aws

Entwicklerhandbuch

Amazon-Kinesis-Data-Streams

Amazon-Kinesis-Data-Streams : Entwicklerhandbuch

Table of Contents

Was ist Amazon Kinesis Data Streams?

Sie können Amazon Kinesis Data Streams zum Erfassen und Verarbeiten großer Streams von Datensätzen in Echtzeit nutzen. Sie können Datenverarbeitungsanwendungen erstellen, die als Anwendungen von Kinesis Data Streams bezeichnet werden. Eine typische Anwendung von Kinesis Data Streams liest Daten aus einem Datenstrom als Datensätze. Diese Anwendungen können die Kinesis Client Library verwenden und auf EC2 Amazon-Instances ausgeführt werden. Sie können die verarbeiteten Datensätze an Dashboards senden, sie zum Erstellen von Warnmeldungen, zum dynamischen Ändern von Preis- und Werbestrategien nutzen oder Daten an verschiedene andere AWS -Services senden. Informationen zu den Features und Preisen von Kinesis Data Streams finden Sie unter Amazon Kinesis Data Streams.

Kinesis Data Streams ist zusammen mit Firehose, Kinesis Video Streams und Managed Service for Apache Flink Teil der Kinesis-Streaming-Datenplattform.

Weitere Informationen zu AWS Big-Data-Lösungen finden Sie unter Big Data auf. AWS Weitere Informationen zu AWS -Streaming-Datenlösungen finden Sie unter Was sind Streaming-Daten?.

Themen

Was kann ich mit Kinesis Data Streams machen?

Sie können Kinesis Data Streams für die schnelle und kontinuierliche Aufnahme und Aggregation von Daten verwenden. Der verwendete Datentyp kann Protokolldaten zur IT-Infrastruktur, Anwendungsprotokolle, Data-Feeds von sozialen Medien, Marktdaten-Feeds sowie Web-Clickstream-Daten einschließen. Da die Reaktionszeit für die Aufnahme und Verarbeitung der Daten in Echtzeit erfolgt, ist die Verarbeitung in der Regel ein leichtgewichtiger Prozess.

Es folgen typische Szenarien für den Einsatz von Kinesis Data Streams:

Beschleunigte Protokoll- und Datenfeedaufnahme und -verarbeitung

> Produzenten können Daten direkt in einen Stream einleiten. Sie können beispielsweise System- und Anwendungsprotokolle übertragen. Diese stehen dann innerhalb von Sekunden für die

Verarbeitung zur Verfügung. Dadurch wird verhindert, dass die Protokolldaten verloren gehen, wenn Front-End-Server oder Anwendungsserver abstürzen. Kinesis Data Streams unterstützt eine beschleunigte Datenfeedaufnahme, da die Daten auf den Servern nicht zu Stapeln zusammengefasst werden, bevor sie zur Aufnahme weitergeleitet werden.

Echtzeitmetriken und -berichte

Sie können Daten, die in Kinesis Data Streams erfasst wurden, für eine einfache Datenanalyse und Berichterstellung in Echtzeit verwenden. So kann Ihre Datenverarbeitungsanwendung beispielsweise während des Daten-Streamings an Metriken und Berichten für System- und Anwendungsprotokolle arbeiten und muss nicht auf einzelne Datenpakete warten.

Echtzeitdatenanalysen

Hier werden die Vorteile von Echtzeitdaten mit der parallelen Verarbeitung kombiniert. So können Sie Website-Clickstreams in Echtzeit verarbeiten und anschließend die Benutzerfreundlichkeit der Website mit verschiedenen Anwendungen von Kinesis Data Streams analysieren, die parallel ausgeführt werden.

Komplexe Stream-Verarbeitung

Sie können gerichtete azyklische Graphen (DAGs) von Kinesis Data Streams Streams-Anwendungen und Datenströmen erstellen. Dazu gehört in der Regel das Übertragen von Daten aus mehreren Anwendungen von Kinesis Data Streams in einen anderen Stream für die nachgeordnete Verarbeitung durch eine andere Anwendung von Kinesis Data Streams.

Vorteile der Verwendung von Kinesis Data Streams

Sie können mit Kinesis Data Streams eine Vielzahl von Problemen lösen, die beim Daten-Streaming auftreten. Typisch ist jedoch der Einsatz der Echtzeit-Aggregation von Daten gefolgt vom Laden der aggregierten Daten in ein Data Warehouse oder einen MapReduce-Cluster.

Die Daten werden in Kinesis-Daten-Streams geschrieben. Dies sorgt für Beständigkeit und Elastizität. Die Verzögerung zwischen dem Zeitpunkt, zu dem ein Datensatz in den Stream aufgenommen wird, und dem Zeitpunkt, zu dem er abgerufen werden kann (put-to-get Verzögerung), beträgt in der Regel weniger als 1 Sekunde. Eine Anwendung von Kinesis Data Streams kann somit die Daten aus dem Stream nahezu sofort nach dem Hinzufügen der Daten nutzen. Die von Kinesis Data Streams verwalteten Serviceaufgaben befreien Sie von dem Aufwand, der beim Erstellen und Ausführen einer Datenzufuhr-Pipeline betrieben werden muss. Sie können Streaming-Anwendungen des Typs Map-

Reduce erstellen. Die Elastizität von Kinesis Data Streams erlaubt das Skalieren des Streams, damit Sie niemals Datensätze verlieren, bevor diese ablaufen.

Mehrere Anwendungen von Kinesis Data Streams können Daten aus einem Stream aufnehmen, sodass mehrere Aktionen, beispielsweise Archivierung und Verarbeitung, gleichzeitig und unabhängig voneinander durchgeführt werden können. So ist es unter anderem möglich, dass zwei Anwendungen Daten aus demselben Stream auslesen. Die erste Anwendung berechnet die Ausführung von Aggregaten und aktualisiert eine Amazon-DynamoDB-Tabelle und die zweite Anwendung komprimiert und archiviert Daten in einem Datenspeicher wie Amazon Simple Storage Service (Amazon S3). Die DynamoDB-Tabelle mit laufenden Aggregaten wird dann von einem Dashboard für Berichte gelesen. up-to-the-minute

Die Kinesis Client Library ermöglicht eine fehlertolerante Nutzung von Daten aus Datenströmen und stellt zudem skalierbaren Support für Anwendungen von Kinesis Data Streams bereit.

Zugehörige Services

Informationen zur Verwendung von EMR Amazon-Clustern zum direkten Lesen und Verarbeiten von Kinesis-Datenströmen finden Sie unter Kinesis Connector.

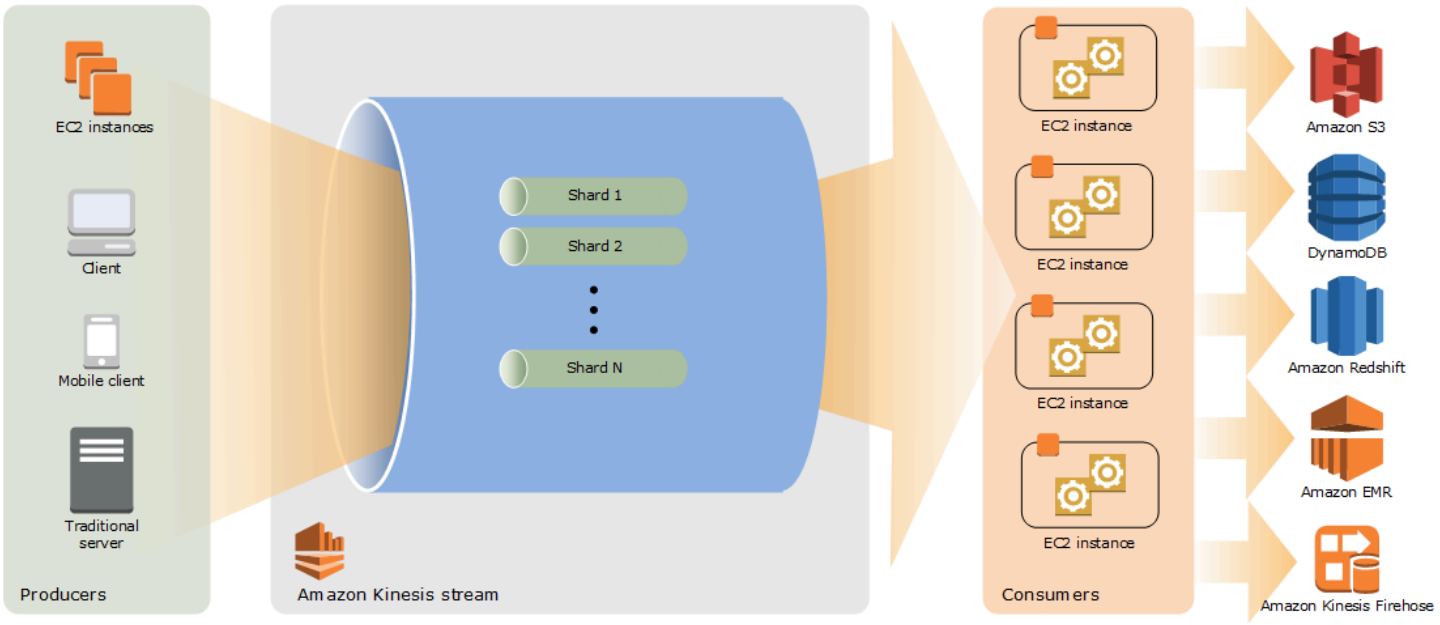
Terminologie und Konzepte von Amazon Kinesis Data Streams

Bevor Sie mit Amazon Kinesis Data Streams beginnen, sollten Sie sich mit dessen Architektur und Terminologie vertraut machen.

Themen

Informieren Sie sich über die High-Level-Architektur von Kinesis Data Streams

Das folgende Diagramm zeigt die High-Level-Architektur von Kinesis Data Streams. Die Produzenten übermitteln kontinuierlich Daten an Kinesis Data Streams und die Verbraucher verarbeiten die Daten in Echtzeit. Verbraucher (z. B. eine benutzerdefinierte Anwendung, die auf Amazon läuft, EC2 oder ein Amazon Firehose Firehose-Lieferstream) können ihre Ergebnisse mit einem AWS Service wie Amazon DynamoDB, Amazon Redshift oder Amazon S3 speichern.



Machen Sie sich mit der Terminologie von Kinesis Data Streams vertraut

Kinesis Data Stream

Ein Kinesis-Datenstrom ist eine Gruppe von Shards. Jeder Shard hat eine Sequenz von Datensätzen. Jedem Datensatz wird von Kinesis Data Streams eine Sequenznummer zugewiesen.

Datensatz

Die von Kinesis Data Streams gespeicherte Dateneinheit wird als Datensatz bezeichnet. Datensätze besehen aus einer Sequenznummer, einem Partitionsschlüssel und einem Daten-Blob. Ein Daten-Blob ist eine unveränderliche Byte-Reihenfolge. Kinesis Data Streams untersucht, interpretiert oder ändert Daten im Blob in keiner Weise. Ein Daten-Blob kann bis zu 1 MB groß sein.

Kapazitätsmodus

Ein Datenstrom-Kapazitätsmodus bestimmt, wie Kapazität verwaltet wird und wie Ihnen die Nutzung Ihres Datenstroms in Rechnung gestellt wird. Derzeit können Sie in Kinesis Data Streams zwischen einem On-Demand-Modus und einem Bereitstellungsmodus für Ihre Datenstreams wählen. Weitere Informationen finden Sie unter Wählen Sie den Datenstream-Kapazitätsmodus.

Im On-Demand-Modus verwaltet Kinesis Data Streams die Shards automatisch, um den erforderlichen Durchsatz bereitzustellen. Sie zahlen nur für den tatsächlich genutzten Durchsatz und Kinesis Data Streams passt sich automatisch dem Durchsatzbedarf Ihrer Workloads an, wenn dieser steigt oder sinkt. Weitere Informationen finden Sie unter Funktionen und Anwendungsfälle des On-Demand-Modus.

Mit einem bereitgestellten Modus müssen Sie die Anzahl der Shards für den Datenstrom angeben. Die Gesamtkapazität eines Datenstroms ist die Summe der Kapazitäten der einzelnen Shards. Sie können die Anzahl der Shards in einem Datenstrom nach Bedarf erhöhen oder verringern, und die Anzahl der Shards wird Ihnen nach einem Stundensatz in Rechnung gestellt. Weitere Informationen finden Sie unter Funktionen und Anwendungsfälle im Bereitstellungsmodus.

Aufbewahrungszeitraum

Der Aufbewahrungszeitraum gibt den Zeitraum am, in dem auf Datensätze zugegriffen werden kann, nachdem sie dem Stream hinzugefügt wurden. Die Standard-Aufbewahrungsdauer eines

Streams beträgt nach Erstellung 24 Stunden. Sie können den Aufbewahrungszeitraum mithilfe des IncreaseStreamRetentionPeriodVorgangs auf bis zu 8760 Stunden (365 Tage) erhöhen und den Aufbewahrungszeitraum mithilfe des Vorgangs auf mindestens 24 Stunden reduzieren. DecreaseStreamRetentionPeriod Für Streams mit einem Aufbewahrungszeitraum von mehr als 24 Stunden fallen zusätzliche Gebühren an. Weitere Informationen finden Sie unter Preise für Amazon Kinesis Daten-Streams.

Produzent

Produzenten speichern Datensätze in Amazon Kinesis Data Streams. Ein Webserver, der Protokolldaten an einen Stream sendet, ist ein Beispiel für einen Produzenten.

Konsument

Verbraucher erhalten Datensätze von Amazon Kinesis Data Streams und verarbeiten sie. Diese Konsumenten werden als Anwendung von Amazon Kinesis Data Streams bezeichnet.

Anwendung von Amazon Kinesis Data Streams

Eine Amazon Kinesis Data Streams Streams-Anwendung nutzt einen Stream, der üblicherweise auf einer Flotte von EC2 Instances ausgeführt wird.

Sie können zwei Arten von Konsumenten entwickeln: geteilte Rundesendekonsumenten und erweiterte Rundesendekonsumenten. Informationen zu den Unterschieden zwischen diesen Typen und zu ihrer Erstellung erhalten Sie unter Daten aus Amazon Kinesis Data Streams lesen.

Die Ausgabe einer Anwendung von Kinesis Data Streams kann als Eingabe für einen anderen Stream genutzt werden. So können Sie komplexe Topologien erstellen, die Daten in Echtzeit verarbeiten. Eine Anwendung kann auch Daten an eine Vielzahl anderer AWS Dienste senden. Es sind mehrere Anwendungen für einen Stream möglich. Dabei kann jede Anwendung Daten aus einem Stream gleichzeitig und unabhängig konsumieren.

