-stream-name"; // replace with your stream name
        Region region = Region.US_EAST_1; // replace with your region
        new SampleConsumer(streamName, region).run();
    }
```

```
}
```

KCL menciptakan konsumen Enhanced Fan-out (EFO) dengan throughput khusus secara default. Untuk informasi selengkapnya tentang Enhanced Fan-out, lihat. Kembangkan konsumen fan-out yang ditingkatkan dengan throughput khusus Jika Anda memiliki kurang dari 2 konsumen atau tidak memerlukan penundaan propagasi baca di bawah 200 ms, Anda harus mengatur konfigurasi berikut di objek scheduler untuk menggunakan konsumen throughput bersama:

```
configsBuilder.retrievalConfig().retrievalSpecificConfig(new PollingConfig(streamName,
 kinesisClient))
```

Kode berikut adalah contoh pembuatan objek penjadwal yang menggunakan konsumen throughput bersama:

Impor:

```
import software.amazon.kinesis.retrieval.polling.PollingConfig;
```

Kode:

```
Scheduler scheduler = new Scheduler(
            configsBuilder.checkpointConfig(),
            configsBuilder.coordinatorConfig(),
            configsBuilder.leaseManagementConfig(),
            configsBuilder.lifecycleConfig(),
            configsBuilder.metricsConfig(),
            configsBuilder.processorConfig(),
            configsBuilder.retrievalConfig().retrievalSpecificConfig(new
 PollingConfig(streamName, kinesisClient))
        );/
```

Mengembangkan konsumen dengan KCL dalam bahasa non-Jawa

Bagian ini mencakup implementasi konsumen menggunakan Kinesis Client Library (KCL) di Python, Node.js, .NET, dan Ruby.

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 KCL dengan bahasa selain Java. Oleh karena itu, jika

Anda menginstal KCL untuk bahasa non-Java dan menulis aplikasi konsumen Anda sepenuhnya dalam bahasa non-Java, 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, wilayah AWS yang terhubung dengannya). Untuk informasi selengkapnya tentang `MultiLangDaemon` on GitHub, lihat MultiLangDaemon proyek KCL.

Sementara konsep inti tetap sama di seluruh bahasa, ada beberapa pertimbangan dan implementasi khusus bahasa. Untuk konsep inti tentang pengembangan konsumen KCL, lihatKembangkan konsumen dengan KCL di Jawa. Untuk informasi lebih rinci tentang cara mengembangkan konsumen KCL dengan Python, Node.js, .NET, dan Ruby dan pembaruan terbaru, silakan merujuk ke repositori berikut: GitHub

- Python: amazon-kinesis-client-python
- Node.js: amazon-kinesis-client-nodejs
- .NET: amazon-kinesis-client-net
- Ruby: amazon-kinesis-client-ruby

> **Important**
>
> Jangan gunakan versi pustaka KCL non-Java berikut jika Anda menggunakan JDK 8. Versi ini berisi dependensi (logback) yang tidak kompatibel dengan JDK 8.
>
> - KCL Python 3.0.2 dan 2.2.0
> - KCL Node.js 2.3.0
> - KCL .NET 3.1.0
> - KCL Ruby 2.2.0
>
> Sebaiknya gunakan versi yang dirilis sebelum atau sesudah versi yang terpengaruh ini saat bekerja dengan JDK 8.

Pemrosesan multi-aliran dengan KCL

Bagian ini menjelaskan perubahan yang diperlukan dalam KCL yang memungkinkan Anda membuat aplikasi konsumen KCL yang dapat memproses lebih dari satu aliran data secara bersamaan.

> **Important**
>
> - Pemrosesan multi-aliran hanya didukung di KCL 2.3 atau yang lebih baru.
> - Pemrosesan multi-aliran tidak didukung untuk konsumen KCL yang ditulis dalam bahasa non-Java yang berjalan dengan. `multilangdaemon`
> - Pemrosesan multi-aliran 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.
  - Yang `streamConfigList` 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;
}
```

  - `InitialPositionInStreamExtended`Bidang `StreamIdentifier` dan wajib, sementara `consumerArn` bersifat opsional. Anda harus menyediakan `consumerArn` satu-satunya jika Anda menggunakan KCL untuk menerapkan aplikasi konsumen fan-out yang disempurnakan.
  - Untuk informasi selengkapnya`StreamIdentifier`, lihat https://github.com/awslabs/amazon-kinesis-client/blob/v2.5.8/amazon-kinesis-client/src/main/java/software/amazon/kinesis/common/StreamIdentifier.java #L129. Untuk membuat`StreamIdentifier`, kami sarankan Anda membuat instance multistream dari `streamArn` dan yang tersedia di KCL

2.5.0 atau `streamCreationEpoch` yang lebih baru. Di KCL v2.3 dan v2.4, yang tidak mendukung`streamArm`, buat instance multistream dengan menggunakan format. `account-id:StreamName:streamCreationTimestamp` Format ini akan usang dan tidak lagi didukung dimulai dengan rilis utama berikutnya.

- MultistreamTracker juga mencakup strategi untuk menghapus sewa aliran lama di tabel sewa (). formerStreamsLeases DeletionStrategy Perhatikan bahwa strategi TIDAK DAPAT diubah selama runtime aplikasi konsumen. Untuk informasi lebih lanjut, lihat https://github.com/awslabs/amazon-kinesis-client/blob/0c5042dadf794fe988438436252a5a8fe70b6b0 b/amazon-kinesis-client/src/main/java/software/amazon/kinesis/processor/FormerStreamsLeasesDeletionStrategy .java.

•

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 multi-stream 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 multi-stream 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 partisi untuk tabel sewa) adalah ID pecahan. Jika Anda mengkonfigurasi ulang aplikasi konsumen KCL yang ada untuk memproses beberapa aliran data, itu merusak tabel sewa Anda, karena `leaseKey` strukturnya harus sebagai berikut: `account-id:StreamName:StreamCreationTimestamp:ShardId` untuk mendukung multi-aliran.

Gunakan registri AWS Glue Skema dengan KCL

Anda dapat mengintegrasikan Kinesis Data Streams AWS Glue dengan registri Schema. Registri AWS Glue Skema 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. Registri AWS Glue Skema memungkinkan Anda meningkatkan kualitas end-to-end data dan tata kelola data dalam aplikasi streaming Anda. Untuk informasi selengkapnya, lihat Skema Registri AWS Glue. Salah satu cara untuk mengatur integrasi ini adalah melalui KCL untuk Java.

> **Important**
>
> - AWS Glue Integrasi registri skema untuk Kinesis Data Streams hanya didukung di KCL 2.3 atau yang lebih baru.

- AWS Glue Integrasi registri skema untuk Kinesis Data Streams tidak didukung untuk konsumen KCL yang ditulis dalam bahasa non-Java yang dijalankan dengan. `multilangdaemon`
- AWS Glue Integrasi registri skema untuk Kinesis Data Streams tidak didukung dalam versi KCL 1.x apa pun.

Untuk petunjuk rinci tentang cara mengatur integrasi Kinesis Data Streamswith AWS Glue Schema registry menggunakan KCL, lihat bagian “Berinteraksi dengan Data Menggunakan KPL/KCL Pustaka” di Kasus Penggunaan: Mengintegrasikan Amazon Kinesis Data Streams dengan Schema Registry. AWS Glue

Izin IAM diperlukan untuk aplikasi konsumen KCL

Anda harus menambahkan izin berikut ke peran IAM atau pengguna yang terkait dengan aplikasi konsumen KCL Anda.

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

Tabel berikut menunjukkan izin IAM minimum yang umumnya diperlukan untuk aplikasi konsumen KCL:

Izin IAM minimum untuk aplikasi konsumen KCL

Layanan	Tindakan	Sumber daya (ARNs)	Tujuan

Amazon Kinesis Data Streams	`DescribeStream`
	
	`DescribeStreamSummary`
	
	`RegisterStreamConsumer`
		Kinesis data stream dari mana aplikasi KCL Anda akan memproses data.
		
		`arn:aws:kinesis:region:account:stream/StreamName`
			Sebelum mencoba membaca catatan, konsumen memeriksa apakah aliran data ada, apakah aktif, dan apakah pecahan terkandung dalam aliran data.

Layanan	Tindakan	Sumber daya (ARNs)	Tujuan
			Mendaftarkan konsumen ke pecahan.

Amazon Kinesis Data Streams	`GetRecords`
	
	`GetShardIterator`
	
	`ListShards`
		Kinesis data stream dari mana aplikasi KCL Anda akan memproses data.
		
		`arn:aws:kinesis:region:account:stream/StreamName`
			Membaca catatan dari pecahan.

Amazon Kinesis Data Streams	`SubscribeToShard`
	
	`DescribeStreamConsumer`
		Kinesis data stream dari mana aplikasi KCL Anda akan memproses data. Tambahkan tindakan ini hanya jika Anda menggunakan konsumen fan-out (EFO) yang disempurnakan.
		
		`arn:aws:kinesis:region:account:stream/StreamName/consumer/*`
			Berlangganan pecahan untuk konsumen fan-out (EFO) yang disempurnakan.

Layanan	Tindakan	Sumber daya (ARNs)	Tujuan

Amazon DynamoDB	`CreateTable`
	`DescribeTable`
	`UpdateTable`
	`Scan`
	`GetItem`
	`PutItem`
	`UpdateItem`
	`DeleteItem`
		Tabel sewa (tabel metadata di DynamoDB dibuat oleh KCL.
		`arn:aws:dynamodb:region:account:table/KCLApplicationName`
			Tindakan ini diperlukan agar KCL dapat menggunakan tabel sewa yang dibuat di DynamoDB.

Layanan	Tindakan	Sumber daya (ARNs)	Tujuan

Amazon DynamoDB	`CreateTable`
	
	`DescribeTable`
	
	`Scan`
	
	`GetItem`
	
	`PutItem`
	
	`UpdateItem`
	
	`DeleteItem`
		Metrik pekerja dan tabel status koordinator (tabel metadata di DynamoDB) dibuat oleh KCL.
		
		`arn:aws:dynamodb:region:account:table/KCLApplicationName-WorkerMetricStats`
		
		`arn:aws:dynamodb:region:account:table/KCLApplicationName-CoordinatorState`
			Tindakan ini diperlukan untuk KCL untuk mengelola metrik pekerja dan tabel metadata status koordinator di DynamoDB.

Amazon DynamoDB	`Query`	Indeks sekunder global pada tabel sewa. `arn:aws:dynamodb:region:account:table/KCLApplicationName/index/*`	Tindakan ini diperlukan agar KCL membaca indeks sekunder global dari tabel sewa yang dibuat di DynamoDB.

Layanan	Tindakan	Sumber daya (ARNs)	Tujuan
Amazon CloudWatch	`PutMetricData`	*	Unggah metrik CloudWatch yang berguna untuk memantau aplikasi. Tanda bintang (*) digunakan karena tidak ada sumber daya spesifik CloudWatch di mana `PutMetricData` tindakan dipanggil.

> **Note**
>
> Ganti “wilayah,” “akun,” “StreamName,” dan “KCLApplicationNama” ARNs dengan nama Anda sendiri Wilayah AWS, Akun AWS nomor, Kinesis data stream, dan nama aplikasi KCL masing-masing. KCL 3.x membuat dua tabel metadata lagi di DynamoDB. Untuk detail tentang tabel metadata DynamoDB yang dibuat oleh KCL, lihat. Tabel metadata DynamoDB dan load balancing di KCL Jika Anda menggunakan konfigurasi untuk menyesuaikan nama tabel metadata yang dibuat oleh KCL, gunakan nama tabel yang ditentukan, bukan nama aplikasi KCL.

Berikut ini adalah contoh dokumen kebijakan untuk aplikasi konsumen KCL.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kinesis:DescribeStream",
                "kinesis:DescribeStreamSummary",
```

```
                "kinesis:RegisterStreamConsumer",
                "kinesis:GetRecords",
                "kinesis:GetShardIterator",
                "kinesis:ListShards"
            ],
            "Resource": "arn:aws:kinesis:us-
east-1:123456789012:stream/STREAM_NAME"
        },
        {
            "Effect": "Allow",
            "Action": [
                "kinesis:SubscribeToShard",
                "kinesis:DescribeStreamConsumer"
            ],
            "Resource": "arn:aws:kinesis:us-
east-1:123456789012:stream/STREAM_NAME/consumer/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:CreateTable",
                "dynamodb:DescribeTable",
                "dynamodb:UpdateTable",
                "dynamodb:GetItem",
                "dynamodb:UpdateItem",
                "dynamodb:PutItem",
                "dynamodb:DeleteItem",
                "dynamodb:Scan"
            ],
            "Resource": [
            "arn:aws:dynamodb:us-east-1:123456789012:table/KCL_APPLICATION_NAME"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:CreateTable",
                "dynamodb:DescribeTable",
                "dynamodb:GetItem",
                "dynamodb:UpdateItem",
                "dynamodb:PutItem",
                "dynamodb:DeleteItem",
                "dynamodb:Scan"
            ],
```

```
            "Resource": [
            "arn:aws:dynamodb:us-east-1:123456789012:table/KCL_APPLICATION_NAME-
WorkerMetricStats",
    "arn:aws:dynamodb:us-east-1:123456789012:table/KCL_APPLICATION_NAME-
CoordinatorState"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:Query"
            ],
            "Resource": [
            "arn:aws:dynamodb:us-east-1:123456789012:table/KCL_APPLICATION_NAME/
index/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:PutMetricData"
            ],
            "Resource": "*"
        }
    ]
}
```

Sebelum Anda menggunakan kebijakan contoh ini, periksa item berikut:

- Ganti REGION dengan Anda Wilayah AWS (misalnya, us-east-1).
- Ganti ACCOUNT_ID dengan ID Anda. Akun AWS
- Ganti STREAM_NAME dengan nama aliran data Kinesis Anda.
- Ganti CONSUMER_NAME dengan nama konsumen Anda, biasanya nama aplikasi Anda saat menggunakan KCL.
- Ganti KCL_APPLICATION_NAME dengan nama aplikasi KCL Anda.

Konfigurasi KCL

Anda dapat mengatur properti konfigurasi untuk menyesuaikan fungsionalitas Perpustakaan Klien Kinesis untuk memenuhi persyaratan spesifik Anda. Tabel berikut menjelaskan properti konfigurasi dan kelas.

> **Important**
>
> Dalam KCL 3.x, algoritma load balancing bertujuan untuk mencapai pemanfaatan CPU yang merata di seluruh pekerja, bukan jumlah sewa yang sama per pekerja. Pengaturan `maxLeasesForWorker` terlalu rendah, Anda mungkin membatasi kemampuan KCL untuk menyeimbangkan beban kerja secara efektif. Jika Anda menggunakan `maxLeasesForWorker` konfigurasi, pertimbangkan untuk meningkatkan nilainya untuk memungkinkan distribusi beban terbaik.

Tabel ini menunjukkan properti konfigurasi untuk KCL

Properti konfigurasi	Kelas konfigurasi	Deskripsi	Nilai default
`applicationName`	ConfigsBuilder	Nama untuk ini aplikasi KCL. Digunakan sebagai default untuk `tableName` dan`consumerName` .	Tidak berlaku
`tableName`	ConfigsBuilder	Memungkinkan penggantian nama tabel yang digunakan untuk tabel sewa Amazon DynamoDB.	Tidak berlaku
`streamName`	ConfigsBuilder	Nama aliran tempat aplikasi ini memproses catatan dari.	Tidak berlaku

Properti konfigurasi	Kelas konfigurasi	Deskripsi	Nilai default
`workerIdentifier`	ConfigsBuilder	Pengenal unik yang mewakili instantiasi prosesor aplikasi ini. Ini pasti unik.	Tidak berlaku
`failoverTimeMillis`	LeaseManagementConfig	Jumlah milidetik yang harus dilewati sebelum Anda dapat mempertimbangkan pemilik sewa telah gagal. Untuk aplikasi yang memiliki sejumlah besar pecahan, ini dapat diatur ke angka yang lebih tinggi untuk mengurangi jumlah DynamoDB IOPS yang diperlukan untuk melacak sewa.	10.000 (10 detik)
`shardSyncIntervalMillis`	LeaseManagementConfig	Waktu antara panggilan sinkronisasi shard.	60.000 (60 detik)
`cleanupLeasesUponShardCompletion`	LeaseManagementConfig	Ketika ditetapkan, sewa dihapus segera setelah sewa anak mulai diproses.	BETUL

Properti konfigurasi	Kelas konfigurasi	Deskripsi	Nilai default
`ignoreUnexpectedChildShards`	LeaseManagementConfig	Saat diatur, pecahan anak yang memiliki pecahan terbuka diabaikan. Ini terutama untuk DynamoDB Streams.	SALAH
`maxLeasesForWorker`	LeaseManagementConfig	Jumlah maksimum sewa yang harus diterima oleh satu pekerja. Mengaturnya terlalu rendah dapat menyebabkan kehilangan data jika pekerja tidak dapat memproses semua pecahan, dan menyebabkan penugasan sewa yang kurang optimal di antara pekerja. Pertimbangkan jumlah pecahan total, jumlah pekerja, dan kapasitas pemrosesan pekerja saat mengonfigurasinya.	Tidak terbatas.

Properti konfigurasi	Kelas konfigurasi	Deskripsi	Nilai default
`maxLeaseRenewalThreads`	LeaseManagementConfig	Mengontrol ukuran kumpulan utas penyewa penyewaan. Semakin banyak sewa yang dapat diambil aplikasi Anda, semakin besar kumpulan ini seharusnya.	20
`billingMode`	LeaseManagementConfig	Menentukan mode kapasitas tabel sewa yang dibuat di DynamoDB. Ada dua opsi: mode on-demand (PAY_PER_REQUEST) dan mode yang disediakan. Sebaiknya gunakan pengaturan default mode on-demand karena secara otomatis menskalakan untuk mengakomodasi beban kerja Anda tanpa perlu perencanaan kapasitas.	PAY_PER_REQUEST (mode sesuai permintaan)

Properti konfigurasi	Kelas konfigurasi	Deskripsi	Nilai default
`initialLeaseTableReadCapacity`	LeaseManagementConfig	Kapasitas baca DynamoDB yang digunakan jika Perpustakaan Klien Kinesis perlu membuat tabel sewa DynamoDB baru dengan mode kapasitas yang disediakan. Anda dapat mengabaikan konfigurasi ini jika Anda menggunakan mode kapasitas sesuai permintaan default dalam `billingMode` konfigurasi.	10

Properti konfigurasi	Kelas konfigurasi	Deskripsi	Nilai default
`initialLeaseTableWriteCapacity`	LeaseManagementConfig	Kapasitas baca DynamoDB yang digunakan jika Perpustakaan Klien Kinesis perlu membuat tabel sewa DynamoDB baru. Anda dapat mengabaikan konfigurasi ini jika Anda menggunakan mode kapasitas sesuai permintaan default dalam `billingMode` konfigurasi.	10
`initialPositionInStreamExtended`	LeaseManagementConfig	Posisi awal dalam aliran tempat aplikasi harus dimulai. Ini hanya digunakan selama pembuatan sewa awal.	InitialPositionInStream.TRIM_HORIZON

Properti konfigurasi	Kelas konfigurasi	Deskripsi	Nilai default
`reBalance Threshold Percentage`	LeaseMana gementConfig	Nilai persentase yang menentuka n kapan algoritma load balancing harus mempertimbangkan untuk menetapkan kembali pecahan di antara pekerja. Ini adalah konfigurasi baru yang diperkena lkan di KCL 3.x.	10
`dampening Percentage`	LeaseMana gementConfig	Nilai persentase yang digunakan untuk meredam jumlah beban yang akan dipindahkan dari pekerja yang kelebihan beban dalam satu operasi penyeimbangan kembali. Ini adalah konfigurasi baru yang diperkena lkan di KCL 3.x.	60

Properti konfigurasi	Kelas konfigurasi	Deskripsi	Nilai default
`allowThroughputOvershoot`	LeaseManagementConfig	Menentukan apakah sewa tambahan masih perlu diambil dari pekerja yang kelebihan beban bahkan jika itu menyebabkan jumlah total throughput sewa yang diambil melebihi jumlah throughput yang diinginkan. Ini adalah konfigurasi baru yang diperkenalkan di KCL 3.x.	BETUL
`disableWorkerMetrics`	LeaseManagementConfig	Menentukan apakah KCL harus mengabaikan metrik sumber daya dari pekerja (seperti pemanfaatan CPU) saat menetapkan kembali sewa dan penyeimbangan beban. Setel ini ke TRUE jika Anda ingin mencegah KCL dari load balancing berdasarkan pemanfaatan CPU. Ini adalah konfigurasi baru yang diperkenalkan di KCL 3.x.	SALAH

Properti konfigurasi	Kelas konfigurasi	Deskripsi	Nilai default
`maxThroughputPerHostKBps`	LeaseManagementConfig	Jumlah throughput maksimum untuk ditugaskan kepada pekerja selama penugasan sewa. Ini adalah konfigurasi baru yang diperkenalkan di KCL 3.x.	Tidak terbatas.

Properti konfigurasi	Kelas konfigurasi	Deskripsi	Nilai default
`isGracefulLeaseHandoffEnabled`	LeaseManagementConfig	Mengontrol perilaku serah terima sewa antar pekerja. Ketika disetel ke true, KCL akan mencoba untuk mentransfer sewa dengan anggun dengan memberikan waktu pecahan yang RecordProcessor cukup untuk menyelesaikan pemrosesan sebelum menyerahkan sewa kepada pekerja lain. Ini dapat membantu memastikan integritas data dan transisi yang lancar tetapi dapat meningkatkan waktu handoff. Ketika disetel ke false, sewa akan segera diserahkan tanpa menunggu RecordProcessor untuk ditutup dengan anggun. Hal ini dapat menyebabkan handoff lebih cepat tetapi dapat berisiko pemrosesan yang tidak lengkap.	BETUL

Properti konfigurasi	Kelas konfigurasi	Deskripsi	Nilai default
		Catatan: Checkpoin ting harus diimpleme ntasikan di dalam metode ShutdownR equested () RecordProcessor untuk mendapatk an manfaat dari fitur handoff sewa yang anggun. Ini adalah konfigurasi baru yang diperkena lkan di KCL 3.x.	

Properti konfigurasi	Kelas konfigurasi	Deskripsi	Nilai default
`gracefulLeaseHandoffTimeoutMillis`	LeaseManagementConfig	Menentukan waktu minimum (dalam milidetik) untuk menunggu pecahan saat ini ditutup dengan anggun sebelum secara paksa mentransfer sewa ke pemilik berikutnya. RecordProcessor Jika metode ProcessRecords Anda biasanya berjalan lebih lama dari nilai default, pertimbangkan untuk meningkatkan setelan ini. Ini memastikan RecordProcessor memiliki waktu yang cukup untuk menyelesaikan pemrosesannya sebelum transfer sewa terjadi. Ini adalah konfigurasi baru yang diperkenalkan di KCL 3.x.	30.000 (30 detik)

Properti konfigurasi	Kelas konfigurasi	Deskripsi	Nilai default
`maxRecords`	PollingConfig	Memungkinkan pengaturan jumlah maksimum catatan yang Kinesis kembalikan.	10.000
`retryGetRecordsInSeconds`	PollingConfig	Mengkonfigurasi penundaan antara GetRecords upaya untuk kegagalan.	Tidak ada
`maxGetRecordsThreadPool`	PollingConfig	Ukuran kolam benang yang digunakan untuk GetRecords.	Tidak ada
`idleTimeBetweenReadsInMillis`	PollingConfig	Menentukan berapa lama KCL menunggu di antara GetRecords panggilan untuk polling data dari aliran data. Satuannya adalah milidetik.	1.500
`callProcessRecordsEvenForEmptyRecordList`	ProcessorConfig	Ketika ditetapkan, prosesor rekaman dipanggil bahkan ketika tidak ada catatan yang diberikan dari Kinesis.	SALAH

Properti konfigurasi	Kelas konfigurasi	Deskripsi	Nilai default
`parentSha rdPollInt ervalMillis`	CoordinatorConfig	Seberapa sering prosesor rekaman harus melakukan polling untuk melihat apakah pecahan induk telah selesai. Satuannya adalah milidetik.	10.000 (10 detik)
`skipShard SyncAtWor kerInitia lizationI fLeaseExist`	CoordinatorConfig	Nonaktifkan sinkronis asi data pecahan jika tabel sewa berisi sewa yang ada.	SALAH
`shardPrio ritization`	CoordinatorConfig	Prioritas pecahan mana yang akan digunakan.	NoOpShardPrioritiz ation

Properti konfigurasi	Kelas konfigurasi	Deskripsi	Nilai default
`ClientVersionConfig`	CoordinatorConfig	Menentukan mode kompatibilitas versi KCL mana aplikasi akan berjalan. Konfigurasi ini hanya untuk migrasi dari versi KCL sebelumnya. Saat bermigrasi ke 3.x, Anda perlu mengatur konfigurasi ini ke. `CLIENT_VERSION_CONFIG_COMPATIBLE_WITH_2X` Anda dapat menghapus konfigurasi ini saat Anda menyelesaikan migrasi.	CLIENT_VERSION_CONFIG_3X
`taskBackoffTimeMillis`	LifecycleConfig	Waktu menunggu untuk mencoba lagi tugas KCL gagal. Satuannya adalah milidetik.	500 (0,5 detik)
`logWarningForTaskAfterMillis`	LifecycleConfig	Berapa lama menunggu sebelum peringatan dicatat jika tugas belum selesai.	Tidak ada

Properti konfigurasi	Kelas konfigurasi	Deskripsi	Nilai default
`listShardsBackoffTimeInMillis`	RetrievalConfig	Jumlah milidetik untuk menunggu di antara panggilan ke `ListShards` saat kegagalan terjadi. Satuannya adalah milidetik.	1.500 (1,5 detik)
`maxListShardsRetryAttempts`	RetrievalConfig	Jumlah maksimum kali yang `ListShards` mencoba lagi sebelum menyerah.	50
`metricsBufferTimeMillis`	MetricsConfig	Menentukan durasi maksimum (dalam milidetik) untuk menyangga metrik sebelum mempublikasikannya. CloudWatch	10.000 (10 detik)
`metricsMaxQueueSize`	MetricsConfig	Menentukan jumlah maksimum metrik untuk buffer sebelum dipublikasikan ke. CloudWatch	10.000

Properti konfigurasi	Kelas konfigurasi	Deskripsi	Nilai default
`metricsLevel`	MetricsConfig	Menentukan tingkat granularitas CloudWatch metrik yang akan diaktifkan dan dipublikasikan. Nilai yang mungkin: TIDAK ADA, RINGKASAN, DETAIL.	MetricsLevel.RINCI
`metricsEnabledDimensions`	MetricsConfig	Kontrol memungkinkan dimensi untuk CloudWatch Metrik.	Semua dimensi

Konfigurasi yang dihentikan di KCL 3.x

Properti konfigurasi berikut dihentikan di KCL 3.x:

Tabel menunjukkan properti konfigurasi yang dihentikan untuk KCL 3.x

Properti konfigurasi	Kelas konfigurasi	Deskripsi
`maxLeasesToStealAtOneTime`	LeaseManagementConfig	Jumlah maksimum sewa aplikasi harus mencoba untuk mencuri pada satu waktu. KCL 3.x akan mengabaikan konfigurasi ini dan menetapkan kembali sewa berdasarkan pemanfaatan sumber daya pekerja.
`enablePriorityLeaseAssignment`	LeaseManagementConfig	Mengontrol apakah pekerja harus memprioritaskan pengambilan sewa yang sangat kedaluwarsa (sewa

Properti konfigurasi	Kelas konfigurasi	Deskripsi
		tidak diperpanjang untuk 3x waktu failover) dan sewa pecahan baru, terlepas dari jumlah sewa target tetapi masih menghormati batas sewa maksimum. KCL 3.x akan mengabaikan konfigura si ini dan selalu menyebarkan sewa kedaluwarsa ke seluruh pekerja.

Important

Anda masih harus memiliki properti konfigurasi yang tidak sesuai selama migrasi dari versi KCL sebelumnya ke KCL 3.x. Selama migrasi, pekerja KCL pertama-tama akan memulai dengan mode kompatibel KCL 2.x dan beralih ke mode fungsionalitas KCL 3.x ketika mendeteksi bahwa semua pekerja KCL aplikasi siap menjalankan KCL 3.x. Konfigurasi yang dihentikan ini diperlukan saat pekerja KCL menjalankan mode yang kompatibel dengan KCL 2.x.

Kebijakan siklus hidup versi KCL

Topik ini menguraikan kebijakan siklus hidup versi untuk Amazon Kinesis Client Library (KCL). AWS secara teratur menyediakan rilis baru untuk versi KCL untuk mendukung fitur dan penyempurnaan baru, perbaikan bug, patch keamanan, dan pembaruan ketergantungan. Kami menyarankan Anda tetap up-to-date menggunakan versi KCL untuk mengikuti fitur-fitur terbaru, pembaruan keamanan, dan dependensi yang mendasarinya. Kami tidak menyarankan untuk terus menggunakan versi KCL yang tidak didukung.

Siklus hidup untuk versi KCL utama terdiri dari tiga fase berikut:

- Ketersediaan umum (GA) — Selama fase ini, versi utama didukung sepenuhnya. AWS menyediakan rilis versi minor dan patch reguler yang mencakup dukungan untuk fitur baru atau pembaruan API untuk Kinesis Data Streams, serta perbaikan bug dan keamanan.

- Mode pemeliharaan — AWS membatasi rilis versi patch untuk mengatasi perbaikan bug kritis dan masalah keamanan saja. Versi utama tidak akan menerima pembaruan untuk fitur baru atau APIs Kinesis Data Streams.
- E nd-of-support — Versi utama tidak akan lagi menerima pembaruan atau rilis. Rilis yang diterbitkan sebelumnya akan terus tersedia melalui manajer paket publik dan kode akan tetap aktif GitHub. Penggunaan versi yang telah end-of-support dicapai dilakukan atas kebijaksanaan pengguna. Kami menyarankan Anda meningkatkan ke versi utama terbaru.

Versi utama	Fase saat ini	Tanggal rilis	Tanggal mode pemeliharaan	End-of-support tanggal
KCL 1.x	Mode pemeliharaan	2013-12-19	2025-04-17	2026-01-30
KCL 2.x	Ketersediaan umum	2018-08-02	--	--
KCL 3.x	Ketersediaan umum	2024-11-06	--	--

Migrasi dari versi KCL sebelumnya

Topik ini menjelaskan cara bermigrasi dari versi Kinesis Client Library (KCL) sebelumnya.

Apa yang baru KCL 3.0?

Kinesis Client Library (KCL) 3.0 memperkenalkan beberapa perangkat tambahan utama dibandingkan dengan versi sebelumnya:

- Ini menurunkan biaya komputasi untuk aplikasi konsumen dengan secara otomatis mendistribusikan kembali pekerjaan dari pekerja yang terlalu dimanfaatkan ke pekerja yang kurang dimanfaatkan dalam armada aplikasi konsumen. Algoritma load balancing baru ini memastikan pemanfaatan CPU yang didistribusikan secara merata di seluruh pekerja dan menghilangkan kebutuhan untuk menyediakan pekerja yang berlebihan.
- Ini mengurangi biaya DynamoDB yang terkait dengan KCL dengan mengoptimalkan operasi baca pada tabel sewa.

- Ini meminimalkan pemrosesan ulang data ketika sewa dipindahkan ke pekerja lain dengan memungkinkan pekerja saat ini menyelesaikan pemeriksaan catatan yang telah diproses.
- Ini digunakan AWS SDK for Java 2.x untuk meningkatkan kinerja dan fitur keamanan, sepenuhnya menghapus ketergantungan pada AWS SDK untuk Java 1.x.

Untuk informasi selengkapnya, lihat catatan rilis KCL 3.0.

Topik

- Migrasi dari KCL 2.x ke KCL 3.x
- Putar kembali ke versi KCL sebelumnya
- Gulung maju ke KCL 3.x setelah rollback
- Praktik terbaik untuk tabel sewa dengan mode kapasitas yang disediakan
- Migrasi dari KCL 1.x ke KCL 3.x

Migrasi dari KCL 2.x ke KCL 3.x

Topik ini memberikan step-by-step petunjuk untuk memigrasikan konsumen Anda dari KCL 2.x ke KCL 3.x. KCL 3.x mendukung migrasi di tempat konsumen KCL 2.x. Anda dapat terus mengkonsumsi data dari aliran data Kinesis Anda sambil memigrasikan pekerja Anda secara bergulir.

> **Important**
>
> KCL 3.x mempertahankan antarmuka dan metode yang sama seperti KCL 2.x. Oleh karena itu, Anda tidak perlu memperbarui kode pemrosesan catatan selama migrasi. Namun, Anda harus mengatur konfigurasi yang tepat dan memeriksa langkah-langkah yang diperlukan untuk migrasi. Kami sangat menyarankan Anda mengikuti langkah-langkah migrasi berikut untuk pengalaman migrasi yang lancar.

Langkah 1: Prasyarat

Sebelum Anda mulai menggunakan KCL 3.x, pastikan Anda memiliki yang berikut:

- Java Development Kit (JDK) 8 atau lebih baru
- AWS SDK untuk Java 2.x
- Maven atau Gradle untuk manajemen ketergantungan

> **Important**
>
> Jangan gunakan AWS SDK untuk Java versi 2.27.19 hingga 2.27.23 dengan KCL 3.x. Versi ini menyertakan masalah yang menyebabkan kesalahan pengecualian terkait dengan penggunaan DynamoDB KCL. Kami menyarankan Anda menggunakan AWS SDK untuk Java versi 2.28.0 atau yang lebih baru untuk menghindari masalah ini.

Langkah 2: Tambahkan dependensi

Jika Anda menggunakan Maven, tambahkan dependensi berikut ke file Anda. `pom.xml` Pastikan Anda mengganti 3.xx ke versi KCL terbaru.

```
<dependency>
    <groupId>software.amazon.kinesis</groupId>
    <artifactId>amazon-kinesis-client</artifactId>
    <version>3.x.x</version> <!-- Use the latest version -->
</dependency>
```

Jika Anda menggunakan Gradle, tambahkan berikut ini ke `build.gradle` file Anda. Pastikan Anda mengganti 3.xx ke versi KCL terbaru.

```
implementation 'software.amazon.kinesis:amazon-kinesis-client:3.x.x'
```

Anda dapat memeriksa versi terbaru KCL di Repositori Pusat Maven.

Langkah 3: Siapkan konfigurasi terkait migrasi

Untuk bermigrasi dari KCL 2.x ke KCL 3.x, Anda harus mengatur parameter konfigurasi berikut:

- CoordinatorConfig. clientVersionConfig: Konfigurasi ini menentukan mode kompatibilitas versi KCL mana aplikasi akan berjalan. Saat bermigrasi dari KCL 2.x ke 3.x, Anda perlu mengatur konfigurasi ini ke. `CLIENT_VERSION_CONFIG_COMPATIBLE_WITH_2X` Untuk mengatur konfigurasi ini, tambahkan baris berikut saat membuat objek scheduler Anda:

```
configsBuilder.coordiantorConfig().clientVersionConfig(ClientVersionConfig.CLIENT_VERSION_CONFI
```

Berikut ini adalah contoh cara mengatur `CoordinatorConfig.clientVersionConfig` untuk bermigrasi dari KCL 2.x ke 3.x. Anda dapat menyesuaikan konfigurasi lain sesuai kebutuhan berdasarkan kebutuhan spesifik Anda:

```
Scheduler scheduler = new Scheduler(
    configsBuilder.checkpointConfig(),

 configsBuilder.coordiantorConfig().clientVersionConfig(ClientVersionConfig.CLIENT_VERSION_CONF
    configsBuilder.leaseManagementConfig(),
    configsBuilder.lifecycleConfig(),
    configsBuilder.metricsConfig(),
    configsBuilder.processorConfig(),
    configsBuilder.retrievalConfig()
);
```

Sangat penting bahwa semua pekerja dalam aplikasi konsumen Anda menggunakan algoritma load balancing yang sama pada waktu tertentu karena KCL 2.x dan 3.x menggunakan algoritma load balancing yang berbeda. Menjalankan pekerja dengan algoritma load balancing yang berbeda dapat menyebabkan distribusi beban suboptimal karena kedua algoritma beroperasi secara independen.

Pengaturan kompatibilitas KCL 2.x ini memungkinkan aplikasi KCL 3.x Anda berjalan dalam mode yang kompatibel dengan KCL 2.x dan menggunakan algoritma load balancing untuk KCL 2.x hingga semua pekerja di aplikasi konsumen Anda telah ditingkatkan ke KCL 3.x. Ketika migrasi selesai, KCL akan secara otomatis beralih ke mode fungsionalitas KCL 3.x penuh dan mulai menggunakan algoritma penyeimbangan beban KCL 3.x baru untuk semua pekerja yang sedang berjalan.

> **Important**
>
> Jika Anda tidak menggunakan `ConfigsBuilder` tetapi membuat `LeaseManagementConfig` objek untuk mengatur konfigurasi, Anda harus menambahkan satu parameter lagi yang disebut `applicationName` dalam KCL versi 3.x atau yang lebih baru. Untuk detailnya, lihat Kesalahan kompilasi dengan LeaseManagementConfig konstruktor. Kami merekomendasikan penggunaan `ConfigsBuilder` untuk mengatur konfigurasi KCL. `ConfigsBuilder`menyediakan cara yang lebih fleksibel dan dapat dipelihara untuk mengkonfigurasi aplikasi KCL Anda.

Langkah 4: Ikuti praktik terbaik untuk implementasi metode ShutdownRequested ()

KCL 3.x memperkenalkan fitur yang disebut handoff sewa anggun untuk meminimalkan pemrosesan ulang data ketika sewa diserahkan kepada pekerja lain sebagai bagian dari proses penggantian sewa. Ini dicapai dengan memeriksa nomor urut yang diproses terakhir di tabel sewa sebelum serah terima sewa. Untuk memastikan handoff sewa anggun berfungsi dengan baik, Anda harus memastikan bahwa Anda memanggil `checkpointer` objek dalam metode di kelas Anda. `shutdownRequested` `RecordProcessor` Jika Anda tidak memanggil `checkpointer` objek dalam `shutdownRequested` metode, Anda dapat menerapkannya seperti yang diilustrasikan dalam contoh berikut.

⚠ Important

- Contoh implementasi berikut adalah persyaratan minimal untuk handoff sewa yang anggun. Anda dapat memperluasnya untuk menyertakan logika tambahan yang terkait dengan pos pemeriksaan jika diperlukan. Jika Anda melakukan pemrosesan asinkron, pastikan semua catatan yang dikirimkan ke hilir diproses sebelum menjalankan checkpointing.
- Sementara handoff sewa yang anggun secara signifikan mengurangi kemungkinan pemrosesan ulang data selama transfer sewa, itu tidak sepenuhnya menghilangkan kemungkinan ini. Untuk menjaga integritas dan konsistensi data, rancang aplikasi konsumen hilir Anda menjadi idempoten. Ini berarti mereka harus dapat menangani pemrosesan rekaman duplikat potensial tanpa efek buruk pada keseluruhan sistem.

```
/**
 * Invoked when either Scheduler has been requested to gracefully shutdown
 * or lease ownership is being transferred gracefully so the current owner
 * gets one last chance to checkpoint.
 *
 * Checkpoints and logs the data a final time.
 *
 * @param shutdownRequestedInput Provides access to a checkpointer, allowing a record
 processor to checkpoint
 *                               before the shutdown is completed.
 */
public void shutdownRequested(ShutdownRequestedInput shutdownRequestedInput) {
    try {
       // Ensure that all delivered records are processed
```

```
        // and has been successfully flushed to the downstream before calling
        // checkpoint
        // If you are performing any asynchronous processing or flushing to
        // downstream, you must wait for its completion before invoking
        // the below checkpoint method.
         log.info("Scheduler is shutting down, checkpointing.");
         shutdownRequestedInput.checkpointer().checkpoint();
    } catch (ShutdownException | InvalidStateException e) {
         log.error("Exception while checkpointing at requested shutdown. Giving up.",
 e);
    }
}
```

Langkah 5: Periksa prasyarat KCL 3.x untuk mengumpulkan metrik pekerja

KCL 3.x mengumpulkan metrik pemanfaatan CPU seperti pemanfaatan CPU dari pekerja untuk menyeimbangkan beban di seluruh pekerja secara merata. Pekerja aplikasi konsumen dapat berjalan di Amazon EC2, Amazon ECS, Amazon EKS, atau. AWS Fargate KCL 3.x dapat mengumpulkan metrik pemanfaatan CPU dari pekerja hanya jika prasyarat berikut terpenuhi:

Amazon Elastic Compute Cloud(Amazon EC2)

- Sistem operasi Anda harus OS Linux.
- Anda harus mengaktifkan IMDSv2di instans EC2 Anda.

Amazon Elastic Container Service (Amazon ECS) di Amazon EC2

- Sistem operasi Anda harus OS Linux.
- Anda harus mengaktifkan titik akhir metadata tugas ECS versi 4.
- Versi agen penampung Amazon ECS Anda harus 1.39.0 atau yang lebih baru.

Amazon ECS aktif AWS Fargate

- Anda harus mengaktifkan titik akhir metadata tugas Fargate versi 4. Jika Anda menggunakan platform Fargate versi 1.4.0 atau yang lebih baru, ini diaktifkan secara default.
- Platform Fargate versi 1.4.0 atau yang lebih baru.

Layanan Amazon Elastic Kubernetes (Amazon EKS) di Amazon EC2

- Sistem operasi Anda harus OS Linux.

Amazon EKS di AWS Fargate

- Platform Fargate 1.3.0 atau yang lebih baru.

> **Important**
>
> Jika KCL 3.x tidak dapat mengumpulkan metrik pemanfaatan CPU dari pekerja karena prasyarat tidak terpenuhi, itu akan menyeimbangkan kembali beban tingkat throughput per sewa. Mekanisme penyeimbangan kembali ini akan memastikan semua pekerja akan mendapatkan tingkat throughput total yang sama dari sewa yang diberikan kepada setiap pekerja. Untuk informasi selengkapnya, lihat Bagaimana KCL memberikan sewa kepada pekerja dan menyeimbangkan beban.

Langkah 6: Perbarui izin IAM untuk KCL 3.x

Anda harus menambahkan izin berikut ke peran atau kebijakan IAM yang terkait dengan aplikasi konsumen KCL 3.x Anda. Ini melibatkan memperbarui kebijakan IAM yang ada yang digunakan oleh aplikasi KCL. Untuk informasi selengkapnya, lihat Izin IAM diperlukan untuk aplikasi konsumen KCL.

> **Important**
>
> Aplikasi KCL Anda yang ada mungkin tidak memiliki tindakan dan sumber daya IAM berikut yang ditambahkan dalam kebijakan IAM karena tidak diperlukan di KCL 2.x. Pastikan Anda telah menambahkannya sebelum menjalankan aplikasi KCL 3.x Anda:
>
> - Tindakan: `UpdateTable`
>   - Sumber daya (ARNs): `arn:aws:dynamodb:region:account:table/KCLApplicationName`
> - Tindakan: `Query`
>   - Sumber daya (ARNs): `arn:aws:dynamodb:region:account:table/KCLApplicationName/index/*`
> - Tindakan:`CreateTable`,`DescribeTable`,`Scan`,`GetItem`,`PutItem`,`UpdateItem`, `DeleteItem`

- Sumber daya (ARNs):`arn:aws:dynamodb:region:account:table/KCLApplicationName-WorkerMetricStats`, `arn:aws:dynamodb:region:account:table/KCLApplicationName-CoordinatorState`

  Ganti “wilayah,” “akun,” dan “KCLApplicationNama” di ARNs dengan nama aplikasi Anda sendiri Wilayah AWS, Akun AWS nomor, dan KCL masing-masing. Jika Anda menggunakan konfigurasi untuk menyesuaikan nama tabel metadata yang dibuat oleh KCL, gunakan nama tabel yang ditentukan, bukan nama aplikasi KCL.

Langkah 7: Menyebarkan kode KCL 3.x ke pekerja Anda

Setelah Anda mengatur konfigurasi yang diperlukan untuk migrasi dan menyelesaikan semua daftar periksa migrasi sebelumnya, Anda dapat membuat dan menerapkan kode Anda ke pekerja Anda.

> **Note**
>
> Jika Anda melihat kesalahan kompilasi dengan `LeaseManagementConfig` konstruktor, lihat Kesalahan kompilasi dengan LeaseManagementConfig konstruktor untuk informasi pemecahan masalah.

Langkah 8: Selesaikan migrasi

Selama penyebaran kode KCL 3.x, KCL terus menggunakan algoritma penetapan sewa dari KCL 2.x. Ketika Anda telah berhasil menerapkan kode KCL 3.x ke semua pekerja Anda, KCL secara otomatis mendeteksi ini dan beralih ke algoritme penetapan sewa baru berdasarkan pemanfaatan sumber daya pekerja. Untuk detail selengkapnya tentang algoritme penetapan sewa baru, lihat. Bagaimana KCL memberikan sewa kepada pekerja dan menyeimbangkan beban

Selama penerapan, Anda dapat memantau proses migrasi dengan metrik berikut yang dipancarkan. CloudWatch Anda dapat memantau metrik di bawah `Migration` operasi. Semua metrik adalah per-KCL-application metrik dan disetel ke tingkat `SUMMARY` metrik. Jika `Sum` statistik `CurrentState:3xWorker` metrik cocok dengan jumlah total pekerja dalam aplikasi KCL Anda, ini menunjukkan bahwa migrasi ke KCL 3.x telah berhasil diselesaikan.

> **Important**
>
> Dibutuhkan setidaknya 10 menit bagi KCL untuk beralih ke algoritme penugasan leasee baru setelah semua pekerja siap menjalankannya.

CloudWatch metrik untuk proses migrasi KCL

Metrik-metrik	Deskripsi
`CurrentState:3xWorker`	Jumlah pekerja KCL berhasil bermigrasi ke KCL 3.x dan menjalankan algoritma penugasan sewa baru. Jika `Sum` hitungan metrik ini cocok dengan jumlah total pekerja Anda, ini menunjukkan bahwa migrasi ke KCL 3.x telah berhasil diselesaikan. • Tingkat metrik: Ringkasan • Unit: Hitungan • Statistik: Statistik yang paling berguna adalah `Sum`
`CurrentState:2xCompatibleWorker`	Jumlah pekerja KCL yang berjalan dalam mode kompatibel KCL 2.x selama proses migrasi. Nilai bukan nol untuk metrik ini menunjukkan bahwa migrasi masih berlangsung. • Tingkat metrik: Ringkasan • Unit: Hitungan • Statistik: Statistik yang paling berguna adalah `Sum`
`Fault`	Jumlah pengecualian yang ditemui selama proses migrasi. Sebagian besar pengecual ian ini adalah kesalahan sementara, dan KCL 3.x akan secara otomatis mencoba lagi untuk menyelesaikan migrasi. Jika Anda mengamati nilai `Fault` metrik persisten, tinjau log Anda

Metrik-metrik	Deskripsi
	dari periode migrasi untuk pemecahan masalah lebih lanjut. Jika masalah berlanjut, hubungi Dukungan. • Tingkat metrik: Ringkasan • Unit: Hitungan • Statistik: Statistik yang paling berguna adalah `Sum`
`GsiStatusReady`	Status penciptaan indeks sekunder global (GSI) pada tabel sewa. Metrik ini menunjukkan apakah GSI pada tabel sewa telah dibuat, prasyarat untuk menjalankan KCL 3.x. Nilainya adalah 0 atau 1, dengan 1 menunjukkan penciptaan yang berhasil. Selama keadaan rollback, metrik ini tidak akan dipancarkan. Setelah Anda maju lagi, Anda dapat melanjutkan pemantauan metrik ini. • Tingkat metrik: Ringkasan • Unit: Hitungan • Statistik: Statistik yang paling berguna adalah `Sum`

Metrik-metrik	Deskripsi
`workerMetricsReady`	Status emisi metrik pekerja dari semua pekerja. Metrik menunjukkan apakah semua pekerja memancarkan metrik seperti pemanfaat an CPU. Nilainya adalah 0 atau 1, dengan 1 menunjukkan semua pekerja berhasil memancarkan metrik dan siap untuk algoritme penetapan sewa baru. Selama keadaan rollback, metrik ini tidak akan dipancarkan. Setelah Anda maju lagi, Anda dapat melanjutk an pemantauan metrik ini. • Tingkat metrik: Ringkasan • Unit: Hitungan • Statistik: Statistik yang paling berguna adalah `Sum`

KCL menyediakan kemampuan rollback ke mode kompatibel 2.x selama migrasi. Setelah migrasi berhasil ke KCL 3.x berhasil, kami sarankan Anda menghapus `CoordinatorConfig.clientVersionConfig` pengaturan `CLIENT_VERSION_CONFIG_COMPATIBLE_WITH_2X` jika rollback tidak lagi diperlukan. Menghapus konfigurasi ini menghentikan emisi metrik terkait migrasi dari aplikasi KCL.

> **Note**
>
> Sebaiknya Anda memantau kinerja dan stabilitas aplikasi selama suatu periode selama migrasi dan setelah menyelesaikan migrasi. Jika Anda mengamati masalah apa pun, Anda dapat mengembalikan pekerja untuk menggunakan fungsionalitas yang kompatibel dengan KCL 2.x menggunakan Alat Migrasi KCL.

Putar kembali ke versi KCL sebelumnya

Topik ini menjelaskan langkah-langkah untuk mengembalikan konsumen Anda kembali ke versi sebelumnya. Ketika Anda perlu memutar kembali, ada proses dua langkah:

1. Jalankan Alat Migrasi KCL.
2. Menerapkan ulang kode versi KCL sebelumnya (opsional).

Langkah 1: Jalankan Alat Migrasi KCL

Ketika Anda perlu memutar kembali ke versi KCL sebelumnya, Anda harus menjalankan Alat Migrasi KCL. Alat Migrasi KCL melakukan dua tugas penting:

- Ini menghapus tabel metadata yang disebut tabel metrik pekerja dan indeks sekunder global pada tabel sewa di DynamoDB. Kedua artefak ini dibuat oleh KCL 3.x tetapi tidak diperlukan saat Anda memutar kembali ke versi sebelumnya.
- Itu membuat semua pekerja berjalan dalam mode yang kompatibel dengan KCL 2.x dan mulai menggunakan algoritma load balancing yang digunakan dalam versi KCL sebelumnya. Jika Anda memiliki masalah dengan algoritme penyeimbangan beban baru di KCL 3.x, ini akan segera mengurangi masalah.

Important

Tabel status koordinator di DynamoDB harus ada dan tidak boleh dihapus selama proses migrasi, rollback, dan rollforward.

Note

Sangat penting bahwa semua pekerja dalam aplikasi konsumen Anda menggunakan algoritma load balancing yang sama pada waktu tertentu. Alat Migrasi KCL memastikan bahwa semua pekerja di aplikasi konsumen KCL 3.x Anda beralih ke mode yang kompatibel dengan KCL 2.x sehingga semua pekerja menjalankan algoritme penyeimbangan beban yang sama selama pelunasan bergulir kembali ke versi KCL Anda sebelumnya.

Anda dapat mengunduh Alat Migrasi KCL di direktori skrip repositori GitHubKCL. Skrip dapat dijalankan dari pekerja Anda atau host mana pun yang memiliki izin yang diperlukan untuk menulis ke tabel status koordinator, menghapus tabel metrik pekerja, dan memperbarui tabel sewa. Anda dapat merujuk ke izin IAM yang diperlukan Izin IAM diperlukan untuk aplikasi konsumen KCL untuk

menjalankan skrip. Anda harus menjalankan skrip hanya sekali per aplikasi KCL. Anda dapat menjalankan KCL Migration Tool dengan perintah berikut:

```
python3 ./KclMigrationTool.py --region <region> --mode rollback [--
application_name <applicationName>] [--lease_table_name <leaseTableName>] [--
coordinator_state_table_name <coordinatorStateTableName>] [--worker_metrics_table_name
 <workerMetricsTableName>]
```

Parameter

- --region: Ganti `<region>` dengan Anda Wilayah AWS.
- --application_name: Parameter ini diperlukan jika Anda menggunakan nama default untuk tabel metadata DynamoDB Anda (tabel sewa, tabel status koordinator, dan tabel metrik pekerja). Jika Anda telah menentukan nama kustom untuk tabel ini, Anda dapat menghilangkan parameter ini. Ganti `<applicationName>` dengan nama aplikasi KCL Anda yang sebenarnya. Alat ini menggunakan nama ini untuk mendapatkan nama tabel default jika nama kustom tidak disediakan.
- --lease_table_name (opsional): Parameter ini diperlukan ketika Anda telah menetapkan nama khusus untuk tabel sewa dalam konfigurasi KCL Anda. Jika Anda menggunakan nama tabel default, Anda dapat menghilangkan parameter ini. Ganti `leaseTableName` dengan nama tabel kustom yang Anda tentukan untuk tabel sewa Anda.
- --coordinator_state_table_name (opsional): Parameter ini diperlukan ketika Anda telah menetapkan nama kustom untuk tabel status koordinator dalam konfigurasi KCL Anda. Jika Anda menggunakan nama tabel default, Anda dapat menghilangkan parameter ini. Ganti `<coordinatorStateTableName>` dengan nama tabel kustom yang Anda tentukan untuk tabel status koordinator Anda.
- --worker_metrics_table_name (opsional): Parameter ini diperlukan ketika Anda telah menetapkan nama khusus untuk tabel metrik pekerja dalam konfigurasi KCL Anda. Jika Anda menggunakan nama tabel default, Anda dapat menghilangkan parameter ini. Ganti `<workerMetricsTableName>` dengan nama tabel kustom yang Anda tentukan untuk tabel metrik pekerja Anda.

Langkah 2: Menerapkan ulang kode dengan versi KCL sebelumnya (opsional)

Setelah menjalankan KCL Migration Tool untuk rollback, Anda akan melihat salah satu pesan berikut:

- Pesan 1: “Rollback selesai. Aplikasi KCL Anda menjalankan mode yang kompatibel dengan KCL 2.x. Jika Anda tidak melihat mitigasi regresi apa pun, harap kembalikan ke binari aplikasi Anda sebelumnya dengan menerapkan kode dengan versi KCL Anda sebelumnya.
  - Tindakan yang diperlukan: Ini berarti pekerja Anda berjalan dalam mode yang kompatibel dengan KCL 2.x. Jika masalah berlanjut, terapkan ulang kode dengan versi KCL sebelumnya ke pekerja Anda.
- Pesan 2: “Rollback selesai. Aplikasi KCL Anda menjalankan mode fungsionalitas KCL 3.x. Rollback ke binari aplikasi sebelumnya tidak diperlukan, kecuali jika Anda tidak melihat mitigasi untuk masalah ini dalam waktu 5 menit. Jika Anda masih memiliki masalah, harap kembalikan ke binari aplikasi Anda sebelumnya dengan menerapkan kode dengan versi KCL Anda sebelumnya.
  - Tindakan yang diperlukan: Ini berarti pekerja Anda berjalan dalam mode KCL 3.x dan Alat Migrasi KCL mengalihkan semua pekerja ke mode yang kompatibel dengan KCL 2.x. Jika masalah teratasi, Anda tidak perlu menerapkan ulang kode dengan versi KCL sebelumnya. Jika masalah berlanjut, terapkan ulang kode dengan versi KCL sebelumnya ke pekerja Anda.

Gulung maju ke KCL 3.x setelah rollback

Topik ini menjelaskan langkah-langkah untuk memajukan konsumen Anda kembali ke KCL 3.x setelah rollback. Ketika Anda perlu maju, Anda harus melalui proses dua langkah:

1. Jalankan Alat Migrasi KCL.
2. Terapkan kode dengan KCL 3.x.

Langkah 1: Jalankan Alat Migrasi KCL

Jalankan Alat Migrasi KCL. Alat Migrasi KCL dengan perintah berikut untuk maju ke KCL 3.x:

```
python3 ./KclMigrationTool.py --region <region> --mode rollforward [--application_name
 <applicationName>] [--coordinator_state_table_name <coordinatorStateTableName>]
```

Parameter

- --region: Ganti <region> dengan Anda Wilayah AWS.
- --application_name: Parameter ini diperlukan jika Anda menggunakan nama default untuk tabel status koordinator Anda. Jika Anda telah menentukan nama kustom untuk tabel status koordinator, Anda dapat menghilangkan parameter ini. Ganti <applicationName> dengan nama aplikasi KCL

Anda yang sebenarnya. Alat ini menggunakan nama ini untuk mendapatkan nama tabel default jika nama kustom tidak disediakan.

- --coordinator_state_table_name (opsional): Parameter ini diperlukan ketika Anda telah menetapkan nama kustom untuk tabel status koordinator dalam konfigurasi KCL Anda. Jika Anda menggunakan nama tabel default, Anda dapat menghilangkan parameter ini. Ganti `<coordinatorStateTableName>` dengan nama tabel kustom yang Anda tentukan untuk tabel status koordinator Anda.

Setelah Anda menjalankan alat migrasi dalam mode roll-forward, KCL membuat sumber daya DynamoDB berikut yang diperlukan untuk KCL 3.x:

- Indeks Sekunder Global pada tabel sewa
- Tabel metrik pekerja

Langkah 2: Menyebarkan kode dengan KCL 3.x

Setelah menjalankan KCL Migration Tool untuk maju, terapkan kode Anda dengan KCL 3.x ke pekerja Anda. Ikuti Langkah 8: Selesaikan migrasi untuk menyelesaikan migrasi Anda.

Praktik terbaik untuk tabel sewa dengan mode kapasitas yang disediakan

Jika tabel sewa aplikasi KCL Anda dialihkan ke mode kapasitas yang disediakan, KCL 3.x membuat indeks sekunder global pada tabel sewa dengan mode penagihan yang disediakan dan unit kapasitas baca (RCU) yang sama dan unit kapasitas tulis (WCU) sebagai tabel sewa dasar. Saat indeks sekunder global dibuat, sebaiknya Anda memantau penggunaan aktual pada indeks sekunder global di konsol DynamoDB dan menyesuaikan unit kapasitas jika diperlukan. Untuk panduan lebih rinci tentang beralih mode kapasitas tabel metadata DynamoDB yang dibuat oleh KCL, lihat. Mode kapasitas DynamoDB untuk tabel metadata yang dibuat oleh KCL

> **Note**
>
> Secara default, KCL membuat tabel metadata seperti tabel sewa, tabel metrik pekerja, dan tabel status koordinator, dan indeks sekunder global pada tabel sewa menggunakan mode kapasitas sesuai permintaan. Kami menyarankan Anda menggunakan mode kapasitas sesuai permintaan untuk menyesuaikan kapasitas secara otomatis berdasarkan perubahan penggunaan Anda.

Migrasi dari KCL 1.x ke KCL 3.x

Topik ini menjelaskan petunjuk untuk memigrasikan konsumen Anda dari KCL 1.x ke KCL 3.x. KCL 1.x menggunakan kelas dan antarmuka yang berbeda dibandingkan dengan KCL 2.x dan KCL 3.x. Anda harus memigrasikan prosesor rekaman, pabrik prosesor rekaman, dan kelas pekerja ke format yang kompatibel dengan KCL 2.x/3.x terlebih dahulu, dan ikuti langkah-langkah migrasi untuk migrasi KCL 2.x ke KCL 3.x. Anda dapat langsung meningkatkan dari KCL 1.x ke KCL 3.x.

- Langkah 1: Migrasikan prosesor rekaman

  Ikuti bagian Migrasi prosesor rekaman di halaman Migrasi konsumen dari KCL 1.x ke KCL 2.x.
- Langkah 2: Migrasikan pabrik prosesor rekaman

  Ikuti bagian Migrasi pabrik prosesor rekaman di halaman Migrasi konsumen dari KCL 1.x ke KCL 2.x.
- Langkah 3: Migrasikan pekerja

  Ikuti bagian Migrasi pekerja di halaman Migrasi konsumen dari KCL 1.x ke KCL 2.x.
- Langkah 4: Migrasikan konfigurasi KCL 1.x

  Ikuti bagian Konfigurasi klien Amazon Kinesis di halaman Migrasi konsumen dari KCL 1.x ke KCL 2.x.
- Langkah 5: Periksa penghapusan waktu idle dan penghapusan konfigurasi klien

  Ikuti bagian penghapusan waktu Idle dan penghapusan konfigurasi Klien di halaman Migrasikan konsumen dari KCL 1.x ke KCL 2.x.
- Langkah 6: Ikuti step-by-step petunjuk dalam panduan migrasi KCL 2.x ke KCL 3.x

  Ikuti petunjuk di Migrasi dari KCL 2.x ke KCL 3.x halaman untuk menyelesaikan migrasi. Jika Anda perlu memutar kembali ke versi KCL sebelumnya atau maju ke KCL 3.x setelah rollback, lihat dan. Putar kembali ke versi KCL sebelumnya Gulung maju ke KCL 3.x setelah rollback

> **Important**
>
> Jangan gunakan AWS SDK untuk Java versi 2.27.19 hingga 2.27.23 dengan KCL 3.x. Versi ini menyertakan masalah yang menyebabkan kesalahan pengecualian terkait dengan

penggunaan DynamoDB KCL. Kami menyarankan Anda menggunakan AWS SDK untuk Java versi 2.28.0 atau yang lebih baru untuk menghindari masalah ini.

Dokumentasi versi KCL sebelumnya

Topik-topik berikut telah diarsipkan. Untuk melihat dokumentasi Perpustakaan Klien Kinesis saat ini, lihat. Gunakan Perpustakaan Klien Kinesis

Important

Perpustakaan Klien Amazon Kinesis (KCL) versi 1.x dan 2.x sudah usang. KCL 1.x akan mencapai end-of-support pada 30 Januari 2026. Kami sangat menyarankan Anda memigrasikan aplikasi KCL Anda menggunakan versi 1.x ke versi KCL terbaru sebelum 30 Januari 2026. Untuk menemukan versi KCL terbaru, lihat halaman Perpustakaan Klien Amazon Kinesis di. GitHub Untuk informasi tentang versi KCL terbaru, lihatGunakan Perpustakaan Klien Kinesis. Untuk informasi tentang migrasi dari KCL 1.x ke KCL 3.x, lihat. Migrasi dari KCL 1.x ke KCL 3.x

Dokumentasi pensiunan

- Informasi KCL 1.x dan 2.x
- Kembangkan konsumen khusus dengan throughput bersama
- Migrasikan konsumen dari KCL 1.x ke KCL 2.x

Informasi KCL 1.x dan 2.x

Important

Perpustakaan Klien Amazon Kinesis (KCL) versi 1.x dan 2.x sudah usang. KCL 1.x akan mencapai end-of-support pada 30 Januari 2026. Kami sangat menyarankan Anda memigrasikan aplikasi KCL Anda menggunakan versi 1.x ke versi KCL terbaru sebelum 30 Januari 2026. Untuk menemukan versi KCL terbaru, lihat halaman Perpustakaan Klien Amazon Kinesis di. GitHub Untuk informasi tentang versi KCL terbaru, lihatGunakan Perpustakaan Klien Kinesis. Untuk informasi tentang migrasi dari KCL 1.x ke KCL 3.x, lihat. Migrasi dari KCL 1.x ke KCL 3.x

Salah satu metode pengembangan aplikasi konsumen kustom yang dapat memproses data dari aliran data KDS adalah dengan menggunakan Kinesis Client Library (KCL).

Topik

- Tentang KCL (versi sebelumnya)
- KCL versi sebelumnya
- Konsep KCL (versi sebelumnya)
- Gunakan tabel sewa untuk melacak pecahan yang diproses oleh aplikasi konsumen KCL
- Memproses beberapa aliran data dengan KCL 2.x yang sama untuk aplikasi konsumen Java
- Gunakan KCL dengan AWS Glue Schema Registry

> **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 selengkapnya, lihat https://github.com/awslabs/amazon-kinesis-client/release.

Tentang KCL (versi sebelumnya)

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 APIs Streams yang tersedia di. AWS SDKs Kinesis APIs Data Streams 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 lebih lanjut tentang MultiLangDaemon on GitHub, lihat MultiLangDaemon proyek KCL.

KCL bertindak sebagai perantara antara logika pemrosesan rekaman Anda dan Kinesis Data Streams.

KCL versi sebelumnya

Saat ini, Anda dapat menggunakan salah satu dari versi KCL yang didukung berikut ini untuk membangun aplikasi konsumen kustom Anda:

- KCL 1.x

  Untuk informasi selengkapnya, lihat Kembangkan konsumen KCL 1.x
- KCL 2.x

  Untuk informasi selengkapnya, lihat Kembangkan 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 Kembangkan 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 Kembangkan konsumen fan-out yang ditingkatkan dengan throughput khusus.

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. Migrasikan konsumen dari KCL 1.x ke KCL 2.x

Konsep KCL (versi sebelumnya)

- 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 kredensil 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/amazon-kinesis-client/.java. blob/v1.x/src/main/java/com/amazonaws/services/kinesis/clientlibrary/lib/worker/Worker 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/.java. blob/master/amazon-kinesis-client/src/main/java/software/amazon/kinesis/coordinator/Scheduler

- 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 ke 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/amazon-kinesis-client/.java untuk KCL 2.x. blob/master/amazon-kinesis-client/src/main/java/software/amazon/kinesis/leases/Lease

- 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 Gunakan tabel 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.

Gunakan tabel sewa untuk melacak pecahan yang diproses oleh aplikasi konsumen KCL

Topik

- Apa itu meja sewa
- Throughput
- Bagaimana tabel sewa disinkronkan dengan pecahan dalam aliran data Kinesis

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 Kinesis

- 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 Anda beradaMemproses 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 yang lebih baru.

- nama aliran Pengidentifikasi aliran data dalam format berikut:`account-id:StreamName:streamCreationTimestamp`.

> **Note**
>
> Data ini hanya ada di tabel sewa jika Anda beradaMemproses 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 yang lebih baru.

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 Kinesis

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
- Sinkronisasi di KCL 2.x, dimulai dengan KCL 2.3 dan yang lebih baru
- Sinkronisasi di KCL 1.x, dimulai dengan KCL 1.14 dan yang lebih baru

Sinkronisasi di KCL 1.0 - 1.13 dan KCL 2.0 - 2.2

Dalam 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 penemuan. `ListShards` `DescribeStream` APIs Dalam semua versi KCL yang tercantum di atas, setiap pekerja aplikasi konsumen KCL menyelesaikan langkah-langkah berikut untuk melakukan proses lease/shard sinkronisasi 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 yang lebih baru

Dimulai dengan versi terbaru yang didukung dari KCL 2.x (KCL 2.3) dan yang lebih baru, perpustakaan sekarang mendukung perubahan berikut pada proses sinkronisasi. Perubahan lease/shard sinkronisasi 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 lease/shard sinkronisasi untuk menjaga tabel sewa tetap up to date dengan pecahan terbaru dalam aliran data, satu pemimpin pekerja terpilih melakukan sinkronisasi sewa/pecahan.
- KCL 2.3 menggunakan parameter `ChildShards` pengembalian `GetRecords` dan `SubscribeToShard` APIs untuk melakukan lease/shard sinkronisasi yang terjadi pada `SHARD_END` pecahan tertutup, memungkinkan pekerja KCL untuk hanya membuat sewa untuk pecahan anak dari pecahan yang selesai diproses. Untuk dibagikan di seluruh aplikasi konsumen, optimalisasi lease/shard sinkronisasi ini menggunakan `ChildShards` parameter `GetRecords` API. Untuk aplikasi konsumen throughput khusus (fan-out yang disempurnakan), optimalisasi lease/shard sinkronisasi ini menggunakan `ChildShards` parameter API. `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 shard/lease pemindaian 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 lease/shard pemindaian berkala.

Untuk informasi selengkapnya tentang `PeriodicShardSyncManager` di KCL 2.3, lihat https://github.com/awslabs/amazon-kinesis-client/blob/master/amazon-kinesis-client/src/main/java/software/amazon/kinesis/leases/LeaseManagementConfig.java #L201 -L213.

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 sewa untuk aliran, maka itu akan memicu sinkronisasi 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 yang lebih baru

Dimulai dengan versi terbaru yang didukung dari KCL 1.x (KCL 1.14) dan yang lebih baru, perpustakaan sekarang mendukung perubahan berikut pada proses sinkronisasi. Perubahan lease/

shard sinkronisasi 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 lease/shard sinkronisasi untuk menjaga tabel sewa tetap up to date dengan pecahan terbaru dalam aliran data, satu pemimpin pekerja terpilih melakukan sinkronisasi sewa/pecahan.
- KCL 1.14 menggunakan parameter `ChildShards` pengembalian `GetRecords` dan `SubscribeToShard` APIs untuk melakukan lease/shard sinkronisasi yang terjadi pada `SHARD_END` pecahan tertutup, memungkinkan pekerja KCL untuk hanya membuat sewa untuk pecahan anak dari pecahan yang selesai diproses. Untuk informasi selengkapnya, lihat 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 shard/

lease pemindaian 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 lease/shard pemindaian 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 yang lebih baru.

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 penggemar yang disempurnakan.

  Untuk informasi selengkapnya`StreamIdentifier`, lihat https://github.com/awslabs/amazon-kinesis-client/blob/v2.5.8/amazon-kinesis-client/src/main/java/software/amazon/kinesis/common/StreamIdentifier.java #L129. Untuk membuat`StreamIdentifier`, kami sarankan Anda membuat instance multistream dari `streamArn` dan yang tersedia di

v2.5.0 dan `streamCreationEpoch` yang lebih baru. Di KCL v2.3 dan v2.4, yang tidak mendukung`streamArm`, buat instance multistream dengan menggunakan format. `account-id:StreamName:streamCreationTimestamp` Format ini akan usang dan tidak lagi didukung dimulai dengan rilis utama berikutnya.

`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-clientb/amazon-kinesis-client/src/main/java/software/amazon/kinesis/processor/FormerStreamsLeasesDeletionStrategy/blob/0c5042dadf794fe988438436252a5a8fe70b6b0 .java

- 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`

Gunakan KCL dengan AWS Glue Schema Registry

Anda dapat mengintegrasikan aliran data Kinesis Anda dengan Registri Skema. AWS Glue Registri AWS 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 AWS Glue Skema memungkinkan Anda 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 KCL di Jawa.

> **Important**
>
> Saat ini, integrasi Kinesis Data AWS Glue Streams dan Schema Registry 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 Pustaka” di Kasus KPL/KCL Penggunaan: Mengintegrasikan Amazon Kinesis Data Streams dengan Glue Schema Registry. AWS

Kembangkan konsumen khusus dengan throughput bersama

> **Important**
>
> Perpustakaan Klien Amazon Kinesis (KCL) versi 1.x dan 2.x sudah usang. KCL 1.x akan mencapai end-of-support pada 30 Januari 2026. Kami sangat menyarankan Anda memigrasikan aplikasi KCL Anda menggunakan versi 1.x ke versi KCL terbaru sebelum 30 Januari 2026. Untuk menemukan versi KCL terbaru, lihat halaman Perpustakaan Klien Amazon Kinesis di. GitHub Untuk informasi tentang versi KCL terbaru, lihatGunakan Perpustakaan Klien Kinesis. Untuk informasi tentang migrasi dari KCL 1.x ke KCL 3.x, lihat. Migrasi dari KCL 1.x ke KCL 3.x

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. Anda dapat menggunakan Kinesis Client Library (KCL) atau. AWS SDK untuk Java

Topik

- Kembangkan konsumen kustom dengan throughput bersama menggunakan KCL

Untuk informasi tentang membangun konsumen yang dapat menerima catatan dari aliran data Kinesis dengan throughput khusus, lihat. Kembangkan konsumen fan-out yang ditingkatkan dengan throughput khusus

Kembangkan konsumen kustom dengan throughput bersama menggunakan KCL

> **Important**
>
> Perpustakaan Klien Amazon Kinesis (KCL) versi 1.x dan 2.x sudah usang. KCL 1.x akan mencapai end-of-support pada 30 Januari 2026. Kami sangat menyarankan Anda memigrasikan aplikasi KCL Anda menggunakan versi 1.x ke versi KCL terbaru sebelum 30 Januari 2026. Untuk menemukan versi KCL terbaru, lihat halaman Perpustakaan Klien Amazon Kinesis di. GitHub Untuk informasi tentang versi KCL terbaru, lihatGunakan Perpustakaan Klien Kinesis. Untuk informasi tentang migrasi dari KCL 1.x ke KCL 3.x, lihat. Migrasi dari KCL 1.x ke KCL 3.x

Salah satu metode pengembangan aplikasi konsumen khusus dengan throughput bersama adalah dengan menggunakan Kinesis Client Library (KCL).

Pilih dari topik berikut untuk versi KCL yang Anda gunakan.

Topik

- Kembangkan konsumen KCL 1.x
- Kembangkan Konsumen KCL 2.x

Kembangkan konsumen KCL 1.x

> **Important**
>
> Perpustakaan Klien Amazon Kinesis (KCL) versi 1.x dan 2.x sudah usang. KCL 1.x akan mencapai end-of-support pada 30 Januari 2026. Kami sangat menyarankan Anda memigrasikan aplikasi KCL Anda menggunakan versi 1.x ke versi KCL terbaru sebelum 30 Januari 2026. Untuk menemukan versi KCL terbaru, lihat halaman Perpustakaan Klien Amazon Kinesis di. GitHub Untuk informasi tentang versi KCL terbaru, lihatGunakan Perpustakaan Klien Kinesis. Untuk informasi tentang migrasi dari KCL 1.x ke KCL 3.x, lihat. Migrasi dari KCL 1.x ke KCL 3.x

Anda dapat mengembangkan aplikasi konsumen untuk Amazon Kinesis Data Streams menggunakan Kinesis Client Library (KCL).

Untuk informasi lebih lanjut tentang KCL, lihatTentang KCL (versi sebelumnya).

Pilih dari topik berikut tergantung pada opsi yang ingin Anda gunakan.

Daftar Isi

- Mengembangkan Konsumen Perpustakaan Klien Kinesis di Jawa
- Mengembangkan konsumen Perpustakaan Klien Kinesis di Node.js
- Mengembangkan konsumen Perpustakaan Klien Kinesis di .NET
- Kembangkan konsumen Perpustakaan Klien Kinesis dengan Python
- Mengembangkan Konsumen Perpustakaan Klien Kinesis di Ruby

Mengembangkan Konsumen Perpustakaan Klien Kinesis di Jawa

> **Important**
>
> Perpustakaan Klien Amazon Kinesis (KCL) versi 1.x dan 2.x sudah usang. KCL 1.x akan mencapai end-of-support pada 30 Januari 2026. Kami sangat menyarankan Anda memigrasikan aplikasi KCL Anda menggunakan versi 1.x ke versi KCL terbaru sebelum 30 Januari 2026. Untuk menemukan versi KCL terbaru, lihat halaman Perpustakaan Klien Amazon Kinesis di. GitHub Untuk informasi tentang versi KCL terbaru, lihatGunakan Perpustakaan Klien Kinesis. Untuk informasi tentang migrasi dari KCL 1.x ke KCL 3.x, lihat. Migrasi dari KCL 1.x ke KCL 3.x

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 AWS Javadoc 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 untuk Java

Anda harus menyelesaikan tugas-tugas berikut saat menerapkan aplikasi konsumen KCL di Jawa:

Tugas

- Menerapkan metode IRecord Processor
- Menerapkan pabrik kelas untuk antarmuka IRecord Processor
- Buat pekerja
- Ubah properti konfigurasi
- Migrasi ke Versi 2 dari antarmuka prosesor rekaman

Menerapkan metode IRecord Processor

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.

IRecordVersi Prosesor

- Antarmuka Asli (Versi 1)
- Antarmuka yang 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, lihatGunakan resharding, scaling, dan parallel processing untuk mengubah jumlah pecahan.

```
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 diteruskan 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 yang 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 antarmuka IRecord Processor

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 metode IRecord Processor, 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 Gunakan tabel sewa untuk melacak pecahan yang diproses oleh aplikasi konsumen KCL.

Siapkan kredensil

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. AWS kredensil 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 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.

Gunakan 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 rekaman

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`. Contoh:

   ```
   final Worker worker = new Worker.Builder()
       .recordProcessorFactory(recordProcessorFactory)
       .config(config)
       .build();
   ```

Mengembangkan konsumen Perpustakaan Klien Kinesis di Node.js

> **Important**
>
> Perpustakaan Klien Amazon Kinesis (KCL) versi 1.x dan 2.x sudah usang. KCL 1.x akan mencapai end-of-support pada 30 Januari 2026. Kami sangat menyarankan Anda memigrasikan aplikasi KCL Anda menggunakan versi 1.x ke versi KCL terbaru sebelum 30 Januari 2026. Untuk menemukan versi KCL terbaru, lihat halaman Perpustakaan Klien Amazon Kinesis di. GitHub Untuk informasi tentang versi KCL terbaru, lihatGunakan Perpustakaan Klien Kinesis. Untuk informasi tentang migrasi dari KCL 1.x ke KCL 3.x, lihat. Migrasi dari KCL 1.x ke KCL 3.x

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. 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

- Implementasikan prosesor rekaman
- Ubah properti konfigurasi

Implementasikan prosesor rekaman

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 setidaknya sekali semantik, 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, lihatGunakan resharding, scaling, dan parallel processing untuk mengubah jumlah pecahan.

```
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 penggabungan, 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 verification/validation layanan 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 Gunakan tabel sewa untuk melacak pecahan yang diproses oleh aplikasi konsumen KCL.

Siapkan kredensil

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. AWS kredensil 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

> **Important**
>
> Perpustakaan Klien Amazon Kinesis (KCL) versi 1.x dan 2.x sudah usang. KCL 1.x akan mencapai end-of-support pada 30 Januari 2026. Kami sangat menyarankan Anda memigrasikan aplikasi KCL Anda menggunakan versi 1.x ke versi KCL terbaru sebelum 30 Januari 2026. Untuk menemukan versi KCL terbaru, lihat halaman Perpustakaan Klien Amazon Kinesis di. GitHub Untuk informasi tentang versi KCL terbaru, lihatGunakan

Perpustakaan Klien Kinesis. Untuk informasi tentang migrasi dari KCL 1.x ke KCL 3.x, lihat. Migrasi dari KCL 1.x ke KCL 3.x

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 Perpustakaan Klien Kinesis (.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 kelas IRecord Processor
- Ubah properti konfigurasi

Menerapkan metode kelas IRecord Processor

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 setidaknya sekali semantik, 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, lihatGunakan resharding, scaling, dan parallel processing untuk mengubah jumlah pecahan.

```
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 penggabungan, 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.

Matikan

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 Gunakan tabel sewa untuk melacak pecahan yang diproses oleh aplikasi konsumen KCL.

Siapkan kredensil

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 kredensil Anda tersedia untuk salah satu penyedia kredensional dalam rantai penyedia kredensi default. Jika Anda menjalankan aplikasi konsumen pada instans EC2, sebaiknya Anda mengonfigurasi instans dengan peran IAM. AWS kredensil 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 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`

Kembangkan konsumen Perpustakaan Klien Kinesis dengan Python

> **Important**
>
> Perpustakaan Klien Amazon Kinesis (KCL) versi 1.x dan 2.x sudah usang. KCL 1.x akan mencapai end-of-support pada 30 Januari 2026. Kami sangat menyarankan Anda

memigrasikan aplikasi KCL Anda menggunakan versi 1.x ke versi KCL terbaru sebelum 30 Januari 2026. Untuk menemukan versi KCL terbaru, lihat halaman Perpustakaan Klien Amazon Kinesis di. GitHub Untuk informasi tentang versi KCL terbaru, lihatGunakan Perpustakaan Klien Kinesis. Untuk informasi tentang migrasi dari KCL 1.x ke KCL 3.x, lihat. Migrasi dari KCL 1.x ke KCL 3.x

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` untuk mengimplementasikan 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 setidaknya sekali semantik, 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, lihatGunakan resharding, scaling, dan parallel processing untuk mengubah jumlah pecahan.

```
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 penggabungan, 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`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 Gunakan tabel sewa untuk melacak pecahan yang diproses oleh aplikasi konsumen KCL.

Siapkan kredensil

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 kredensil Anda tersedia untuk salah satu penyedia kredensional dalam rantai penyedia kredensi default. Jika Anda menjalankan aplikasi konsumen di instans Amazon EC2, sebaiknya Anda mengonfigurasi instans dengan peran IAM. AWS kredensil 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

> **Important**
>
> Perpustakaan Klien Amazon Kinesis (KCL) versi 1.x dan 2.x sudah usang. KCL 1.x akan mencapai end-of-support pada 30 Januari 2026. Kami sangat menyarankan Anda memigrasikan aplikasi KCL Anda menggunakan versi 1.x ke versi KCL terbaru sebelum 30 Januari 2026. Untuk menemukan versi KCL terbaru, lihat halaman Perpustakaan Klien Amazon Kinesis di. GitHub Untuk informasi tentang versi KCL terbaru, lihatGunakan Perpustakaan Klien Kinesis. Untuk informasi tentang migrasi dari KCL 1.x ke KCL 3.x, lihat. Migrasi dari KCL 1.x ke KCL 3.x

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.

Kembangkan Konsumen KCL 2.x

> **Important**
>
> Perpustakaan Klien Amazon Kinesis (KCL) versi 1.x dan 2.x sudah usang. KCL 1.x akan mencapai end-of-support pada 30 Januari 2026. Kami sangat menyarankan Anda

memigrasikan aplikasi KCL Anda menggunakan versi 1.x ke versi KCL terbaru sebelum 30 Januari 2026. Untuk menemukan versi KCL terbaru, lihat halaman Perpustakaan Klien Amazon Kinesis di. GitHub Untuk informasi tentang versi KCL terbaru, lihatGunakan Perpustakaan Klien Kinesis. Untuk informasi tentang migrasi dari KCL 1.x ke KCL 3.x, lihat. Migrasi dari KCL 1.x ke KCL 3.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.

Pilih dari topik berikut tergantung pada opsi yang ingin Anda gunakan.

Topik

- Mengembangkan Konsumen Perpustakaan Klien Kinesis di Jawa
- Kembangkan konsumen Perpustakaan Klien Kinesis dengan Python
- Kembangkan konsumen fan-out yang disempurnakan dengan KCL 2.x

Mengembangkan Konsumen Perpustakaan Klien Kinesis di Jawa

Important

Perpustakaan Klien Amazon Kinesis (KCL) versi 1.x dan 2.x sudah usang. KCL 1.x akan mencapai end-of-support pada 30 Januari 2026. Kami sangat menyarankan Anda memigrasikan aplikasi KCL Anda menggunakan versi 1.x ke versi KCL terbaru sebelum 30 Januari 2026. Untuk menemukan versi KCL terbaru, lihat halaman Perpustakaan Klien Amazon Kinesis di. GitHub Untuk informasi tentang versi KCL terbaru, lihatGunakan Perpustakaan Klien Kinesis. Untuk informasi tentang migrasi dari KCL 1.x ke KCL 3.x, lihat. Migrasi dari KCL 1.x ke KCL 3.x

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);
            }
        }
    }

}
```

Kembangkan konsumen Perpustakaan Klien Kinesis dengan Python

> **Important**
>
> Perpustakaan Klien Amazon Kinesis (KCL) versi 1.x dan 2.x sudah usang. KCL 1.x akan mencapai end-of-support pada 30 Januari 2026. Kami sangat menyarankan Anda memigrasikan aplikasi KCL Anda menggunakan versi 1.x ke versi KCL terbaru sebelum 30 Januari 2026. Untuk menemukan versi KCL terbaru, lihat halaman Perpustakaan Klien Amazon Kinesis di. GitHub Untuk informasi tentang versi KCL terbaru, lihatGunakan Perpustakaan Klien Kinesis. Untuk informasi tentang migrasi dari KCL 1.x ke KCL 3.x, lihat. Migrasi dari KCL 1.x ke KCL 3.x

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 menerapkan 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 Gunakan tabel sewa untuk melacak pecahan yang diproses oleh aplikasi konsumen KCL.

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. Jika Anda menjalankan aplikasi konsumen di instans Amazon EC2, sebaiknya Anda mengonfigurasi instans dengan peran IAM. AWS kredensil 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.

Kembangkan konsumen fan-out yang disempurnakan dengan KCL 2.x

> **Important**
>
> Perpustakaan Klien Amazon Kinesis (KCL) versi 1.x dan 2.x sudah usang. KCL 1.x akan mencapai end-of-support pada 30 Januari 2026. Kami sangat menyarankan Anda memigrasikan aplikasi KCL Anda menggunakan versi 1.x ke versi KCL terbaru sebelum 30 Januari 2026. Untuk menemukan versi KCL terbaru, lihat halaman Perpustakaan Klien Amazon Kinesis di. GitHub Untuk informasi tentang versi KCL terbaru, lihatGunakan Perpustakaan Klien Kinesis. Untuk informasi tentang migrasi dari KCL 1.x ke KCL 3.x, lihat. Migrasi dari KCL 1.x ke KCL 3.x

Konsumen yang menggunakan fan-out yang disempurnakan di Amazon Kinesis Data Streams dapat menerima catatan dari aliran data dengan throughput khusus hingga 2 MB data per detik per pecahan. Konsumen jenis ini tidak harus bersaing dengan konsumen lain yang menerima data dari aliran. Untuk informasi selengkapnya, lihat Kembangkan konsumen fan-out yang ditingkatkan dengan throughput khusus.

Anda dapat menggunakan versi 2.0 atau yang lebih baru dari Kinesis Client Library (KCL) untuk mengembangkan aplikasi yang menggunakan fan-out yang disempurnakan untuk menerima data dari aliran. KCL secara otomatis berlangganan aplikasi Anda ke semua pecahan aliran, dan memastikan bahwa aplikasi konsumen Anda dapat membaca dengan nilai throughput 2 per pecahan. MB/ sec Jika Anda ingin menggunakan KCL tanpa mengaktifkan fan-out yang disempurnakan, lihat Mengembangkan Konsumen Menggunakan Kinesis Client Library 2.0.

Topik

- Kembangkan konsumen fan-out yang disempurnakan menggunakan KCL 2.x di Jawa

Kembangkan konsumen fan-out yang disempurnakan menggunakan KCL 2.x di Jawa

> **Important**
>
> Perpustakaan Klien Amazon Kinesis (KCL) versi 1.x dan 2.x sudah usang. KCL 1.x akan mencapai end-of-support pada 30 Januari 2026. Kami sangat menyarankan Anda memigrasikan aplikasi KCL Anda menggunakan versi 1.x ke versi KCL terbaru sebelum 30 Januari 2026. Untuk menemukan versi KCL terbaru, lihat halaman Perpustakaan Klien Amazon Kinesis di. GitHub Untuk informasi tentang versi KCL terbaru, lihatGunakan Perpustakaan Klien Kinesis. Untuk informasi tentang migrasi dari KCL 1.x ke KCL 3.x, lihat. Migrasi dari KCL 1.x ke KCL 3.x

Anda dapat menggunakan Kinesis Client Library (KCL) versi 2.0 atau yang lebih baru 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 dari `ProcessorFactory` dan`RecordProcessor`.

Disarankan agar Anda menggunakan `KinesisClientUtil` untuk membuat `KinesisAsyncClient` dan mengkonfigurasi `maxConcurrency``KinesisAsyncClient`.

Important

Klien Amazon Kinesis mungkin melihat peningkatan latensi yang signifikan, kecuali jika Anda mengonfigurasi `KinesisAsyncClient` untuk memiliki cukup `maxConcurrency` 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);
            }
        }
    }

}
```

Migrasikan konsumen dari KCL 1.x ke KCL 2.x

> **Important**
>
> Perpustakaan Klien Amazon Kinesis (KCL) versi 1.x dan 2.x sudah usang. KCL 1.x akan mencapai end-of-support pada 30 Januari 2026. Kami sangat menyarankan Anda memigrasikan aplikasi KCL Anda menggunakan versi 1.x ke versi KCL terbaru sebelum 30 Januari 2026. Untuk menemukan versi KCL terbaru, lihat halaman Perpustakaan Klien Amazon Kinesis di. GitHub Untuk informasi tentang versi KCL terbaru, lihatGunakan Perpustakaan Klien Kinesis. Untuk informasi tentang migrasi dari KCL 1.x ke KCL 3.x, lihat. Migrasi dari KCL 1.x ke KCL 3.x

Topik ini menjelaskan perbedaan antara versi 1.x dan 2.x dari Kinesis Client Library (KCL). Ini juga menunjukkan kepada Anda cara memigrasikan konsumen Anda dari versi 1.x ke versi 2.x dari KCL. Setelah Anda memigrasikan klien Anda, klien akan mulai memproses catatan dari lokasi terakhir yang diperiksa.

Versi 2.0 dari KCL memperkenalkan perubahan antarmuka berikut:

Perubahan Antarmuka KCL

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 menjadi `software.amazon.kinesis.processor.ShardRecordProcessor`

Topik

- Migrasikan prosesor rekaman

- Migrasikan pabrik prosesor rekaman
- Migrasi pekerja
- Konfigurasikan klien Amazon Kinesis
- Penghapusan waktu idle
- Penghapusan konfigurasi klien

Migrasikan prosesor rekaman

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 memigrasikan kelas prosesor rekaman

1. Ubah antarmuka dari `com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor` dan `com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IShutdownNotificat` `kesoftware.amazon.kinesis.processor.ShardRecordProcessor`, 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. Perbarui `import` pernyataan untuk `initialize` dan `processRecords` metode.

```
// 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. Ganti `shutdown` metode dengan metode baru berikut:`leaseLost`,`shardEnded`, dan`shutdownRequested`.

```
//    @Override
//    public void shutdownRequested(IRecordProcessorCheckpointer checkpointer) {
//        //
//        // This is moved to shardEnded(...)
//        //
//        try {
//            checkpointer.checkpoint();
//        } catch (ShutdownException | InvalidStateException e) {
//            //
//            // Swallow exception
//            //
//            e.printStackTrace();
//        }
//    }

    @Override
    public void leaseLost(LeaseLostInput leaseLostInput) {

    }

    @Override
    public void shardEnded(ShardEndedInput shardEndedInput) {
        try {
            shardEndedInput.checkpointer().checkpoint();
        } catch (ShutdownException | InvalidStateException e) {
            //
            // Swallow the exception
            //
            e.printStackTrace();
        }
    }

//    @Override
```

```
//    public void shutdownRequested(IRecordProcessorCheckpointer checkpointer) {
//        //
//        // This is moved to shutdownRequested(ShutdownReauestedInput)
//        //
//        try {
//            checkpointer.checkpoint();
//        } catch (ShutdownException | InvalidStateException e) {
//            //
//            // Swallow exception
//            //
//            e.printStackTrace();
//        }
//    }

    @Override
    public void shutdownRequested(ShutdownRequestedInput shutdownRequestedInput) {
        try {
            shutdownRequestedInput.checkpointer().checkpoint();
        } catch (ShutdownException | InvalidStateException e) {
            //
            // Swallow the exception
            //
            e.printStackTrace();
        }
    }
```

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();
        }
    }
}
```

Migrasikan pabrik prosesor rekaman

Pabrik prosesor rekaman bertanggung jawab untuk membuat prosesor rekaman ketika 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 dari `com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessorFa` `kesoftware.amazon.kinesis.processor.ShardRecordProcessorFactory`, 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 rekaman 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, yang disebut`Scheduler`, menggantikan kelas. `Worker` Berikut ini adalah contoh pekerja KCL 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. Ubah `import` pernyataan untuk `Worker` kelas ke 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 pada contoh berikut.

   Disarankan agar Anda menggunakan `KinesisClientUtil` untuk membuat `KinesisAsyncClient` dan mengkonfigurasi `maxConcurrencyKinesisAsyncClient`.

   > **Important**
   >
   > Klien Amazon Kinesis mungkin melihat peningkatan latensi yang signifikan, kecuali jika Anda mengonfigurasi `KinesisAsyncClient` untuk memiliki cukup `maxConcurrency`

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()
    );
```

Konfigurasikan 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 Baru	Deskripsi
`applicationName`	`ConfigsBuilder`	Nama untuk ini aplikasi KCL. Digunakan sebagai default untuk `tableName` dan `consumerName` .
`tableName`	`ConfigsBuilder`	Memungkinkan penggantian nama tabel yang digunakan untuk tabel sewa Amazon DynamoDB.
`streamName`	`ConfigsBuilder`	Nama aliran tempat aplikasi ini memproses catatan dari.
`kinesisEndpoint`	`ConfigsBuilder`	Opsi ini telah dihapus. Lihat Penghapusan Konfigurasi Klien.
`dynamoDBEndpoint`	`ConfigsBuilder`	Opsi ini telah dihapus. Lihat Penghapusan Konfigurasi Klien.
`initialPositionInStreamExtended`	`RetrievalConfig`	Lokasi di pecahan tempat KCL mulai mengambil catatan, dimulai dengan proses awal aplikasi.
`kinesisCredentialsProvider`	`ConfigsBuilder`	Opsi ini telah dihapus. Lihat Penghapusan Konfigurasi Klien.
`dynamoDBCredentialsProvider`	`ConfigsBuilder`	Opsi ini telah dihapus. Lihat Penghapusan Konfigurasi Klien.

Bidang Asli	Kelas Konfigurasi Baru	Deskripsi
`cloudWatchCredentialsProvider`	`ConfigsBuilder`	Opsi ini telah dihapus. Lihat Penghapusan Konfigurasi Klien.
`failoverTimeMillis`	LeaseManagementConfig	Jumlah milidetik yang harus dilewati sebelum Anda dapat mempertimbangkan pemilik sewa telah gagal.
`workerIdentifier`	`ConfigsBuilder`	Pengenal unik yang mewakili instantiasi prosesor aplikasi ini. Ini pasti unik.
`shardSyncIntervalMillis`	LeaseManagementConfig	Waktu antara panggilan sinkronisasi shard.
`maxRecords`	`PollingConfig`	Memungkinkan pengaturan jumlah maksimum catatan yang Kinesis kembalikan.
`idleTimeBetweenReadsInMillis`	`CoordinatorConfig`	Opsi ini telah dihapus. Lihat Penghapusan Waktu Idle.
`callProcessRecordsEvenForEmptyRecordList`	`ProcessorConfig`	Ketika ditetapkan, prosesor rekaman dipanggil bahkan ketika tidak ada catatan yang diberikan dari Kinesis.
`parentShardPollIntervalMillis`	`CoordinatorConfig`	Seberapa sering prosesor rekaman harus melakukan polling untuk melihat apakah pecahan induk telah selesai.
`cleanupLeasesUponShardCompletion`	`LeaseManagementConfig`	Ketika ditetapkan, sewa dihapus segera setelah sewa anak mulai diproses.

Bidang Asli	Kelas Konfigurasi Baru	Deskripsi
`ignoreUnexpectedChildShards`	`LeaseManagementConfig`	Saat 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 yang gagal.
`metricsBufferTimeMillis`	`MetricsConfig`	Mengontrol penerbitan CloudWatch metrik.
`metricsMaxQueueSize`	`MetricsConfig`	Mengontrol penerbitan CloudWatch metrik.
`metricsLevel`	`MetricsConfig`	Mengontrol penerbitan CloudWatch metrik.
`metricsEnabledDimensions`	`MetricsConfig`	Mengontrol penerbitan CloudWatch metrik.
`validateSequenceNumberBeforeCheckpointing`	`CheckpointConfig`	Opsi ini telah dihapus. Lihat Validasi Nomor Urutan Pos Pemeriksaan.

Bidang Asli	Kelas Konfigurasi Baru	Deskripsi
`regionName`	`ConfigsBuilder`	Opsi ini telah dihapus. Lihat Penghapusan Konfigurasi Klien.
`maxLeasesForWorker`	`LeaseManagementConfig`	Jumlah maksimum sewa satu contoh aplikasi harus diterima.
`maxLeasesToStealAtOneTime`	`LeaseManagementConfig`	Jumlah maksimum sewa aplikasi harus mencoba untuk mencuri pada satu waktu.
`initialLeaseTableReadCapacity`	`LeaseManagementConfig`	DynamoDB IOPs membaca yang digunakan jika Kinesis Client Library perlu membuat tabel sewa DynamoDB baru.
`initialLeaseTableWriteCapacity`	`LeaseManagementConfig`	DynamoDB IOPs membaca yang digunakan jika Kinesis Client Library perlu membuat tabel sewa DynamoDB baru.
`initialPositionInStreamExtended`	LeaseManagementConfig	Posisi awal dalam aliran tempat aplikasi harus dimulai. Ini hanya digunakan selama pembuatan sewa awal.
`skipShardSyncAtWorkerInitializationIfLeasesExist`	`CoordinatorConfig`	Nonaktifkan sinkronisasi data pecahan jika tabel sewa berisi sewa yang ada. TODO: KinesisEco -438
`shardPrioritization`	`CoordinatorConfig`	Prioritas pecahan mana yang akan digunakan.
`shutdownGraceMillis`	N/A	Opsi ini telah dihapus. Lihat MultiLang Penghapusan.

Bidang Asli	Kelas Konfigurasi Baru	Deskripsi
`timeoutInSeconds`	N/A	Opsi ini telah dihapus. Lihat MultiLang Penghapusan.
`retryGetRecordsInSeconds`	`PollingConfig`	Mengkonfigurasi penundaan antara GetRecords upaya untuk kegagalan.
`maxGetRecordsThreadPool`	`PollingConfig`	Ukuran kolam benang yang digunakan untuk GetRecords.
`maxLeaseRenewalThreads`	`LeaseManagementConfig`	Mengontrol ukuran kumpulan utas penyewa penyewaan. Semakin banyak sewa yang dapat diambil aplikasi Anda, semakin besar kumpulan ini seharusnya.
`recordsFetcherFactory`	`PollingConfig`	Memungkinkan penggantian pabrik yang digunakan untuk membuat fetcher yang mengambil dari aliran.
`logWarningForTaskAfterMillis`	`LifecycleConfig`	Berapa lama menunggu sebelum peringatan dicatat jika tugas belum selesai.
`listShardsBackoffTimeInMillis`	`RetrievalConfig`	Jumlah milidetik untuk menunggu di antara panggilan ke `ListShards` saat kegagalan terjadi.
`maxListShardsRetryAttempts`	`RetrievalConfig`	Jumlah maksimum kali yang `ListShards` mencoba lagi sebelum menyerah.