Shard

Ein Shard ist eine eindeutig identifizierbare Sequenz von Datensätzen in einem Stream. Ein Stream besteht aus einem oder mehreren Shards. Jeder Shard stellt eine feste Einheit der Kapazität bereit. Jeder Shard kann bis zu 5 Transaktionen pro Sekunde für Lesevorgänge unterstützen, bis zu einer maximalen Gesamtdatenleserate von 2 MB pro Sekunde und bis zu 1 000 Datensätze pro Sekunde für Schreibvorgänge, bis zu einer maximalen Gesamtdatenschreibrate von 1 MB pro Sekunde

(einschließlich Partitionsschlüssel). Die Datenkapazität Ihres Streams bezieht sich auf die Anzahl der Shards, die Sie für den Stream festlegen. Die Gesamtkapazität des Streams ist die Summe der Kapazitäten der einzelnen Shards.

Wenn sich Ihre Datenrate erhöht, können Sie die Anzahl der zu Ihrem Stream zugewiesenen Shards erhöhen oder verringern. Weitere Informationen finden Sie unter Einen Stream erneut teilen.

Partitionsschlüssel

Ein Partitionsschlüssel wird verwendet, um Daten nach Shard in einem Stream zu gruppieren. Kinesis Data Streams teilt die Datensätze eines Streams auf mehrere Shards auf. Dabei wird der Partitionsschlüssel verwendet, der jedem Datensatz zugeordnet ist, um zu ermitteln, zu welchem Shard ein gegebener Datensatz gehört. Partitionsschlüssel sind Unicode-Zeichenfolgen mit einer maximalen Länge von 256 Zeichen pro Schlüssel. Eine MD5 Hash-Funktion wird verwendet, um Partitionsschlüssel 128-Bit-Ganzzahlwerten zuzuordnen und zugehörige Datensätze unter Verwendung der Hash-Schlüsselbereiche der Shards Shards den Shards zuzuordnen. Wenn eine Anwendung Daten in einen Stream schreibt, muss sie einen Partitionsschlüssel angeben.

Sequenznummer

Jeder Datensatz verfügt über eine Sequenznummer, die pro Partitionsschlüssel im betreffenden Shard eindeutig ist. Die Sequenznummer wird von Kinesis Data Streams zugewiesen, nachdem Sie mit `client.putRecords` oder `client.putRecord` in den Stream geschrieben haben. Sequenznummern für denselben Partitionsschlüssel steigen in der Regel im Laufe der Zeit. Je länger die Zeitdauer zwischen Schreibanforderungen ist, desto größer wird die Sequenznummer.

> **Note**
>
> Sequenznummern können nicht als Index für Datensätze innerhalb desselben Streams verwendet werden. Nutzen Sie für die logische Unterteilung von Datensätzen Partitionsschlüssel oder erstellen Sie für jeden Datensatz einen separaten Stream.

Kinesis Client Library

Die Kinesis Client Library wird in Ihre Anwendung kompiliert, um eine fehlertolerante Nutzung von Daten aus dem Stream zu gewährleisten. Die Kinesis Client Library stellt sicher, dass für jeden Shard ein Datensatzprozessor verfügbar ist, der diesen Shard ausführt und verarbeitet. Die Bibliothek

vereinfacht zudem das Auslesen von Daten aus dem Stream. Die Kinesis Client Library verwendet eine Amazon-DynamoDB-Tabelle zum Speichern von Kontrolldaten. Sie generiert eine Anwendung pro Tabelle, die die Daten verarbeitet.

Es gibt zwei Hauptversionen der Kinesis Client Library. Welche Sie verwenden, hängt vom Typ des zu erstellenden Konsumenten ab. Weitere Informationen finden Sie unter Daten aus Amazon Kinesis Data Streams lesen.

Anwendungsname

Der Name einer Anwendung von Amazon Kinesis Data Streams identifiziert die Anwendung. Jede Ihrer Anwendungen muss einen eindeutigen Namen haben, der auf das AWS Konto und die Region zugeschnitten ist, die von der Anwendung verwendet werden. Dieser Name wird als Name für die Steuertabelle in Amazon DynamoDB und als Namespace für Amazon-Metriken verwendet. CloudWatch

Serverseitige Verschlüsselung

Amazon Kinesis Data Streams kann automatisch sensible Daten verschlüsseln, wenn ein Produzent diese an einen Stream übergibt. Kinesis Data Streams verwendet AWS KMS-Masterschlüssel für die Verschlüsselung. Weitere Informationen finden Sie unter Datenschutz in Amazon Kinesis Data Streams.

> **Note**
>
> Zum Auslesen aus einem verschlüsselten Stream oder zum Schreiben in einen solchen benötigen Konsumentenanwendungen eine Berechtigung für den Zugriff auf den Masterschlüssel. Informationen zum Gewähren von Berechtigungen für Produzenten- und Konsumentenanwendungen finden Sie unter the section called “Berechtigungen zur Verwendung von benutzergenerierten Schlüsseln KMS”.

> **Note**
>
> Die Verwendung serverseitiger Verschlüsselung verursacht AWS Key Management Service () Kosten.AWS KMS Weitere Informationen finden Sie unter AWS Key Management Service – Preise.

Kontingente und -Einschränkungen

In der folgenden Tabelle werden Stream- und Shard-Kontingente und -Limits für Amazon Kinesis Data Streams beschrieben.

Kontingent	On-Demand-Modus	Modus bereitgestellter Kapazität
Anzahl der Datenströme	Es gibt kein oberes Kontingent für die Anzahl der Streams in Ihrem AWS Konto. Standardmäßig können Sie im On-Demand-Kapazitätsmodus bis zu 50 Datenströme erstellen. Wenn Sie eine Erhöhung dieses Kontingents benötigen, erstellen Sie bitte ein Support-Ticket.	Es gibt kein maximales Kontingent für die Anzahl der Streams im Bereitstellungsmodus innerhalb eines Kontos.
Anzahl der Shards	Es gibt keine Obergrenze. Die Anzahl der Shards hängt von der Menge der erfassten Daten und dem erforderlichen Durchsatz ab. Kinesis Data Streams skaliert die Anzahl der Shards automatisch als Reaktion auf Änderungen des Datenvolumens und des Datenverkehrs.	Es gibt keine Obergrenze. Das Standard-Shard-Kontingent beträgt 500 Shards pro AWS Konto für die folgenden AWS Regionen: USA Ost (Nord-Virginia), USA West (Oregon) und Europa (Irland). Für alle anderen Regionen beträgt das Standardkontingent 200 Shards pro AWS -Konto. Informationen zur Beantragung einer Erhöhung des shards-per-data Stream-Kontingents finden Sie unter Eine Erhöhung der Quote beantragen.

Kontingent	On-Demand-Modus	Modus bereitgestellter Kapazität
Datenstrom-Durchsatz	Standardmäßig haben neue Datenströme, die mit dem On-Demand-Kapazitätsmodus erstellt wurden, einen Schreibdurchsatz von 4 Mbit/s und einen Lesedurchsatz von 8 MB/s. Mit steigendem Datenverkehr skalieren Datenströme mit dem On-Demand-Kapazitätsmodus auf bis zu 200 Mbit/s Schreib- und 400 Mbit/s Lesedurchsatz. Wenn Sie eine Erhöhung auf 2 Gbit/s Schreib- und 4 Gbit/s Lesekapazität benötigen, reichen Sie ein Support-Ticket ein	Es gibt keine Obergrenze. Der maximale Durchsatz hängt von der Anzahl der für den Stream bereitgestellten Shards ab. Jeder Shard kann einen Schreibdurchsatz von bis zu 1 Mbit/s oder 1 000 Datensätze/Sekunde oder einen Lesedurchsatz von bis zu 2 Mbit/s oder 2 000 Datensätze/Sekunde unterstützen. Wenn Sie mehr Aufnahmekapazität benötigen, können Sie die Anzahl der Shards im Stream einfach mit AWS Management Console oder dem erhöhen. UpdateShardCountAPI
Daten-Nutzlastgröße	Die maximale Größe der Daten-Nutzlast eines Datensatzes vor der `base64-encoding` beträgt bis zu 1 MB.	
`GetRecords` -Transaktionsgröße	GetRecordskann bis zu 10 MB Daten pro Anruf von einem einzelnen Shard und bis zu 10.000 Datensätze pro Anruf abrufen. Jeder Aufruf von `GetRecords` gilt als eine Lese-Transaktion. Ein Shard kann bis zu fünf Lese-Transaktionen pro Sekunde unterstützen. Jede Lese-Transaktion kann bis zu 10.000 Datensätze mit einem Kontingent von 10 MB pro Transaktion liefern.	
Datenleserate pro Shard	Jeder Shard kann eine maximale Gesamtdatenleserate von bis zu 2 MB pro Sekunde über unterstützen. GetRecords Wenn ein Aufruf von `GetRecords` 10 MB zurückgibt, lösen nachfolgende Aufrufe innerhalb der nächsten 5 Sekunden eine Ausnahme aus.	

Kontingent	On-Demand-Modus	Modus bereitgestellter Kapazität
Anzahl der registrierten Verbraucher pro Datenstrom	Sie können bis zu 20 registrierte Verbraucher (Enhanced Fan-Out Limit) für jeden Datenstrom erstellen.	
Zwischen Bereitstellungs- und On-Demand-Modus wechseln	Für jeden Datenstream in Ihrem AWS Konto können Sie innerhalb von 24 Stunden zweimal zwischen den Modi „On-Demand-Kapazität“ und „Bereitgestellte Kapazität“ wechseln.	

Limits für API

Wie die meisten AWS APIs API Operationen sind Kinesis Data Streams Streams-Operationen ratenbegrenzt. Die folgenden Beschränkungen gelten pro AWS Konto und Region. Weitere Informationen zu Kinesis Data Streams APIs finden Sie in der Amazon Kinesis Kinesis-Referenz API.

KDSGrenzwerte für die Steuerungsebene API

Im folgenden Abschnitt werden die Grenzwerte für die KDS Steuerungsebene beschriebenAPIs. KDSMit der APIs Steuerungsebene können Sie Ihre Datenströme erstellen und verwalten. Diese Limits gelten pro AWS -Konto pro Region.

APIGrenzwerte der Kontrollebene

API	APIAnruflimit	Pro Konto/Stream	Beschreibung
AddTagsToStream	5 Transaktionen pro Sekunde (TPS)	Pro Konto	50 Tags pro Datenstrom
CreateStream	5 TPS	Pro Konto	Es gibt kein Kontingent für die Anzahl der Streams in einem Konto. Sie erhalten beim Erstellen einer `CreateStream` -Anforderung eine `LimitExceededException` ,

API	APIAnruflimit	Pro Konto/Stream	Beschreibung
			wenn Sie versuchen , eine der folgenden Aktionen auszuführen: • Es befinden sich zu einem beliebige n Zeitpunkt mehr als fünf Streams im `CREATING`-Zustand. • Sie erstellen mehr Shards als für Ihr Konto zulässig sind.
DecreaseStreamRete ntionPeriod	5 TPS	Pro Stream	Der Mindestwert für den Aufbewahr ungszeitraum eines Datenstroms beträgt 24 Stunden.
DeleteResourcePolicy	5 TPS	Pro Konto	Wenn Sie eine Erhöhung dieses Limits benötigen, erstellen Sie bitte ein Support-Ticket.
DeleteStream	5 TPS	Pro Konto	
DeregisterStreamCo nsumer	5 TPS	Pro Stream	
DescribeLimits	1 TPS	Pro Konto	
DescribeStream	10 TPS	Pro Konto	

API	APIAnruflimit	Pro Konto/Stream	Beschreibung
DescribeStreamConsumer	20TPS	Pro Stream	
DescribeStreamSummary	20TPS	Pro Konto	
DisableEnhancedMonitoring	5 TPS	Pro Stream	
EnableEnhancedMonitoring	5 TPS	Pro Stream	
GetResourcePolicy	5 TPS	Pro Konto	Wenn Sie eine Erhöhung dieses Limits benötigen, erstellen Sie bitte ein Support-Ticket.
IncreaseStreamRetentionPeriod	5 TPS	Pro Stream	Der maximale Wert für den Aufbewahrungszeitraum eines Streams beträgt 8760 Stunden (365 Tage).
ListShards	1000 TPS	Pro Stream	
ListStreamConsumers	5 TPS	Pro Stream	
ListStreams	5 TPS	Pro Konto	
ListTagsForStream	5 TPS	Pro Stream	
MergeShards	5 TPS	Pro Stream	Gilt nur für bereitgestellte.

API	APIAnruflimit	Pro Konto/Stream	Beschreibung
PutResourcePolicy	5 TPS	Pro Konto	Wenn Sie eine Erhöhung dieses Limits benötigen, erstellen Sie bitte ein Support-Ticket.
RegisterStreamConsumer	5 TPS	Pro Stream	Sie können pro Datenstrom bis zu 20 Verbraucher registrieren. Ein bestimmter Verbraucher kann jeweils nur für einen Datenstrom registriert werden. Es können nur 5 Verbraucher gleichzeitig erstellt werden. Mit anderen Worten, es können nicht mehr als 5 Verbraucher gleichzeitig den Status `CREATING` haben. Registrierung eines sechsten Verbrauchers, bei 5 Verbrauchern in einem `CREATING`
RemoveTagsFromStream	5 TPS	Pro Stream	
SplitShard	5 TPS	Pro Stream	Gilt nur für bereitgestellte

API	APIAnruflimit	Pro Konto/Stream	Beschreibung
StartStreamEncryption		Pro Stream	Sie können innerhalb von 24 Stunden 25 Mal erfolgreich einen neuen AWS KMS Schlüssel für die serverseitige Verschlüsselung anwenden.
StopStreamEncryption		Pro Stream	Sie können die serverseitige Verschlüsselung in einem fortlaufenden 24-Stunden-Zeitraum 25 Mal erfolgreich deaktivieren.
UpdateShardCount		Pro Stream	Gilt nur für bereitgestellte. Das Standardlimit für die Anzahl der Shards beträgt 10 000. Diesbezüglich gibt es zusätzliche Beschränkungen. API Weitere Informationen finden Sie unter UpdateShardCount.

API	APIAnruflimit	Pro Konto/Stream	Beschreibung
UpdateStreamMode		Pro Stream	Für jeden Datenstream am in Ihrem AWS Konto können Sie innerhalb von 24 Stunden zweimal zwischen den Modi „On-Demand-Kapazität“ und „Bereitgestellte Kapazität“ wechseln.

KDSGrenzwerte für die Datenebene API

Im folgenden Abschnitt werden die Grenzwerte für die KDS Datenebene beschriebenAPIs. KDSMit der APIs Datenebene können Sie Ihre Datenströme für die Erfassung und Verarbeitung von Datensätzen in Echtzeit verwenden. Diese Limits gelten pro Shard innerhalb Ihrer Datenströme.

APIGrenzen der Datenebene

API	APIAnruflimit	Nutzlast-Limit	Weitere Details
GetRecords	5 TPS	Die maximale Anzahl von Datensätzen, die pro Anruf zurückgegeben werden können, beträgt 10.000. Die maximale Größe der Daten, die `GetRecords` zurückgeben kann, ist 10 MB.	Wenn ein Aufruf diese Datenmenge zurückgibt, lösen nachfolgende Aufrufe innerhalb der nächsten 5 Sekunden eine `ProvisionedThroughputExceededException` aus. Wenn im Stream nicht

API	APIAnruflimit	Nutzlast-Limit	Weitere Details
			genügend Durchsatz bereitgestellt ist, lösen nachfolgende Aufrufe innerhalb der nächsten 1 Sekunde `ProvisionedThroughputExceededException` aus.
GetShardIterator	5 TPS		Ein Shard-Iterator läuft 5 Minuten nach Rückgabe an den Anforderer ab. Wenn eine GetShardIterator Anfrage zu oft gestellt wird, erhalten Sie eine ProvisionedThroughputExceededException.
PutRecord	1000 TPS	Jeder Shard unterstützt Schreibvorgänge von bis zu 1.000 Datensätzen pro Sekunde bis zu einem maximalen Datenschreibvolumen von 1 MB pro Sekunde.	

API	APIAnruflimit	Nutzlast-Limit	Weitere Details
PutRecords		Jede PutRecords Anfrage kann bis zu 500 Datensätze unterstützen. Die maximale Größe jedes Datensatzes in der Anforderung beträgt 1 MB bis zu einem Limit von 5 MB für die gesamte Anforderung einschließlich Partitionsschlüsseln. Jeder Shard unterstützt Schreibvorgänge von bis zu 1.000 Datensätzen pro Sekunde bis zu einem maximalen Datenschreibvolumen von 1 MB pro Sekunde.	
SubscribeToShard	Sie können SubscribeToShard pro Sekunde pro registriertem Verbraucher pro Shard einen Anruf tätigen.		Wenn Sie SubscribeToShard erneut mit demselben Verbraucher anrufen ARN und ShardId innerhalb von 5 Sekunden nach einem erfolgreichen Anruf, erhalten Sie eineResourceInUseException.

Erhöhung der Kontingente

Sie können mit Service Quotas eine Erhöhung für ein Kontingent beantragen, sofern das Kontingent anpassbar ist. Einige Anfragen werden automatisch gelöst, andere werden an AWS Support übermittelt. Sie können den Status einer beantragten Kontingenterhöhung verfolgen, die an AWS Support gesendet wird. Anfragen zur Erhöhung der Servicekontingente erhalten keinen bevorzugten Support. Wenn Sie eine dringende Anfrage haben, wenden Sie sich an den AWS Support. Weitere Informationen finden Sie unter Was sind Service Quotas?.

Um eine Erhöhung des Servicekontingents anzufordern, befolgen Sie das unter Kontingenterhöhung beantragen beschriebenen Verfahren.

Vollständige Voraussetzungen für die Einrichtung von Amazon Kinesis Data Streams

Bevor Sie Amazon Kinesis Data Streams zum ersten Mal verwenden, führen Sie die folgenden Aufgaben durch, um Ihre Umgebung einzurichten.

Aufgaben

- Melden Sie sich an für AWS
- Bibliotheken und Tools herunterladen
- Konfigurieren Sie Ihre Entwicklungsumgebung

Melden Sie sich an für AWS

Wenn Sie sich für Amazon Web Services (AWS) registrieren, wird Ihr AWS Konto automatisch für alle Dienste angemeldet AWS, einschließlich Kinesis Data Streams. Berechnet werden Ihnen aber nur die Services, die Sie nutzen.

Wenn Sie bereits ein AWS Konto haben, fahren Sie mit der nächsten Aufgabe fort. Wenn Sie kein AWS -Konto haben, führen Sie die folgenden Schritte zum Erstellen eines Kontos aus.

Um sich für ein AWS Konto zu registrieren

1. Öffnen Sie https://portal.aws.amazon.com/billing/die Anmeldung.
2. Folgen Sie den Online-Anweisungen.

   Bei der Anmeldung müssen Sie auch einen Telefonanruf entgegennehmen und einen Verifizierungscode über die Telefontasten eingeben.

   Wenn Sie sich für eine anmelden AWS-Konto, Root-Benutzer des AWS-Kontoswird eine erstellt. Der Root-Benutzer hat Zugriff auf alle AWS -Services und Ressourcen des Kontos. Als bewährte Sicherheitsmethode weisen Sie einem Administratorbenutzer Administratorzugriff zu und verwenden Sie nur den Root-Benutzer, um Aufgaben auszuführen, die Root-Benutzerzugriff erfordern.

Bibliotheken und Tools herunterladen

Die folgenden Bibliotheken und Tools unterstützen Sie bei der Arbeit mit Kinesis Data Streams:

- Die Amazon Kinesis API Kinesis-Referenz ist der grundlegende Satz von Vorgängen, die Kinesis Data Streams unterstützt. Weitere Informationen zum Durchführen grundlegender Operationen mit Java-Code finden Sie in den folgenden Ressourcen:
  - Entwickeln Sie Produzenten, die Amazon Kinesis Data Streams verwenden, API mit dem AWS SDK for Java
  - Entwickeln Sie benutzerdefinierte Verbraucher mit gemeinsamem Durchsatz mithilfe der AWS SDK for Java
  - Kinesis-Datenstreams erstellen und verwalten
- Das AWS SDKs für Go, Java, JavaScript. NET, Node.js PHP, Python und Ruby bieten Unterstützung und Beispiele für Kinesis Data Streams. Wenn Ihre Version von AWS SDK for Java keine Beispiele für Kinesis Data Streams enthält, können Sie sie auch von GitHubherunterladen.
- Die Kinesis Client Library (KCL) bietet ein easy-to-use Programmiermodell für die Datenverarbeitung. Das KCL kann Ihnen helfen, schnell mit Kinesis Data Streams in Java, Node.js, zu beginnen. NET, Python und Ruby. Weitere Informationen finden Sie unter Lesen von Daten aus Streams.
- Das AWS Command Line Interface unterstützt Kinesis Data Streams. Das AWS CLI ermöglicht es Ihnen, mehrere AWS Dienste von der Befehlszeile aus zu steuern und sie mithilfe von Skripten zu automatisieren.

Konfigurieren Sie Ihre Entwicklungsumgebung

Um das verwenden zu könnenKCL, stellen Sie sicher, dass Ihre Java-Entwicklungsumgebung die folgenden Anforderungen erfüllt:

- Java 1.7 (Java SE 7JDK) oder höher. Sie können die neueste Java-Software von der Oracle-Website unter Java SE-Downloads herunterladen.
- Apache Commons-Paket (Code, HTTP Client und Protokollierung)
- JSONJackson-Prozessor

Beachten Sie, dass das AWS SDK for Java Apache Commons und Jackson im Drittanbieter-Ordner enthält. Die SDK für Java funktioniert jedoch mit Java 1.6, während die Kinesis Client Library Java 1.7 benötigt.

Verwenden Sie die AWS CLI , um Amazon Kinesis Data Streams Streams-Operationen durchzuführen

In diesem Abschnitt erfahren Sie, wie Sie grundlegende Amazon Kinesis Data Streams Streams-Operationen mit dem AWS Command Line Interface durchführen. Sie erlernen Grundlagen für den Datenfluss von Kinesis Data Streams und die erforderlichen Schritte zum Senden von Daten an einen Kinesis-Datenstrom sowie zum Abrufen von Daten von dort.

Wenn Sie Kinesis Data Streams zum ersten Mal verwenden, sollten Sie sich zunächst mit den Konzepten und der Terminologie in Terminologie und Konzepte von Amazon Kinesis Data Streams vertraut machen.

Themen

- Tutorial: Installation und Konfiguration von AWS CLI for Kinesis Data Streams
- Tutorial: Führen Sie grundlegende Kinesis Data Streams Streams-Operationen mit dem AWS CLI

Für CLI den Zugriff benötigen Sie eine Zugriffsschlüssel-ID und einen geheimen Zugriffsschlüssel. Verwenden Sie möglichst temporäre Anmeldeinformationen anstelle langfristiger Zugriffsschlüssel. Temporäre Anmeldeinformationen bestehen aus einer Zugriffsschlüssel-ID, einem geheimen Zugriffsschlüssel und einem Sicherheits-Token, das angibt, wann die Anmeldeinformationen ablaufen. Weitere Informationen finden Sie im IAMBenutzerhandbuch unter Verwenden temporärer Anmeldeinformationen mit AWS Ressourcen.

Eine ausführliche step-by-step IAM Anleitung zur Einrichtung von Sicherheitsschlüsseln finden Sie unter IAM Benutzer erstellen.

In diesem Abschnitt werden die behandelten spezifischen Befehle unverändert übernommen, es sei denn, bestimmte Werte sind für jede Ausführung unbedingt unterschiedlich. Außerdem verwenden die Beispiele die USA West (Oregon)-Region, aber die Schritte in diesem Abschnitt funktionieren in allen Regionen, in denen Kinesis Data Streams unterstützt wird.

Tutorial: Installation und Konfiguration von AWS CLI for Kinesis Data Streams

Installieren Sie das AWS CLI

Ausführliche Anweisungen zur Installation der Betriebssysteme AWS CLI für Windows und Linux, OS X und Unix finden Sie unter Installation von AWS CLI.

Verwenden Sie den folgenden Befehl, um die verfügbaren Optionen und Services anzuzeigen:

```
aws help
```

Sie werden den Kinesis Data Streams-Dienst verwenden, sodass Sie die AWS CLI Unterbefehle für Kinesis Data Streams mit dem folgenden Befehl überprüfen können:

```
aws kinesis help
```

Mit diesem Befehl erhalten Sie eine Ausgabe mit den verfügbaren Befehlen für Kinesis Data Streams:

```
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
```

Diese Befehlsliste entspricht den Kinesis Data Streams, die in der Amazon Kinesis Service API Reference API dokumentiert sind. Der `create-stream` Befehl entspricht beispielsweise der Aktion. `CreateStream` API

Der AWS CLI ist jetzt erfolgreich installiert, aber nicht konfiguriert. Dies wird im nächsten Abschnitt veranschaulicht.

Konfigurieren Sie das AWS CLI

Für den allgemeinen Gebrauch ist der `aws configure` Befehl der schnellste Weg, Ihre AWS CLI Installation einzurichten. Weitere Informationen finden Sie unter Konfigurieren der AWS CLI.

Tutorial: Führen Sie grundlegende Kinesis Data Streams Streams-Operationen mit dem AWS CLI

In diesem Abschnitt wird die grundlegende Verwendung eines Kinesis-Datenstroms über die Befehlszeile AWS CLI beschrieben. Stellen Sie sicher, dass Sie mit den unter Terminologie und Konzepte von Amazon Kinesis Data Streams behandelten Konzepten vertraut sind.

> **Note**
>
> Nachdem Sie einen Stream erstellt haben, fallen für Ihr Konto geringe Gebühren für die Nutzung von Kinesis Data Streams an, da Kinesis Data Streams nicht für das AWS kostenlose Kontingent in Frage kommt. Wenn Sie mit diesem Tutorial fertig sind, löschen Sie Ihre AWS Ressourcen, damit keine Gebühren mehr anfallen. Weitere Informationen finden Sie unter Schritt 4: Bereinigen.

Themen

Schritt 1: Erstellen Sie einen Stream

Der erste Schritt besteht darin, einen Stream zu erstellen und sicherzustellen, dass dieser erfolgreich erstellt wurde. Verwenden Sie den folgenden Befehl zum Erstellen eines Streams mit dem Namen „Foo”.

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

Als Nächstes führen Sie den folgenden Befehl aus, um die Erstellung des Streams zu überprüfen:

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

Sie sollten eine Ausgabe ähnlich dem folgenden Beispiel erhalten:

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

In diesem Beispiel hat der Stream einen StatusCREATING, was bedeutet, dass er noch nicht einsatzbereit ist. Prüfen Sie den Status nach wenigen Minuten erneut. Jetzt sollten Sie eine Ausgabe ähnlich dem folgenden Beispiel erhalten:

```
{
    "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
    }
}
```

Diese Ausgabe enthält Informationen, die Sie für dieses Tutorial nicht benötigen. Die wichtigsten Informationen für den Moment sind`"StreamStatus": "ACTIVE"`, dass Sie wissen, dass der Stream bereit ist, verwendet zu werden, und die Informationen zu dem einzelnen Shard, den Sie angefordert haben. Sie können die Existenz Ihres neuen Streams auch mit dem Befehl `list-streams` überprüfen, wie in der folgenden Abbildung dargestellt:

```
aws kinesis list-streams
```

Ausgabe:

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

Schritt 2: Einen Datensatz erstellen

Jetzt, da Sie über einen aktiven Stream verfügen, können Sie Daten an diesen senden. In diesem Tutorial verwenden Sie einen möglichst einfachen Befehl, `put-record`. Dieser sendet einen einzelnen Datensatz mit dem Text „testdata” in den Stream.

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

Ist der Befehl erfolgreich, wird eine Ausgabe zurückgegeben, die wie folgt aussehen sollte:

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

Herzlichen Glückwunsch, Sie haben gerade Daten zu einem Stream hinzugefügt! Als Nächstes erfahren Sie, wie Sie Daten aus dem Stream abrufen können.

Schritt 3: Holen Sie sich den Datensatz

GetShardIterator

Bevor Sie Daten aus dem Stream abrufen können, müssen Sie den Shard-Iterator für den Shard abrufen, an dem Sie interessiert sind. Ein Shard-Iterator stellt die Position des Streams und des Shards dar, aus denen der Konsument (in diesem Fall der Befehl `get-record`) Daten ausliest. Sie verwenden den `get-shard-iterator` Befehl wie folgt:

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

Denken Sie daran, dass API hinter den `aws kinesis` Befehlen ein Kinesis Data Streams steckt. Wenn Sie sich also für einen der angezeigten Parameter interessieren, können Sie im `GetShardIterator`APIReferenzthema mehr darüber lesen. Eine erfolgreiche Ausführung führt zu einer Ausgabe, die dem folgenden Beispiel ähnelt:

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

Die lange Zeichenfolge scheinbar zufälliger Zeichen ist der Shard-Iterator (Ihrer lautet jedoch anders). Sie müssen den Shard-Iterator kopieren/in den Befehl get einfügen, der als Nächstes gezeigt wird. Shard-Iteratoren verfügen über eine Gültigkeitsdauer von 300 Sekunden. Dies sollte ausreichen, um den Shard-Iterator zu kopieren und in den nächsten Befehl einzufügen. Sie müssen alle Zeilenumbrüche aus Ihrem Shard-Iterator entfernen, bevor Sie sie in den nächsten Befehl einfügen. Wenn Sie eine Fehlermeldung erhalten, dass der Shard-Iterator nicht mehr gültig ist, führen Sie den Befehl erneut aus. `get-shard-iterator`

GetRecords

Der `get-records` Befehl ruft Daten aus dem Stream ab und wird in einem Aufruf von `GetRecords`in den Kinesis Data Streams aufgelöst. API Der Shard-Iterator gibt die Position im Shard an, ab der Datensätze sequenziell ausgelesen werden sollen: Wenn keine Datensätze in dem Teil des Shards verfügbar sind, auf die der Iterator zeigt, gibt `GetRecords` eine leere Liste zurück. Möglicherweise sind mehrere Aufrufe erforderlich, um zu einem Teil des Shards zu gelangen, der Datensätze enthält.