Penghapusan waktu idle

Dalam versi 1.x dari KCL, yang `idleTimeBetweenReadsInMillis` berhubungan dengan dua kuantitas:

- Jumlah waktu antara pemeriksaan pengiriman tugas. Anda sekarang dapat mengonfigurasi waktu ini di antara tugas dengan mengaturCoordinatorConfig#shardConsumerDispatchPollIntervalMillis.
- Jumlah waktu untuk tidur ketika tidak ada catatan yang dikembalikan dari Kinesis Data Streams. Dalam versi 2.0, catatan fan-out yang disempurnakan didorong dari retriever masing-masing. Aktivitas pada konsumen shard hanya terjadi ketika permintaan yang didorong tiba.

Penghapusan konfigurasi klien

Di versi 2.0, KCL tidak lagi menciptakan klien. Itu tergantung pada pengguna untuk menyediakan klien yang valid. Dengan perubahan ini, semua parameter konfigurasi yang mengontrol pembuatan klien telah dihapus. Jika Anda membutuhkan parameter ini, Anda dapat mengaturnya pada klien sebelum memberikan klienConfigsBuilder.

Bidang yang Dihapus	Konfigurasi Setara
kinesisEndpoint	Konfigurasikan SDK KinesisAsyncClient dengan titik akhir pilihan:. KinesisAsyncClient.builder().endpointOverride(URI.create("https://<kinesis endpoint>")).build()
dynamoDBEndpoint	Konfigurasikan SDK DynamoDbAsyncClient dengan titik akhir pilihan:. DynamoDbAsyncClient.builder().endpointOverride(URI.create("https://<dynamodb endpoint>")).build()
kinesisClientConfig	Konfigurasikan SDK KinesisAsyncClient dengan konfigurasi yang diperlukan:KinesisAsyncClient.builder().overrideConfiguration(<your configuration>).build() .
dynamoDBClientConfig	Konfigurasikan SDK DynamoDbAsyncClient dengan konfigurasi yang diperlukan:DynamoDbAsyncClient.builder().overrideConfiguration(<your configuration>).build() .
cloudWatchClientConfig	Konfigurasikan SDK CloudWatchAsyncClient dengan konfigurasi yang diperlukan:CloudWatchAsyncClient.builder().overrideConfiguration(<your configuration>).build() .

Bidang yang Dihapus	Konfigurasi Setara
`regionName`	Konfigurasikan SDK dengan Wilayah pilihan. Ini sama untuk semua klien SDK. Misalnya, `KinesisAsyncClient.builder().region(Region.US_WEST_2).build()` .

Mengembangkan konsumen dengan AWS SDK untuk Java

Anda dapat mengembangkan konsumen khusus menggunakan Amazon Kinesis APIs Data Streams. Bagian ini menjelaskan penggunaan Kinesis APIs Data AWS SDK untuk Java Streams dengan file.

> **Important**
>
> Metode yang direkomendasikan untuk mengembangkan Kinesis Data Streams khusus yang dibagikan oleh konsumen adalah dengan menggunakan Kinesis Client Library (KCL). KCL membantu Anda mengkonsumsi dan memproses data dari aliran data Kinesis dengan menangani banyak tugas kompleks yang terkait dengan komputasi terdistribusi. Untuk informasi selengkapnya, lihat Kembangkan konsumen dengan KCL di Jawa.

Topik

- Mengembangkan konsumen throughput bersama dengan AWS SDK untuk Java
- Mengembangkan konsumen fan-out yang ditingkatkan dengan AWS SDK untuk Java
- Berinteraksi dengan data menggunakan AWS Glue Schema Registry

Mengembangkan konsumen throughput bersama dengan AWS SDK untuk Java

Salah satu metode untuk mengembangkan Kinesis Data Streams kustom yang dibagikan oleh konsumen adalah dengan menggunakan Amazon APIs Kinesis Data Streams dengan file. AWS SDK untuk Java Bagian ini menjelaskan penggunaan Kinesis APIs Data AWS SDK untuk Java Streams dengan file. Anda dapat memanggil Kinesis APIs Data Streams menggunakan bahasa pemrograman

lain yang berbeda. Untuk informasi selengkapnya tentang semua yang tersedia AWS SDKs, lihat Mulai Mengembangkan dengan Amazon Web Services.

Contoh kode Java di bagian ini menunjukkan cara melakukan operasi API Kinesis Data Streams 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 kinerja.

Topik

- Dapatkan data dari aliran
- Gunakan iterator shard
- Gunakan GetRecords
- Beradaptasi dengan reshard

Dapatkan data dari aliran

Kinesis APIs Data Streams `getShardIterator` menyertakan dan metode `getRecords` 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 Kinesis APIs Data Streams. Misalnya, untuk menerapkan alat khusus untuk memantau atau men-debug aliran data Anda.

> **Important**
>
> Kinesis Data Streams mendukung perubahan pada periode retensi rekaman data aliran data Anda. Untuk informasi selengkapnya, lihat Ubah periode retensi data.

Gunakan iterator shard

Anda mengambil catatan dari aliran pada basis per-shard. Untuk setiap pecahan, dan untuk setiap batch catatan yang Anda ambil dari pecahan itu, Anda harus mendapatkan iterator shard. Iterator shard digunakan dalam `getRecordsRequest` objek untuk menentukan pecahan dari mana catatan akan diambil. Jenis yang terkait dengan iterator shard menentukan titik dalam pecahan dari mana catatan harus diambil (lihat nanti di bagian ini untuk lebih jelasnya). Sebelum Anda dapat bekerja dengan iterator shard, Anda harus mengambil shard. Untuk informasi selengkapnya, lihat Daftar pecahan.

Dapatkan iterator shard awal menggunakan metode ini`getShardIterator`. Dapatkan iterator pecahan untuk kumpulan catatan tambahan menggunakan `getNextShardIterator` metode `getRecordsResult` objek yang dikembalikan oleh metode. `getRecords` Sebuah iterator shard berlaku selama 5 menit. Jika Anda menggunakan iterator shard saat valid, Anda mendapatkan yang baru. Setiap iterator shard tetap valid selama 5 menit, bahkan setelah digunakan.

Untuk mendapatkan iterator shard awal, buat instance `GetShardIteratorRequest` dan teruskan ke metode. `getShardIterator` Untuk mengkonfigurasi permintaan, tentukan stream dan ID shard. Untuk informasi tentang cara mendapatkan aliran di AWS akun Anda, lihatDaftar aliran. Untuk informasi tentang cara mendapatkan pecahan dalam aliran, lihatDaftar pecahan.

```
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 sampel ini menentukan `TRIM_HORIZON` sebagai tipe iterator ketika mendapatkan iterator shard awal. Jenis iterator ini berarti bahwa catatan harus dikembalikan dimulai dengan catatan pertama ditambahkan ke shard—daripada dimulai dengan catatan yang paling baru ditambahkan, juga dikenal sebagai tip. Berikut ini adalah jenis iterator yang mungkin:

- `AT_SEQUENCE_NUMBER`
- `AFTER_SEQUENCE_NUMBER`
- `AT_TIMESTAMP`

- `TRIM_HORIZON`
- `LATEST`

Untuk informasi selengkapnya, lihat ShardIteratorType.

Beberapa tipe iterator mengharuskan Anda menentukan nomor urut selain jenisnya; 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 peningkatan urutan catatan secara ketat. Untuk informasi selengkapnya, lihat contoh kode diPutRecord contoh.

Gunakan GetRecords

Setelah Anda mendapatkan iterator shard, buat instance objek. `GetRecordsRequest` Tentukan iterator untuk permintaan menggunakan `setShardIterator` metode.

Secara opsional, Anda juga dapat mengatur jumlah catatan untuk diambil menggunakan metode ini`setLimit`. Jumlah catatan yang dikembalikan `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 ini menetapkan batas ini menjadi 25 catatan.

Jika tidak ada catatan yang dikembalikan, itu berarti tidak ada catatan data saat ini tersedia dari pecahan ini pada nomor urut yang direferensikan oleh iterator pecahan. 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 shard yang dikembalikan oleh panggilan sebelumnya ke. `getRecords`

Lewati `getRecordsRequest` ke `getRecords` metode, dan tangkap nilai yang dikembalikan sebagai `getRecordsResult` objek. Untuk mendapatkan catatan data, panggil `getRecords` metode pada `getRecordsResult` objek.

```
GetRecordsRequest getRecordsRequest = new GetRecordsRequest();
getRecordsRequest.setShardIterator(shardIterator);
getRecordsRequest.setLimit(25);

GetRecordsResult getRecordsResult = client.getRecords(getRecordsRequest);
List<Record> records = getRecordsResult.getRecords();
```

Untuk mempersiapkan panggilan lain`getRecords`, dapatkan iterator shard berikutnya dari. `getRecordsResult`

```
shardIterator = getRecordsResult.getNextShardIterator();
```

Untuk hasil terbaik, tidurlah setidaknya 1 detik (1.000 milidetik) di antara panggilan `getRecords` untuk menghindari melebihi batas frekuensi. `getRecords`

```
try {
  Thread.sleep(1000);
}
catch (InterruptedException e) {}
```

Biasanya, Anda harus memanggil `getRecords` dalam satu lingkaran, bahkan ketika Anda mengambil satu catatan dalam skenario pengujian. Satu panggilan ke `getRecords` mungkin mengembalikan daftar catatan kosong, bahkan ketika pecahan berisi lebih banyak catatan di nomor urutan selanjutnya. Ketika ini terjadi, yang `NextShardIterator` dikembalikan bersama dengan daftar catatan kosong mereferensikan nomor urut selanjutnya dalam pecahan, dan `getRecords` panggilan berturut-turut akhirnya mengembalikan catatan. Sampel berikut menunjukkan penggunaan loop.

Contoh: GetRecords

Contoh kode berikut mencerminkan `getRecords` tips di bagian ini, termasuk membuat 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 Perpustakaan Klien Kinesis, mungkin akan melakukan beberapa panggilan sebelum mengembalikan data. Perilaku ini dirancang dan tidak menunjukkan masalah dengan KCL atau data Anda.

Beradaptasi dengan reshard

Jika `getRecordsResult.getNextShardIterator` kembali`null`, ini menunjukkan bahwa pecahan pecahan atau penggabungan telah terjadi yang melibatkan pecahan ini. Pecahan ini sekarang dalam `CLOSED` keadaan dan Anda telah membaca semua catatan data yang tersedia dari pecahan ini.

Dalam skenario ini, Anda dapat menggunakan `getRecordsResult.childShards` untuk mempelajari tentang pecahan anak baru dari pecahan yang sedang diproses yang dibuat oleh split atau merge. Untuk informasi selengkapnya, lihat ChildShard.

Dalam kasus split, dua pecahan baru keduanya `parentShardId` sama dengan ID pecahan pecahan yang Anda proses sebelumnya. Nilai `adjacentParentShardId` untuk kedua pecahan ini adalah`null`.

Dalam kasus penggabungan, pecahan baru tunggal yang dibuat oleh penggabungan memiliki `parentShardId` sama dengan ID pecahan dari salah satu pecahan induk dan `adjacentParentShardId` sama dengan ID pecahan induk lainnya. Aplikasi Anda telah membaca semua data dari salah satu pecahan ini. Ini adalah pecahan yang `getRecordsResult.getNextShardIterator` dikembalikan`null`. 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 penggabungan.

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 ke bawah untuk beradaptasi dengan perubahan jumlah pecahan.

Untuk informasi lebih lanjut tentang resharding, termasuk diskusi tentang status pecahan — seperti —lihat. `CLOSED` Reshard aliran

Mengembangkan konsumen fan-out yang ditingkatkan dengan AWS SDK untuk Java

Penggemar yang disempurnakan adalah fitur Amazon Kinesis Data Streams yang memungkinkan konsumen menerima catatan dari aliran data dengan throughput khusus hingga 2 MB data per detik per pecahan. Konsumen yang menggunakan fan-out yang ditingkatkan tidak harus bersaing dengan konsumen lain yang menerima data dari streaming. Untuk informasi selengkapnya, lihat Kembangkan konsumen fan-out yang ditingkatkan dengan throughput khusus.

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. Hubungi RegisterStreamConsumeruntuk mendaftarkan aplikasi Anda sebagai konsumen yang menggunakan fan-out yang disempurnakan. Kinesis Data Streams menghasilkan Nama Sumber Daya Amazon (ARN) untuk konsumen dan mengembalikannya sebagai respons.
2. Untuk mulai mendengarkan pecahan tertentu, berikan ARN konsumen dalam panggilan ke. SubscribeToShard Kinesis Data Streams kemudian mulai mendorong catatan dari pecahan itu kepada Anda, dalam bentuk peristiwa bertipe melalui koneksi SubscribeToShardEventHTTP/2. Koneksi tetap terbuka hingga 5 menit. Panggil SubscribeToShardlagi jika Anda ingin terus menerima catatan dari pecahan setelah `future` yang dikembalikan oleh panggilan untuk SubscribeToShardmenyelesaikan secara normal atau luar biasa.

   > **Note**
   >
   > `SubscribeToShard`API juga mengembalikan daftar pecahan anak dari pecahan saat ini saat akhir pecahan saat ini tercapai.

3. Untuk membatalkan pendaftaran konsumen yang menggunakan fan-out yang ditingkatkan, hubungi. DeregisterStreamConsumer

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` kembali`null`, ini menunjukkan bahwa pecahan pecahan atau penggabungan telah terjadi yang melibatkan pecahan ini. Pecahan ini sekarang dalam `CLOSED` keadaan, dan Anda telah membaca semua catatan data yang tersedia dari pecahan ini. Dalam skenario ini, per contoh di atas, Anda dapat menggunakan `event.childShards` untuk mempelajari tentang pecahan anak baru dari pecahan yang sedang diproses yang dibuat oleh split atau merge. Untuk informasi selengkapnya, lihat ChildShard.

Berinteraksi dengan data menggunakan AWS Glue Schema Registry

Anda dapat mengintegrasikan aliran data Kinesis Anda dengan Registri Skema. AWS Glue Registri AWS 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 AWS Glue Skema 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 `GetRecords` Kinesis Data Streams API yang tersedia AWS di Java SDK.

Untuk petunjuk terperinci tentang cara mengatur integrasi Aliran Data Kinesis dengan Registri `GetRecords` Skema menggunakan Kinesis Data Streams, lihat bagian “Berinteraksi dengan Data Menggunakan Aliran Data APIs Kinesis” di Kasus Penggunaan: Mengintegrasikan Amazon Kinesis Data APIs Streams dengan Registri Skema Glue. AWS

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?

Kembangkan 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.

Kembangkan 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.

Membaca data dari Kinesis Data AWS Streams menggunakan layanan lain

AWS Layanan berikut dapat langsung diintegrasikan dengan Amazon Kinesis Data Streams untuk membaca data dari aliran data Kinesis. Tinjau informasi untuk setiap layanan yang Anda minati dan lihat referensi yang disediakan.

Topik

- Baca data dari Kinesis Data Streams menggunakan Amazon EMR
- Baca data dari Kinesis Data Streams EventBridge menggunakan Amazon Pipes
- Membaca data dari Kinesis Data Streams menggunakan AWS Glue
- Baca data dari Kinesis Data Streams menggunakan Amazon Redshift

Baca data dari Kinesis Data Streams menggunakan Amazon EMR

Cluster EMR Amazon dapat membaca dan memproses aliran Kinesis secara langsung, menggunakan alat yang sudah dikenal di ekosistem Hadoop seperti Hive, Pig,, Hadoop Streaming API, dan Cascading. MapReduce Anda juga dapat menggabungkan data real-time dari Kinesis Data Streams dengan data yang ada di Amazon S3, Amazon DynamoDB, dan HDFS di cluster yang sedang 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.

Baca data dari Kinesis Data Streams EventBridge menggunakan Amazon Pipes

Amazon EventBridge Pipes mendukung aliran data Kinesis 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 rekaman 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.

Membaca data dari Kinesis Data Streams 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.

Baca data dari Kinesis Data Streams 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.

Baca dari Kinesis Data Streams 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. Pilih opsi yang ingin Anda pelajari lebih lanjut dan temukan sumber daya dan tautan ke dokumentasi yang relevan.

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 tingkat tinggi APIs 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.

Memecahkan Masalah Kinesis Data Streams konsumen

Topik berikut menawarkan solusi untuk masalah umum dengan konsumen Amazon Kinesis Data Streams:

- Kesalahan kompilasi dengan LeaseManagementConfig konstruktor
- Beberapa catatan Kinesis Data Streams dilewati saat menggunakan Perpustakaan Klien Kinesis
- Catatan milik pecahan yang sama diproses oleh prosesor rekaman yang berbeda pada saat yang sama
- Aplikasi konsumen membaca pada tingkat yang lebih lambat dari yang diharapkan
- GetRecords mengembalikan array catatan kosong bahkan ketika ada data dalam aliran
- Iterator shard kedaluwarsa secara tak terduga
- Pemrosesan catatan konsumen tertinggal

- Kesalahan izin kunci KMS yang tidak sah
- DynamoDbException: Jalur dokumen yang disediakan dalam ekspresi pembaruan tidak valid untuk pembaruan
- Memecahkan masalah umum lainnya bagi konsumen

Kesalahan kompilasi dengan LeaseManagementConfig konstruktor

Saat memutakhirkan ke Kinesis Client Library (KCL) versi 3.x atau yang lebih baru, Anda mungkin mengalami kesalahan kompilasi yang terkait dengan konstruktor. `LeaseManagementConfig` Jika Anda langsung membuat `LeaseManagementConfig` objek untuk mengatur konfigurasi alih-alih menggunakan `ConfigsBuilder` dalam KCL versi 3.x atau yang lebih baru, Anda mungkin melihat pesan kesalahan berikut saat mengkompilasi kode aplikasi KCL Anda.

```
Cannot resolve constructor 'LeaseManagementConfig(String, DynamoDbAsyncClient,
 KinesisAsyncClient, String)'
```

KCL dengan versi 3.x atau yang lebih baru mengharuskan Anda untuk menambahkan satu parameter lagi, applicationName (type: String), setelah parameter TableName.

- Sebelum: leaseManagementConfig = baru LeaseManagementConfig (TableName, DBClient dynamo, KinesisClient, streamName, WorkerIdentifier)
- Setelah: leaseManagementConfig = baru LeaseManagementConfig (TableName, applicationName, dynamo, KinesisClient, streamNameDBClient, workerIdentifier)

Alih-alih langsung membuat LeaseManagementConfig objek, sebaiknya gunakan `ConfigsBuilder` untuk mengatur konfigurasi di KCL 3.x dan versi yang lebih baru. `ConfigsBuilder`menyediakan cara yang lebih fleksibel dan dapat dipelihara untuk mengkonfigurasi aplikasi KCL Anda.

Berikut ini adalah contoh penggunaan `ConfigsBuilder` untuk mengatur konfigurasi KCL.

```
ConfigsBuilder configsBuilder = new ConfigsBuilder(
    streamName,
    applicationName,
    kinesisClient,
    dynamoClient,
    cloudWatchClient,
    UUID.randomUUID().toString(),
    new SampleRecordProcessorFactory()
```

```
);

Scheduler scheduler = new Scheduler(
    configsBuilder.checkpointConfig(),
    configsBuilder.coordinatorConfig(),
    configsBuilder.leaseManagementConfig()
    .failoverTimeMillis(60000), // this is an example
    configsBuilder.lifecycleConfig(),
    configsBuilder.metricsConfig(),
    configsBuilder.processorConfig(),
    configsBuilder.retrievalConfig()
);
```

Beberapa catatan Kinesis Data Streams 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 timbul 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. Jika instance pekerja 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.

Tetapkan 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 yang 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-shard. 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, I/O memblokir, 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, lihatGunakan resharding, scaling, dan parallel processing untuk mengubah jumlah pecahan.

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 yang dilakukan dalam 5 detik berikutnya akan melempar a`ProvisionedThroughputExceededException`.

GetRecords mengembalikan array catatan kosong bahkan ketika ada data dalam aliran

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 sebelumnya 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.

Iterator shard kedaluwarsa secara tak terduga

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 kedaluwarsa segera 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 Gunakan tabel sewa untuk melacak pecahan yang diproses oleh aplikasi konsumen KCL.

Pemrosesan catatan 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 yang cepat adalah meningkatkan periode retensi. Ini menghentikan hilangnya data penting saat Anda memecahkan masalah lebih lanjut. Untuk informasi selengkapnya, lihat Pantau 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 Pantau 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. Selidiki 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, antara lain) pada node pemrosesan yang mendasarinya selama permintaan puncak.

Kesalahan izin kunci KMS yang tidak sah

Kesalahan ini terjadi ketika aplikasi konsumen membaca dari aliran terenkripsi tanpa izin pada kunci. AWS KMS Untuk menetapkan izin ke aplikasi untuk mengakses kunci KMS, lihat Menggunakan Kebijakan Utama di KMS dan Menggunakan Kebijakan IAM dengan AWS KMS. AWS

DynamoDbException: Jalur dokumen yang disediakan dalam ekspresi pembaruan tidak valid untuk pembaruan

Saat menggunakan KCL 3.x dengan AWS SDK untuk Java versi 2.27.19 hingga 2.27.23, Anda mungkin menemukan pengecualian DynamoDB berikut:

“software.amazon.awssdk.services.dynamodb.model. DynamoDbException: Jalur dokumen yang disediakan dalam ekspresi pembaruan tidak valid untuk pembaruan (Layanan: DynamoDb, Kode Status: 400, ID Permintaan: xxx)”

Kesalahan ini terjadi karena masalah yang diketahui di yang mempengaruhi tabel metadata DynamoDB AWS SDK untuk Java yang dikelola oleh KCL 3.x. Masalah ini diperkenalkan di versi 2.27.19 dan berdampak pada semua versi hingga 2.27.23. Masalah telah diselesaikan di AWS SDK untuk Java versi 2.27.24. Untuk kinerja dan stabilitas yang optimal, kami sarankan untuk meningkatkan ke versi 2.28.0 atau yang lebih baru.

Memecahkan masalah umum lainnya 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?

Optimalkan konsumen Amazon Kinesis Data Streams

Anda dapat lebih mengoptimalkan konsumen Amazon Kinesis Data Streams berdasarkan perilaku tertentu yang Anda lihat.

Tinjau topik berikut untuk mengidentifikasi solusi.

Topik

- Tingkatkan pemrosesan latensi rendah
- Memproses data serial menggunakan Perpustakaan AWS Lambda Produser Amazon Kinesis
- Gunakan resharding, scaling, dan parallel processing untuk mengubah jumlah pecahan
- Menangani catatan duplikat
- Menangani startup, shutdown, dan throttling

Tingkatkan pemrosesan latensi rendah

Penundaan propagasi didefinisikan sebagai end-to-end latensi dari saat catatan ditulis ke aliran hingga 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 polling setiap pecahan satu kali per detik per aplikasi. Ini memungkinkan Anda memiliki beberapa aplikasi konsumen yang memproses aliran secara bersamaan tanpa mencapai batas Amazon Kinesis Data Streams 5 panggilan per detik. `GetRecords` 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 memanfaatkan ini dan memerlukan konsumsi data dari aliran segera setelah tersedia. Anda dapat mengurangi penundaan propagasi secara signifikan dengan mengganti pengaturan default KCL untuk melakukan polling lebih sering, seperti yang ditunjukkan pada contoh berikut.

Kode konfigurasi Java KCL:

```
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 `GetRecords` 5 panggilan per detik, per shard, `idleTimeBetweenReadsInMillis` menyetel properti lebih rendah dari 200ms dapat mengakibatkan aplikasi Anda mengamati pengecualian. `ProvisionedThroughputExceededException` Terlalu banyak pengecualian ini dapat

mengakibatkan back-off eksponensial dan dengan demikian menyebabkan latensi tak terduga yang signifikan dalam pemrosesan. Jika Anda menyetel properti ini pada atau tepat di atas 200 ms dan memiliki lebih dari satu aplikasi pemrosesan, Anda akan mengalami pelambatan serupa.

Memproses data serial menggunakan Perpustakaan AWS Lambda Produser Amazon Kinesis

Amazon Kinesis Producer Library (KPL) menggabungkan rekaman kecil yang diformat pengguna ke dalam rekaman yang lebih besar hingga 1 MB untuk memanfaatkan throughput Amazon Kinesis Data Streams dengan lebih baik. Sementara KCL untuk Java mendukung deagregasi catatan ini, Anda perlu menggunakan modul khusus untuk mendeagregasi catatan saat menggunakan AWS Lambda sebagai konsumen aliran Anda. Anda dapat memperoleh kode proyek dan instruksi yang diperlukan dari GitHub Modul Deagregasi Perpustakaan Produser Amazon Kinesis untuk Lambda. AWS Komponen dalam proyek ini memberi Anda kemampuan untuk memproses data serial KPL dalam AWS Lambda, di Java, Node.js dan Python. Komponen ini juga dapat digunakan sebagai bagian dari aplikasi KCL multi-lang.

Gunakan resharding, scaling, dan parallel processing untuk mengubah jumlah pecahan

Resharding memungkinkan 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 memantau metrik penanganan data shard. Meskipun KCL itu sendiri tidak memulai operasi resharding, KCL dirancang untuk beradaptasi dengan perubahan jumlah pecahan yang dihasilkan dari resharding.

Seperti disebutkan dalamGunakan tabel 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 dalam pecahan sebelum resharding diproses terlebih dahulu. Setelah data 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 parallel processing

Contoh berikut menggambarkan bagaimana 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 memulai pada instance kedua, ia menyeimbangkan beban dengan instance pertama, sehingga setiap instance sekarang memproses dua pecahan.
- Jika Anda kemudian memutuskan untuk membagi empat pecahan menjadi lima pecahan. KCL kembali mengoordinasikan pemrosesan di seluruh instance: satu instance memproses tiga pecahan, dan yang lainnya memproses dua pecahan. Koordinasi serupa terjadi saat 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 tepat satu pekerja KCL dan memiliki persis satu prosesor rekaman yang sesuai, jadi 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 instans (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 menskalakan instans secara otomatis berdasarkan metrik yang sesuai. Untuk informasi selengkapnya, lihat Panduan Pengguna Amazon EC2 Auto Scaling.

Ketika resharding meningkatkan jumlah pecahan dalam aliran, peningkatan yang sesuai dalam jumlah prosesor rekaman meningkatkan beban pada instans EC2 yang menghosting mereka. Jika instans merupakan bagian dari grup Auto Scaling, dan beban meningkat cukup, grup Auto Scaling menambahkan lebih banyak instance untuk menangani peningkatan beban. Anda harus mengonfigurasi instans untuk meluncurkan aplikasi Amazon Kinesis Data Streams saat startup, sehingga pekerja tambahan dan prosesor rekaman langsung aktif di instans baru.

Untuk informasi lebih lanjut tentang resharding, lihat. Reshard 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.

Produsen mencoba lagi

Pertimbangkan produser yang mengalami batas waktu terkait jaringan setelah melakukan panggilan`PutRecord`, tetapi sebelum dapat menerima pengakuan dari Amazon Kinesis Data Streams. Produser tidak dapat memastikan apakah rekaman dikirim ke Kinesis Data Streams. Dengan asumsi bahwa setiap catatan penting untuk aplikasi, produser akan ditulis untuk mencoba lagi panggilan dengan data yang sama. Jika kedua `PutRecord` panggilan pada data yang sama berhasil dilakukan ke Kinesis Data Streams, maka akan ada dua catatan Kinesis Data Streams. Meskipun kedua catatan memiliki data yang identik, mereka juga memiliki nomor urut yang unik. Aplikasi yang membutuhkan jaminan ketat harus menyematkan 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 percobaan ulang konsumen.

> **Note**
>
> Jika Anda menggunakan AWS SDK`PutRecord`, pelajari tentang perilaku Coba Ulang SDK di panduan pengguna AWS SDKs dan Alat.

Coba ulang konsumen

Percobaan ulang konsumen (aplikasi pemrosesan data) terjadi ketika prosesor rekaman dimulai ulang. Rekam prosesor untuk restart shard yang sama dalam kasus berikut:

1. Seorang pekerja berhenti secara tak terduga
2. Contoh pekerja ditambahkan atau dihapus
3. Pecahan digabungkan atau dibagi
4. Aplikasi ini digunakan

Dalam semua kasus ini, pemetaan shards-to-worker-to -record-processor terus diperbarui untuk pemrosesan keseimbangan beban. Prosesor shard yang dimigrasikan ke instance lain memulai ulang catatan pemrosesan dari pos pemeriksaan terakhir. Ini menghasilkan pemrosesan rekaman duplikat seperti yang ditunjukkan pada contoh di bawah ini. Untuk informasi selengkapnya tentang load-balancing, lihat. Gunakan resharding, scaling, dan parallel processing untuk mengubah jumlah pecahan

Contoh: Percobaan ulang konsumen menghasilkan catatan yang dikirim ulang

Dalam contoh ini, Anda memiliki aplikasi yang terus membaca catatan dari aliran, menggabungkan catatan ke dalam file lokal, dan mengunggah file ke Amazon S3. Untuk mempermudah, asumsikan hanya ada 1 pecahan dan 1 pekerja yang memproses pecahan. Perhatikan contoh urutan peristiwa berikut, dengan asumsi bahwa pos pemeriksaan terakhir berada pada nomor rekor 10000:

1. Seorang pekerja membaca kumpulan catatan berikutnya dari pecahan, mencatat 10001 hingga 20000.
2. Pekerja kemudian meneruskan batch catatan ke prosesor rekaman terkait.
3. Pemroses rekaman mengumpulkan data, membuat file Amazon S3, dan mengunggah file ke Amazon S3 dengan sukses.
4. Pekerja berhenti secara tak terduga sebelum pos pemeriksaan baru dapat terjadi.
5. Aplikasi, pekerja, dan prosesor rekaman 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 terhadap percobaan ulang konsumen

Meskipun catatan dapat diproses lebih dari satu kali, aplikasi Anda mungkin ingin menyajikan efek samping seolah-olah catatan diproses hanya satu kali (pemrosesan idempoten). Solusi untuk masalah ini bervariasi dalam kompleksitas dan akurasi. Jika tujuan data akhir dapat menangani duplikat dengan baik, kami sarankan untuk mengandalkan tujuan akhir untuk mencapai pemrosesan idempoten. Misalnya, dengan Opensearch Anda dapat menggunakan kombinasi versi dan unik IDs untuk mencegah pemrosesan duplikat.

Dalam contoh aplikasi di bagian sebelumnya, ia terus membaca catatan dari aliran, menggabungkan catatan ke dalam file lokal, dan mengunggah file ke Amazon S3. Seperti yang diilustrasikan, catatan 10001 -20000 dikonsumsi lebih dari satu kali sehingga menghasilkan beberapa file Amazon S3 dengan data yang sama. Salah satu cara untuk mengurangi duplikat dari contoh ini adalah memastikan bahwa langkah 3 menggunakan skema berikut:

1. Record Processor menggunakan jumlah catatan tetap per file Amazon S3, seperti 5000.
2. Nama file menggunakan skema ini: awalan Amazon S3, shard-id, dan file. `First-Sequence-Num` Dalam hal ini, bisa jadi seperti`sample-shard000001-10001`.
3. Setelah Anda mengunggah file Amazon S3, periksa dengan menentukan. `Last-Sequence-Num` Dalam hal ini, Anda akan memeriksa di nomor rekor 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. Percobaan ulang hanya menghasilkan penulisan data yang sama ke file yang sama lebih dari satu kali.

Dalam kasus operasi reshard, jumlah catatan yang tersisa di pecahan mungkin kurang dari jumlah tetap yang Anda inginkan. Dalam hal ini, `shutdown()` metode Anda harus menyiram file ke Amazon S3 dan pos pemeriksaan pada nomor urut terakhir. Skema di atas juga kompatibel dengan operasi reshard.

Menangani startup, shutdown, dan throttling

Berikut adalah beberapa pertimbangan tambahan untuk dimasukkan ke dalam desain aplikasi Amazon Kinesis Data Streams Anda.

Topik

- Memulai produsen data dan konsumen data

- Matikan aplikasi Amazon Kinesis Data Streams
- Baca pelambatan

Memulai produsen data dan konsumen data

Secara default, KCL mulai membaca catatan dari ujung aliran, yang merupakan catatan yang paling baru ditambahkan. Dalam konfigurasi ini, jika aplikasi penghasil data menambahkan catatan ke aliran sebelum prosesor rekaman penerima 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 disebut periode retensi. Mengatur posisi awal ke `TRIM_HORIZON` akan memulai prosesor rekaman dengan data tertua dalam aliran, seperti yang didefinisikan oleh periode retensi. Bahkan dengan `TRIM_HORIZON` pengaturannya, jika prosesor rekaman dimulai setelah waktu yang lebih lama berlalu daripada periode retensi, maka beberapa catatan dalam aliran tidak akan lagi tersedia. Untuk alasan ini, Anda harus selalu meminta aplikasi konsumen membaca dari aliran dan menggunakan CloudWatch metrik `GetRecords.IteratorAgeMilliseconds` untuk memantau bahwa aplikasi mengikuti data yang masuk.