Im folgenden Beispiel für den `get-records` Befehl:

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

Wenn Sie dieses Tutorial von einem UNIX-Befehlsprozessor wie der Bash aus ausführen, können Sie die Erfassung des Shard-Iterators mit einem verschachtelten Befehl wie dem Folgenden automatisieren:

```
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
```

Wenn Sie dieses Tutorial von einem System aus ausführen, das dies unterstützt PowerShell, können Sie die Erfassung des Shard-Iterators mit einem Befehl wie dem folgenden automatisieren:

```
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])
```

Das erfolgreiche Ergebnis des `get-records` Befehls fordert Datensätze aus Ihrem Stream für den Shard an, den Sie beim Abrufen des Shard-Iterators angegeben haben, wie im folgenden Beispiel:

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

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

Beachten Sie, dass `get-records` dies oben als Anfrage beschrieben wurde, was bedeutet, dass Sie möglicherweise keine oder mehr Datensätze erhalten, selbst wenn Ihr Stream Datensätze enthält. Alle zurückgegebenen Datensätze stellen möglicherweise nicht alle Datensätze dar, die sich derzeit in Ihrem Stream befinden. Das ist normal, und der Produktionscode fragt den Stream in angemessenen Intervallen nach Datensätzen ab. Diese Abfragegeschwindigkeit hängt von Ihren spezifischen Anforderungen an das Anwendungsdesign ab.

In Ihren Aufzeichnungen in diesem Teil des Tutorials werden Sie feststellen, dass die Daten unbrauchbar zu sein scheinen — und es ist nicht der Klartext, den `testdata` wir gesendet haben. Dies liegt an der Art und Weise, auf die `put-record` Base64-Codierung verwendet, um Ihnen das Senden von Binärdaten zu ermöglichen. Die Kinesis Data Streams Streams-Unterstützung in bietet jedoch AWS CLI keine Base64-Decodierung, da die Base64-Decodierung von rohen Binärinhalten, die auf stdout gedruckt werden, auf bestimmten Plattformen und Terminals zu unerwünschtem Verhalten und potenziellen Sicherheitsproblemen führen kann. Wenn Sie einen Base64-Decoder verwenden (z. B. https://www.base64decode.org/), um `dGVzdGRhdGE=` manuell zu decodieren, sehen Sie, dass es tatsächlich `testdata` ist. Dies ist für dieses Tutorial ausreichend, da der in der Praxis nur selten zur Nutzung von Daten verwendet wird. AWS CLI Häufiger wird es verwendet, um den Status des Streams zu überwachen und Informationen abzurufen, wie zuvor gezeigt (`describe-stream`und`list-streams`). Weitere Informationen zu finden Sie unter Entwickeln benutzerdefinierter Verbraucher mit gemeinsamem Durchsatz unter Verwendung von KCL. KCL

`get-records`gibt nicht immer alle Datensätze im angegebenen Stream/Shard zurück. Verwenden Sie in diesem Fall den `NextShardIterator` aus dem letzten Ergebnis, um die nächste

Datensatzgruppe abzurufen. Wenn mehr Daten in den Stream eingegeben würden, was bei Produktionsanwendungen der Normalfall ist, könnten Sie jedes Mal nach nutzbaren Daten suchen. `get-records` Wenn Sie jedoch innerhalb der Lebensdauer von 300 Sekunden nicht `get-records` mit dem nächsten Shard-Iterator aufrufen, erhalten Sie eine Fehlermeldung, und Sie müssen den `get-shard-iterator` Befehl verwenden, um einen neuen Shard-Iterator zu erhalten.

In dieser Ausgabe wird auch angegeben`MillisBehindLatest`, wie viele Millisekunden die Antwort der GetRecordsOperation von der Spitze des Streams ausgeht, was angibt, wie weit der Verbraucher hinter der aktuellen Zeit zurückliegt. Der Wert Null gibt an, dass die Datenverarbeitung aktuell ist und dass zurzeit keine neuen zu verarbeitenden Datensätze vorhanden sind. In diesem Tutorial sehen Sie möglicherweise eine recht große Zeit, wenn Sie sich die Zeit dafür genommen haben, stets mitzulesen. Standardmäßig verbleiben Datensätze 24 Stunden lang in einem Stream und warten darauf, dass Sie sie abrufen. Dieser Zeitraum wird als Aufbewahrungszeitraum bezeichnet und ist bis zu 365 Tage konfigurierbar.

Ein erfolgreiches `get-records` Ergebnis wird immer angezeigt, `NextShardIterator` auch wenn sich derzeit keine weiteren Datensätze im Stream befinden. Hierbei handelt es sich um ein Abrufmodell, bei dem davon ausgegangen wird, dass ein Produzent zu einem bestimmten Zeitpunkt möglicherweise mehr Datensätze in den Stream einfügt. Sie können zwar Ihre eigenen Abfrageroutinen schreiben, aber wenn Sie die zuvor genannten KCL für die Entwicklung von Verbraucheranwendungen verwenden, wird diese Abfrage für Sie erledigt.

Wenn Sie aufrufen, `get-records` bis der Stream und der Shard, aus dem Sie abrufen, keine Datensätze mehr enthalten, erhalten Sie eine Ausgabe mit leeren Datensätzen, die dem folgenden Beispiel ähnelt:

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

Schritt 4: Bereinigen

Lösche deinen Stream, um Ressourcen freizugeben und ungewollte Gebühren für dein Konto zu vermeiden. Tun Sie dies jedes Mal, wenn Sie einen Stream erstellt haben und ihn nicht verwenden werden, da Gebühren pro Stream anfallen, unabhängig davon, ob Sie Daten mit ihm übertragen und abrufen oder nicht. Der Befehl zum Aufräumen lautet wie folgt:

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

Erfolg führt zu keiner Ausgabe. Verwenden Sie`describe-stream`, um den Fortschritt des Löschvorgangs zu überprüfen:

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

Wenn Sie diesen Befehl unmittelbar nach dem Löschbefehl ausführen, erhalten Sie eine Ausgabe, die dem folgenden Beispiel ähnelt:

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

Nachdem der Stream vollständig gelöscht wurde, führt `describe-stream` zum Fehler „nicht gefunden”.

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

Tutorials „Erste Schritte“ für Amazon Kinesis Data Streams

Amazon Kinesis Data Streams bietet eine Reihe verschiedener Lösungen für die Aufnahme und Nutzung von Daten aus Kinesis-Datenströmen. Die Tutorials in diesem Abschnitt sollen Ihnen helfen, die Konzepte und Funktionen von Amazon Kinesis Data Streams besser zu verstehen und die Lösung zu finden, die Ihren Anforderungen entspricht.

Themen

Tutorial: Verarbeiten Sie Aktiendaten in Echtzeit mit KPL und KCL 2.x

Das Szenario für dieses Tutorial beinhaltet die Aufnahme von Aktiengeschäften in einen Datenstream und das Schreiben einer grundlegenden Amazon Kinesis Data Streams Streams-Anwendung, die Berechnungen im Stream durchführt. Sie lernen, wie Sie einen Stream von Datensätzen an Kinesis Data Streams senden und eine Anwendung implementieren, die die Datensätze nahezu in Echtzeit verarbeitet und verarbeitet.

> **Important**
>
> Nachdem Sie einen Stream erstellt haben, fallen für Ihr Konto geringe Gebühren für die Nutzung von Kinesis Data Streams an, da Kinesis Data Streams nicht für das AWS kostenlose Kontingent in Frage kommt. Nach dem Start der Konsumentenanwendung fallen nominale Gebühren für die Amazon-DynamoDB-Nutzung an. Die Konsumentenanwendung verwendet DynamoDB zum Verfolgen des Verarbeitungsstatus. Wenn Sie mit dieser Anwendung fertig sind, sollten Sie Ihre AWS -Ressourcen löschen, damit keine weiteren Gebühren anfallen. Weitere Informationen finden Sie unter Bereinigen von -Ressourcen.

Der Code greift nicht auf tatsächliche Wertpapierdaten zu, sondern simuliert nur deren Strom. Dazu werden zufällige Wertpapierdaten erzeugt. Ausgangspunkt sind dabei echte Marktdaten der 25 führenden Aktien gemäß Börsenkapitalisierung von Februar 2015. Wenn Sie Zugriff auf einen Echtzeit-Stream von Wertpapierdaten haben, möchten Sie vermutlich nützliche, zeitnahe Statistiken aus den Stream-Daten erzeugen. Sie können beispielsweise eine Zeitfensteranalyse durchführen, um festzustellen, welche Aktie in den letzten 5 Minuten am häufigsten erworben wurde. Oder Sie möchten im Falle eines zu großen Verkaufsauftrags (d. h. zu viele Anteile) benachrichtigt werden. Der Code in diesem Tutorial kann erweitert werden, um solche Funktionen bereitzustellen.

Sie können die in diesem Tutorial aufgeführten Schritte mit einem Desktop-PC oder Laptop durchführen und sowohl den Produzenten- als auch den Verbrauchercode auf demselben Rechner oder auf jeder Plattform ausführen, die den definierten Anforderungen entspricht.

Bei den gezeigten Beispielen wird die Region USA West (Oregon) verwendet. Sie funktionieren aber auch für alle anderen AWS -Regionen, die Kinesis Data Streams unterstützen.

Aufgaben

- Erfüllen der Voraussetzungen
- Erstellen Sie einen Datenstream
- Erstellen Sie eine IAM Richtlinie und einen Benutzer
- Laden Sie den Code herunter und erstellen Sie ihn
- Implementieren Sie den Hersteller
- Implementieren Sie den Verbraucher
- (Optional) Erweitern Sie den Consumer-Bereich
- Bereinigen von -Ressourcen

Erfüllen der Voraussetzungen

Sie müssen die folgenden Voraussetzungen erfüllen, um dieses Tutorial abschließen zu können:

Erstellen und verwenden Sie ein Amazon Web Services Services-Konto

Bevor Sie beginnen, stellen Sie sicher, dass Sie mit den unter besprochenen Konzepten vertraut sindTerminologie und Konzepte von Amazon Kinesis Data Streams, insbesondere mit Streams, Shards, Produzenten und Verbrauchern. Es ist zudem sinnvoll, die Schritte in der folgenden Anleitung durchzuführen: Tutorial: Installation und Konfiguration von AWS CLI for Kinesis Data Streams.

Sie benötigen ein AWS Konto und einen Webbrowser, um auf die AWS Management Console zugreifen zu können.

Verwenden Sie für den Konsolenzugriff Ihren IAM Benutzernamen und Ihr Passwort, um sich von der Anmeldeseite AWS Management Consoleaus auf der IAM Anmeldeseite anzumelden. Informationen zu AWS Sicherheitsanmeldedaten, einschließlich programmatischem Zugriff und Alternativen zu langfristigen Anmeldeinformationen, finden Sie unter AWS Sicherheitsanmeldedaten im IAMBenutzerhandbuch. Einzelheiten zur Anmeldung bei Ihrem AWS-Konto finden Sie unter So melden Sie sich an AWS im AWS-Anmeldung Benutzerhandbuch.

Weitere Informationen IAM und Anweisungen zur Einrichtung eines Sicherheitsschlüssels finden Sie unter IAM Benutzer erstellen.

Erfüllen Sie die Anforderungen an die Systemsoftware

Im System, das Sie zum Ausführen der Anwendung verwenden, muss Java 7 oder höher installiert sein. Um das neueste Java Development Kit (JDK) herunterzuladen und zu installieren, gehen Sie zur Java SE-Installationsseite von Oracle.

Sie benötigen die neueste AWS SDK for Java-Version.

Die Consumer-Anwendung benötigt die Kinesis Client Library (KCL) Version 2.2.9 oder höher, die Sie GitHub unter https://github.com/awslabs/ amazon-kinesis-client /tree/master abrufen können.

Nächste Schritte

Erstellen Sie einen Datenstream

Erstellen Sie einen Datenstream

Zuerst müssen Sie den Daten-Stream erstellen, den Sie in den weiteren Schritten dieses Tutorials verwenden.

So erstellen Sie einen Stream

1. Melden Sie sich bei der an AWS Management Console und öffnen Sie die Kinesis-Konsole unter https://console.aws.amazon.com/kinesis.
2. Klicken Sie im Navigationsbereich auf Data Streams (Daten-Streams).
3. Erweitern Sie in der Navigationsleiste die Regionsauswahl und wählen Sie eine Region aus.

4. Wählen Sie Create Kinesis Stream (Kinesis-Stream erstellen).
5. Geben Sie einen Namen für den Daten-Stream ein (z. B. **`StockTradeStream`**).
6. Geben Sie **1** die Anzahl der Shards ein, lassen Sie aber Estimate the number of shards you'll need (Schätzung der Anzahl der Shards, die Sie benötigen) reduziert.
7. Wählen Sie Create Kinesis Stream (Kinesis-Stream erstellen).

Auf der Listenseite Kinesis Streams (Kinesis-Streams) lautet der Status des Streams `CREATING`, während der Stream erstellt wird. Sobald der Stream verwendet werden kann, ändert sich der Status in `ACTIVE`.

Wenn Sie den Namen des Streams auswählen, wird auf der angezeigten Seite auf der Registerkarte Details eine Zusammenfassung der Konfiguration des Daten-Streams angezeigt. Der Abschnitt Monitoring (Überwachung) zeigt Überwachungsinformationen für den Stream an.

Nächste Schritte

Erstellen Sie eine IAM Richtlinie und einen Benutzer

Erstellen Sie eine IAM Richtlinie und einen Benutzer

Bewährte Sicherheitsmethoden AWS schreiben die Verwendung detaillierter Berechtigungen vor, um den Zugriff auf verschiedene Ressourcen zu kontrollieren. AWS Identity and Access Management (IAM) ermöglicht Ihnen die Verwaltung von Benutzern und Benutzerberechtigungen in. AWS In einer IAMRichtlinie werden die zulässigen Aktionen und die Ressourcen, für die die Aktionen gelten, explizit aufgeführt.

Im Folgenden werden die Berechtigungen für Produzenten und Konsumenten von Kinesis Data Streams aufgelistet, die mindestens erforderlich sind.

Produzent

Aktionen	Ressource	Zweck
`DescribeStream` , `DescribeStreamSummary` , `DescribeStreamConsumer`	Kinesis Data Stream	Vor dem Versuch, Datensätze zu lesen, prüft der Verbrauch eam vorhanden und ob er aktiv ist und ob Shards im Daten

Aktionen	Ressource	Zweck
`SubscribeToShard` , `RegisterStreamConsumer`	Kinesis Data Stream	Abonniert und registriert Verbraucher bei einem Shard.
`PutRecord` , `PutRecords`	Kinesis Data Stream	Schreibt Datensätze in Kinesis Data Streams.

Konsument

Aktionen	Ressource	Zweck
`DescribeStream`	Kinesis Data Stream	Vor dem Versuch, Datensätze zu lesen, prüft der Verbrauch eam vorhanden und ob er aktiv ist und ob Shards im Daten
`GetRecords` , `GetShardIterator`	Kinesis Data Stream	Auslesen von Datensätzen aus einem Shard.
`CreateTable` , `DescribeTable` , `GetItem`, `PutItem`, `Scan`, `UpdateItem`	Amazon-DynamoDB-Tabelle.	Wenn der Consumer mit der Kinesis Client Library (KCL) (e oder 2.x) entwickelt wurde, benötigt er Berechtigungen für e Tabelle, um den Verarbeitungsstatus der Anwendung verfo
`DeleteItem`	Amazon-DynamoDB-Tabelle.	Für den Fall, dass der Konsument Split/Merge-Operationen Kinesis Data Streams ausführt.
`PutMetricData`	CloudWatch Amazon-Protokoll	KCLEs lädt auch Metriken hoch CloudWatch, die für die Üb Anwendung nützlich sind.

In diesem Tutorial erstellen Sie eine einzige IAM Richtlinie, die alle zuvor genannten Berechtigungen gewährt. Im Praxiseinsatz können Sie zwei Richtlinien erstellen, eine für Produzenten und eine für Verbraucher.

So erstellen Sie eine IAM-Richtlinie

1. Suchen Sie den Amazon-Ressourcennamen (ARN) für den neuen Datenstream, den Sie im vorherigen Schritt erstellt haben. Sie finden ihn oben ARN auf der Registerkarte „Details“ als Stream. ARN Das ARN Format ist wie folgt:

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

   Region

   Der AWS Regionalcode; zum Beispiel`us-west-2`. Weitere Informationen finden Sie unter Regionen und Verfügbarkeitskonzepte.

   Konto

   Die AWS Konto-ID, wie in den Kontoeinstellungen angezeigt.

   Name

   Der Name des Datenstroms, den Sie im vorherigen Schritt erstellt haben, nämlich`StockTradeStream`.

2. Ermitteln Sie ARN die DynamoDB-Tabelle, die vom Verbraucher verwendet (und von der ersten Consumer-Instance erstellt werden soll). Er muss das folgende Format aufweisen:

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

   Die Region und die Konto-ID sind identisch mit den Werten im Datenstream, den Sie für dieses Tutorial verwenden, aber der Name ist der Name der DynamoDB-Tabelle, die von der Verbraucheranwendung erstellt und verwendet wird. ARN KCLverwendet den Namen der Anwendung als Tabellennamen. Verwenden Sie `StockTradesProcessor` in diesem Schritt für den DynamoDB-Tabellennamen, da dies der Anwendungsname ist, der in späteren Schritten in diesem Tutorial verwendet wird.

3. Wählen Sie in der IAM Konsole unter Policies (https://console.aws.amazon.com/iam/home#policies) die Option Create policy aus. Wenn Sie zum ersten Mal mit IAM Richtlinien arbeiten, wählen Sie Erste Schritte, Richtlinie erstellen aus.
4. Wählen Sie Select (Auswählen) neben Policy Generator (Richtliniengenerator) aus.
5. Wählen Sie Amazon Kinesis als AWS Service.

6. Legen Sie `DescribeStream`, `GetShardIterator`, `GetRecords`, `PutRecord` und `PutRecords` als zulässige Aktionen fest.
7. Geben Sie ARN den Datenstrom ein, den Sie in diesem Tutorial verwenden.
8. Verwenden Sie Add Statement (Statement hinzufügen) für die folgenden Elemente:

AWS Dienst	Aktionen	ARN
Amazon-DynamoDB	`CreateTable` , `DeleteItem` , `DescribeTable` , `GetItem`, `PutItem`, `Scan`, `UpdateItem`	Die ARN der DynamoDB-Tabelle, die Sie in Schritt 2 dieses Verfahrens erstellt haben.
Amazon CloudWatch	`PutMetricData`	*

   Das Sternchen (*), das bei der Angabe von verwendet wird, ARN ist nicht erforderlich. In diesem Fall liegt das daran, dass es keine bestimmte Ressource gibt, für die die `PutMetricData` Aktion aufgerufen wird. CloudWatch
9. Wählen Sie Next Step (Weiter) aus.
10. Ändern Sie Policy Name (Richtlinienname) in `StockTradeStreamPolicy`, prüfen Sie den Code und wählen sie Create Policy (Richtlinie erstellen) aus.

Das resultierende Richtliniendokument sollte folgendermaßen aussehen:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmt123",
      "Effect": "Allow",
      "Action": [
        "kinesis:DescribeStream",
        "kinesis:PutRecord",
        "kinesis:PutRecords",
        "kinesis:GetShardIterator",
        "kinesis:GetRecords",
        "kinesis:ListShards",
```

```
        "kinesis:DescribeStreamSummary",
        "kinesis:RegisterStreamConsumer"
      ],
      "Resource": [
        "arn:aws:kinesis:us-west-2:123:stream/StockTradeStream"
      ]
    },
    {
      "Sid": "Stmt234",
      "Effect": "Allow",
      "Action": [
        "kinesis:SubscribeToShard",
        "kinesis:DescribeStreamConsumer"
      ],
      "Resource": [
        "arn:aws:kinesis:us-west-2:123:stream/StockTradeStream/*"
      ]
    },
    {
      "Sid": "Stmt456",
      "Effect": "Allow",
      "Action": [
        "dynamodb:*"
      ],
      "Resource": [
        "arn:aws:dynamodb:us-west-2:123:table/StockTradesProcessor"
      ]
    },
    {
      "Sid": "Stmt789",
      "Effect": "Allow",
      "Action": [
        "cloudwatch:PutMetricData"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

So erstellen Sie einen IAM-Benutzer

1. Öffnen Sie die IAM Konsole unter. https://console.aws.amazon.com/iam/
2. Wählen Sie auf der Seite Users (Benutzer) die Option Add user (Benutzer hinzufügen) aus.
3. Geben Sie für User name `StockTradeStreamUser` ein.
4. Wählen Sie für Access type (Zugriffstyp) die Option Programmatic access (Programmgesteuerter Zugriff) und wählen Sie dann Next: Permissions (Weiter: Berechtigungen).
5. Wählen Sie Vorhandene Richtlinien direkt zuordnen.
6. Suchen Sie nach dem Namen der Richtlinie, die Sie im vorherigen Verfahren erstellt haben (`StockTradeStreamPolicy`. Markieren Sie das Kontrollkästchen links neben dem Richtliniennamen und wählen Sie dann Next: Review (Weiter: Prüfen).
7. Überprüfen Sie die Details und die Zusammenfassung und wählen Sie dann Create user (Benutzer erstellen) aus.
8. Kopieren Sie die Access key ID (Zugriffsschlüssel-ID) und speichern Sie sie privat. Wählen Sie unter Secret access key (Geheimer Zugriffsschlüssel) die Option Show (Anzeigen) und speichern Sie auch diesen Schlüssel privat.
9. Fügen Sie die Zugriffs- und Geheimschlüssel in eine lokalen Datei an einem sichern Ort ein, auf den nur Sie Zugriff haben. Erstellen Sie für diese Anwendung eine Datei namens `~/.aws/credentials` (mit strikten Berechtigungen). Die Datei sollte das folgende Format aufweisen:

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

Um eine IAM Richtlinie an einen Benutzer anzuhängen

1. Öffnen Sie in der IAM Konsole Richtlinien und wählen Sie Richtlinienaktionen aus.
2. Klicken Sie auf `StockTradeStreamPolicy` und Attach (Verknüpfen).
3. Wählen Sie `StockTradeStreamUser` und Attach Policy (Richtlinie anfügen) aus.

Nächste Schritte

Laden Sie den Code herunter und erstellen Sie ihn

Laden Sie den Code herunter und erstellen Sie ihn

In diesem Thema finden Sie Beispielcode zur Implementierung der Beispiel-Wertpapiertransaktionen im Daten-Stream (Produzent) und zur Verarbeitung dieser Daten (Verbraucher).

So laden Sie den Code herunter und erstellen ihn

1. Laden Sie den Quellcode aus dem https://github.com/aws-samples/amazon-kinesis-learning GitHub Repo auf Ihren Computer herunter.
2. Erstellen Sie ein Projekt in Ihrem IDE mit dem Quellcode und halten Sie sich dabei an die bereitgestellte Verzeichnisstruktur.
3. Fügen Sie dem Projekt die folgenden Bibliotheken hinzu:
   - Amazon Kinesis Kinesis-Clientbibliothek () KCL
   - AWS SDK
   - Apache HttpCore
   - Apache HttpClient
   - Apache Commons Lang
   - Apache Commons Logging
   - Guava (Google-Kernbibliotheken für Java)
   - Jackson Annotations
   - Jackson Core
   - Jackson Databind
   - Jackson-Datenformat: CBOR
   - Joda Time
4. Abhängig von Ihrem IDE wird das Projekt möglicherweise automatisch erstellt. Wenn nicht, erstellen Sie das Projekt mit den entsprechenden Schritten für SieIDE.

Wenn Sie diese Schritte erfolgreich abgeschlossen haben, können Sie zum nächsten Abschnitt wechseln, the section called “Implementieren Sie den Hersteller”.

Nächste Schritte

Implementieren Sie den Hersteller

Dieses Tutorial verwendet das reale Szenario einer Überwachung des Wertpapierhandels. Die folgenden Prinzipien erläutern kurz, wie dieses Szenario zum Produzenten und seiner unterstützenden Codestruktur passt.

Beachten Sie den Quellcode und prüfen Sie die folgenden Informationen.

StockTrade Klasse

Ein einzelner Aktienhandel wird durch eine Instanz der StockTrade Klasse repräsentiert. Diese Instance enthält folgende Attribute: Tickersymbol, Preis, Anzahl der Anteile, Art des Handels (Kauf oder Verkauf) und ID zur eindeutigen Identifizierung der Handelsaktion. Dieser Klasse wird für Sie implementiert.

Stream-Datensatz

Ein Stream ist eine Sequenz von Datensätzen. Ein Datensatz ist eine Serialisierung einer `StockTrade` Instanz im JSON Format. Beispielsweise:

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

StockTradeGenerator Klasse

StockTradeGenerator hat eine Methode namens`getRandomTrade()`, die bei jedem Aufruf einen neuen zufällig generierten Aktienhandel zurückgibt. Dieser Klasse wird für Sie implementiert.

StockTradesWriter Klasse

Die `main` Methode des Herstellers ruft StockTradesWriter kontinuierlich einen zufälligen Trade ab und sendet ihn dann an Kinesis Data Streams, indem sie die folgenden Aufgaben ausführt:

1. Liest den Namen des Datenstreams und den Namen der Region als Eingabe.
2. Verwendet die`KinesisAsyncClientBuilder`, um die Region, die Anmeldeinformationen und die Client-Konfiguration festzulegen.

3. Sicherstellen, dass der Stream vorhanden und aktiv ist (wenn nicht, kommt es zu einer Beendigung mit Fehler).
4. Aufrufen der `StockTradeGenerator.getRandomTrade()`-Methode in einer Dauerschleife und anschließend Aufruf der `sendStockTrade`-Methode, um die Handelsdaten alle 100 Millisekunden an den Stream zu senden.

Die `sendStockTrade`-Methode der `StockTradesWriter`-Klasse hat den folgenden Code:

```
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);
        }
    }
```

Beachten Sie die folgende Code-Struktur:

- The `PutRecord` API erwartet ein Byte-Array, und Sie müssen den Handel in JSON ein Format konvertieren. Diese einzelne Codezeile führt die Operation aus:

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

- Bevor Sie die Transaktion senden können, erstellen Sie eine neue `PutRecordRequest`-Instance (in diesem Fall Anforderung genannt). Jede `request` benötigt den Namen des Streams, einen Partitionsschlüssel und einen Daten-Blob.

```
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();
```

  Das Beispiel verwendet einen Börsenticker als Partitionsschlüssel, der den Datensatz einem bestimmten Shard zuordnet. In der Praxis sollten Sie Hunderte oder gar Tausende von Partitionsschlüsseln pro Shard haben, sodass die Datensätze in Ihrem Stream gleichmäßig verteilt sind. Weitere Informationen zum Hinzufügen von Daten zu einem Stream finden Sie unter Daten in Amazon Kinesis Data Streams schreiben.

  `request` kann die Daten jetzt an den Client senden (PUT-Operation):

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

- Eine Fehlerüberprüfung und Protokollierung sind immer nützliche Ergänzungen. Dieser Code protokolliert Fehlerbedingungen:

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

  Platzieren Sie den try/catch-Block um die `put`-Operation herum:

```
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);
}
```

Der Grund besteht darin, dass eine Kinesis Data Streams-PUT-Operation aufgrund eines Netzwerkfehlers fehlschlagen kann oder gedrosselt wird, weil die Durchsatzgrenze des Streams erreicht wird. Es wird empfohlen, dass Sie Ihre Wiederholungsrichtlinie für `put` Operationen zur Vermeidung von Datenverlusten, wie z. B. die Verwendung eines Wiederholungsversuchs, sorgfältig prüfen.

- Eine Statusprotokollierung ist hilfreich, wenn auch optional:

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

Der hier gezeigte Hersteller verwendet die API Einzeldatensatzfunktion von Kinesis Data Streams,`PutRecord`. In der Praxis ist es oft effizienter, die Eignung von `PutRecords` für mehrere Datensätze zu nutzen und mehrere Datensatzstapel gleichzeitig zu senden, wenn ein Produzent viele Datensätze erstellt. Weitere Informationen finden Sie unter Daten in Amazon Kinesis Data Streams schreiben.