Dalam beberapa skenario, mungkin baik-baik saja bagi prosesor rekaman untuk melewatkan beberapa catatan pertama dalam aliran. Misalnya, Anda mungkin menjalankan beberapa catatan awal melalui aliran untuk menguji apakah aliran berfungsi end-to-end seperti 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 Gunakan iterator shard.

Matikan 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 menjalankannya. Anda dapat mengakhiri instance menggunakan Konsol Manajemen AWSatau. AWS CLI

Setelah mematikan aplikasi Amazon Kinesis Data Streams, Anda harus menghapus tabel Amazon DynamoDB yang digunakan KCL untuk melacak status aplikasi.

Baca pelambatan

Throughput aliran disediakan pada tingkat pecahan. Setiap pecahan memiliki throughput baca hingga 5 transaksi per detik untuk pembacaan, hingga total kecepatan baca data maksimum 2 MB per detik. Jika aplikasi (atau sekelompok aplikasi yang beroperasi pada aliran yang sama) mencoba mendapatkan data dari pecahan dengan kecepatan yang lebih cepat, Kinesis Data Streams membatasi operasi Get yang sesuai.

Dalam aplikasi Amazon Kinesis Data Streams, jika prosesor rekaman memproses data lebih cepat dari batas — seperti dalam kasus failover — pelambatan terjadi. Karena KCL mengelola interaksi antara aplikasi dan Kinesis Data Streams, pengecualian throttling terjadi dalam kode KCL daripada dalam kode aplikasi. Namun, karena KCL mencatat pengecualian ini, Anda melihatnya di log.

Jika Anda menemukan bahwa aplikasi Anda dibatasi secara konsisten, Anda harus mempertimbangkan untuk meningkatkan jumlah pecahan untuk aliran.

Pantau Kinesis Data Streams

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.
- Perpustakaan Produser Kinesis — Amazon 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?

Pantau 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 streaming dan metrik tingkat shard yang Anda konfigurasikan secara otomatis dikumpulkan dan didorong ke setiap menit. CloudWatch 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 Amazon Kinesis Data Streams

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 disempurnakan
- Dimensi untuk metrik Amazon Kinesis Data Streams
- 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 lagi digunakan. 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
	Unit: Milidetik
`GetRecords.Latency`	Waktu yang dibutuhkan per operasi `GetRecords` , diukur selama periode waktu yang ditentukan. Dimensi: StreamName Statistik: Minimum, Maksimum, Rata-rata Unit: 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: Hitungan
`GetRecords.Success`	Jumlah `GetRecords` operasi yang berhasil per aliran, diukur selama periode waktu yang ditentukan. Dimensi: StreamName Statistik: Rata-rata, Jumlah, Sampel Unit: Hitungan

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: Hitungan

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 Unit: 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: Hitungan

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 Unit: Milidetik
PutRecords.Records	Metrik ini sudah usang. Gunakan PutRecords.SuccessfulRecords . Dimensi: StreamName Statistik: Minimum, Maksimum, Rata-rata, Jumlah, Sampel Unit: Hitungan
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: Hitungan

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: Hitungan
`PutRecords.Success fulRecords`	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: Hitungan
`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: Hitungan

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: Hitungan
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: Hitungan
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`	Jumlah milidetik catatan baca berasal dari ujung aliran, menunjukkan seberapa jauh di belakang waktu konsumen saat ini. Dimensi: StreamName, ConsumerName Statistik: Minimum, Maksimum, Rata-rata, Sampel Unit: 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: Hitungan
`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: Hitungan

Metrik	Deskripsi
`WriteProvisionedTh roughputExceeded`	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: `WriteProvisionedTh roughputExceeded` Dimensi: StreamName Statistik: Minimum, Maksimum, Rata-rata, Jumlah, Sampel Unit: Hitungan

Metrik tingkat shard yang disempurnakan

`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: Hitungan
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 Unit: 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: Hitungan

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: Hitungan

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: Hitungan

Dimensi untuk metrik Amazon Kinesis Data Streams

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 catatan 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 Amazon Kinesis Producer Library (KPL), gunakan. `PutRecords.Success`
`GetRecords.Success`	Sebaiknya gunakan CloudWatch alarm pada statistik Rata-rata untuk menunjukkan kapan catatan gagal dari aliran.

Akses 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. Pada panel navigasi, silakan 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 sebelumnya, kecuali untuk Throughput Baca dan Throughput Tulis. 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.

Pantau kesehatan Agen Aliran Data Kinesis dengan Amazon CloudWatch

Agen menerbitkan CloudWatch metrik kustom dengan namespace dari. AWS KinesisAgent 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. CloudWatch metrik yang diterbitkan dari beberapa agen Kinesis dikumpulkan atau digabungkan. Untuk informasi selengkapnya tentang konfigurasi agen, silakan lihat Tentukan pengaturan konfigurasi agen.

Monitor dengan CloudWatch

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: Hitungan
`RecordSendErrors`	Jumlah catatan yang menghasilkan status gagal dalam panggilan ke `PutRecords` , termasuk percobaan ulang, selama periode waktu yang ditentukan.

Metrik	Deskripsi
	Unit: Hitungan
`ServiceErrors`	Jumlah panggilan ke `PutRecords` yang mengakibatkan kesalahan layanan (selain kesalahan throttling) selama periode waktu yang ditentukan. Unit: Hitungan

Log 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 Kinesis Data Streams 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 acara terbaru di AWS akun Anda. 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 konsol, jejak tersebut berlaku untuk semua AWS Wilayah. Jejak mencatat peristiwa dari semua Wilayah di AWS partisi 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
- GetRecords
- GetShardIterator
- IncreaseStreamRetentionPeriod
- ListStreamConsumers
- ListStreams
- ListTagsForStream
- MergeShards
- PutRecord
- PutRecords

- RegisterStreamConsumer
- RemoveTagsFromStream
- SplitShard
- StartStreamEncryption
- StopStreamEncryption
- SubscribeToShard
- UpdateShardCount
- UpdateStreamMode

Setiap entri peristiwa atau log berisi informasi tentang entitas yang membuat permintaan tersebut. Informasi identitas membantu Anda menentukan hal berikut ini:

- Apakah permintaan itu dibuat dengan kredenal pengguna root atau AWS Identity and Access Management (IAM).
- Apakah permintaan tersebut dibuat dengan kredensial keamanan sementara untuk satu peran atau pengguna gabungan.
- Apakah permintaan itu dibuat oleh AWS layanan lain.

Untuk informasi lain, lihat Elemen userIdentity CloudTrail.

Contoh: Entri file log Kinesis Data Streams

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. Sebuah kejadian mewakili permintaan tunggal dari sumber apa pun dan mencakup informasi tentang tindakan yang diminta, tanggal dan waktu tindakan, parameter permintaan, dan sebagainya. Berkas log CloudTrail bukan merupakan jejak tumpukan terurut dari panggilan API publik, sehingga 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"
        }
    ]
}
```

Pantau 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

- Per-KCL-application metrik
- Metrik per pekerja
- Metrik per shard

Per-KCL-application metrik

Metrik ini digabungkan di semua pekerja KCL dalam lingkup aplikasi, seperti yang didefinisikan oleh namespace Amazon. CloudWatch

Topik

- LeaseAssignmentManager
- InitializeTask
- ShutdownTask
- ShardSyncTask
- BlockOnParentTask
- PeriodicShardSyncManager
- MultistreamTracker

LeaseAssignmentManager

`LeaseAssignmentManager`Operasi ini bertanggung jawab untuk menetapkan sewa kepada pekerja dan menyeimbangkan kembali sewa di antara pekerja untuk mencapai pemanfaatan sumber daya

pekerja yang merata. Logika untuk operasi ini termasuk membaca metadata terkait sewa dari tabel sewa dan metrik dari tabel metrik pekerja, dan melakukan penugasan sewa.

Metrik	Deskripsi
LeaseAndWorkerMetricsLoad.Waktu	Waktu yang dibutuhkan untuk memuat semua entri metrik sewa dan pekerja di manajer penugasan sewa (LAM), penugasan sewa baru dan algoritme penyeimbangan beban yang diperkenalkan di KCL 3.x. Tingkat metrik: Terperinci Unit: Milidetik
TotalLeases	Jumlah total sewa untuk aplikasi KCL saat ini. Tingkat metrik: Ringkasan Unit: Hitungan
NumWorkers	Jumlah total pekerja dalam aplikasi KCL saat ini. Tingkat metrik: Ringkasan Unit: Hitungan
AssignExpiredOrUnassignedLeases.Waktu	Waktu untuk melakukan penugasan dalam memori dari sewa yang kedaluwarsa. Tingkat metrik: Terperinci Unit: Milidetik
LeaseSpillover	Jumlah sewa yang tidak ditetapkan karena mencapai batas jumlah maksimum sewa atau throughput maksimum per pekerja. Tingkat metrik: Ringkasan Unit: Hitungan
BalanceWorkerVariance.Waktu	Saatnya melakukan penyeimbangan sewa dalam memori antar pekerja. Tingkat metrik: Terperinci

Metrik	Deskripsi
	Unit: Milidetik
NumOfLeasesReassignment	Jumlah total penggantian sewa yang dilakukan dalam iterasi penugasan kembali saat ini. Tingkat metrik: Ringkasan Unit: Hitungan
FailedAssignmentCount	Jumlah kegagalan dalam AssignLease panggilan ke tabel sewa DynamoDB. Tingkat metrik: Terperinci Unit: Hitungan
ParallelyAssignLeases.Waktu	Saatnya menyiram tugas baru ke tabel sewa DynamoDB. Tingkat metrik: Terperinci Unit: Milidetik
ParallelyAssignLeases.Sukses	Jumlah pembilasan tugas baru yang berhasil. Tingkat metrik: Terperinci Unit: Hitungan
TotalStaleWorkerMetricsEntry	Jumlah total entri metrik pekerja yang harus dibersihkan. Tingkat metrik: Terperinci Unit: Hitungan
StaleWorkerMetricsCleanup.Waktu	Waktu untuk melakukan penghapusan entri metrik pekerja dari tabel metrik pekerja DynamoDB. Tingkat metrik: Terperinci Unit: Milidetik

Metrik	Deskripsi
Waktu	Waktu yang dibutuhkan oleh `LeaseAssignmentManager` operasi. Tingkat metrik: Ringkasan Unit: Milidetik
Berhasil	Berapa kali `LeaseAssignmentManager` operasi berhasil diselesai kan. Tingkat metrik: Ringkasan Unit: Hitungan
ForceLeaderRelease	Menunjukkan bahwa manajer penugasan sewa telah gagal 3 kali berturut-turut dan pekerja pemimpin melepaskan kepemimpinan. Tingkat metrik: Ringkasan Unit: Hitungan
NumWorkersWithInva lidEntry	Jumlah entri metrik pekerja yang dianggap tidak valid. Tingkat metrik: Ringkasan Unit: Hitungan
NumWorkersWithFail ingWorkerMetric	Jumlah entri metrik pekerja yang memiliki -1 (mewakili nilai metrik pekerja tidak tersedia) sebagai salah satu nilai untuk metrik pekerja. Tingkat metrik: Ringkasan Unit: Hitungan
LeaseDeserializati onFailureCount	Entri sewa dari meja sewa yang gagal deserialisasi. Tingkat metrik: Ringkasan Unit: Hitungan

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 Unit: Hitungan
KinesisDataFetcher .getIterator.waktu	Waktu yang dibutuhkan per `GetShardIterator` operasi untuk aplikasi KCL yang diberikan. Tingkat metrik: Terperinci Unit: Milidetik
RecordProcessor.in isialize.waktu	Waktu yang dibutuhkan oleh metode inisialisasi prosesor rekaman. Tingkat metrik: Ringkasan Unit: Milidetik
Berhasil	Jumlah inisialisasi prosesor rekaman yang berhasil. Tingkat metrik: Ringkasan Unit: Hitungan
Waktu	Waktu yang dibutuhkan oleh pekerja KCL untuk inisialisasi prosesor rekaman. Tingkat metrik: Ringkasan Unit: 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. Tingkat metrik: Terperinci Unit: Hitungan
CreateLease.Waktu	Waktu yang dibutuhkan untuk menambahkan informasi pecahan anak baru dalam tabel DynamoDB aplikasi KCL. Tingkat metrik: Terperinci Unit: Milidetik
UpdateLease.Sukses	Jumlah pos pemeriksaan akhir yang berhasil selama shutdown prosesor rekaman. Tingkat metrik: Terperinci Unit: Hitungan
UpdateLease.Waktu	Waktu yang dibutuhkan oleh operasi pos pemeriksaan selama shutdown prosesor rekaman. Tingkat metrik: Terperinci Unit: Milidetik
RecordProcessor.shutdown.waktu	Waktu yang dibutuhkan oleh metode shutdown prosesor rekaman. Tingkat metrik: Ringkasan

Metrik	Deskripsi
	Unit: Milidetik
Berhasil	Jumlah tugas shutdown yang berhasil. Tingkat metrik: Ringkasan Unit: Hitungan
Waktu	Waktu yang dibutuhkan oleh pekerja KCL untuk tugas shutdown. Tingkat metrik: Ringkasan Unit: 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: Hitungan
CreateLease.Waktu	Waktu yang dibutuhkan untuk menambahkan informasi pecahan baru dalam tabel DynamoDB aplikasi KCL. Tingkat metrik: Terperinci Unit: Milidetik
Berhasil	Jumlah operasi sinkronisasi shard yang berhasil. Tingkat metrik: Ringkasan

Metrik	Deskripsi
	Unit: Hitungan
Waktu	Waktu yang dibutuhkan untuk operasi sinkronisasi shard. Tingkat metrik: Ringkasan Unit: 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: Hitungan
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: Hitungan
NumStreamsWithPartialLeases	Jumlah aliran data (per AWS akun) yang diproses aplikasi konsumen yang berisi sewa sebagian. Tingkat metrik: Ringkasan Unit: Hitungan
Berhasil	Berapa kali `PeriodicShardSyncManager` berhasil mengidentifikasi sewa sebagian dalam aliran data yang diproses oleh aplikasi konsumen. Tingkat metrik: Ringkasan Unit: Hitungan
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 Unit: Milidetik

MultistreamTracker

`MultistreamTracker`Antarmuka memungkinkan Anda 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: Hitungan
ActiveStreams.Hitung	Jumlah aliran data aktif yang sedang diproses. Tingkat metrik: Ringkasan Unit: Hitungan
StreamsPendingDeletion.Hitung	Jumlah aliran data yang tertunda penghapusan berdasarkan. `FormerStreamsLeasesDeletionStrategy` Tingkat metrik: Ringkasan Unit: Hitungan

Metrik per pekerja

Metrik ini dikumpulkan di semua prosesor rekaman yang menggunakan data dari aliran data Kinesis, seperti instans Amazon EC2.

Topik

- WorkerMetricStatsReporter
- LeaseDiscovery
- RenewAllLeases
- TakeLeases

WorkerMetricStatsReporter

`WorkerMetricStatReporter`Operasi bertanggung jawab untuk menerbitkan metrik pekerja saat ini secara berkala ke tabel metrik pekerja. Metrik ini digunakan oleh `LeaseAssignmentManager` operasi untuk melakukan penugasan sewa.

Metrik	Deskripsi
InMemoryMetricStatsReporterFailure	Jumlah kegagalan untuk menangkap nilai metrik pekerja dalam memori, karena kegagalan beberapa metrik pekerja. Tingkat metrik: Ringkasan Unit: Hitungan
WorkerMetricStatsReporter.Waktu	Waktu yang dibutuhkan oleh `WorkerMetricsStats` operasi. Tingkat metrik: Ringkasan Unit: Milidetik
WorkerMetricStatsReporter.Sukses	Berapa kali `WorkerMetricsStats` operasi berhasil diselesaikan. Tingkat metrik: Ringkasan Unit: Hitungan

LeaseDiscovery

`LeaseDiscovery`Operasi bertanggung jawab untuk mengidentifikasi sewa baru yang diberikan kepada pekerja saat ini oleh `LeaseAssignmentManager` operasi. Logika untuk operasi ini melibatkan identifikasi sewa yang diberikan kepada pekerja saat ini dengan membaca indeks sekunder global dari tabel sewa.

Metrik	Deskripsi
ListLeaseKeysForWorker.Waktu	Saatnya memanggil indeks sekunder global pada tabel sewa dan mendapatkan kunci sewa yang ditugaskan untuk pekerja saat ini. Tingkat metrik: Detil Unit: Milidetik
FetchNewLeases.Waktu	Saatnya mengambil semua sewa baru dari tabel sewa. Tingkat metrik: Detil

Metrik	Deskripsi
	Unit: Milidetik
NewLeasesDiscovered	Jumlah total sewa baru yang diberikan kepada pekerja. Tingkat metrik: Detil Unit: Hitungan
Waktu	Waktu yang dibutuhkan oleh `LeaseDiscovery` operasi. Tingkat metrik: Ringkasan Unit: Milidetik
Berhasil	Berapa kali `LeaseDiscovery` operasi berhasil diselesaikan. Tingkat metrik: Ringkasan Unit: Hitungan
OwnerMismatch	Jumlah ketidakcocokan pemilik dari respons GSI dan tabel sewa dibaca konsisten. Tingkat metrik: Detil Unit: Hitungan

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: Detil Unit: Hitungan

Metrik	Deskripsi
RenewLease.Waktu	Waktu yang dibutuhkan oleh operasi perpanjangan sewa. Tingkat metrik: Detil Unit: Milidetik
CurrentLeases	Jumlah sewa pecahan yang dimiliki oleh pekerja setelah semua sewa diperbarui. Tingkat metrik: Ringkasan Unit: Hitungan
LostLeases	Jumlah sewa pecahan yang hilang setelah upaya untuk memperbarui semua sewa yang dimiliki oleh pekerja. Tingkat metrik: Ringkasan Unit: Hitungan
Berhasil	Berapa kali operasi pembaruan sewa berhasil bagi pekerja. Tingkat metrik: Ringkasan Unit: Hitungan
Waktu	Waktu yang dibutuhkan untuk memperbarui semua sewa untuk pekerja. Tingkat metrik: Ringkasan Unit: 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. Tingkat metrik: Detil Unit: Hitungan
ListLeases.Waktu	Waktu yang dibutuhkan untuk mengambil semua sewa shard dari tabel DynamoDB aplikasi KCL. Tingkat metrik: Detil Unit: Milidetik
TakeLease.Sukses	Berapa kali pekerja berhasil mengambil sewa pecahan dari pekerja KCL lainnya. Tingkat metrik: Detil Unit: Hitungan
TakeLease.Waktu	Waktu yang dibutuhkan untuk memperbarui tabel sewa dengan sewa yang diambil oleh pekerja. Tingkat metrik: Detil Unit: Milidetik
NumWorkers	Jumlah total pekerja, sebagaimana diidentifikasi oleh pekerja tertentu. Tingkat metrik: Ringkasan Unit: Hitungan
NeededLeases	Jumlah sewa pecahan yang dibutuhkan pekerja saat ini untuk beban pemrosesan pecahan yang seimbang. Tingkat metrik: Detil

Metrik	Deskripsi
	Unit: Hitungan
LeasesToTake	Jumlah sewa yang akan diambil pekerja. Tingkat metrik: Detil Unit: Hitungan
TakenLeases	Jumlah sewa yang berhasil diambil oleh pekerja. Tingkat metrik: Ringkasan Unit: Hitungan
TotalLeases	Jumlah pecahan yang diproses aplikasi KCL. Tingkat metrik: Detil Unit: Hitungan
ExpiredLeases	Jumlah total pecahan yang tidak diproses oleh pekerja mana pun, sebagaimana diidentifikasi oleh pekerja tertentu. Tingkat metrik: Ringkasan Unit: Hitungan
Berhasil	Berapa kali `TakeLeases` operasi berhasil diselesaikan. Tingkat metrik: Ringkasan Unit: Hitungan
Waktu	Waktu yang dibutuhkan oleh `TakeLeases` operasi untuk seorang pekerja. Tingkat metrik: Ringkasan Unit: 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: Detil Unit: Hitungan
KinesisDataFetcher .getRecords.waktu	Waktu yang dibutuhkan per `GetRecords` operasi untuk pecahan aliran data Kinesis. Tingkat metrik: Detil Unit: Milidetik
UpdateLease.Sukses	Jumlah pos pemeriksaan yang berhasil dibuat oleh prosesor rekaman untuk pecahan yang diberikan. Tingkat metrik: Detil Unit: Hitungan
UpdateLease.Waktu	Waktu yang dibutuhkan untuk setiap operasi pos pemeriksaan untuk pecahan yang diberikan. Tingkat metrik: Detil Unit: 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: Hitungan
ExpiredIterator	Jumlah yang ExpiredIteratorException diterima saat menelepon `GetRecords` . Tingkat metrik: Ringkasan Unit: Hitungan
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 semua stempel waktu di setiap catatan. Tingkat metrik: Ringkasan Unit: Milidetik
RecordProcessor.Pr ocessRecords.time	Waktu yang dibutuhkan oleh `processRecords` metode prosesor rekaman. Tingkat metrik: Ringkasan Unit: Milidetik

Metrik	Deskripsi
Berhasil	Jumlah operasi tugas proses yang berhasil. Tingkat metrik: Ringkasan Unit: Hitungan
Waktu	Waktu yang dibutuhkan untuk operasi tugas proses. Tingkat metrik: Ringkasan Unit: Milidetik

Pantau Perpustakaan Produser Kinesis dengan Amazon CloudWatch

Perpustakaan Produsen Amazon 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 unggahan 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 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 pada 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` Ydalam diagram sebelumnya).

Saat Anda menentukan dimensi tambahan, Anda harus memberikan nilai untuk`tuple:<DimensionName, DimensionValue, Granularity>`. Granularitas digunakan untuk menentukan di mana dimensi kustom dimasukkan dalam hierarki: `GLOBAL` berarti bahwa dimensi tambahan dimasukkan 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 unggahan 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: Detil 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: Detil Unit: Jumlah
`UserRecordsPut`	Hitung berapa banyak catatan pengguna logis yang berhasil dimasukkan. 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
`UserRecordsDataPut`	Byte dalam catatan pengguna logis berhasil dimasukkan.

Metrik	Deskripsi
	Tingkat metrik: Detil 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: Detil 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: Detil 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: Detil Satuan: Milidetik

Metrik	Deskripsi
`User Records per Kinesis Record`	Jumlah catatan pengguna logis yang dikumpulkan ke dalam catatan Kinesis Data Streams tunggal. Tingkat metrik: Detil Unit: Jumlah
`Amazon Kinesis Records per PutRecord sRequest`	Jumlah catatan Kinesis Data Streams dikumpulkan menjadi satu. `PutRecordsRequest` Tidak tersedia pada tingkat pecahan. Tingkat metrik: Detil 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: Detil 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 menggambarkan 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 kepatuhan AWS. 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, kebutuhan organisasi, 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
- Mengontrol akses ke sumber daya Amazon Kinesis Data Streams menggunakan IAM
- Validasi kepatuhan untuk Amazon Kinesis Data Streams
- Ketahanan dalam Aliran Data Amazon Kinesis
- Keamanan infrastruktur di Kinesis Data Streams
- Praktik keamanan terbaik untuk Kinesis Data Streams

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?
- Biaya, Wilayah, dan pertimbangan kinerja
- Bagaimana cara memulai dengan enkripsi sisi server?
- Membuat dan menggunakan kunci KMS buatan pengguna
- Izin untuk menggunakan kunci KMS buatan pengguna
- Verifikasi dan Memecahkan Masalah izin kunci KMS
- Gunakan Amazon Kinesis Data Streams dengan titik akhir VPC antarmuka

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.

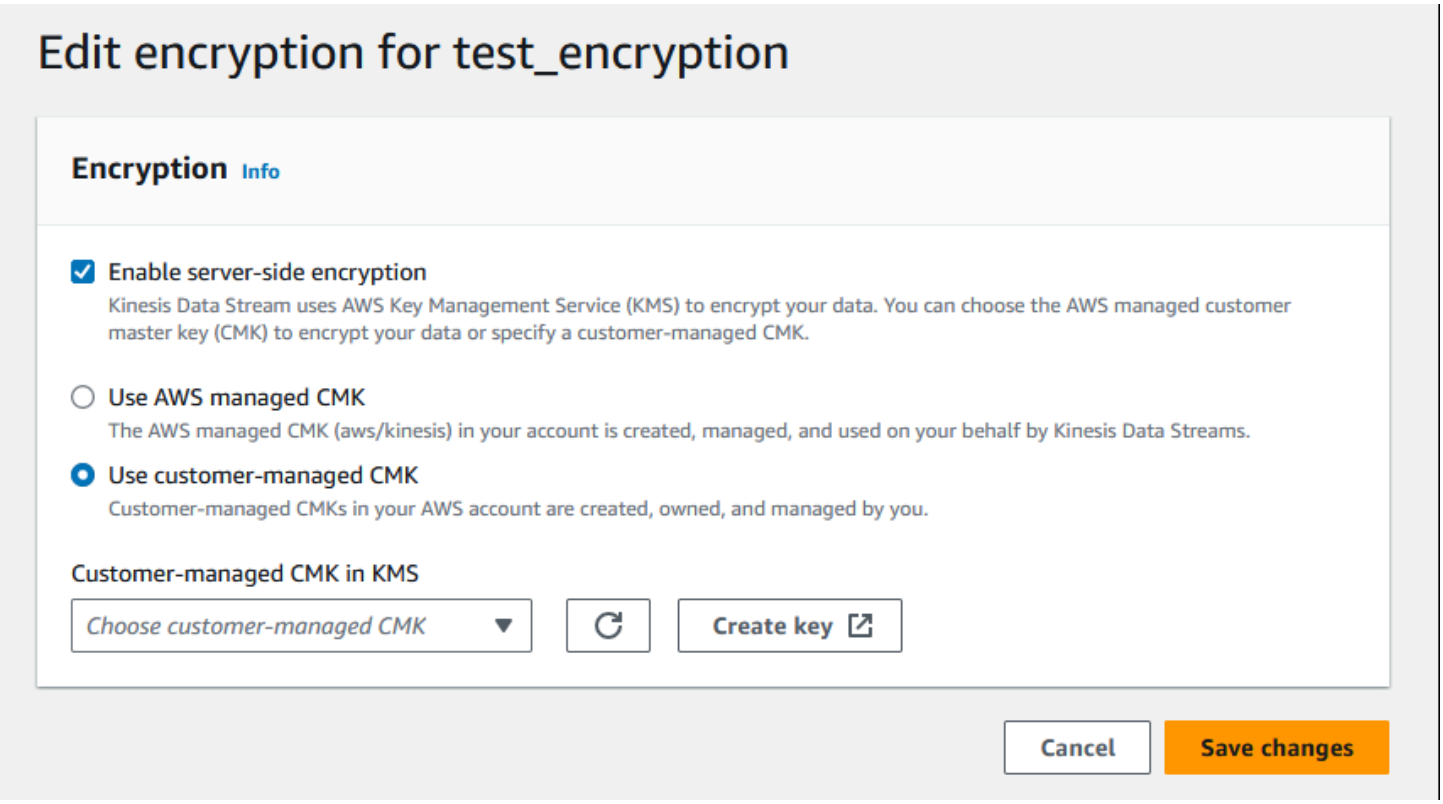
Dengan enkripsi sisi server, produsen dan konsumen Kinesis stream Anda tidak perlu mengelola kunci utama atau operasi kriptografi. Data Anda secara otomatis dienkripsi saat masuk dan meninggalkan layanan Kinesis Data Streams, sehingga data Anda saat istirahat dienkripsi. AWS KMS menyediakan semua kunci master yang digunakan oleh fitur enkripsi sisi server. AWS KMS

memudahkan penggunaan CMK untuk Kinesis yang dikelola AWS oleh, 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 baik 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

Saat 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 KMS Bila 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 Manajemen AWS 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 data dan konsumen 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 kunci account/user akses IAM tunggal di seluruh pembaca dan penulis, layanan Kinesis memanggil AWS KMS layanan sekitar 12 kali setiap 5 menit. 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 Manajemen AWS 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 Tiongkok. AWS GovCloud Untuk informasi selengkapnya tentang Wilayah yang didukung untuk Kinesis Data https://docs.aws.amazon.com/general/latest/gr/akStreams, lihat .html.

Pertimbangan Kinerja

Karena overhead layanan penerapan enkripsi, menerapkan enkripsi sisi server meningkatkan latensi tipikal`PutRecord`,`PutRecords`, dan `GetRecords` kurang dari 100µs.

Bagaimana cara memulai dengan enkripsi sisi server?

Cara termudah untuk memulai dengan enkripsi sisi server adalah dengan menggunakan Konsol Manajemen AWS 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 Konsol Manajemen AWS dan buka konsol Amazon Kinesis Data Streams.
2. Buat atau pilih aliran Kinesis di file. Konsol Manajemen AWS
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. Konsol Manajemen AWS

Membuat dan menggunakan kunci KMS buatan pengguna

Bagian ini menjelaskan cara membuat dan menggunakan kunci KMS Anda sendiri, alih-alih menggunakan kunci master yang dikelola oleh Amazon Kinesis.

Buat kunci KMS buatan pengguna

Untuk petunjuk cara membuat kunci Anda sendiri, lihat Membuat Kunci di Panduan AWS Key Management Service Pengembang. Setelah Anda membuat kunci untuk akun Anda, layanan Kinesis Data Streams mengembalikan kunci ini dalam daftar kunci master KMS.

Gunakan kunci KMS buatan pengguna

Setelah izin yang benar diterapkan ke konsumen, produsen, dan administrator Anda, Anda dapat menggunakan kunci 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. Konsol Manajemen AWS

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. Konsol Manajemen AWS

Izin untuk menggunakan kunci KMS buatan pengguna

Sebelum Anda dapat menggunakan enkripsi sisi server dengan kunci KMS buatan pengguna, Anda harus mengonfigurasi kebijakan AWS KMS kunci untuk mengizinkan enkripsi aliran dan enkripsi dan dekripsi catatan aliran. Untuk contoh dan informasi selengkapnya tentang AWS KMS izin, lihat Izin API AWS 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 hilangnya 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 produsen

Produsen aliran Kinesis Anda harus memiliki izin. `kms:GenerateDataKey`

JSON

```
{
  "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`

JSON

```
{
  "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 streaming

Administrator aliran Kinesis harus memiliki otorisasi untuk menelepon dan. `kms:List*` `kms:DescribeKey*`

Verifikasi 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`

Untuk informasi selengkapnya, lihat Pantau Kinesis Data Streams

Gunakan Amazon Kinesis Data Streams dengan titik akhir VPC antarmuka

Anda dapat menggunakan titik akhir VPC antarmuka untuk mencegah lalu lintas antara Amazon VPC dan Kinesis Data Streams Anda meninggalkan jaringan Amazon. Endpoint VPC antarmuka tidak memerlukan gateway internet, perangkat NAT, koneksi VPN, atau koneksi. Direct Connect Endpoint VPC antarmuka didukung oleh AWS PrivateLink, sebuah AWS teknologi yang memungkinkan komunikasi pribadi antar AWS layanan menggunakan antarmuka elastic network dengan private di Amazon VPC IPs Anda. Untuk informasi selengkapnya, lihat Amazon Virtual Private Cloud dan Interface VPC Endpoints ().AWS PrivateLink

Topik

- Gunakan titik akhir VPC antarmuka untuk Kinesis Data Streams
- Kontrol akses ke titik akhir VPC untuk Kinesis Data Streams
- Ketersediaan kebijakan endpoint VPC untuk Kinesis Data Streams

Gunakan titik akhir VPC antarmuka untuk Kinesis Data Streams

Untuk memulai, Anda tidak perlu mengubah pengaturan untuk aliran, produsen, atau konsumen Anda. Buat titik akhir VPC antarmuka untuk Kinesis Data Streams untuk memulai lalu lintas yang mengalir dari dan ke sumber daya VPC Amazon Anda melalui titik akhir VPC antarmuka. Titik akhir VPC antarmuka berkemampuan FIPS tersedia untuk Wilayah AS. Untuk informasi selengkapnya, lihat Membuat Titik Akhir Antarmuka.

Layanan panggilan Amazon 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.

Kontrol akses ke titik akhir VPC 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 agar hanya terjadi melalui titik akhir VPC yang ditentukan. Gunakan kebijakan ini untuk membatasi akses ke aliran tertentu ke titik akhir VPC tertentu saat menggunakannya 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 aliran tertentu dari titik akhir VPC tertentu saja - kebijakan sampel ini dapat dilampirkan ke pengguna, peran, atau grup IAM. Ini membatasi akses ke aliran data Kinesis tertentu hanya terjadi dari titik akhir VPC tertentu.

  JSON

```
{
   "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 endpoint 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)
- Europe (Milan)
- Africa (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)
- Europe (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 mengasumsikannya
- 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, lihatKembangkan konsumen KCL 1.x.