So führen Sie den Produzenten aus

1. Verifizieren Sie, dass der in Erstellen Sie eine IAM Richtlinie und einen Benutzer abgerufene Zugriffsschlüssel samt geheimem Schlüsselpaar in der Datei `~/.aws/credentials` gespeichert wurde.
2. Führen Sie die `StockTradeWriter`-Klasse mit den folgenden Argumenten aus:

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

Wenn Sie den Stream in einer anderen Region als `us-west-2` erstellt haben, müssen Sie stattdessen hier diese Region angeben.

Die Ausgabe sollte folgendermaßen oder ähnlich aussehen:

```
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
```

Ihre Wertpapierdaten werden nun von Kinesis Data Streams eingespeist.

Nächste Schritte

Implementieren Sie den Verbraucher

Implementieren Sie den Verbraucher

Die Verbraucheranwendung in diesem Tutorial verarbeitet die Wertpapiertransaktionen im Daten-Stream kontinuierlich. Sie gibt dann für jede Minute die beliebtesten Aktien aus, die gekauft und verkauft wurden. Die Anwendung basiert auf der Kinesis Client Library (KCL), die einen Großteil der Arbeit erledigt, die für Verbraucher-Apps üblich ist. Weitere Informationen finden Sie unter Verwenden Sie die Kinesis Client Library.

Überprüfen Sie die folgenden Informationen in Bezug auf den Quellcode.

StockTradesProcessor Klasse

Die für Sie bereitgestellte Hauptklasse des Verbrauchers, die die folgenden Aufgaben erfüllt:

- Liest die als Argumente übergebenen Anwendungs-, Datenstrom- und Regionsnamen.

- Erzeugt eine `KinesisAsyncClient` Instanz mit dem Namen der Region.
- Erstellt eine `StockTradeRecordProcessorFactory`-Instance für die Instances von `ShardRecordProcessor`, implementiert von einer `StockTradeRecordProcessor`-Instance.
- Erzeugt eine `ConfigsBuilder` Instanz mit der `StockTradeRecordProcessorFactory` Instanz `KinesisAsyncClient` `StreamNameApplicationName`,, und. Dies ist für das Erstellen aller Konfigurationen mit Standardwerten nützlich.
- Erstellt mit der Instanz einen KCL Scheduler (in KCL Version 1.x wurde er früher als KCL Worker bezeichnet). `ConfigsBuilder`
- Der Scheduler erstellt für jeden Shard (der dieser Verbraucher-Instance zugeordnet ist) einen neuen Thread, der in einer Schleife die Datensätze aus dem Daten-Stream liest. Anschließend wird die `StockTradeRecordProcessor`-Instance aufgerufen, um die empfangenen Datensatzstapel zu verarbeiten.

StockTradeRecordProcessor Klasse

Implementierung der `StockTradeRecordProcessor`-Instance, die wiederum fünf erforderliche Methoden implementiert: `initialize`, `processRecords`, `leaseLost`, `shardEnded` und `shutdownRequested`.

Die `shutdownRequested` Methoden `initialize` und werden von der verwendet, KCL um dem Aufzeichnungsprozessor mitzuteilen, wann er bereit sein sollte, mit dem Empfang von Datensätzen zu beginnen bzw. wann er damit rechnen sollte, keine Aufzeichnungen mehr zu empfangen, sodass er alle anwendungsspezifischen Einrichtungs- und Beendungsaufgaben ausführen kann. `leaseLost`und `shardEnded` werden verwendet, um eine beliebige Logik dafür zu implementieren, was zu tun ist, wenn ein Lease verloren geht oder eine Verarbeitung das Ende eines Shards erreicht hat. In diesem Beispiel protokollieren wir einfach Meldungen dieser Ereignisse.

Der Code für diese Methoden wird für Sie bereitgestellt. Die wesentliche Verarbeitung erfolgt mit der `processRecords` Methode, die wiederum `processRecord` für die einzelnen Datensätze nutzt. Die letztgenannte Methode wird als nahezu leerer Skeleton-Code bereitgestellt und im nächsten Schritt (der weitere Informationen enthält) implementiert.

Beachten Sie außerdem die Implementierung der Hilfsmethoden für `processRecord`: `reportStats` und `resetStats`, die im ursprünglichen Quellcode leer sind.

Die `processRecords`-Methode wurde für Sie implementiert und führt die folgenden Schritte aus:

- Für jeden übergebenen Datensatz wird `processRecord` aufgerufen.
- Ruft `reportStats()` zum Drucken der neuesten Statistiken auf, wenn seit dem letzten Bericht mindestens 1 Minute vergangen ist, und dann `resetStats()`, um die Statistiken zu löschen, damit das nächste Intervall nur neue Datensätze enthält.
- Legt den Zeitpunkt für die nächste Berichterstellung fest.
- Ruft `checkpoint()` auf, wenn seit dem letzten Prüfpunkt mindestens 1 Minute vergangen ist.
- Legt den Zeitpunkt für das nächste Checkpointing fest.

Diese Methode verwendet für das Checkpointing und die Berichterstellung ein Intervall von 60 Sekunden. Weitere Informationen zum Checkpointing finden Sie unter Verwendung der Kinesis Client Library.

StockStats Klasse

Diese Klasse stellt eine Datenaufbewahrung und eine Nachverfolgung von Statistiken für die beliebtesten Aktien bereit. Dieser Code wird für Sie bereitgestellt und enthält folgende Methoden:

- `addStockTrade(StockTrade)`: fügt die angegebene `StockTrade` in die ausgeführten Statistiken ein.
- `toString()`: gibt die Statistiken als formatierte Zeichenfolge zurück.

Diese Klasse verfolgt die beliebtesten Aktien, indem sie fortlaufend die Gesamtzahl der Trades für jede Aktie und deren maximale Anzahl zählt. Sie aktualisiert diese Werte, sobald eine neue Handelstransaktion empfangen wird.

Fügen Sie Code zu den Methoden der `StockTradeRecordProcessor`-Klasse hinzu, wie in den folgenden Schritten gezeigt.

So implementieren Sie den Konsumenten

1. Implementieren Sie die `processRecord`-Methode, indem Sie ein richtig bemessenes `StockTrade`-Objekt instanziieren und die Datensatzdaten zu diesem hinzufügen, sodass im Falle eines Problems eine Warnung protokolliert wird.

```
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. Implementieren Sie eine `reportStats` Methode. Ändern Sie das Ausgabeformat nach Ihren Wünschen.

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

3. Implementieren Sie die Methode `resetStats`, die eine neue `stockStats`-Instance erstellt.

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

4. Implementieren Sie die folgenden Methoden, die für die `ShardRecordProcessor` Schnittstelle erforderlich sind:

```
@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);
    }
}
```

So führen Sie den Konsumenten aus

1. Führen Sie den unter erstellten Produzenten aus, um simulierte Wertpapiertransaktionsdatensätze in den Stream zu schreiben.
2. Stellen Sie sicher, dass der Zugriffsschlüssel und das geheime key pair, das Sie zuvor (beim Erstellen des IAM Benutzers) abgerufen haben, in der Datei gespeichert sind`~/.aws/credentials`.
3. Führen Sie die `StockTradesProcessor`-Klasse mit den folgenden Argumenten aus:

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

   Beachten Sie, dass Sie, wenn Sie Ihren Stream in einer anderen Region als `us-west-2` erstellt haben, stattdiesen diese Region hier angeben müssen.

Nach einer Minute sollen Sie eine Ausgabe ähnlich der folgenden sehen, die anschließend einmal pro Minute aktualisiert wird:

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

Nächste Schritte

(Optional) Erweitern Sie den Consumer-Bereich

(Optional) Erweitern Sie den Consumer-Bereich

Dieser optionale Abschnitt zeigt, wie Sie den Konsumentencode erweitern können, um einem komplexeren Szenario gerecht zu werden.

Wenn Sie minütlich über die größten Verkaufsaufträge informiert werden möchten, können Sie die `StockStats`-Klasse an drei Stellen bearbeiten.

So erweitern Sie den Konsumenten

1. Fügen Sie neue Instance-Variablen hinzu:

```
// 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. Fügen Sie folgenden Code zu hinz `addStockTrade`:

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

```
 }
```

3. Ändern Sie die `toString`-Methode, um die zusätzlichen Informationen zu drucken:

```
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);
}
```

Wenn Sie den Konsumenten jetzt ausführen (führen Sie auch den Produzenten), sollten Sie eine Ausgabe ähnlich der folgenden sehen:

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

Nächste Schritte

Bereinigen von -Ressourcen

Bereinigen von -Ressourcen

Da Sie für die Nutzung des Kinesis Data Streams zahlen, sollten Sie diesen ebenso wie die entsprechende Amazon-DynamoDB-Tabelle löschen, wenn sie nicht mehr benötigt werden. Für einen aktiven Stream fallen auch dann nominale Gebühren an, wenn Sie keine Datensätze senden oder abrufen. Der Grund hierfür ist, dass ein aktiver Stream durch kontinuierlichen „Horchen“ darauf,

ob neue Datensätze oder Anforderungen zum Abrufen von Datensätzen eingehen, Ressourcen verbraucht.

So löschen Sie den Stream und die Tabelle

1. Schalten Sie alle Produzenten und Verbraucher aus, die Sie möglicherweise noch in Betrieb haben.
2. Öffnen Sie die Kinesis-Konsole unter https://console.aws.amazon.com/kinesis.
3. Wählen Sie den Stream aus, den Sie für diese Anwendung erstellt haben (`StockTradeStream`).
4. Wählen Sie Delete Stream (Stream löschen) aus.
5. Öffnen Sie die DynamoDB-Konsole unter. https://console.aws.amazon.com/dynamodb/
6. Löschen Sie die `StockTradesProcessor`-Tabelle.

Übersicht

Die Verarbeitung einer großen Datenmenge nahezu in Echtzeit erfordert weder das Schreiben von kompliziertem Code noch die Entwicklung einer riesigen Infrastruktur. Es ist genauso einfach wie die Schreiblogik, eine kleine Datenmenge zu verarbeiten (wie Schreiben`processRecord(Record)`), aber Kinesis Data Streams zu verwenden, um so zu skalieren, dass es für eine große Menge an Streaming-Daten funktioniert. Sie müssen sich keine Gedanken über die Skalierung der Verarbeitung machen, da Kinesis Data Streams das für Sie übernimmt. Sie müssen lediglich Ihre Streaming-Datensätze an Kinesis Data Streams senden und eine Logik für die Verarbeitung neu empfangener Datensätze schreiben.

Nachfolgend einige mögliche Erweiterungen für diese Anwendung.

Shard-übergreifende Aggregation

> Derzeit erhalten Sie Statistiken, die aus der Aggregation von Datensätzen resultieren, die von einem einzelnen Auftragnehmer eines einzelnen Shards empfangen werden. (Ein Shard kann jeweils nur von einem Auftragnehmer in einer einzelnen Anwendung verarbeitet werden.) Möglicherweise möchten Sie, wenn Sie eine Skalierung durchführen und mehr als einen Shard haben, eine Shard-übergreifende Aggregation vornehmen. Dies kann mit einer Pipeline-Architektur realisiert werden, bei der die Ausgabe eines jeden Auftragnehmers in einen anderen Stream mit einem einzelnen Shard eingespeist wird, der von einem Auftragnehmer verarbeitet wird, der die Ausgaben der ersten Phase aggregiert. Da die Daten aus der ersten Phase

begrenzt sind (eine Ausgabe pro Minute pro Shard), können sie problemlos von nur einem Shard verarbeitet werden.

Skalierung

Wenn der Stream skaliert wird, damit mehr Shards zur Verfügung stehen (da viele Produzenten Daten senden), geschieht dies dadurch, dass mehr Worker hinzugefügt werden. Sie können die Worker in EC2 Amazon-Instances ausführen und Auto Scaling Scaling-Gruppen verwenden.

Verwenden Sie Konnektoren zu Amazon S3/DynamoDB/Amazon Redshift/Storm

Da ein Stream kontinuierlich verarbeitet wird, kann seine Ausgabe an andere Ziele gesendet werden. AWS bietet Konnektoren zur Integration von Kinesis Data Streams mit anderen AWS Diensten und Tools von Drittanbietern.

Tutorial: Verarbeiten Sie Aktiendaten in Echtzeit mit KPL und KCL 1.x

Im Szenario dieses Tutorials werden Wertpapierdaten in einen Datenstrom geschrieben. Zudem wird eine einfache Anwendung mit Amazon Kinesis Data Streams erstellt, die Berechnungen mit dem Stream durchführt. Sie lernen, wie Sie einen Stream von Datensätzen an Kinesis Data Streams senden und eine Anwendung implementieren, die die Datensätze nahezu in Echtzeit verarbeitet und verarbeitet.

Important

Nachdem Sie einen Stream erstellt haben, fallen für Ihr Konto geringe Gebühren für die Nutzung von Kinesis Data Streams an, da Kinesis Data Streams nicht für das AWS kostenlose Kontingent in Frage kommt. Nach dem Start der Konsumentenanwendung fallen nominale Gebühren für die Amazon-DynamoDB-Nutzung an. Die Konsumentenanwendung verwendet DynamoDB zum Verfolgen des Verarbeitungsstatus. Wenn Sie mit dieser Anwendung fertig sind, sollten Sie Ihre AWS -Ressourcen löschen, damit keine weiteren Gebühren anfallen. Weitere Informationen finden Sie unter Bereinigen von -Ressourcen.

Der Code greift nicht auf tatsächliche Wertpapierdaten zu, sondern simuliert nur deren Strom. Dazu werden zufällige Wertpapierdaten erzeugt. Ausgangspunkt sind dabei echte Marktdaten der 25 führenden Aktien gemäß Börsenkapitalisierung von Februar 2015. Wenn Sie Zugriff auf einen

Echtzeit-Stream von Wertpapierdaten haben, möchten Sie vermutlich nützliche, zeitnahe Statistiken aus den Stream-Daten erzeugen. Sie können beispielsweise eine Zeitfensteranalyse durchführen, um festzustellen, welche Aktie in den letzten 5 Minuten am häufigsten erworben wurde. Oder Sie möchten im Falle eines zu großen Verkaufsauftrags (d. h. zu viele Anteile) benachrichtigt werden. Der Code in diesem Tutorial kann erweitert werden, um solche Funktionen bereitzustellen.

Sie können die Schritte in diesem Tutorial auf Ihrem Desktop- oder Laptop-Computer durcharbeiten und sowohl den Producer- als auch den Consumer-Code auf demselben Computer oder auf einer beliebigen Plattform ausführen, die die definierten Anforderungen unterstützt, z. B. Amazon Elastic Compute Cloud (AmazonEC2).

Bei den gezeigten Beispielen wird die Region USA West (Oregon) verwendet. Sie funktionieren aber auch für alle anderen AWS -Regionen, die Kinesis Data Streams unterstützen.

Aufgaben

- Erfüllen der Voraussetzungen
- Erstellen Sie einen Datenstream
- Erstellen Sie eine IAM Richtlinie und einen Benutzer
- Laden Sie den Implementierungscode herunter und erstellen Sie ihn
- Implementieren Sie den Hersteller
- Implementieren Sie den Verbraucher
- (Optional) Erweitern Sie die Zahl der Verbraucher
- Bereinigen von -Ressourcen

Erfüllen der Voraussetzungen

Für Tutorial: Verarbeiten Sie Aktiendaten in Echtzeit mit KPL und KCL 1.x gelten die folgenden Anforderungen.

Erstellen und verwenden Sie ein Amazon Web Services Services-Konto

Bevor Sie beginnen, müssen Sie sich mit den in Terminologie und Konzepte von Amazon Kinesis Data Streams behandelten Konzepten vertraut machen, insbesondere mit Streams, Shards, Produzenten und Konsumenten. Das Durcharbeiten von Tutorial: Installation und Konfiguration von AWS CLI for Kinesis Data Streams ist ebenfalls hilfreich.

Sie benötigen ein AWS Konto und einen Webbrowser, um auf das zugreifen zu können AWS Management Console.

Verwenden Sie für den Konsolenzugriff Ihren IAM Benutzernamen und Ihr Passwort, um sich von der Anmeldeseite AWS Management Consoleaus auf der IAM Anmeldeseite anzumelden. Informationen zu AWS Sicherheitsanmeldedaten, einschließlich programmatischem Zugriff und Alternativen zu langfristigen Anmeldeinformationen, finden Sie unter AWS Sicherheitsanmeldedaten im IAMBenutzerhandbuch. Einzelheiten zur Anmeldung bei Ihrem AWS-Konto finden Sie unter So melden Sie sich an AWS im AWS-Anmeldung Benutzerhandbuch.

Weitere Informationen IAM und Anweisungen zur Einrichtung eines Sicherheitsschlüssels finden Sie unter IAM Benutzer erstellen.

Erfüllen Sie die Anforderungen an die Systemsoftware

Auf dem System, dass die Anwendung ausführt, muss Java 7 oder höher installiert sein. Um das neueste Java Development Kit (JDK) herunterzuladen und zu installieren, gehen Sie zur Java SE-Installationsseite von Oracle.

Wenn Sie über ein IDE Java-Betriebssystem wie Eclipse verfügen, können Sie den Quellcode öffnen, bearbeiten, erstellen und ausführen.

Sie benötigen die neueste AWS SDK for Java-Version. Wenn Sie Eclipse als Ihr Gerät verwendenIDE, können Sie stattdessen das AWS Toolkit for Eclipse installieren.

Die Consumer-Anwendung benötigt die Kinesis Client Library (KCL) Version 1.2.1 oder höher, die Sie GitHub unter Kinesis Client Library (Java) beziehen können.

Nächste Schritte

Erstellen Sie einen Datenstream

Erstellen Sie einen Datenstream

Im ersten Schritt von Tutorial: Verarbeiten Sie Aktiendaten in Echtzeit mit KPL und KCL 1.x erstellen Sie den Stream, der in den weiteren Schritten genutzt wird.

So erstellen Sie einen Stream

1. Melden Sie sich bei der an AWS Management Console und öffnen Sie die Kinesis-Konsole unter https://console.aws.amazon.com/kinesis.

2. Klicken Sie im Navigationsbereich auf Data Streams (Daten-Streams).
3. Erweitern Sie in der Navigationsleiste die Regionsauswahl und wählen Sie eine Region aus.
4. Wählen Sie Create Kinesis Stream (Kinesis-Stream erstellen).
5. Geben Sie einen Namen für Ihren Stream ein (z. B. **`StockTradeStream`**).
6. Geben Sie **1** die Anzahl der Shards ein, lassen Sie aber Estimate the number of shards you'll need (Schätzung der Anzahl der Shards, die Sie benötigen) reduziert.
7. Wählen Sie Create Kinesis Stream (Kinesis-Stream erstellen).

Auf der Seite mit der Auflistung der Kinesis Streams ist der Status Ihres Streams während des Erstellens auf `CREATING` gesetzt. Sobald der Stream verwendet werden kann, ändert sich der Status in `ACTIVE`. Wählen Sie den Namen des Streams aus. Auf der angezeigten Seite zeigt die Registerkarte Details eine Zusammenfassung Ihrer Streams-Konfiguration an. Der Abschnitt Monitoring (Überwachung) zeigt Überwachungsinformationen für den Stream an.

Zusätzliche Informationen zu Shards

Wenn Sie Kinesis Data Streams außerhalb dieses Tutorials verwenden, sollten Sie das Erstellen des Streams sorgfältiger planen. Bei der Bereitstellung von Shards sollten Sie vom höchsten erwarteten Bedarf ausgehen. In unserem Beispielszenario kommt es an der US-Börse tagsüber zu Datenverkehrsspitzen (Eastern Time). Der Bedarf sollte so geschätzt werden, dass diesen Verkehrsspitzen Rechnung getragen wird. Sie können dann entweder für den höchst möglichen Bedarf vorsorgen oder Ihren Stream an die Schwankungen anpassen.

Ein Shard ist eine Einheit für die Durchsatzkapazität. Erweitern Sie auf der Seite Kinesis Stream erstellen Schätzen Sie die Anzahl der Shards, die Sie brauchen. Geben Sie entsprechend der folgenden Informationen die durchschnittliche Datensatzgröße, die maximale Anzahl der Datensätze, die pro Sekunde geschrieben werden, und die Anzahl der Konsumentenanwendungen an:

Durchschnittliche Datensatzgröße

Eine Schätzung der berechneten durchschnittlichen Größe Ihrer Datensätze. Wenn Sie diesen Wert nicht kennen, verwenden Sie die geschätzte maximale Datensatzgröße.

Max. geschriebene Datensätze

Berücksichtigen Sie die Anzahl der Entitäten, die Daten bereitstellen, und die ungefähre Anzahl der jeweils erstellten Datensätze pro Sekunde. Beispiel: Wenn Sie Börsendaten von

20 Handelsservern erhalten und jeder pro Sekunde 250 Handelsabschlüsse generiert, beträgt die Anzahl der Handelsabschlüsse (Datensätze) 5.000/Sekunde.

Anzahl der Konsumentenanwendungen

Die Anzahl der Anwendungen, die unabhängig voneinander Daten aus dem Stream auslesen, die Stream-Daten unterschiedlich verarbeiten und unterschiedliche Ausgaben generieren. Es können mehrere Instances einer Anwendung auf verschiedenen Rechnern (d. h. in einem Cluster) ausgeführt werden, sodass ein großer Daten-Stream verarbeitet werden kann.

Wenn der gezeigte Schätzwert der Shards das aktuelle Shard-Limit übersteigt, müssen Sie ggf. eine Erhöhung des Limits beantragen, bevor Sie einen Stream mit dieser Anzahl an Shards erstellen können. Nutzen Sie zum Beantragen einer Erhöhung des Shard-Limits das Formular für Limits der Kinesis Data Streams. Weitere Informationen zu Streams und Shards finden Sie unter Kinesis-Datenstreams erstellen und verwalten.

Nächste Schritte

Erstellen Sie eine IAM Richtlinie und einen Benutzer

Erstellen Sie eine IAM Richtlinie und einen Benutzer

Bewährte Sicherheitsmethoden AWS schreiben die Verwendung detaillierter Berechtigungen vor, um den Zugriff auf verschiedene Ressourcen zu kontrollieren. AWS Identity and Access Management (IAM) ermöglicht Ihnen die Verwaltung von Benutzern und Benutzerberechtigungen in. AWS In einer IAMRichtlinie werden die zulässigen Aktionen und die Ressourcen, für die die Aktionen gelten, explizit aufgeführt.

Im Folgenden werden die Berechtigungen für einen Produzenten und Konsumenten von Kinesis Data Streams aufgelistet, die mindestens erforderlich sind.

Produzent

Aktionen	Ressource	Zweck
`DescribeStream` , `DescribeStreamSummary` , `DescribeStreamConsumer`	Kinesis Data Streams	Bevor er versucht, Datensätze zu schreiben, prüft der Produ vorhanden und aktiv ist, ob die Shards im Stream enthalten einen Consumer hat.

Aktionen	Ressource	Zweck
`SubscribeToShard` , `RegisterStreamConsumer`	Kinesis Data Streams	Abonnieren und Registrieren eines Konsumenten bei einem Shard.
`PutRecord` , `PutRecords`	Kinesis Data Streams	Schreiben von Datensätzen in Kinesis Data Streams.

Konsument

Aktionen	Ressource	Zweck
`DescribeStream`	Kinesis Data Streams	Vor dem Versuch, Daten zu lesen, prüft der Konsument, ob und aktiv ist und ob Shards im Stream enthalten sind.
`GetRecords` , `GetShardIterator`	Kinesis Data Streams	Lesen von Datensätzen aus einem Shard von Kinesis Data
`CreateTable` , `DescribeTable` , `GetItem`, `PutItem`, `Scan`, `UpdateItem`	Amazon-DynamoDB-Tabelle.	Wenn der Consumer mit der Kinesis Client Library (KCL) er er Berechtigungen für eine DynamoDB-Tabelle, um den Ve Anwendung verfolgen zu können. Der erste gestartete Kons Tabelle.
`DeleteItem`	Amazon-DynamoDB-Tabelle.	Für den Fall, dass der Konsument Split/Merge-Operationen Kinesis Data Streams ausführt.
`PutMetricData`	CloudWatch Amazon-Protokoll	KCLEs lädt auch Metriken hoch CloudWatch, die für die Üb Anwendung nützlich sind.

Für diese Anwendung erstellen Sie eine einzige IAM Richtlinie, die alle zuvor genannten Berechtigungen gewährt. In der Praxis empfiehlt es sich möglicherweise, zwei Richtlinien zu erstellen: eine für Produzenten und eine für Konsumenten.

So erstellen Sie eine IAM-Richtlinie

1. Suchen Sie den Amazon-Ressourcennamen (ARN) für den neuen Stream. Sie finden ihn oben ARN auf der Registerkarte „Details“ als Stream. ARN Das ARN Format ist wie folgt:

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

   Region

   Der Regionscode, beispielsweise `us-west-2`. Weitere Informationen finden Sie unter Regionen und Verfügbarkeitskonzepte.

   Konto

   Die AWS Konto-ID, wie in den Kontoeinstellungen angezeigt.

   Name

   Der Name des Streams von Erstellen Sie einen Datenstream, hier `StockTradeStream`.
2. Ermitteln Sie ARN die DynamoDB-Tabelle, die vom Consumer verwendet (und von der ersten Consumer-Instance erstellt) werden soll. Er muss das folgende Format aufweisen:

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

   Region und Konto stammen vom selben Ort wie im vorherigen Schritt. Hier ist der Name jedoch der Name der Tabelle, die von der Verbraucheranwendung erstellt und verwendet wird. Die vom Verbraucher KCL verwendete Datei verwendet den Namen der Anwendung als Tabellennamen. Verwenden Sie `StockTradesProcessor`, dies ist der Anwendungsname, der später genutzt wird.
3. Wählen Sie in der IAM Konsole unter Policies (https://console.aws.amazon.com/iam/home#policies) die Option Create policy aus. Wenn Sie zum ersten Mal mit IAM Richtlinien arbeiten, wählen Sie Erste Schritte, Richtlinie erstellen aus.
4. Wählen Sie Select (Auswählen) neben Policy Generator (Richtliniengenerator) aus.
5. Wählen Sie Amazon Kinesis als AWS Service.
6. Legen Sie `DescribeStream`, `GetShardIterator`, `GetRecords`, `PutRecord` und `PutRecords` als zulässige Aktionen fest.
7. Geben Sie den einARN, den Sie in Schritt 1 erstellt haben.
8. Verwenden Sie Add Statement (Statement hinzufügen) für die folgenden Elemente:

AWS Dienst	Aktionen	ARN
Amazon-DynamoDB	`CreateTable` , `DeleteItem` , `DescribeTable` , `GetItem`, `PutItem`, `Scan`, `UpdateItem`	DerARN, den Sie in Schritt 2 erstellt haben
Amazon CloudWatch	`PutMetricData`	*

Das Sternchen (*), das bei der Angabe von verwendet wird, ARN ist nicht erforderlich. In diesem Fall liegt das daran, dass es keine bestimmte Ressource gibt, für die die `PutMetricData` Aktion aufgerufen wird. CloudWatch

9. Wählen Sie Next Step (Weiter) aus.
10. Ändern Sie Policy Name (Richtlinienname) in `StockTradeStreamPolicy`, prüfen Sie den Code und wählen sie Create Policy (Richtlinie erstellen) aus.