Untuk informasi selengkapnya tentang IAM, lihat berikut ini:

- AWS Identity and Access Management (IAM)
- Memulai dengan IAM
- 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

- Sintaksis kebijakan
- Tindakan untuk Kinesis Data Streams
- Nama Sumber Daya Amazon (ARNs) untuk Kinesis Data Streams

- Contoh kebijakan untuk Kinesis Data Streams
- Bagikan aliran data Anda dengan akun lain
- Konfigurasikan AWS Lambda fungsi untuk membaca dari Kinesis Data Streams di akun lain
- Berbagi akses menggunakan kebijakan berbasis sumber daya

Sintaksis 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 (ARNs) untuk Kinesis Data Streams

Setiap pernyataan kebijakan IAM berlaku untuk sumber daya yang Anda tentukan dengan menggunakan ARNs.

Gunakan format sumber daya ARN berikut untuk aliran data Kinesis:

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

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.

JSON

```
{
    "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.

JSON

```
{
    "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.

JSON

```
{
    "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.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": [
                "arn:aws:kinesis:*:111122223333:stream/*"
            ]
        }
    ]
}
```

Bagikan aliran data Anda dengan akun lain

> **Note**
>
> Kinesis Producer Library saat ini tidak mendukung penentuan aliran ARN saat menulis ke aliran data. Gunakan AWS SDK jika Anda ingin menulis ke aliran data lintas akun.

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 principal 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.

Aktifkan akses lintas akun

Untuk mengaktifkan akses lintas akun, Anda dapat menentukan secara spesifik seluruh akun atau entitas IAM di akun lain sebagai principal dalam kebijakan berbasis sumber daya. Menambahkan principal akun silang ke kebijakan berbasis sumber daya hanya setengah 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 ke principal dalam akun yang sama, tidak diperlukan kebijakan berbasis identitas tambahan.

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, di mana utama dapat melakukan tindakan pada sumber daya, 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

Tindakan	Tingkat akses
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 aliran data Kinesis

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

JSON

```
{
    "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

JSON

```
{
    "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:

JSON

```
{
    "Version":"2012-10-17",
    "Id": "__default_sharedthroughput_read_policy_ID",
    "Statement": [
        {
            "Sid": "consumerreadstatement",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:role/role-name"
            },
            "Action": [
                "kinesis:DescribeStreamSummary",
                "kinesis:ListShards"
            ],
            "Resource": "arn:aws:kinesis:us-east-2:123456789012:stream/
datastreamABC"
        }
```

```
    ]
}
```

Pernyataan kebijakan konsumen:

JSON

```
{
    "Version":"2012-10-17",
    "Id": "__default_efo_read_policy_ID",
    "Statement": [
        {
            "Sid": "eforeadstatement",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333: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 Konsol Manajemen AWS, 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 membantu mencegah akses luas diberikan melalui kebijakan sumber daya yang secara langsung dilampirkan ke aliran data atau konsumen terdaftar. Selain itu, periksa dengan cermat kebijakan berikut untuk mengonfirmasi bahwa kebijakan tersebut tidak memberikan akses luas:
  - Kebijakan berbasis identitas yang dilampirkan pada AWS prinsipal terkait (misalnya, peran IAM)
  - Kebijakan berbasis sumber daya yang dilampirkan pada AWS sumber daya terkait (misalnya, kunci KMS) AWS Key Management Service
- AWS Kepala Pelayanan tidak didukung oleh kepala sekolah untuk mencegah calon deputi yang bingung.
- Prinsipal federasi tidak didukung.
- Pengguna kanonik tidak IDs didukung.
- Ukuran polis tidak boleh melebihi 20KB.

Bagikan 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. Bagikan 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.

Bagikan 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 aliran data Kinesis 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 sediakan 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 aliran data Kinesis Streams, lihat.

4. Pilih Buat kebijakan atau gunakan PutResourcePolicyuntuk melampirkan kebijakan ke sumber daya Anda.

Bagikan 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 Kepatuhan AWS.

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 CSPM— 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 secara otomatis melakukan failover di antara Zona Ketersediaan tanpa gangguan. Zona Ketersediaan memiliki ketersediaan dan toleransi kesalahan yang lebih baik, dan dapat diskalakan dibandingkan infrastruktur biasa yang terdiri dari satu atau beberapa pusat data.

Untuk informasi selengkapnya tentang AWS Wilayah dan Availability Zone, lihat Infrastruktur AWS 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 pelambatan.

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 menggunakan ID kunci akses dan kunci akses rahasia yang terkait dengan principal IAM. Atau Anda dapat menggunakan AWS Security Token Service (AWS STS) untuk menghasilkan kredensial keamanan sementara untuk menandatangani permintaan.

Praktik keamanan terbaik 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 kredensil 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 kredensyal 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

Menerapkan enkripsi sisi server dalam 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 CloudTrail terintegrasi 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.

Untuk informasi selengkapnya, lihat the section called “Log panggilan API Amazon Kinesis Data Streams dengan AWS CloudTrail”.

Menggunakan layanan ini dengan AWS SDK

AWS kit pengembangan perangkat lunak (SDKs) tersedia untuk banyak bahasa pemrograman populer. Setiap SDK menyediakan API, contoh kode, dan dokumentasi yang memudahkan developer untuk membangun aplikasi dalam bahasa pilihan mereka.

Dokumentasi SDK	Contoh kode
AWS SDK untuk C++	AWS SDK untuk C++ contoh kode
AWS CLI	AWS CLI contoh kode
AWS SDK untuk Go	AWS SDK untuk Go contoh kode
AWS SDK untuk Java	AWS SDK untuk Java contoh kode
AWS SDK untuk JavaScript	AWS SDK untuk JavaScript contoh kode
AWS SDK untuk Kotlin	AWS SDK untuk Kotlin contoh kode
AWS SDK untuk .NET	AWS SDK untuk .NET contoh kode
AWS SDK untuk PHP	AWS SDK untuk PHP contoh kode
Alat AWS untuk PowerShell	Alat AWS untuk PowerShell contoh kode
AWS SDK untuk Python (Boto3)	AWS SDK untuk Python (Boto3) contoh kode
AWS SDK untuk Ruby	AWS SDK untuk Ruby contoh kode
AWS SDK untuk Rust	AWS SDK untuk Rust contoh kode
AWS SDK untuk SAP ABAP	AWS SDK untuk SAP ABAP contoh kode
AWS SDK untuk Swift	AWS SDK untuk Swift contoh kode

Ketersediaan contoh

Tidak dapat menemukan apa yang Anda butuhkan? Minta contoh kode menggunakan tautan Berikan umpan balik di bagian bawah halaman ini.

Contoh kode untuk Kinesis menggunakan AWS SDKs

Contoh kode berikut menunjukkan cara menggunakan Kinesis dengan kit pengembangan AWS perangkat lunak (SDK).

Dasar-dasar adalah contoh kode yang menunjukkan kepada Anda bagaimana melakukan operasi penting dalam suatu layanan.

Tindakan merupakan kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Sementara tindakan menunjukkan cara memanggil fungsi layanan individual, Anda dapat melihat tindakan dalam konteks dalam skenario terkait.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan layanan ini dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Contoh kode

- Contoh dasar untuk Kinesis menggunakan AWS SDKs
  - Pelajari dasar-dasar Kinesis dengan SDK AWS
  - Tindakan untuk Kinesis menggunakan AWS SDKs
    - Gunakan AddTagsToStream dengan AWS SDK atau CLI
    - Gunakan CreateStream dengan AWS SDK atau CLI
    - Gunakan DeleteStream dengan AWS SDK atau CLI
    - Gunakan DeregisterStreamConsumer dengan AWS SDK atau CLI
    - Gunakan DescribeStream dengan AWS SDK atau CLI
    - Gunakan GetRecords dengan AWS SDK atau CLI
    - Gunakan GetShardIterator dengan CLI
    - Gunakan ListStreamConsumers dengan AWS SDK
    - Gunakan ListStreams dengan AWS SDK atau CLI
    - Gunakan ListTagsForStream dengan AWS SDK atau CLI
    - Gunakan PutRecord dengan AWS SDK atau CLI
    - Gunakan PutRecords dengan AWS SDK atau CLI
    - Gunakan RegisterStreamConsumer dengan AWS SDK atau CLI
- Contoh tanpa server untuk Kinesis

- Memanggil fungsi Lambda dari pemicu Kinesis
- Melaporkan kegagalan item batch untuk fungsi Lambda dengan pemicu Kinesis

Contoh dasar untuk Kinesis menggunakan AWS SDKs

Contoh kode berikut menunjukkan cara menggunakan dasar-dasar Amazon Kinesis dengan. AWS SDKs

Contoh

- Pelajari dasar-dasar Kinesis dengan SDK AWS
- Tindakan untuk Kinesis menggunakan AWS SDKs
  - Gunakan AddTagsToStream dengan AWS SDK atau CLI
  - Gunakan CreateStream dengan AWS SDK atau CLI
  - Gunakan DeleteStream dengan AWS SDK atau CLI
  - Gunakan DeregisterStreamConsumer dengan AWS SDK atau CLI
  - Gunakan DescribeStream dengan AWS SDK atau CLI
  - Gunakan GetRecords dengan AWS SDK atau CLI
  - Gunakan GetShardIterator dengan CLI
  - Gunakan ListStreamConsumers dengan AWS SDK
  - Gunakan ListStreams dengan AWS SDK atau CLI
  - Gunakan ListTagsForStream dengan AWS SDK atau CLI
  - Gunakan PutRecord dengan AWS SDK atau CLI
  - Gunakan PutRecords dengan AWS SDK atau CLI
  - Gunakan RegisterStreamConsumer dengan AWS SDK atau CLI

Pelajari dasar-dasar Kinesis dengan SDK AWS

Contoh kode berikut ini menunjukkan cara untuk melakukan:

- Buat aliran dan letakkan catatan di dalamnya.
- Buat iterator pecahan.
- Baca catatan, lalu bersihkan sumber daya.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   DATA lo_stream_describe_result TYPE REF TO /aws1/cl_knsdescrstreamoutput.
   DATA lo_stream_description TYPE REF TO /aws1/cl_knsstreamdescription.
   DATA lo_sharditerator TYPE REF TO /aws1/cl_knsgetsharditerator01.
   DATA lo_record_result TYPE REF TO /aws1/cl_knsputrecordoutput.

   "Create stream."
   TRY.
       lo_kns->createstream(
           iv_streamname = iv_stream_name
           iv_shardcount = iv_shard_count ).
       MESSAGE 'Stream created.' TYPE 'I'.
     CATCH /aws1/cx_knsinvalidargumentex.
       MESSAGE 'The specified argument was not valid.' TYPE 'E'.
     CATCH /aws1/cx_knslimitexceededex.
       MESSAGE 'The request processing has failed because of a limit exceeded
exception.' TYPE 'E'.
     CATCH /aws1/cx_knsresourceinuseex.
       MESSAGE 'The request processing has failed because the resource is in
use.' TYPE 'E'.
   ENDTRY.

   "Wait for stream to becomes active."
   lo_stream_describe_result = lo_kns->describestream( iv_streamname =
iv_stream_name ).
   lo_stream_description = lo_stream_describe_result->get_streamdescription( ).
   WHILE lo_stream_description->get_streamstatus( ) <> 'ACTIVE'.
     IF sy-index = 30.
       EXIT.               "maximum 5 minutes"
     ENDIF.
     WAIT UP TO 10 SECONDS.
```

```
      lo_stream_describe_result = lo_kns->describestream( iv_streamname =
 iv_stream_name ).
      lo_stream_description = lo_stream_describe_result-
>get_streamdescription( ).
    ENDWHILE.

    "Create record."
    TRY.
        lo_record_result = lo_kns->putrecord(
            iv_streamname = iv_stream_name
            iv_data       = iv_data
            iv_partitionkey = iv_partition_key ).
        MESSAGE 'Record created.' TYPE 'I'.
      CATCH /aws1/cx_knsinvalidargumentex.
        MESSAGE 'The specified argument was not valid.' TYPE 'E'.
      CATCH /aws1/cx_knskmsaccessdeniedex.
        MESSAGE 'You do not have permission to perform this AWS KMS action.' TYPE
 'E'.
      CATCH /aws1/cx_knskmsdisabledex.
        MESSAGE 'KMS key used is disabled.' TYPE 'E'.
      CATCH /aws1/cx_knskmsinvalidstateex.
        MESSAGE 'KMS key used is in an invalid state. ' TYPE 'E'.
      CATCH /aws1/cx_knskmsnotfoundex.
        MESSAGE 'KMS key used is not found.' TYPE 'E'.
      CATCH /aws1/cx_knskmsoptinrequired.
        MESSAGE 'KMS key option is required.' TYPE 'E'.
      CATCH /aws1/cx_knskmsthrottlingex.
        MESSAGE 'The rate of requests to AWS KMS is exceeding the request
 quotas.' TYPE 'E'.
      CATCH /aws1/cx_knsprovthruputexcdex.
        MESSAGE 'The request rate for the stream is too high, or the requested
 data is too large for the available throughput.' TYPE 'E'.
      CATCH /aws1/cx_knsresourcenotfoundex.
        MESSAGE 'Resource being accessed is not found.' TYPE 'E'.
    ENDTRY.

    "Create a shard iterator in order to read the record."
    TRY.
        lo_sharditerator = lo_kns->getsharditerator(
          iv_shardid = lo_record_result->get_shardid( )
          iv_sharditeratortype = iv_sharditeratortype
          iv_streamname = iv_stream_name ).
        MESSAGE 'Shard iterator created.' TYPE 'I'.
      CATCH /aws1/cx_knsinvalidargumentex.
```

```
        MESSAGE 'The specified argument was not valid.' TYPE 'E'.
      CATCH /aws1/cx_knsprovthruputexcdex.
        MESSAGE 'The request rate for the stream is too high, or the requested
data is too large for the available throughput.' TYPE 'E'.
      CATCH /aws1/cx_sgmresourcenotfound.
        MESSAGE 'Resource being accessed is not found.' TYPE 'E'.
    ENDTRY.

    "Read the record."
    TRY.
        oo_result = lo_kns->getrecords(                    " oo_result is
returned for testing purposes. "
            iv_sharditerator   = lo_sharditerator->get_sharditerator( ) ).
        MESSAGE 'Shard iterator created.' TYPE 'I'.
      CATCH /aws1/cx_knsexpirediteratorex.
        MESSAGE 'Iterator expired.' TYPE 'E'.
      CATCH /aws1/cx_knsinvalidargumentex.
        MESSAGE 'The specified argument was not valid.' TYPE 'E'.
      CATCH /aws1/cx_knskmsaccessdeniedex.
        MESSAGE 'You do not have permission to perform this AWS KMS action.' TYPE
'E'.
      CATCH /aws1/cx_knskmsdisabledex.
        MESSAGE 'KMS key used is disabled.' TYPE 'E'.
      CATCH /aws1/cx_knskmsinvalidstateex.
        MESSAGE 'KMS key used is in an invalid state. ' TYPE 'E'.
      CATCH /aws1/cx_knskmsnotfoundex.
        MESSAGE 'KMS key used is not found.' TYPE 'E'.
      CATCH /aws1/cx_knskmsoptinrequired.
        MESSAGE 'KMS key option is required.' TYPE 'E'.
      CATCH /aws1/cx_knskmsthrottlingex.
        MESSAGE 'The rate of requests to AWS KMS is exceeding the request
quotas.' TYPE 'E'.
      CATCH /aws1/cx_knsprovthruputexcdex.
        MESSAGE 'The request rate for the stream is too high, or the requested
data is too large for the available throughput.' TYPE 'E'.
      CATCH /aws1/cx_knsresourcenotfoundex.
        MESSAGE 'Resource being accessed is not found.' TYPE 'E'.
    ENDTRY.

    "Delete stream."
    TRY.
        lo_kns->deletestream(
            iv_streamname = iv_stream_name ).
        MESSAGE 'Stream deleted.' TYPE 'I'.
```

```
      CATCH /aws1/cx_knslimitexceededex.
        MESSAGE 'The request processing has failed because of a limit exceeded
exception.' TYPE 'E'.
      CATCH /aws1/cx_knsresourceinuseex.
        MESSAGE 'The request processing has failed because the resource is in
use.' TYPE 'E'.
    ENDTRY.
```

- Untuk mengetahui hal detail mengenai API, silakan lihat topik-topik berikut di referensi API AWS SDK untuk ABAP SAP.
  - CreateStream
  - DeleteStream
  - GetRecords
  - GetShardIterator
  - PutRecord

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan layanan ini dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Tindakan untuk Kinesis menggunakan AWS SDKs

Contoh kode berikut menunjukkan bagaimana melakukan tindakan Kinesis individu dengan. AWS SDKs Setiap contoh menyertakan tautan ke GitHub, di mana Anda dapat menemukan instruksi untuk mengatur dan menjalankan kode.

Contoh berikut hanya mencakup tindakan yang paling umum digunakan. Untuk daftar lengkapnya, lihat Referensi API Amazon Kinesis.

Contoh

- Gunakan AddTagsToStream dengan AWS SDK atau CLI
- Gunakan CreateStream dengan AWS SDK atau CLI
- Gunakan DeleteStream dengan AWS SDK atau CLI
- Gunakan DeregisterStreamConsumer dengan AWS SDK atau CLI
- Gunakan DescribeStream dengan AWS SDK atau CLI
- Gunakan GetRecords dengan AWS SDK atau CLI

- Gunakan GetShardIterator dengan CLI
- Gunakan ListStreamConsumers dengan AWS SDK
- Gunakan ListStreams dengan AWS SDK atau CLI
- Gunakan ListTagsForStream dengan AWS SDK atau CLI
- Gunakan PutRecord dengan AWS SDK atau CLI
- Gunakan PutRecords dengan AWS SDK atau CLI
- Gunakan RegisterStreamConsumer dengan AWS SDK atau CLI

Gunakan **`AddTagsToStream`** dengan AWS SDK atau CLI

Contoh kode berikut menunjukkan cara menggunakan`AddTagsToStream`.

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.Kinesis;
using Amazon.Kinesis.Model;

/// <summary>
/// This example shows how to apply key/value pairs to an Amazon Kinesis
/// stream.
/// </summary>
public class TagStream
{
    public static async Task Main()
    {
        IAmazonKinesis client = new AmazonKinesisClient();

        string streamName = "AmazonKinesisStream";
```

```
            var tags = new Dictionary<string, string>
            {
                { "Project", "Sample Kinesis Project" },
                { "Application", "Sample Kinesis App" },
            };

            var success = await ApplyTagsToStreamAsync(client, streamName, tags);

            if (success)
            {
                Console.WriteLine($"Taggs successfully added to {streamName}.");
            }
            else
            {
                Console.WriteLine("Tags were not added to the stream.");
            }
        }

        /// <summary>
        /// Applies the set of tags to the named Kinesis stream.
        /// </summary>
        /// <param name="client">The initialized Kinesis client.</param>
        /// <param name="streamName">The name of the Kinesis stream to which
        /// the tags will be attached.</param>
        /// <param name="tags">A sictionary containing key/value pairs which
        /// will be used to create the Kinesis tags.</param>
        /// <returns>A Boolean value which represents the success or failure
        /// of AddTagsToStreamAsync.</returns>
        public static async Task<bool> ApplyTagsToStreamAsync(
            IAmazonKinesis client,
            string streamName,
            Dictionary<string, string> tags)
        {
            var request = new AddTagsToStreamRequest
            {
                StreamName = streamName,
                Tags = tags,
            };

            var response = await client.AddTagsToStreamAsync(request);

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
    }
```

- Untuk detail API, lihat AddTagsToStreamdi Referensi AWS SDK untuk .NET API.

CLI

AWS CLI

Untuk menambahkan tag ke aliran data

`add-tags-to-stream`Contoh berikut memberikan tag dengan kunci `samplekey` dan nilai `example` untuk aliran tertentu.

```
aws kinesis add-tags-to-stream \
    --stream-name samplestream \
    --tags samplekey=example
```

Perintah ini tidak menghasilkan output.

Untuk informasi selengkapnya, lihat Menandai Streaming Anda di Panduan Pengembang Amazon Kinesis Data Streams.

- Untuk detail API, lihat AddTagsToStreamdi Referensi AWS CLI Perintah.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan layanan ini dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan **`CreateStream`** dengan AWS SDK atau CLI

Contoh kode berikut menunjukkan cara menggunakan`CreateStream`.

Contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Anda dapat melihat tindakan ini dalam konteks dalam contoh kode berikut:

- Pelajari dasar-dasarnya

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
using System;
using System.Threading.Tasks;
using Amazon.Kinesis;
using Amazon.Kinesis.Model;

/// <summary>
/// This example shows how to create a new Amazon Kinesis stream.
/// </summary>
public class CreateStream
{
    public static async Task Main()
    {
        IAmazonKinesis client = new AmazonKinesisClient();

        string streamName = "AmazonKinesisStream";
        int shardCount = 1;

        var success = await CreateNewStreamAsync(client, streamName,
shardCount);
        if (success)
        {
            Console.WriteLine($"The stream, {streamName} successfully
created.");
        }
    }

    /// <summary>
    /// Creates a new Kinesis stream.
    /// </summary>
    /// <param name="client">An initialized Kinesis client.</param>
    /// <param name="streamName">The name for the new stream.</param>
    /// <param name="shardCount">The number of shards the new stream will
```

```
        /// use. The throughput of the stream is a function of the number of
        /// shards; more shards are required for greater provisioned
        /// throughput.</param>
        /// <returns>A Boolean value indicating whether the stream was created.</
returns>
        public static async Task<bool> CreateNewStreamAsync(IAmazonKinesis
 client, string streamName, int shardCount)
        {
            var request = new CreateStreamRequest
            {
                StreamName = streamName,
                ShardCount = shardCount,
            };

            var response = await client.CreateStreamAsync(request);

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
    }
```

- Untuk detail API, lihat CreateStreamdi Referensi AWS SDK untuk .NET API.

CLI

AWS CLI

Untuk membuat aliran data

`create-stream`Contoh berikut membuat aliran data bernama samplestream dengan 3 pecahan.

```
aws kinesis create-stream \
    --stream-name samplestream \
    --shard-count 3
```

Perintah ini tidak menghasilkan output.

Untuk informasi selengkapnya, lihat Membuat Stream di Panduan Pengembang Amazon Kinesis Data Streams.

- Untuk detail API, lihat CreateStreamdi Referensi AWS CLI Perintah.

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.kinesis.KinesisClient;
import software.amazon.awssdk.services.kinesis.model.CreateStreamRequest;
import software.amazon.awssdk.services.kinesis.model.KinesisException;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class CreateDataStream {
    public static void main(String[] args) {

        final String usage = """

                Usage:
                    <streamName>

                Where:
                    streamName - The Amazon Kinesis data stream (for example,
 StockTradeStream).
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }
```

```
        String streamName = args[0];
        Region region = Region.US_EAST_1;
        KinesisClient kinesisClient = KinesisClient.builder()
                .region(region)
                .build();
        createStream(kinesisClient, streamName);
        System.out.println("Done");
        kinesisClient.close();
    }

    public static void createStream(KinesisClient kinesisClient, String
 streamName) {
        try {
            CreateStreamRequest streamReq = CreateStreamRequest.builder()
                    .streamName(streamName)
                    .shardCount(1)
                    .build();

            kinesisClient.createStream(streamReq);

        } catch (KinesisException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- Untuk detail API, lihat CreateStreamdi Referensi AWS SDK for Java 2.x API.

PowerShell

Alat untuk PowerShell V4

Contoh 1: Membuat aliran baru. Secara default cmdlet ini tidak mengembalikan output sehingga PassThru sakelar - ditambahkan untuk mengembalikan nilai yang diberikan ke StreamName parameter - untuk penggunaan selanjutnya.

```
$streamName = New-KINStream -StreamName "mystream" -ShardCount 1 -PassThru
```

- Untuk detail API, lihat CreateStreamdi Referensi Alat AWS untuk PowerShell Cmdlet (V4).

Alat untuk PowerShell V5

Contoh 1: Membuat aliran baru.

```
New-KINStream -StreamName "mystream" -ShardCount 1
```

- Untuk detail API, lihat CreateStreamdi Referensi Alat AWS untuk PowerShell Cmdlet (V5).

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
class KinesisStream:
    """Encapsulates a Kinesis stream."""

    def __init__(self, kinesis_client):
        """
        :param kinesis_client: A Boto3 Kinesis client.
        """
        self.kinesis_client = kinesis_client
        self.name = None
        self.details = None
        self.stream_exists_waiter = kinesis_client.get_waiter("stream_exists")


    def create(self, name, wait_until_exists=True):
        """
        Creates a stream.

        :param name: The name of the stream.
        :param wait_until_exists: When True, waits until the service reports that
                                  the stream exists, then queries for its
 metadata.
        """
        try:
```

```
            self.kinesis_client.create_stream(StreamName=name, ShardCount=1)
            self.name = name
            logger.info("Created stream %s.", name)
            if wait_until_exists:
                logger.info("Waiting until exists.")
                self.stream_exists_waiter.wait(StreamName=name)
                self.describe(name)
        except ClientError:
            logger.exception("Couldn't create stream %s.", name)
            raise
```

- Untuk detail API, lihat CreateStreamdi AWS SDK for Python (Boto3) Referensi API.

Rust

SDK for Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
async fn make_stream(client: &Client, stream: &str) -> Result<(), Error> {
    client
        .create_stream()
        .stream_name(stream)
        .shard_count(4)
        .send()
        .await?;

    println!("Created stream");

    Ok(())
}
```

- Untuk detail API, lihat CreateStreamreferensi AWS SDK for Rust API.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   TRY.
       lo_kns->createstream(
           iv_streamname = iv_stream_name
           iv_shardcount = iv_shard_count ).
       MESSAGE 'Stream created.' TYPE 'I'.
     CATCH /aws1/cx_knsinvalidargumentex.
       MESSAGE 'The specified argument was not valid.' TYPE 'E'.
     CATCH /aws1/cx_knslimitexceededex.
       MESSAGE 'The request processing has failed because of a limit exceed
exception.' TYPE 'E'.
     CATCH /aws1/cx_knsresourceinuseex.
       MESSAGE 'The request processing has failed because the resource is in
use.' TYPE 'E'.
   ENDTRY.
```

- Untuk detail API, lihat CreateStreamdi AWS SDK untuk referensi SAP ABAP API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan layanan ini dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan **`DeleteStream`** dengan AWS SDK atau CLI

Contoh kode berikut menunjukkan cara menggunakan`DeleteStream`.

Contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Anda dapat melihat tindakan ini dalam konteks dalam contoh kode berikut:

- Pelajari dasar-dasarnya

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
using System;
using System.Threading.Tasks;
using Amazon.Kinesis;
using Amazon.Kinesis.Model;

/// <summary>
/// Shows how to delete an Amazon Kinesis stream.
/// </summary>
public class DeleteStream
{
    public static async Task Main()
    {
        IAmazonKinesis client = new AmazonKinesisClient();
        string streamName = "AmazonKinesisStream";

        var success = await DeleteStreamAsync(client, streamName);

        if (success)
        {
            Console.WriteLine($"Stream, {streamName} successfully deleted.");
        }
        else
        {
            Console.WriteLine("Stream not deleted.");
        }
    }

    /// <summary>
    /// Deletes a Kinesis stream.
    /// </summary>
    /// <param name="client">An initialized Kinesis client object.</param>
    /// <param name="streamName">The name of the string to delete.</param>
```

```
        /// <returns>A Boolean value representing the success of the operation.</
returns>
        public static async Task<bool> DeleteStreamAsync(IAmazonKinesis client,
 string streamName)
        {
            // If EnforceConsumerDeletion is true, any consumers
            // of this stream will also be deleted. If it is set
            // to false and this stream has any consumers, the
            // call will fail with a ResourceInUseException.
            var request = new DeleteStreamRequest
            {
                StreamName = streamName,
                EnforceConsumerDeletion = true,
            };

            var response = await client.DeleteStreamAsync(request);

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
    }
```

- Untuk detail API, lihat DeleteStreamdi Referensi AWS SDK untuk .NET API.

CLI

AWS CLI

Untuk menghapus aliran data

`delete-stream`Contoh berikut menghapus aliran data yang ditentukan.

```
aws kinesis delete-stream \
    --stream-name samplestream
```

Perintah ini tidak menghasilkan output.

Untuk informasi selengkapnya, lihat Menghapus Stream di Panduan Pengembang Amazon Kinesis Data Streams.

- Untuk detail API, lihat DeleteStreamdi Referensi AWS CLI Perintah.

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.kinesis.KinesisClient;
import software.amazon.awssdk.services.kinesis.model.DeleteStreamRequest;
import software.amazon.awssdk.services.kinesis.model.KinesisException;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DeleteDataStream {

    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <streamName>

                Where:
                    streamName - The Amazon Kinesis data stream (for example,
 StockTradeStream)
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }
```

```
        String streamName = args[0];
        Region region = Region.US_EAST_1;
        KinesisClient kinesisClient = KinesisClient.builder()
                .region(region)
                .build();

        deleteStream(kinesisClient, streamName);
        kinesisClient.close();
        System.out.println("Done");
    }

    public static void deleteStream(KinesisClient kinesisClient, String
 streamName) {
        try {
            DeleteStreamRequest delStream = DeleteStreamRequest.builder()
                    .streamName(streamName)
                    .build();

            kinesisClient.deleteStream(delStream);

        } catch (KinesisException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- Untuk detail API, lihat DeleteStreamdi Referensi AWS SDK for Java 2.x API.

PowerShell

Alat untuk PowerShell V4

Contoh 1: Menghapus aliran yang ditentukan. Anda diminta untuk konfirmasi sebelum perintah dijalankan. Untuk menekan konfirmasi yang diminta, gunakan sakelar -Force.

```
Remove-KINStream -StreamName "mystream"
```

- Untuk detail API, lihat DeleteStreamdi Referensi Alat AWS untuk PowerShell Cmdlet (V4).

Alat untuk PowerShell V5

Contoh 1: Menghapus aliran yang ditentukan. Anda diminta untuk konfirmasi sebelum perintah dijalankan. Untuk menekan konfirmasi yang diminta, gunakan sakelar -Force.

```
Remove-KINStream -StreamName "mystream"
```

- Untuk detail API, lihat DeleteStreamdi Referensi Alat AWS untuk PowerShell Cmdlet (V5).

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
class KinesisStream:
    """Encapsulates a Kinesis stream."""

    def __init__(self, kinesis_client):
        """
        :param kinesis_client: A Boto3 Kinesis client.
        """
        self.kinesis_client = kinesis_client
        self.name = None
        self.details = None
        self.stream_exists_waiter = kinesis_client.get_waiter("stream_exists")


    def delete(self):
        """
        Deletes a stream.
        """
        try:
            self.kinesis_client.delete_stream(StreamName=self.name)
            self._clear()
            logger.info("Deleted stream %s.", self.name)
        except ClientError:
```

```
            logger.exception("Couldn't delete stream %s.", self.name)
            raise
```

- Untuk detail API, lihat DeleteStreamdi AWS SDK for Python (Boto3) Referensi API.

Rust

SDK for Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
async fn remove_stream(client: &Client, stream: &str) -> Result<(), Error> {
    client.delete_stream().stream_name(stream).send().await?;

    println!("Deleted stream.");

    Ok(())
}
```

- Untuk detail API, lihat DeleteStreamreferensi AWS SDK for Rust API.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    TRY.
```

```
        lo_kns->deletestream(
            iv_streamname = iv_stream_name ).
        MESSAGE 'Stream deleted.' TYPE 'I'.
      CATCH /aws1/cx_knslimitexceededex.
        MESSAGE 'The request processing has failed because of a limit exceed
exception.' TYPE 'E'.
      CATCH /aws1/cx_knsresourceinuseex.
        MESSAGE 'The request processing has failed because the resource is in
use.' TYPE 'E'.
    ENDTRY.
```

- Untuk detail API, lihat DeleteStreamdi AWS SDK untuk referensi SAP ABAP API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan layanan ini dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan **`DeregisterStreamConsumer`** dengan AWS SDK atau CLI

Contoh kode berikut menunjukkan cara menggunakan`DeregisterStreamConsumer`.

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
using System;
using System.Threading.Tasks;
using Amazon.Kinesis;
using Amazon.Kinesis.Model;

/// <summary>
/// Shows how to deregister a consumer from an Amazon Kinesis stream.
/// </summary>
public class DeregisterConsumer
```

```
    {
        public static async Task Main(string[] args)
        {
            IAmazonKinesis client = new AmazonKinesisClient();

            string streamARN = "arn:aws:kinesis:us-west-2:000000000000:stream/
AmazonKinesisStream";
            string consumerName = "CONSUMER_NAME";
            string consumerARN = "arn:aws:kinesis:us-west-2:000000000000:stream/
AmazonKinesisStream/consumer/CONSUMER_NAME:000000000000";

            var success = await DeregisterConsumerAsync(client, streamARN,
 consumerARN, consumerName);

            if (success)
            {
                Console.WriteLine($"{consumerName} successfully deregistered.");
            }
            else
            {
                Console.WriteLine($"{consumerName} was not successfully
 deregistered.");
            }
        }

        /// <summary>
        /// Deregisters a consumer from a Kinesis stream.
        /// </summary>
        /// <param name="client">An initialized Kinesis client object.</param>
        /// <param name="streamARN">The ARN of a Kinesis stream.</param>
        /// <param name="consumerARN">The ARN of the consumer.</param>
        /// <param name="consumerName">The name of the consumer.</param>
        /// <returns>A Boolean value representing the success of the operation.</
returns>
        public static async Task<bool> DeregisterConsumerAsync(
            IAmazonKinesis client,
            string streamARN,
            string consumerARN,
            string consumerName)
        {
            var request = new DeregisterStreamConsumerRequest
            {
                StreamARN = streamARN,
                ConsumerARN = consumerARN,
```

```
                ConsumerName = consumerName,
            };

            var response = await client.DeregisterStreamConsumerAsync(request);

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
    }
```

- Untuk detail API, lihat DeregisterStreamConsumerdi Referensi AWS SDK untuk .NET API.

CLI

AWS CLI

Untuk membatalkan pendaftaran konsumen aliran data

`deregister-stream-consumer`Contoh berikut membatalkan pendaftaran konsumen tertentu dari aliran data yang ditentukan.

```
aws kinesis deregister-stream-consumer \
    --stream-arn arn:aws:kinesis:us-west-2:123456789012:stream/samplestream \
    --consumer-name KinesisConsumerApplication
```

Perintah ini tidak menghasilkan output.

Untuk informasi selengkapnya, lihat Mengembangkan Konsumen dengan Fan-Out yang Ditingkatkan Menggunakan API Aliran Data Kinesis di Panduan Pengembang Amazon Kinesis Data Streams.

- Untuk detail API, lihat DeregisterStreamConsumerdi Referensi AWS CLI Perintah.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan layanan ini dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan **`DescribeStream`** dengan AWS SDK atau CLI

Contoh kode berikut menunjukkan cara menggunakan`DescribeStream`.

CLI

AWS CLI

Untuk menggambarkan aliran data

`describe-stream`Contoh berikut mengembalikan rincian aliran data yang ditentukan.

```
aws kinesis describe-stream \
    --stream-name samplestream
```

Output:

```
{
    "StreamDescription": {
        "Shards": [
            {
                "ShardId": "shardId-000000000000",
                "HashKeyRange": {
                    "StartingHashKey": "0",
                    "EndingHashKey": "113427455640312821154458202477256070484"
                },
                "SequenceNumberRange": {
                    "StartingSequenceNumber":
 "496008716829570364423650249261910734372510605801286533 14"
                }
            },
            {
                "ShardId": "shardId-000000000001",
                "HashKeyRange": {
                    "StartingHashKey": "113427455640312821154458202477256070485",
                    "EndingHashKey": "226854911280625642308916404954512140969"
                },
                "SequenceNumberRange": {
                    "StartingSequenceNumber":
 "496008716829793371875635555493326091555237089416346337 46"
                }
            },
            {
                "ShardId": "shardId-000000000002",
                "HashKeyRange": {
                    "StartingHashKey": "226854911280625642308916404954512140970",
                    "EndingHashKey": "340282366920938463463374607431768211455"
```

```
                },
                "SequenceNumberRange": {
                    "StartingSequenceNumber":
 "496008716830016379327620861724741448737963573031406141 78"
                }
            }
        ],
        "StreamARN": "arn:aws:kinesis:us-west-2:123456789012:stream/
samplestream",
        "StreamName": "samplestream",
        "StreamStatus": "ACTIVE",
        "RetentionPeriodHours": 24,
        "EnhancedMonitoring": [
            {
                "ShardLevelMetrics": []
            }
        ],
        "EncryptionType": "NONE",
        "KeyId": null,
        "StreamCreationTimestamp": 1572297168.0
    }
}
```

Untuk informasi selengkapnya, lihat Membuat dan Mengelola Streaming di Panduan Pengembang Amazon Kinesis Data Streams.

- Untuk detail API, lihat DescribeStreamdi Referensi AWS CLI Perintah.

PowerShell

Alat untuk PowerShell V4

Contoh 1: Mengembalikan rincian aliran tertentu.

```
Get-KINStream -StreamName "mystream"
```

Output:

```
HasMoreShards        : False
RetentionPeriodHours : 24
Shards               : {}
StreamARN            : arn:aws:kinesis:us-west-2:123456789012:stream/mystream
StreamName           : mystream
```

```
StreamStatus         : ACTIVE
```

- Untuk detail API, lihat DescribeStreamdi Referensi Alat AWS untuk PowerShell Cmdlet (V4).

Alat untuk PowerShell V5

Contoh 1: Mengembalikan rincian aliran tertentu.

```
Get-KINStream -StreamName "mystream"
```

Output:

```
HasMoreShards        : False
RetentionPeriodHours : 24
Shards               : {}
StreamARN            : arn:aws:kinesis:us-west-2:123456789012:stream/mystream
StreamName           : mystream
StreamStatus         : ACTIVE
```

- Untuk detail API, lihat DescribeStreamdi Referensi Alat AWS untuk PowerShell Cmdlet (V5).

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
class KinesisStream:
    """Encapsulates a Kinesis stream."""

    def __init__(self, kinesis_client):
        """
        :param kinesis_client: A Boto3 Kinesis client.
        """
        self.kinesis_client = kinesis_client
        self.name = None
        self.details = None
```

```
        self.stream_exists_waiter = kinesis_client.get_waiter("stream_exists")


    def describe(self, name):
        """
        Gets metadata about a stream.

        :param name: The name of the stream.
        :return: Metadata about the stream.
        """
        try:
            response = self.kinesis_client.describe_stream(StreamName=name)
            self.name = name
            self.details = response["StreamDescription"]
            logger.info("Got stream %s.", name)
        except ClientError:
            logger.exception("Couldn't get %s.", name)
            raise
        else:
            return self.details
```

- Untuk detail API, lihat DescribeStreamdi AWS SDK for Python (Boto3) Referensi API.

Rust

SDK for Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
async fn show_stream(client: &Client, stream: &str) -> Result<(), Error> {
    let resp = client.describe_stream().stream_name(stream).send().await?;

    let desc = resp.stream_description.unwrap();

    println!("Stream description:");
    println!("  Name:             {}:", desc.stream_name());
```

```
    println!("  Status:            {:?}", desc.stream_status());
    println!("  Open shards:       {:?}", desc.shards.len());
    println!("  Retention (hours): {}", desc.retention_period_hours());
    println!("  Encryption:        {:?}", desc.encryption_type.unwrap());

    Ok(())
}
```

- Untuk detail API, lihat DescribeStreamreferensi AWS SDK for Rust API.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    TRY.
        oo_result = lo_kns->describestream(
            iv_streamname = iv_stream_name ).
        DATA(lt_stream_description) = oo_result->get_streamdescription( ).
        MESSAGE 'Streams retrieved.' TYPE 'I'.
      CATCH /aws1/cx_knslimitexceededex.
        MESSAGE 'The request processing has failed because of a limit exceed
exception.' TYPE 'E'.
      CATCH /aws1/cx_knsresourcenotfoundex.
        MESSAGE 'Resource being accessed is not found.' TYPE 'E'.
    ENDTRY.
```

- Untuk detail API, lihat DescribeStreamdi AWS SDK untuk referensi SAP ABAP API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan layanan ini dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan **`GetRecords`** dengan AWS SDK atau CLI

Contoh kode berikut menunjukkan cara menggunakan`GetRecords`.

Contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Anda dapat melihat tindakan ini dalam konteks dalam contoh kode berikut:

- Pelajari dasar-dasarnya

CLI

AWS CLI

Untuk mendapatkan catatan dari pecahan

`get-records`Contoh berikut mendapatkan catatan data dari pecahan Kinesis aliran data menggunakan iterator shard tertentu.

```
aws kinesis get-records \
    --shard-iterator AAAAAAAAAAF7/0mWD7IuHj1yGv/
TKuNgx2ukD5xipCY4cy4gU96orWwZwcSXh3K9tAmGYeOZyLZrvzzeOFVf9iN99hUPw/w/
b0YWYeehfNvnf1DYt5XpDJghLKr3DzgznkTmMymDP3R+3wRKeuEw6/kdxY2yKJH0veaiekaVc4N2VwK/
GvaGP2Hh9Fg7N++q0Adg6fIDQPt4p8RpavDbk+A4sL9SWGE1
```

Output:

```
{
    "Records": [],
    "MillisBehindLatest": 80742000
}
```

Untuk informasi selengkapnya, lihat Mengembangkan Konsumen Menggunakan API Kinesis Data Streams AWS dengan SDK for Java di Panduan Pengembang Amazon Kinesis Data Streams.

- Untuk detail API, lihat GetRecordsdi Referensi AWS CLI Perintah.

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.kinesis.KinesisClient;
import software.amazon.awssdk.services.kinesis.model.DescribeStreamResponse;
import software.amazon.awssdk.services.kinesis.model.DescribeStreamRequest;
import software.amazon.awssdk.services.kinesis.model.Shard;
import software.amazon.awssdk.services.kinesis.model.GetShardIteratorRequest;
import software.amazon.awssdk.services.kinesis.model.GetShardIteratorResponse;
import software.amazon.awssdk.services.kinesis.model.Record;
import software.amazon.awssdk.services.kinesis.model.GetRecordsRequest;
import software.amazon.awssdk.services.kinesis.model.GetRecordsResponse;
import java.util.ArrayList;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class GetRecords {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <streamName>

                Where:
```

```
                    streamName - The Amazon Kinesis data stream to read from (for
example, StockTradeStream).
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String streamName = args[0];
        Region region = Region.US_EAST_1;
        KinesisClient kinesisClient = KinesisClient.builder()
                .region(region)
                .build();

        getStockTrades(kinesisClient, streamName);
        kinesisClient.close();
    }

    public static void getStockTrades(KinesisClient kinesisClient, String
streamName) {
        String shardIterator;
        String lastShardId = null;
        DescribeStreamRequest describeStreamRequest =
DescribeStreamRequest.builder()
                .streamName(streamName)
                .build();

        List<Shard> shards = new ArrayList<>();
        DescribeStreamResponse streamRes;
        do {
            streamRes = kinesisClient.describeStream(describeStreamRequest);
            shards.addAll(streamRes.streamDescription().shards());

            if (shards.size() > 0) {
                lastShardId = shards.get(shards.size() - 1).shardId();
            }
        } while (streamRes.streamDescription().hasMoreShards());

        GetShardIteratorRequest itReq = GetShardIteratorRequest.builder()
                .streamName(streamName)
                .shardIteratorType("TRIM_HORIZON")
                .shardId(lastShardId)
                .build();
```

```
        GetShardIteratorResponse shardIteratorResult =
 kinesisClient.getShardIterator(itReq);
        shardIterator = shardIteratorResult.shardIterator();

        // Continuously read data records from shard.
        List<Record> records;

        // Create new GetRecordsRequest with existing shardIterator.
        // Set maximum records to return to 1000.
        GetRecordsRequest recordsRequest = GetRecordsRequest.builder()
                .shardIterator(shardIterator)
                .limit(1000)
                .build();

        GetRecordsResponse result = kinesisClient.getRecords(recordsRequest);

        // Put result into record list. Result may be empty.
        records = result.records();

        // Print records
        for (Record record : records) {
            SdkBytes byteBuffer = record.data();
            System.out.printf("Seq No: %s - %s%n", record.sequenceNumber(), new
 String(byteBuffer.asByteArray()));
        }
    }
}
```

- Untuk detail API, lihat GetRecordsdi Referensi AWS SDK for Java 2.x API.

PowerShell

Alat untuk PowerShell V4

Contoh 1: Contoh ini menunjukkan cara mengembalikan dan mengekstrak data dari serangkaian satu atau lebih catatan. Iterator supplierd untuk Get-KINRecord menentukan posisi awal dari catatan untuk kembali yang dalam contoh ini ditangkap ke dalam variabel, $records. Setiap catatan individu kemudian dapat diakses dengan mengindeks koleksi $records. Dengan asumsi data dalam catatan adalah teks yang dikodekan UTF-8,

perintah terakhir menunjukkan bagaimana Anda dapat mengekstrak data dari objek dan mengembalikannya sebagai teks ke konsol. MemoryStream

```
$records
$records = Get-KINRecord -ShardIterator "AAAAAAAAAAGIc....9VnbiRNaP"
```

Output:

```
MillisBehindLatest NextShardIterator            Records
------------------ -----------------            -------
0                  AAAAAAAAAAERNIq...uDn11HuUs  {Key1, Key2}
```

```
$records.Records[0]
```

Output:

```
ApproximateArrivalTimestamp Data                   PartitionKey SequenceNumber
--------------------------- ----                   ------------ --------------
3/7/2016 5:14:33 PM         System.IO.MemoryStream Key1
 4955986459776...931586
```

```
[Text.Encoding]::UTF8.GetString($records.Records[0].Data.ToArray())
```

Output:

```
test data from string
```

- Untuk detail API, lihat GetRecordsdi Referensi Alat AWS untuk PowerShell Cmdlet (V4).

Alat untuk PowerShell V5

Contoh 1: Contoh ini menunjukkan cara mengembalikan dan mengekstrak data dari serangkaian satu atau lebih catatan. Iterator supplierd untuk Get-KINRecord menentukan posisi awal dari catatan untuk kembali yang dalam contoh ini ditangkap ke dalam variabel, $records. Setiap catatan individu kemudian dapat diakses dengan mengindeks koleksi $records. Dengan asumsi data dalam catatan adalah teks yang dikodekan UTF-8, perintah terakhir menunjukkan bagaimana Anda dapat mengekstrak data dari objek dan mengembalikannya sebagai teks ke konsol. MemoryStream

```
$records
```

```
$records = Get-KINRecord -ShardIterator "AAAAAAAAAAGIc....9VnbiRNaP"
```

Output:

```
MillisBehindLatest NextShardIterator           Records
------------------ -----------------           -------
0                  AAAAAAAAAAERNIq...uDn11HuUs  {Key1, Key2}
```

```
$records.Records[0]
```

Output:

```
ApproximateArrivalTimestamp Data                   PartitionKey SequenceNumber
--------------------------- ----                   ------------ --------------
3/7/2016 5:14:33 PM         System.IO.MemoryStream Key1
 4955986459776...931586
```

```
[Text.Encoding]::UTF8.GetString($records.Records[0].Data.ToArray())
```

Output:

```
test data from string
```

- Untuk detail API, lihat GetRecordsdi Referensi Alat AWS untuk PowerShell Cmdlet (V5).

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
class KinesisStream:
    """Encapsulates a Kinesis stream."""

    def __init__(self, kinesis_client):
```

```
        """
        :param kinesis_client: A Boto3 Kinesis client.
        """
        self.kinesis_client = kinesis_client
        self.name = None
        self.details = None
        self.stream_exists_waiter = kinesis_client.get_waiter("stream_exists")


    def get_records(self, max_records):
        """
        Gets records from the stream. This function is a generator that first
gets
        a shard iterator for the stream, then uses the shard iterator to get
records
        in batches from the stream. The shard iterator can be accessed through
the
        'details' property, which is populated using the 'describe' function of
this class.
        Each batch of records is yielded back to the caller until the specified
        maximum number of records has been retrieved.

        :param max_records: The maximum number of records to retrieve.
        :return: Yields the current batch of retrieved records.
        """
        try:
            response = self.kinesis_client.get_shard_iterator(
                StreamName=self.name,
                ShardId=self.details["Shards"][0]["ShardId"],
                ShardIteratorType="LATEST",
            )
            shard_iter = response["ShardIterator"]
            record_count = 0
            while record_count < max_records:
                response = self.kinesis_client.get_records(
                    ShardIterator=shard_iter, Limit=10
                )
                shard_iter = response["NextShardIterator"]
                records = response["Records"]
                logger.info("Got %s records.", len(records))
                record_count += len(records)
                yield records
        except ClientError:
            logger.exception("Couldn't get records from stream %s.", self.name)
```

```
            raise



    def describe(self, name):
        """
        Gets metadata about a stream.

        :param name: The name of the stream.
        :return: Metadata about the stream.
        """
        try:
            response = self.kinesis_client.describe_stream(StreamName=name)
            self.name = name
            self.details = response["StreamDescription"]
            logger.info("Got stream %s.", name)
        except ClientError:
            logger.exception("Couldn't get %s.", name)
            raise
        else:
            return self.details
```

- Untuk detail API, lihat GetRecordsdi AWS SDK for Python (Boto3) Referensi API.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    TRY.
        oo_result = lo_kns->getrecords(             " oo_result is returned for
testing purposes. "
            iv_sharditerator = iv_shard_iterator ).
        DATA(lt_records) = oo_result->get_records( ).
        MESSAGE 'Record retrieved.' TYPE 'I'.
```

```
      CATCH /aws1/cx_knsexpirediteratorex.
        MESSAGE 'Iterator expired.' TYPE 'E'.
      CATCH /aws1/cx_knsinvalidargumentex.
        MESSAGE 'The specified argument was not valid.' TYPE 'E'.
      CATCH /aws1/cx_knskmsaccessdeniedex.
        MESSAGE 'You do not have permission to perform this AWS KMS action.' TYPE
'E'.
      CATCH /aws1/cx_knskmsdisabledex.
        MESSAGE 'KMS key used is disabled.' TYPE 'E'.
      CATCH /aws1/cx_knskmsinvalidstateex.
        MESSAGE 'KMS key used is in an invalid state. ' TYPE 'E'.
      CATCH /aws1/cx_knskmsnotfoundex.
        MESSAGE 'KMS key used is not found.' TYPE 'E'.
      CATCH /aws1/cx_knskmsoptinrequired.
        MESSAGE 'KMS key option is required.' TYPE 'E'.
      CATCH /aws1/cx_knskmsthrottlingex.
        MESSAGE 'The rate of requests to AWS KMS is exceeding the request
quotas.' TYPE 'E'.
      CATCH /aws1/cx_knsprovthruputexcdex.
        MESSAGE 'The request rate for the stream is too high, or the requested
data is too large for the available throughput.' TYPE 'E'.
      CATCH /aws1/cx_knsresourcenotfoundex.
        MESSAGE 'Resource being accessed is not found.' TYPE 'E'.
    ENDTRY.
```

- Untuk detail API, lihat GetRecordsdi AWS SDK untuk referensi SAP ABAP API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan layanan ini dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan **`GetShardIterator`** dengan CLI

Contoh kode berikut menunjukkan cara menggunakan`GetShardIterator`.

Contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Anda dapat melihat tindakan ini dalam konteks dalam contoh kode berikut:

- Pelajari dasar-dasarnya

CLI

AWS CLI

Untuk mendapatkan iterator shard

`get-shard-iterator`Contoh berikut menggunakan tipe iterator `AT_SEQUENCE_NUMBER` shard dan menghasilkan iterator shard untuk mulai membaca catatan data persis dari posisi yang dilambangkan dengan nomor urut yang ditentukan.

```
aws kinesis get-shard-iterator \
    --stream-name samplestream \
    --shard-id shardId-000000000001 \
    --shard-iterator-type LATEST
```

Output:

```
{
    "ShardIterator": "AAAAAAAAAAFEvJjIYI+3jw/4aqgH9FifJ+n48XWTh/
IFIsbILP6o5eDueD39NXNBfpZ10WL5K6ADXk8w+5H+Qhd9cFA9k268CPXCz/kebq1TGYI7Vy
+lUkA9BuN3xvATxMBGxRY3zYK05gqgvaIRn94O8SqeEqwhigwZxNWxID3Ej7YYYcxQi8Q/fIrCjGAy/
n2r5Z9G864YpWDfN9upNNQAR/iiOWKs"
}
```

Untuk informasi selengkapnya, lihat Mengembangkan Konsumen Menggunakan API Kinesis Data Streams AWS dengan SDK for Java di Panduan Pengembang Amazon Kinesis Data Streams.

- Untuk detail API, lihat GetShardIteratordi Referensi AWS CLI Perintah.

PowerShell

Alat untuk PowerShell V4

Contoh 1: Mengembalikan iterator shard untuk shard tertentu dan posisi awal. Rincian pengidentifikasi pecahan dan nomor urut dapat diperoleh dari output Get-KINStream cmdlet, dengan mereferensikan koleksi Shards dari objek aliran yang dikembalikan. Iterator yang dikembalikan dapat digunakan dengan Get-KINRecord cmdlet untuk menarik catatan data dalam pecahan.

```
Get-KINShardIterator -StreamName "mystream" -ShardId "shardId-000000000000" -
ShardIteratorType AT_SEQUENCE_NUMBER -StartingSequenceNumber "495598645..."
```

Output:

```
AAAAAAAAAAGIc....9VnbiRNaP
```

- Untuk detail API, lihat GetShardIteratordi Referensi Alat AWS untuk PowerShell Cmdlet (V4).

Alat untuk PowerShell V5

Contoh 1: Mengembalikan iterator shard untuk shard tertentu dan posisi awal. Rincian pengidentifikasi pecahan dan nomor urut dapat diperoleh dari output Get-KINStream cmdlet, dengan mereferensikan koleksi Shards dari objek aliran yang dikembalikan. Iterator yang dikembalikan dapat digunakan dengan Get-KINRecord cmdlet untuk menarik catatan data dalam pecahan.

```
Get-KINShardIterator -StreamName "mystream" -ShardId "shardId-000000000000" -
ShardIteratorType AT_SEQUENCE_NUMBER -StartingSequenceNumber "495598645..."
```

Output:

```
AAAAAAAAAAGIc....9VnbiRNaP
```

- Untuk detail API, lihat GetShardIteratordi Referensi Alat AWS untuk PowerShell Cmdlet (V5).

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan layanan ini dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan **`ListStreamConsumers`** dengan AWS SDK

Contoh kode berikut menunjukkan cara menggunakan`ListStreamConsumers`.

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    using System;
    using System.Collections.Generic;
    using System.Threading.Tasks;
    using Amazon.Kinesis;
    using Amazon.Kinesis.Model;

    /// <summary>
    /// List the consumers of an Amazon Kinesis stream.
    /// </summary>
    public class ListConsumers
    {
        public static async Task Main()
        {
            IAmazonKinesis client = new AmazonKinesisClient();

            string streamARN = "arn:aws:kinesis:us-east-2:000000000000:stream/
AmazonKinesisStream";
            int maxResults = 10;

            var consumers = await ListConsumersAsync(client, streamARN,
 maxResults);

            if (consumers.Count > 0)
            {
                consumers
                    .ForEach(c => Console.WriteLine($"Name: {c.ConsumerName} ARN:
 {c.ConsumerARN}"));
            }
            else
            {
                Console.WriteLine("No consumers found.");
            }
```

```
        }

        /// <summary>
        /// Retrieve a list of the consumers for a Kinesis stream.
        /// </summary>
        /// <param name="client">An initialized Kinesis client object.</param>
        /// <param name="streamARN">The ARN of the stream for which we want to
        /// retrieve a list of clients.</param>
        /// <param name="maxResults">The maximum number of results to return.</
param>
        /// <returns>A list of Consumer objects.</returns>
        public static async Task<List<Consumer>>
 ListConsumersAsync(IAmazonKinesis client, string streamARN, int maxResults)
        {
            var request = new ListStreamConsumersRequest
            {
                StreamARN = streamARN,
                MaxResults = maxResults,
            };

            var response = await client.ListStreamConsumersAsync(request);

            return response.Consumers;
        }
    }
```

- Untuk detail API, lihat ListStreamConsumersdi Referensi AWS SDK untuk .NET API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan layanan ini dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan **`ListStreams`** dengan AWS SDK atau CLI

Contoh kode berikut menunjukkan cara menggunakan`ListStreams`.

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.Kinesis;
using Amazon.Kinesis.Model;

/// <summary>
/// Retrieves and displays a list of existing Amazon Kinesis streams.
/// </summary>
public class ListStreams
{
    public static async Task Main(string[] args)
    {
        IAmazonKinesis client = new AmazonKinesisClient();
        var response = await client.ListStreamsAsync(new
ListStreamsRequest());

        List<string> streamNames = response.StreamNames;

        if (streamNames.Count > 0)
        {
            streamNames
                .ForEach(s => Console.WriteLine($"Stream name: {s}"));
        }
        else
        {
            Console.WriteLine("No streams were found.");
        }
    }
}
```

- Untuk detail API, lihat ListStreamsdi Referensi AWS SDK untuk .NET API.

CLI

AWS CLI

Untuk membuat daftar aliran data

`list-streams`Contoh berikut mencantumkan semua aliran data aktif di akun dan wilayah saat ini.

```
aws kinesis list-streams
```

Output:

```
{
    "StreamNames": [
        "samplestream",
        "samplestream1"
    ]
}
```

Untuk informasi selengkapnya, lihat Daftar Aliran di Panduan Pengembang Amazon Kinesis Data Streams.

- Untuk detail API, lihat ListStreamsdi Referensi AWS CLI Perintah.

Rust

SDK for Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
async fn show_streams(client: &Client) -> Result<(), Error> {
    let resp = client.list_streams().send().await?;
```

```
    println!("Stream names:");

    let streams = resp.stream_names;
    for stream in &streams {
        println!("  {}", stream);
    }

    println!("Found {} stream(s)", streams.len());

    Ok(())
}
```

- Untuk detail API, lihat ListStreamsreferensi AWS SDK for Rust API.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   TRY.
       oo_result = lo_kns->liststreams(        " oo_result is returned for
testing purposes. "
           "Set Limit to specify that a maximum of streams should be returned."
           iv_limit = iv_limit ).
       DATA(lt_streams) = oo_result->get_streamnames( ).
       MESSAGE 'Streams listed.' TYPE 'I'.
     CATCH /aws1/cx_knslimitexceededex.
       MESSAGE 'The request processing has failed because of a limit exceed
exception.' TYPE 'E'.
   ENDTRY.
```

- Untuk detail API, lihat ListStreamsdi AWS SDK untuk referensi SAP ABAP API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan layanan ini dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan **`ListTagsForStream`** dengan AWS SDK atau CLI

Contoh kode berikut menunjukkan cara menggunakan`ListTagsForStream`.

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.Kinesis;
using Amazon.Kinesis.Model;

/// <summary>
/// Shows how to list the tags that have been attached to an Amazon Kinesis
/// stream.
/// </summary>
public class ListTags
{
    public static async Task Main()
    {
        IAmazonKinesis client = new AmazonKinesisClient();
        string streamName = "AmazonKinesisStream";

        await ListTagsAsync(client, streamName);
    }

    /// <summary>
    /// List the tags attached to a Kinesis stream.
    /// </summary>
    /// <param name="client">An initialized Kinesis client object.</param>
```

```
        /// <param name="streamName">The name of the Kinesis stream for which you
        /// wish to display tags.</param>
        public static async Task ListTagsAsync(IAmazonKinesis client, string
streamName)
        {
            var request = new ListTagsForStreamRequest
            {
                StreamName = streamName,
                Limit = 10,
            };

            var response = await client.ListTagsForStreamAsync(request);
            DisplayTags(response.Tags);

            while (response.HasMoreTags)
            {
                request.ExclusiveStartTagKey = response.Tags[response.Tags.Count
- 1].Key;
                response = await client.ListTagsForStreamAsync(request);
            }
        }

        /// <summary>
        /// Displays the items in a list of Kinesis tags.
        /// </summary>
        /// <param name="tags">A list of the Tag objects to be displayed.</param>
        public static void DisplayTags(List<Tag> tags)
        {
            tags
                .ForEach(t => Console.WriteLine($"Key: {t.Key} Value:
{t.Value}"));
        }
    }
```

- Untuk detail API, lihat ListTagsForStreamdi Referensi AWS SDK untuk .NET API.

CLI

AWS CLI

Untuk mencantumkan tag untuk aliran data

`list-tags-for-stream`Contoh berikut mencantumkan tag yang dilampirkan ke aliran data yang ditentukan.

```
aws kinesis list-tags-for-stream \
    --stream-name samplestream
```

Output:

```
{
    "Tags": [
        {
            "Key": "samplekey",
            "Value": "example"
        }
    ],
    "HasMoreTags": false
}
```

Untuk informasi selengkapnya, lihat Menandai Streaming Anda di Panduan Pengembang Amazon Kinesis Data Streams.

- Untuk detail API, lihat ListTagsForStreamdi Referensi AWS CLI Perintah.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan layanan ini dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan **`PutRecord`** dengan AWS SDK atau CLI

Contoh kode berikut menunjukkan cara menggunakan`PutRecord`.

Contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Anda dapat melihat tindakan ini dalam konteks dalam contoh kode berikut:

- Pelajari dasar-dasarnya

CLI

AWS CLI

Untuk menulis catatan ke dalam aliran data

`put-record`Contoh berikut menulis catatan data tunggal ke dalam aliran data yang ditentukan menggunakan kunci partisi yang ditentukan.

```
aws kinesis put-record \
    --stream-name samplestream \
    --data sampledatarecord \
    --partition-key samplepartitionkey
```

Output:

```
{
    "ShardId": "shardId-000000000009",
    "SequenceNumber": "49600902273357540915989931256901506243878407835297513618",
    "EncryptionType": "KMS"
}
```

Untuk informasi selengkapnya, lihat Mengembangkan Produsen Menggunakan API Amazon Kinesis Data Streams AWS dengan SDK for Java di Panduan Pengembang Amazon Kinesis Data Streams.

- Untuk detail API, lihat PutRecorddi Referensi AWS CLI Perintah.

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.kinesis.KinesisClient;
import software.amazon.awssdk.services.kinesis.model.PutRecordRequest;
import software.amazon.awssdk.services.kinesis.model.KinesisException;
import software.amazon.awssdk.services.kinesis.model.DescribeStreamRequest;
import software.amazon.awssdk.services.kinesis.model.DescribeStreamResponse;

/**
```

```
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class StockTradesWriter {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <streamName>

                Where:
                    streamName - The Amazon Kinesis data stream to which records
 are written (for example, StockTradeStream)
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String streamName = args[0];
        Region region = Region.US_EAST_1;
        KinesisClient kinesisClient = KinesisClient.builder()
                .region(region)
                .build();

        // Ensure that the Kinesis Stream is valid.
        validateStream(kinesisClient, streamName);
        setStockData(kinesisClient, streamName);
        kinesisClient.close();
    }

    public static void setStockData(KinesisClient kinesisClient, String
 streamName) {
        try {
            // Repeatedly send stock trades with a 100 milliseconds wait in
 between.
            StockTradeGenerator stockTradeGenerator = new StockTradeGenerator();
```

```
            // Put in 50 Records for this example.
            int index = 50;
            for (int x = 0; x < index; x++) {
                StockTrade trade = stockTradeGenerator.getRandomTrade();
                sendStockTrade(trade, kinesisClient, streamName);
                Thread.sleep(100);
            }

        } catch (KinesisException | InterruptedException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        System.out.println("Done");
    }

    private static void sendStockTrade(StockTrade trade, KinesisClient
kinesisClient,
            String streamName) {
        byte[] bytes = trade.toJsonAsBytes();

        // The bytes could be null if there is an issue with the JSON
serialization by
        // the Jackson JSON library.
        if (bytes == null) {
            System.out.println("Could not get JSON bytes for stock trade");
            return;
        }

        System.out.println("Putting trade: " + trade);
        PutRecordRequest request = PutRecordRequest.builder()
                .partitionKey(trade.getTickerSymbol()) // We use the ticker
symbol as the partition key, explained in
                                                       // the Supplemental
Information section below.
                .streamName(streamName)
                .data(SdkBytes.fromByteArray(bytes))
                .build();

        try {
            kinesisClient.putRecord(request);
        } catch (KinesisException e) {
            System.err.println(e.getMessage());
        }
    }
```

```
    private static void validateStream(KinesisClient kinesisClient, String
 streamName) {
        try {
            DescribeStreamRequest describeStreamRequest =
 DescribeStreamRequest.builder()
                    .streamName(streamName)
                    .build();

            DescribeStreamResponse describeStreamResponse =
 kinesisClient.describeStream(describeStreamRequest);

            if (!
describeStreamResponse.streamDescription().streamStatus().toString().equals("ACTIVE"))
 {
                System.err.println("Stream " + streamName + " is not active.
 Please wait a few moments and try again.");
                System.exit(1);
            }

        } catch (KinesisException e) {
            System.err.println("Error found while describing the stream " +
 streamName);
            System.err.println(e);
            System.exit(1);
        }
    }
}
```

- Untuk detail API, lihat PutRecorddi Referensi AWS SDK for Java 2.x API.

PowerShell

Alat untuk PowerShell V4

Contoh 1: Menulis catatan yang berisi string yang dipasok ke parameter -Text.

```
Write-KINRecord -Text "test data from string" -StreamName "mystream" -
PartitionKey "Key1"
```

Contoh 2: Menulis catatan yang berisi data yang terkandung dalam file yang ditentukan. File diperlakukan sebagai urutan byte sehingga jika berisi teks, itu harus ditulis dengan pengkodean yang diperlukan sebelum menggunakannya dengan cmdlet ini.

```
Write-KINRecord -FilePath "C:\TestData.txt" -StreamName "mystream" -PartitionKey
 "Key2"
```

- Untuk detail API, lihat PutRecorddi Referensi Alat AWS untuk PowerShell Cmdlet (V4).

Alat untuk PowerShell V5

Contoh 1: Menulis catatan yang berisi string yang dipasok ke parameter -Text.

```
Write-KINRecord -Text "test data from string" -StreamName "mystream" -
PartitionKey "Key1"
```

Contoh 2: Menulis catatan yang berisi data yang terkandung dalam file yang ditentukan. File diperlakukan sebagai urutan byte sehingga jika berisi teks, itu harus ditulis dengan pengkodean yang diperlukan sebelum menggunakannya dengan cmdlet ini.

```
Write-KINRecord -FilePath "C:\TestData.txt" -StreamName "mystream" -PartitionKey
 "Key2"
```

- Untuk detail API, lihat PutRecorddi Referensi Alat AWS untuk PowerShell Cmdlet (V5).

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
class KinesisStream:
    """Encapsulates a Kinesis stream."""

    def __init__(self, kinesis_client):
```

```
        """
        :param kinesis_client: A Boto3 Kinesis client.
        """
        self.kinesis_client = kinesis_client
        self.name = None
        self.details = None
        self.stream_exists_waiter = kinesis_client.get_waiter("stream_exists")


    def put_record(self, data, partition_key):
        """
        Puts data into the stream. The data is formatted as JSON before it is
passed
        to the stream.

        :param data: The data to put in the stream.
        :param partition_key: The partition key to use for the data.
        :return: Metadata about the record, including its shard ID and sequence
number.
        """
        try:
            response = self.kinesis_client.put_record(
                StreamName=self.name, Data=json.dumps(data),
PartitionKey=partition_key
            )
            logger.info("Put record in stream %s.", self.name)
        except ClientError:
            logger.exception("Couldn't put record in stream %s.", self.name)
            raise
        else:
            return response
```

- Untuk detail API, lihat PutRecorddi AWS SDK for Python (Boto3) Referensi API.

Rust

SDK for Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
async fn add_record(client: &Client, stream: &str, key: &str, data: &str) ->
 Result<(), Error> {
    let blob = Blob::new(data);

    client
        .put_record()
        .data(blob)
        .partition_key(key)
        .stream_name(stream)
        .send()
        .await?;

    println!("Put data into stream.");

    Ok(())
}
```

- Untuk detail API, lihat PutRecordreferensi AWS SDK for Rust API.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   TRY.
       oo_result = lo_kns->putrecord(            " oo_result is returned for
testing purposes. "
           iv_streamname = iv_stream_name
           iv_data       = iv_data
           iv_partitionkey = iv_partition_key ).
       MESSAGE 'Record created.' TYPE 'I'.
     CATCH /aws1/cx_knsinvalidargumentex.
       MESSAGE 'The specified argument was not valid.' TYPE 'E'.
     CATCH /aws1/cx_knskmsaccessdeniedex.
       MESSAGE 'You do not have permission to perform this AWS KMS action.' TYPE
'E'.
     CATCH /aws1/cx_knskmsdisabledex.
       MESSAGE 'KMS key used is disabled.' TYPE 'E'.
     CATCH /aws1/cx_knskmsinvalidstateex.
       MESSAGE 'KMS key used is in an invalid state. ' TYPE 'E'.
     CATCH /aws1/cx_knskmsnotfoundex.
       MESSAGE 'KMS key used is not found.' TYPE 'E'.
     CATCH /aws1/cx_knskmsoptinrequired.
       MESSAGE 'KMS key option is required.' TYPE 'E'.
     CATCH /aws1/cx_knskmsthrottlingex.
       MESSAGE 'The rate of requests to AWS KMS is exceeding the request
quotas.' TYPE 'E'.
     CATCH /aws1/cx_knsprovthruputexcdex.
       MESSAGE 'The request rate for the stream is too high, or the requested
data is too large for the available throughput.' TYPE 'E'.
     CATCH /aws1/cx_knsresourcenotfoundex.
       MESSAGE 'Resource being accessed is not found.' TYPE 'E'.
   ENDTRY.
```

- Untuk detail API, lihat PutRecorddi AWS SDK untuk referensi SAP ABAP API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan layanan ini dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan **`PutRecords`** dengan AWS SDK atau CLI

Contoh kode berikut menunjukkan cara menggunakan`PutRecords`.

CLI

AWS CLI

Untuk menulis beberapa catatan ke dalam aliran data

`put-records`Contoh berikut menulis catatan data menggunakan kunci partisi yang ditentukan dan catatan data lain menggunakan kunci partisi yang berbeda dalam satu panggilan.

```
aws kinesis put-records \
    --stream-name samplestream \
    --
records Data=blob1,PartitionKey=partitionkey1 Data=blob2,PartitionKey=partitionkey2
```

Output:

```
{
    "FailedRecordCount": 0,
    "Records": [
        {
            "SequenceNumber":
 "49600883331171471519674795588238531498465399900093808706",
            "ShardId": "shardId-000000000004"
        },
        {
            "SequenceNumber":
 "49600902273357540915989931256902715169698037101720764562",
            "ShardId": "shardId-000000000009"
        }
    ],
    "EncryptionType": "KMS"
}
```

Untuk informasi selengkapnya, lihat Mengembangkan Produsen Menggunakan API Amazon Kinesis Data Streams AWS dengan SDK for Java di Panduan Pengembang Amazon Kinesis Data Streams.

- Untuk detail API, lihat PutRecordsdi Referensi AWS CLI Perintah.

JavaScript

SDK untuk JavaScript (v3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
import { PutRecordsCommand, KinesisClient } from "@aws-sdk/client-kinesis";

/**
 * Put multiple records into a Kinesis stream.
 * @param {{ streamArn: string }} config
 */
export const main = async ({ streamArn }) => {
  const client = new KinesisClient({});
  try {
    await client.send(
      new PutRecordsCommand({
        StreamARN: streamArn,
        Records: [
          {
            Data: new Uint8Array(),
            /**
             * Determines which shard in the stream the data record is assigned
 to.
             * Partition keys are Unicode strings with a maximum length limit of
 256
             * characters for each key. Amazon Kinesis Data Streams uses the
 partition
             * key as input to a hash function that maps the partition key and
             * associated data to a specific shard.
             */
            PartitionKey: "TEST_KEY",
          },
          {
            Data: new Uint8Array(),
            PartitionKey: "TEST_KEY",
          },
```

```
        ],
      }),
    );
  } catch (caught) {
    if (caught instanceof Error) {
      //
    } else {
      throw caught;
    }
  }
};

// Call function if run directly.
import { fileURLToPath } from "node:url";
import { parseArgs } from "node:util";

if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const options = {
    streamArn: {
      type: "string",
      description: "The ARN of the stream.",
    },
  };

  const { values } = parseArgs({ options });
  main(values);
}
```

- Untuk detail API, lihat PutRecordsdi Referensi AWS SDK untuk JavaScript API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan layanan ini dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan **`RegisterStreamConsumer`** dengan AWS SDK atau CLI

Contoh kode berikut menunjukkan cara menggunakan`RegisterStreamConsumer`.

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    using System;
    using System.Threading.Tasks;
    using Amazon.Kinesis;
    using Amazon.Kinesis.Model;

    /// <summary>
    /// This example shows how to register a consumer to an Amazon Kinesis
    /// stream.
    /// </summary>
    public class RegisterConsumer
    {
        public static async Task Main()
        {
            IAmazonKinesis client = new AmazonKinesisClient();
            string consumerName = "NEW_CONSUMER_NAME";
            string streamARN = "arn:aws:kinesis:us-east-2:000000000000:stream/
AmazonKinesisStream";

            var consumer = await RegisterConsumerAsync(client, consumerName,
 streamARN);

            if (consumer is not null)
            {
                Console.WriteLine($"{consumer.ConsumerName}");
            }
        }

        /// <summary>
        /// Registers the consumer to a Kinesis stream.
        /// </summary>
        /// <param name="client">The initialized Kinesis client object.</param>
```

```
        /// <param name="consumerName">A string representing the consumer.</
param>
        /// <param name="streamARN">The ARN of the stream.</param>
        /// <returns>A Consumer object that contains information about the
 consumer.</returns>
        public static async Task<Consumer> RegisterConsumerAsync(IAmazonKinesis
 client, string consumerName, string streamARN)
        {
            var request = new RegisterStreamConsumerRequest
            {
                ConsumerName = consumerName,
                StreamARN = streamARN,
            };

            var response = await client.RegisterStreamConsumerAsync(request);
            return response.Consumer;
        }
    }
```

- Untuk detail API, lihat RegisterStreamConsumerdi Referensi AWS SDK untuk .NET API.

CLI

AWS CLI

Untuk mendaftarkan konsumen aliran data

`register-stream-consumer`Contoh berikut mendaftarkan konsumen yang dipanggil `KinesisConsumerApplication` dengan aliran data yang ditentukan.

```
aws kinesis register-stream-consumer \
    --stream-arn arn:aws:kinesis:us-west-2:012345678912:stream/samplestream \
    --consumer-name KinesisConsumerApplication
```

Output:

```
{
    "Consumer": {
        "ConsumerName": "KinesisConsumerApplication",
```

```
        "ConsumerARN": "arn:aws:kinesis:us-west-2: 123456789012:stream/
samplestream/consumer/KinesisConsumerApplication:1572383852",
        "ConsumerStatus": "CREATING",
        "ConsumerCreationTimestamp": 1572383852.0
    }
}
```

Untuk informasi selengkapnya, lihat Mengembangkan Konsumen dengan Fan-Out yang Ditingkatkan Menggunakan API Aliran Data Kinesis di Panduan Pengembang Amazon Kinesis Data Streams.

- Untuk detail API, lihat RegisterStreamConsumerdi Referensi AWS CLI Perintah.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   TRY.
       oo_result = lo_kns->registerstreamconsumer(       " oo_result is returned
for testing purposes. "
           iv_streamarn = iv_stream_arn
           iv_consumername = iv_consumer_name ).
       MESSAGE 'Stream consumer registered.' TYPE 'I'.
     CATCH /aws1/cx_knsinvalidargumentex.
       MESSAGE 'The specified argument was not valid.' TYPE 'E'.
     CATCH /aws1/cx_sgmresourcelimitexcd.
       MESSAGE 'You have reached the limit on the number of resources.' TYPE
'E'.
     CATCH /aws1/cx_sgmresourceinuse.
       MESSAGE 'Resource being accessed is in use.' TYPE 'E'.
     CATCH /aws1/cx_sgmresourcenotfound.
       MESSAGE 'Resource being accessed is not found.' TYPE 'E'.
   ENDTRY.
```

- Untuk detail API, lihat RegisterStreamConsumerdi AWS SDK untuk referensi SAP ABAP API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan layanan ini dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Contoh tanpa server untuk Kinesis

Contoh kode berikut menunjukkan cara menggunakan Kinesis dengan. AWS SDKs

Contoh

- Memanggil fungsi Lambda dari pemicu Kinesis
- Melaporkan kegagalan item batch untuk fungsi Lambda dengan pemicu Kinesis

Memanggil fungsi Lambda dari pemicu Kinesis

Contoh kode berikut menunjukkan bagaimana menerapkan fungsi Lambda yang menerima peristiwa yang dipicu dengan menerima catatan dari aliran Kinesis. Fungsi mengambil payload Kinesis, mendekode dari Base64, dan mencatat konten rekaman.

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara Kinesis dengan Lambda menggunakan.NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.KinesisEvents;
```

```
using AWS.Lambda.Powertools.Logging;

// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeri

namespace KinesisIntegrationSampleCode;

public class Function
{
    // Powertools Logger requires an environment variables against your function
    // POWERTOOLS_SERVICE_NAME
    [Logging(LogEvent = true)]
    public async Task FunctionHandler(KinesisEvent evnt, ILambdaContext context)
    {
        if (evnt.Records.Count == 0)
        {
            Logger.LogInformation("Empty Kinesis Event received");
            return;
        }

        foreach (var record in evnt.Records)
        {
            try
            {
                Logger.LogInformation($"Processed Event with EventId:
 {record.EventId}");
                string data = await GetRecordDataAsync(record.Kinesis, context);
                Logger.LogInformation($"Data: {data}");
                // TODO: Do interesting work based on the new data
            }
            catch (Exception ex)
            {
                Logger.LogError($"An error occurred {ex.Message}");
                throw;
            }
        }
        Logger.LogInformation($"Successfully processed {evnt.Records.Count}
 records.");
    }

    private async Task<string> GetRecordDataAsync(KinesisEvent.Record record,
 ILambdaContext context)
```

```
    {
        byte[] bytes = record.Data.ToArray();
        string data = Encoding.UTF8.GetString(bytes);
        await Task.CompletedTask; //Placeholder for actual async work
        return data;
    }
}
```

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara Kinesis dengan Lambda menggunakan Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "log"

 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

func handler(ctx context.Context, kinesisEvent events.KinesisEvent) error {
 if len(kinesisEvent.Records) == 0 {
  log.Printf("empty Kinesis event received")
  return nil
 }

 for _, record := range kinesisEvent.Records {
  log.Printf("processed Kinesis event with EventId: %v", record.EventID)
  recordDataBytes := record.Kinesis.Data
  recordDataText := string(recordDataBytes)
```

```
  log.Printf("record data: %v", recordDataText)
  // TODO: Do interesting work based on the new data
 }
 log.Printf("successfully processed %v records", len(kinesisEvent.Records))
 return nil
}

func main() {
 lambda.Start(handler)
}
```

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara Kinesis dengan Lambda menggunakan Java.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.LambdaLogger;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.KinesisEvent;

public class Handler implements RequestHandler<KinesisEvent, Void> {
    @Override
    public Void handleRequest(final KinesisEvent event, final Context context) {
        LambdaLogger logger = context.getLogger();
        if (event.getRecords().isEmpty()) {
            logger.log("Empty Kinesis Event received");
            return null;
        }
        for (KinesisEvent.KinesisEventRecord record : event.getRecords()) {
```

```
            try {
                logger.log("Processed Event with EventId: "+record.getEventID());
                String data = new String(record.getKinesis().getData().array());
                logger.log("Data:"+ data);
                // TODO: Do interesting work based on the new data
            }
            catch (Exception ex) {
                logger.log("An error occurred:"+ex.getMessage());
                throw ex;
            }
        }
        logger.log("Successfully processed:"+event.getRecords().size()+"
 records");
        return null;
    }

}
```

JavaScript

SDK untuk JavaScript (v3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara Kinesis dengan menggunakan Lambda. JavaScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const record of event.Records) {
    try {
      console.log(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      console.log(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      console.error(`An error occurred ${err}`);
```

```
      throw err;
    }
  }
  console.log(`Successfully processed ${event.Records.length} records.`);
};

async function getRecordDataAsync(payload) {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}
```

Mengkonsumsi acara Kinesis dengan menggunakan Lambda. TypeScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
  KinesisStreamEvent,
  Context,
  KinesisStreamHandler,
  KinesisStreamRecordPayload,
} from "aws-lambda";
import { Buffer } from "buffer";
import { Logger } from "@aws-lambda-powertools/logger";

const logger = new Logger({
  logLevel: "INFO",
  serviceName: "kinesis-stream-handler-sample",
});

export const functionHandler: KinesisStreamHandler = async (
  event: KinesisStreamEvent,
  context: Context
): Promise<void> => {
  for (const record of event.Records) {
    try {
      logger.info(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      logger.info(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      logger.error(`An error occurred ${err}`);
```

```
      throw err;
    }
    logger.info(`Successfully processed ${event.Records.length} records.`);
  }
};

async function getRecordDataAsync(
  payload: KinesisStreamRecordPayload
): Promise<string> {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}
```

PHP

SDK untuk PHP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara Kinesis dengan Lambda menggunakan PHP.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\Kinesis\KinesisEvent;
use Bref\Event\Kinesis\KinesisHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler extends KinesisHandler
{
```

```
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
     * @throws \Bref\Event\InvalidLambdaEvent
     */
    public function handleKinesis(KinesisEvent $event, Context $context): void
    {
        $this->logger->info("Processing records");
        $records = $event->getRecords();
        foreach ($records as $record) {
            $data = $record->getData();
            $this->logger->info(json_encode($data));
            // TODO: Do interesting work based on the new data

            // Any exception thrown will be logged and the invocation will be
 marked as failed
        }
        $totalRecords = count($records);
        $this->logger->info("Successfully processed $totalRecords records");
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara Kinesis dengan Lambda menggunakan Python.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
import base64
def lambda_handler(event, context):

    for record in event['Records']:
        try:
            print(f"Processed Kinesis Event - EventID: {record['eventID']}")
            record_data = base64.b64decode(record['kinesis']
['data']).decode('utf-8')
            print(f"Record Data: {record_data}")
            # TODO: Do interesting work based on the new data
        except Exception as e:
            print(f"An error occurred {e}")
            raise e
    print(f"Successfully processed {len(event['Records'])} records.")
```

Ruby

SDK untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara Kinesis dengan Lambda menggunakan Ruby.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
require 'aws-sdk'

def lambda_handler(event:, context:)
  event['Records'].each do |record|
    begin
      puts "Processed Kinesis Event - EventID: #{record['eventID']}"
      record_data = get_record_data_async(record['kinesis'])
      puts "Record Data: #{record_data}"
      # TODO: Do interesting work based on the new data
```

```
    rescue => err
      $stderr.puts "An error occurred #{err}"
      raise err
    end
  end
  puts "Successfully processed #{event['Records'].length} records."
end

def get_record_data_async(payload)
  data = Base64.decode64(payload['data']).force_encoding('UTF-8')
  # Placeholder for actual async work
  # You can use Ruby's asynchronous programming tools like async/await or fibers
 here.
  return data
end
```

Rust

SDK for Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara Kinesis dengan Lambda menggunakan Rust.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::event::kinesis::KinesisEvent;
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

async fn function_handler(event: LambdaEvent<KinesisEvent>) -> Result<(), Error>
 {
    if event.payload.records.is_empty() {
        tracing::info!("No records found. Exiting.");
        return Ok(());
    }

    event.payload.records.iter().for_each(|record| {
```

```
        tracing::info!("EventId:
 {}",record.event_id.as_deref().unwrap_or_default());

        let record_data = std::str::from_utf8(&record.kinesis.data);

        match record_data {
            Ok(data) => {
                // log the record data
                tracing::info!("Data: {}", data);
            }
            Err(e) => {
                tracing::error!("Error: {}", e);
            }
        }
    });

    tracing::info!(
        "Successfully processed {} records",
        event.payload.records.len()
    );

    Ok(())
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        // disable printing the name of the module in every log line.
        .with_target(false)
        // disabling time is handy because CloudWatch will add the ingestion
 time.
        .without_time()
        .init();

    run(service_fn(function_handler)).await
}
```

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan layanan ini dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Melaporkan kegagalan item batch untuk fungsi Lambda dengan pemicu Kinesis

Contoh kode berikut menunjukkan cara mengimplementasikan respons batch sebagian untuk fungsi Lambda yang menerima peristiwa dari aliran Kinesis. Fungsi melaporkan kegagalan item batch dalam respons, memberi sinyal ke Lambda untuk mencoba lagi pesan tersebut nanti.

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch Kinesis dengan Lambda menggunakan.NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text;
using System.Text.Json.Serialization;
using Amazon.Lambda.Core;
using Amazon.Lambda.KinesisEvents;
using AWS.Lambda.Powertools.Logging;

// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeri

namespace KinesisIntegration;

public class Function
{
    // Powertools Logger requires an environment variables against your function
    // POWERTOOLS_SERVICE_NAME
    [Logging(LogEvent = true)]
    public async Task<StreamsEventResponse> FunctionHandler(KinesisEvent evnt,
 ILambdaContext context)
```

```
   {
       if (evnt.Records.Count == 0)
       {
           Logger.LogInformation("Empty Kinesis Event received");
           return new StreamsEventResponse();
       }

       foreach (var record in evnt.Records)
       {
           try
           {
               Logger.LogInformation($"Processed Event with EventId:
{record.EventId}");
               string data = await GetRecordDataAsync(record.Kinesis, context);
               Logger.LogInformation($"Data: {data}");
               // TODO: Do interesting work based on the new data
           }
           catch (Exception ex)
           {
               Logger.LogError($"An error occurred {ex.Message}");
               /* Since we are working with streams, we can return the failed
item immediately.
                  Lambda will immediately begin to retry processing from this
failed item onwards. */
               return new StreamsEventResponse
               {
                   BatchItemFailures = new
List<StreamsEventResponse.BatchItemFailure>
                   {
                       new StreamsEventResponse.BatchItemFailure
{ ItemIdentifier = record.Kinesis.SequenceNumber }
                   }
               };
           }
       }
       Logger.LogInformation($"Successfully processed {evnt.Records.Count}
records.");
       return new StreamsEventResponse();
   }

   private async Task<string> GetRecordDataAsync(KinesisEvent.Record record,
ILambdaContext context)
   {
       byte[] bytes = record.Data.ToArray();
```

```
        string data = Encoding.UTF8.GetString(bytes);
        await Task.CompletedTask; //Placeholder for actual async work
        return data;
    }
}

public class StreamsEventResponse
{
    [JsonPropertyName("batchItemFailures")]
    public IList<BatchItemFailure> BatchItemFailures { get; set; }
    public class BatchItemFailure
    {
        [JsonPropertyName("itemIdentifier")]
        public string ItemIdentifier { get; set; }
    }
}
```

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch Kinesis dengan Lambda menggunakan Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "fmt"
 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

func handler(ctx context.Context, kinesisEvent events.KinesisEvent)
 (map[string]interface{}, error) {
```

```
 batchItemFailures := []map[string]interface{}{}

 for _, record := range kinesisEvent.Records {
  curRecordSequenceNumber := ""

  // Process your record
  if /* Your record processing condition here */ {
   curRecordSequenceNumber = record.Kinesis.SequenceNumber
  }

  // Add a condition to check if the record processing failed
  if curRecordSequenceNumber != "" {
   batchItemFailures = append(batchItemFailures, map[string]interface{}
{"itemIdentifier": curRecordSequenceNumber})
  }
 }

 kinesisBatchResponse := map[string]interface{}{
  "batchItemFailures": batchItemFailures,
 }
 return kinesisBatchResponse, nil
}

func main() {
 lambda.Start(handler)
}
```

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch Kinesis dengan Lambda menggunakan Java.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
```

```
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.KinesisEvent;
import com.amazonaws.services.lambda.runtime.events.StreamsEventResponse;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;

public class ProcessKinesisRecords implements RequestHandler<KinesisEvent,
 StreamsEventResponse> {

    @Override
    public StreamsEventResponse handleRequest(KinesisEvent input, Context
 context) {

        List<StreamsEventResponse.BatchItemFailure> batchItemFailures = new
 ArrayList<>();
        String curRecordSequenceNumber = "";

        for (KinesisEvent.KinesisEventRecord kinesisEventRecord :
 input.getRecords()) {
            try {
                //Process your record
                KinesisEvent.Record kinesisRecord =
 kinesisEventRecord.getKinesis();
                curRecordSequenceNumber = kinesisRecord.getSequenceNumber();

            } catch (Exception e) {
                /* Since we are working with streams, we can return the failed
 item immediately.
                   Lambda will immediately begin to retry processing from this
 failed item onwards. */
                batchItemFailures.add(new
 StreamsEventResponse.BatchItemFailure(curRecordSequenceNumber));
                return new StreamsEventResponse(batchItemFailures);
            }
        }

       return new StreamsEventResponse(batchItemFailures);
    }
}
```

JavaScript

SDK untuk JavaScript (v3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch Kinesis dengan Lambda menggunakan Javascript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const record of event.Records) {
    try {
      console.log(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      console.log(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      console.error(`An error occurred ${err}`);
      /* Since we are working with streams, we can return the failed item
 immediately.
           Lambda will immediately begin to retry processing from this failed
 item onwards. */
      return {
        batchItemFailures: [{ itemIdentifier: record.kinesis.sequenceNumber }],
      };
    }
  }
  console.log(`Successfully processed ${event.Records.length} records.`);
  return { batchItemFailures: [] };
};

async function getRecordDataAsync(payload) {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}
```

Melaporkan kegagalan item batch Kinesis dengan penggunaan Lambda. TypeScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
  KinesisStreamEvent,
  Context,
  KinesisStreamHandler,
  KinesisStreamRecordPayload,
  KinesisStreamBatchResponse,
} from "aws-lambda";
import { Buffer } from "buffer";
import { Logger } from "@aws-lambda-powertools/logger";

const logger = new Logger({
  logLevel: "INFO",
  serviceName: "kinesis-stream-handler-sample",
});

export const functionHandler: KinesisStreamHandler = async (
  event: KinesisStreamEvent,
  context: Context
): Promise<KinesisStreamBatchResponse> => {
  for (const record of event.Records) {
    try {
      logger.info(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      logger.info(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      logger.error(`An error occurred ${err}`);
      /* Since we are working with streams, we can return the failed item
 immediately.
           Lambda will immediately begin to retry processing from this failed
 item onwards. */
      return {
        batchItemFailures: [{ itemIdentifier: record.kinesis.sequenceNumber }],
      };
    }
  }
  logger.info(`Successfully processed ${event.Records.length} records.`);
  return { batchItemFailures: [] };
};
```

```
async function getRecordDataAsync(
  payload: KinesisStreamRecordPayload
): Promise<string> {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}
```

PHP

SDK untuk PHP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch Kinesis dengan Lambda menggunakan PHP.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\Kinesis\KinesisEvent;
use Bref\Event\Handler as StdHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler implements StdHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }
```

```
    /**
     * @throws JsonException
     * @throws \Bref\Event\InvalidLambdaEvent
     */
    public function handle(mixed $event, Context $context): array
    {
        $kinesisEvent = new KinesisEvent($event);
        $this->logger->info("Processing records");
        $records = $kinesisEvent->getRecords();

        $failedRecords = [];
        foreach ($records as $record) {
            try {
                $data = $record->getData();
                $this->logger->info(json_encode($data));
                // TODO: Do interesting work based on the new data
            } catch (Exception $e) {
                $this->logger->error($e->getMessage());
                // failed processing the record
                $failedRecords[] = $record->getSequenceNumber();
            }
        }
        $totalRecords = count($records);
        $this->logger->info("Successfully processed $totalRecords records");

        // change format for the response
        $failures = array_map(
            fn(string $sequenceNumber) => ['itemIdentifier' => $sequenceNumber],
            $failedRecords
        );

        return [
            'batchItemFailures' => $failures
        ];
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch Kinesis dengan Lambda menggunakan Python.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def handler(event, context):
    records = event.get("Records")
    curRecordSequenceNumber = ""

    for record in records:
        try:
            # Process your record
            curRecordSequenceNumber = record["kinesis"]["sequenceNumber"]
        except Exception as e:
            # Return failed record's sequence number
            return {"batchItemFailures":[{"itemIdentifier":
 curRecordSequenceNumber}]}

    return {"batchItemFailures":[]}
```

Ruby

SDK untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch Kinesis dengan Lambda menggunakan Ruby.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
require 'aws-sdk'

def lambda_handler(event:, context:)
  batch_item_failures = []

  event['Records'].each do |record|
    begin
      puts "Processed Kinesis Event - EventID: #{record['eventID']}"
      record_data = get_record_data_async(record['kinesis'])
      puts "Record Data: #{record_data}"
      # TODO: Do interesting work based on the new data
    rescue StandardError => err
      puts "An error occurred #{err}"
      # Since we are working with streams, we can return the failed item
 immediately.
      # Lambda will immediately begin to retry processing from this failed item
 onwards.
      return { batchItemFailures: [{ itemIdentifier: record['kinesis']
['sequenceNumber'] }] }
    end
  end

  puts "Successfully processed #{event['Records'].length} records."
  { batchItemFailures: batch_item_failures }
end

def get_record_data_async(payload)
  data = Base64.decode64(payload['data']).force_encoding('utf-8')
  # Placeholder for actual async work
  sleep(1)
  data
end
```

Rust

SDK for Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch Kinesis dengan Lambda menggunakan Rust.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::{
    event::kinesis::KinesisEvent,
    kinesis::KinesisEventRecord,
    streams::{KinesisBatchItemFailure, KinesisEventResponse},
};
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

async fn function_handler(event: LambdaEvent<KinesisEvent>) ->
 Result<KinesisEventResponse, Error> {
    let mut response = KinesisEventResponse {
        batch_item_failures: vec![],
    };

    if event.payload.records.is_empty() {
        tracing::info!("No records found. Exiting.");
        return Ok(response);
    }

    for record in &event.payload.records {
        tracing::info!(
            "EventId: {}",
            record.event_id.as_deref().unwrap_or_default()
        );

        let record_processing_result = process_record(record);

        if record_processing_result.is_err() {
            response.batch_item_failures.push(KinesisBatchItemFailure {
                item_identifier: record.kinesis.sequence_number.clone(),
```

```
            });
            /* Since we are working with streams, we can return the failed item
 immediately.
            Lambda will immediately begin to retry processing from this failed
 item onwards. */
            return Ok(response);
        }
    }

    tracing::info!(
        "Successfully processed {} records",
        event.payload.records.len()
    );

    Ok(response)
}

fn process_record(record: &KinesisEventRecord) -> Result<(), Error> {
    let record_data = std::str::from_utf8(record.kinesis.data.as_slice());

    if let Some(err) = record_data.err() {
        tracing::error!("Error: {}", err);
        return Err(Error::from(err));
    }

    let record_data = record_data.unwrap_or_default();

    // do something interesting with the data
    tracing::info!("Data: {}", record_data);

    Ok(())
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        // disable printing the name of the module in every log line.
        .with_target(false)
        // disabling time is handy because CloudWatch will add the ingestion
 time.
        .without_time()
        .init();
```

```
    run(service_fn(function_handler)).await
}
```

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan layanan ini dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Riwayat dokumen

Tabel berikut menjelaskan perubahan penting pada dokumentasi Amazon Kinesis Data Streams.

Ubah	Deskripsi	Tanggal Diubah
Menambahkan dukungan untuk pengujian ketahanan dengan. AWS Fault Injection Service	Ditambahkan Lakukan pengujian ketahanan dengan AWS Fault Injection Service.	Oktober 15, 2025
Menambahkan dukungan untuk ukuran rekaman besar hingga 10 MiB.	Ditambahkan Menangani catatan besar.	Oktober 27, 2025
End-of-support tanggal dan kebijakan siklus hidup versi untuk KPL dan KCL.	Menambahkan informasi tentang end-of-support tanggal dan kebijakan siklus hidup versi untuk Amazon Kinesis Producer Library (KPL) dan Amazon Kinesis Client Library (KCL). Untuk informasi selengkapnya, lihat Kebijakan siklus hidup versi KPL dan Kebijakan siklus hidup versi KCL.	13 Maret 2025
Menambahkan dukungan untuk berbagi aliran data di seluruh akun.	Ditambahkan Bagikan aliran data Anda dengan akun lain.	22 November 2023
Menambahkan dukungan untuk mode kapasitas aliran data sesuai permintaan dan disediakan.	Ditambahkan Pilih mode yang tepat untuk streaming.	29 November 2021

Ubah	Deskripsi	Tanggal Diubah
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.	DiperbaruiPantau Kinesis Data Streams.	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
Perbarui konten KPL untuk rilis 0.10.0.	Ditambahkan Kembangkan produsen menggunakan Amazon Kinesis Producer Library (KPL).	Juli 15, 2015
Perbarui topik metrik KCL untuk metrik yang dapat dikonfigurasi.	Ditambahkan Pantau Perpustakaan Klien Kinesis dengan Amazon CloudWatch.	9 Juli 2015
Konten yang diatur ulang.	Topik konten yang diatur ulang secara signifika n untuk tampilan pohon yang lebih ringkas dan pengelompokan yang lebih logis.	01 Juli 2015
Topik panduan pengembang KPL baru.	Ditambahkan Kembangkan produsen menggunakan Amazon Kinesis Producer Library (KPL).	02 Juni 2015
Topik metrik KCL baru.	Ditambahkan Pantau Perpustakaan Klien Kinesis dengan Amazon CloudWatch.	19 Mei 2015

Ubah	Deskripsi	Tanggal Diubah
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.	Maret 26, 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 “Tambahkan beberapa catatan dengan PutRecords”.	15 Desember 2014
Dukungan untuk penandaan	Ditambahkan Tandai sumber daya Amazon Kinesis Data Streams Anda.	11 September 2014
CloudWatch Metrik baru	Menambahkan metrik `GetRecords.IteratorAgeMilliseconds` keDimensi dan metrik Amazon Kinesis Data Streams.	3 September 2014
Bab pemantauan baru	Ditambahkan Pantau Kinesis Data Streams danPantau 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 awal	Rilis awal Panduan Pengembang Amazon Kinesis.	14 November 2013

Terjemahan disediakan oleh mesin penerjemah. Jika konten terjemahan yang diberikan bertentangan dengan versi bahasa Inggris aslinya, utamakan versi bahasa Inggris.