Das erstellte Richtliniendokument sollte etwa wie folgt aussehen:

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

```
      ]
    },
    {
      "Sid": "Stmt234",
      "Effect": "Allow",
      "Action": [
        "kinesis:SubscribeToShard",
        "kinesis:DescribeStreamConsumer"
      ],
      "Resource": [
        "arn:aws:kinesis:us-west-2:123:stream/StockTradeStream/*"
      ]
    },
    {
      "Sid": "Stmt456",
      "Effect": "Allow",
      "Action": [
        "dynamodb:*"
      ],
      "Resource": [
        "arn:aws:dynamodb:us-west-2:123:table/StockTradesProcessor"
      ]
    },
    {
      "Sid": "Stmt789",
      "Effect": "Allow",
      "Action": [
        "cloudwatch:PutMetricData"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

So erstellen Sie einen IAM-Benutzer

1. Öffnen Sie die IAM Konsole unter. https://console.aws.amazon.com/iam/
2. Wählen Sie auf der Seite Users (Benutzer) die Option Add user (Benutzer hinzufügen) aus.
3. Geben Sie für User name `StockTradeStreamUser` ein.

4. Wählen Sie für Access type (Zugriffstyp) die Option Programmatic access (Programmgesteuerter Zugriff) und wählen Sie dann Next: Permissions (Weiter: Berechtigungen).
5. Wählen Sie Vorhandene Richtlinien direkt zuordnen.
6. Suche die von Ihnen erstellte Richtlinie dem Namen nach. Markieren Sie das Kontrollkästchen links neben dem Richtliniennamen und wählen Sie dann Next: Review (Weiter: Prüfen).
7. Überprüfen Sie die Details und die Zusammenfassung und wählen Sie dann Create user (Benutzer erstellen) aus.
8. Kopieren Sie die Access key ID (Zugriffsschlüssel-ID) und speichern Sie sie privat. Wählen Sie unter Secret access key (Geheimer Zugriffsschlüssel) die Option Show (Anzeigen) und speichern Sie auch diesen Schlüssel privat.
9. Fügen Sie die Zugriffs- und Geheimschlüssel in eine lokalen Datei an einem sichern Ort ein, auf den nur Sie Zugriff haben. Erstellen Sie für diese Anwendung eine Datei namens `~/.aws/credentials` (mit strikten Berechtigungen). Die Datei sollte das folgende Format aufweisen:

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

Um einem Benutzer eine IAM Richtlinie zuzuweisen

1. Öffnen Sie in der IAM Konsole Richtlinien und wählen Sie Richtlinienaktionen aus.
2. Klicken Sie auf `StockTradeStreamPolicy` und Attach (Verknüpfen).
3. Wählen Sie `StockTradeStreamUser` und Attach Policy (Richtlinie anfügen) aus.

Nächste Schritte

Laden Sie den Implementierungscode herunter und erstellen Sie ihn

Laden Sie den Implementierungscode herunter und erstellen Sie ihn

Für the section called “Tutorial: Verarbeiten Sie Aktiendaten in Echtzeit mit KPL und KCL 1.x” wird Skeleton-Code bereitgestellt. Er enthält eine Stub-Implementierung für die Übernahme des Wertpapierdaten-Streams (Produzent) und die Verarbeitung der Daten (Verbraucher). Das folgende Verfahren zeigt, wie die Implementierung abgeschlossen wird.

So laden und erstellen Sie den Implementierungscode

1. Laden Sie den Quellcode auf den Computer herunter.
2. Erstellen Sie ein Projekt in Ihrem Favoriten IDE mit dem Quellcode und halten Sie sich dabei an die bereitgestellte Verzeichnisstruktur.
3. Fügen Sie dem Projekt die folgenden Bibliotheken hinzu:
   - Amazon Kinesis Kinesis-Clientbibliothek () KCL
   - AWS SDK
   - Apache HttpCore
   - Apache HttpClient
   - Apache Commons Lang
   - Apache Commons Logging
   - Guava (Google-Kernbibliotheken für Java)
   - Jackson Annotations
   - Jackson Core
   - Jackson Databind
   - Jackson-Datenformat: CBOR
   - Joda Time
4. Abhängig von Ihrem IDE wird das Projekt möglicherweise automatisch erstellt. Wenn nicht, erstellen Sie das Projekt mit den entsprechenden Schritten für SieIDE.

Wenn Sie diese Schritte erfolgreich abgeschlossen haben, können Sie zum nächsten Abschnitt wechseln, the section called “Implementieren Sie den Hersteller”. Wenn der Build Fehler erzeugt, müssen Sie diese untersuchen und beheben, bevor Sie fortfahren.

Nächste Schritte

Implementieren Sie den Hersteller

Die Anwendung im Tutorial: Verarbeiten Sie Aktiendaten in Echtzeit mit KPL und KCL 1.x verwendet das reale Szenario einer Überwachung des Wertpapierhandels. Im Folgenden wird kurz erläutert, wie dieses Szenario zum Produzenten und der unterstützenden Codestruktur zugeordnet wird.

Überprüfen Sie die folgenden Informationen in Bezug auf den Quellcode.

StockTrade Klasse

Ein bestimmter Wertpapierhandel wird durch eine Instance der `StockTrade`-Klasse dargestellt. Diese Instance enthält folgende Attribute: Tickersymbol, Preis, Anzahl der Anteile, Art des Handels (Kauf oder Verkauf) und ID zur eindeutigen Identifizierung der Handelsaktion. Dieser Klasse wird für Sie implementiert.

Stream-Datensatz

Ein Stream ist eine Sequenz von Datensätzen. Ein Datensatz ist eine Serialisierung einer `StockTrade` Instanz im JSON Format. Beispielsweise:

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

StockTradeGenerator Klasse

`StockTradeGenerator` verfügt über eine Methode namens `getRandomTrade()`, die bei Aufruf Daten eines zufällig generierten Wertpapierhandels zurückgibt. Dieser Klasse wird für Sie implementiert.

StockTradesWriter Klasse

Die `main`-Methode des Produzenten, `StockTradesWriter`, ruft kontinuierlich eine zufällige Handelsaktion ab und sendet die Daten an Kinesis Data Streams, indem sie die folgenden Aufgaben durchführt:

1. Lesen des Stream- und Regionsnamen als Eingabe.
2. Erstellen eines `AmazonKinesisClientBuilder`.
3. Verwenden des Client Builder, um die Region, die Anmeldeinformationen und die Client-Konfiguration festzulegen.
4. Erstellen eines `AmazonKinesis`-Clients mit dem Client-Builder.
5. Sicherstellen, dass der Stream vorhanden und aktiv ist (wenn nicht, kommt es zu einer Beendigung mit Fehler).

6. Aufrufen der `StockTradeGenerator.getRandomTrade()`-Methode in einer Dauerschleife und anschließend Aufruf der `sendStockTrade`-Methode, um die Handelsdaten alle 100 Millisekunden an den Stream zu senden.

Die `sendStockTrade`-Methode der `StockTradesWriter`-Klasse hat den folgenden Code:

```
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);
    }
}
```

Beachten Sie die folgende Code-Struktur:

- Die `PutRecord` API erwartet ein Byte-Array, und Sie müssen es in ein JSON Format `trade` konvertieren. Diese einzelne Codezeile führt die Operation aus:

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

- Bevor Sie die Handelsdaten senden können, erstellen Sie eine neue `PutRecordRequest`-Instance (namens `putRecord` in diesem Fall):

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

Jeder `PutRecord`-Aufruf erfordert den Namen des Streams, einen Partitionsschlüssel und einen Daten-Blob. Der folgende Code füllt diese Felder im `putRecord`-Objekt mit dessen `setXxxx()`-Methoden:

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

Im Beispiel wird ein Stock Ticket als Partitionsschlüssel verwendet, wodurch der Datensatz einem bestimmten Shard zugeordnet wird. In der Praxis sollten Sie Hunderte oder gar Tausende von Partitionsschlüsseln pro Shard haben, sodass die Datensätze in Ihrem Stream gleichmäßig verteilt sind. Weitere Informationen zum Hinzufügen von Daten zu einem Stream finden Sie unter Daten zu einem Stream hinzufügen.

`putRecord` ist nun für das Senden an den Client bereit (`put`-Operation):

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

- Eine Fehlerüberprüfung und Protokollierung sind immer nützliche Ergänzungen. Dieser Code protokolliert Fehlerbedingungen:

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

Platzieren Sie den try/catch-Block um die `put`-Operation herum:

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

Der Grund besteht darin, dass eine Kinesis-Data-Streams-`put`-Operation aufgrund eines Netzwerkfehlers fehlschlagen kann oder gedrosselt wird, weil die Durchsatzgrenze des Streams erreicht wird. Wir empfehlen, Ihre Wiederholungsrichtlinie für `put` Operationen

sorgfältig zu überdenken, um Datenverlust zu vermeiden, z. B. die Verwendung eines Wiederholungsversuchs.

- Eine Statusprotokollierung ist hilfreich, wenn auch optional:

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

Der hier gezeigte Hersteller verwendet die API Einzeldatensatzfunktion von Kinesis Data Streams,`PutRecord`. In der Praxis ist es oft effizienter, die Eignung von `PutRecords` für mehrere Datensätze zu nutzen und mehrere Datensatzstapel gleichzeitig zu senden, wenn ein Produzent viele Datensätze erstellt. Weitere Informationen finden Sie unter Daten zu einem Stream hinzufügen.

So führen Sie den Produzenten aus

1. Stellen Sie sicher, dass der Zugriffsschlüssel und das geheime key pair, das Sie zuvor (beim Erstellen des IAM Benutzers) abgerufen haben, in der Datei gespeichert sind`~/.aws/credentials`.
2. Führen Sie die `StockTradeWriter`-Klasse mit den folgenden Argumenten aus:

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

Wenn Sie Ihren Stream in einer anderen Region als `us-west-2` erstellt haben, müssen Sie stattdiesen hier diese Region angeben.

Die Ausgabe sollte folgendermaßen oder ähnlich aussehen:

```
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
```

Ihr Stream für die Wertpapierdaten wird nun von Kinesis Data Streams eingespeist.

Nächste Schritte

Implementieren Sie den Verbraucher

Implementieren Sie den Verbraucher

Die Konsumentenanwendung in Tutorial: Verarbeiten Sie Aktiendaten in Echtzeit mit KPL und KCL 1.x verarbeitet kontinuierlich den in erstellten Stream mit Wertpapiertransaktionen. Sie gibt dann für jede Minute die beliebtesten Aktien aus, die gekauft und verkauft wurden. Die Anwendung basiert auf der Kinesis Client Library (KCL), die einen Großteil der Arbeit erledigt, die für Verbraucher-Apps üblich ist. Weitere Informationen finden Sie unter Entwickeln Sie KCL 1.x-Verbraucher.

Überprüfen Sie die folgenden Informationen in Bezug auf den Quellcode.

StockTradesProcessor Klasse

Hauptklasse des Konsumenten, die für Sie bereitgestellt wird und folgende Aufgaben übernimmt:

- Liest die Namen von Anwendung, Stream und Region, die als Argumente übergeben werden
- Liest Anmeldeinformationen aus `~/.aws/credentials`
- Erstellt eine `RecordProcessorFactory`-Instance für die Instances von `RecordProcessor`, implementiert von einer `StockTradeRecordProcessor`-Instance.
- Erstellt einen KCL Worker mit der `RecordProcessorFactory` Instanz und einer Standardkonfiguration, einschließlich des Stream-Namens, der Anmeldeinformationen und des Anwendungsnamens.
- Die Worker erstellt für jeden Shard (der dieser Konsumenten-Instance zugeordnet ist) einen neuen Thread, der die Datensätze in einer Schleife aus Kinesis Data Streams liest. Anschließend wird die `RecordProcessor`-Instance aufgerufen, um die empfangenen Datensatzstapel zu verarbeiten.

StockTradeRecordProcessor Klasse

Implementierung der `RecordProcessor`-Instance, die wiederum drei erforderliche Methoden implementiert: `initialize`, `processRecords` und `shutdown`.

Wie an den Namen zu erkennen ist, werden `initialize` und `shutdown` von der Kinesis Client Library verwendet, um dem Datensatzprozessor mitzuteilen, wann er mit dem Empfang

von Datensätzen beginnen bzw. wann er diesen stoppen soll, sodass er entsprechende anwendungsspezifische Einrichtungs- und Beendigungsaufgaben ausführen kann. Der Code hierfür wird für Sie bereitgestellt. Die wesentliche Verarbeitung erfolgt mit der `processRecords` Methode, die wiederum `processRecord` für die einzelnen Datensätze nutzt. Die letztgenannte Methode wird als nahezu leeres Code-Skelett bereitgestellt, das im nächsten Schritt näher erläutert und von Ihnen implementiert wird.

Darüber hinaus hervorzuheben ist die Implementierung von Support-Methoden für `processRecord`: `reportStats` und `resetStats`, die im ursprünglichen Quellcode leer sind.

Die `processRecords`-Methode wurde für Sie implementiert und führt die folgenden Schritte aus:

- Ruft `processRecord` für jeden übergebenen Datensatz auf.
- Ruft `reportStats()` zum Drucken der neuesten Statistiken auf, wenn seit dem letzten Bericht mindestens 1 Minute vergangen ist, und dann `resetStats()`, um die Statistiken zu löschen, damit das nächste Intervall nur neue Datensätze enthält.
- Legt den Zeitpunkt für die nächste Berichterstellung fest.
- Ruft `checkpoint()` auf, wenn seit dem letzten Prüfpunkt mindestens 1 Minute vergangen ist.
- Legt den Zeitpunkt für das nächste Checkpointing fest.

Diese Methode verwendet für das Checkpointing und die Berichterstellung ein Intervall von 60 Sekunden. Weitere Informationen zum Checkpointing finden Sie unter Zusätzliche Informationen über den Verbraucher.

StockStats Klasse

Diese Klasse stellt eine Datenaufbewahrung und eine Nachverfolgung von Statistiken für die beliebtesten Aktien bereit. Dieser Code wird für Sie bereitgestellt und enthält folgende Methoden:

- `addStockTrade(StockTrade)`: Fügt den angegebenen `StockTrade` in die ausgeführten Statistiken ein.
- `toString()`: Gibt die Statistiken in einer formatierten Zeichenfolge zurück.

Diese Klasse verfolgt die beliebtesten Aktien, indem sie fortlaufend die Gesamtzahl der Trades für jede Aktie und deren maximale Anzahl zählt. Sie aktualisiert diese Werte, sobald eine neue Handelstransaktion empfangen wird.

Fügen Sie Code zu den Methoden der `StockTradeRecordProcessor`-Klasse hinzu, wie in den folgenden Schritten gezeigt.

So implementieren Sie den Konsumenten

1. Implementieren Sie die `processRecord`-Methode, indem Sie ein richtig bemessenes `StockTrade`-Objekt instanziieren und die Datensatzdaten zu diesem hinzufügen, sodass im Falle eines Problems eine Warnung protokolliert wird.

```
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. Implementieren Sie eine einfache `reportStats`-Methode. Sie können das Ausgabeformat an Ihre Bedürfnisse anpassen.

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

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

3. Implementieren Sie abschließend die `resetStats`-Methode, die eine neue `stockStats`-Instance erstellt.

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

So führen Sie den Konsumenten aus

1. Führen Sie den unter erstellten Produzenten aus, um simulierte Wertpapiertransaktionsdatensätze in den Stream zu schreiben.
2. Stellen Sie sicher, dass der Zugriffsschlüssel und das geheime key pair, das Sie zuvor (beim Erstellen des IAM Benutzers) abgerufen haben, in der Datei gespeichert sind`~/.aws/credentials`.
3. Führen Sie die `StockTradesProcessor`-Klasse mit den folgenden Argumenten aus:

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

Beachten Sie, dass Sie, wenn Sie Ihren Stream in einer anderen Region als `us-west-2` erstellt haben, stattdiesen diese Region hier angeben müssen.

Nach einer Minute sollen Sie eine Ausgabe ähnlich der folgenden sehen, die anschließend einmal pro Minute aktualisiert wird:

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

Zusätzliche Informationen über den Verbraucher

Wenn Sie mit den Vorteilen der Kinesis Client Library vertraut sind, die unter Entwickeln Sie KCL 1.x-Verbraucher und anderswo erörtert werden, fragen Sie sich möglicherweise, warum Sie sie hier nutzen sollten. Sie verwenden zwar nur einen einzigen Shard-Stream und eine einzige Consumer-Instance, um ihn zu verarbeiten, aber es ist immer noch einfacher, den Consumer mithilfe von zu implementieren. KCL Vergleichen Sie die Implementierungsschritte im Produzentenabschnitt mit denen für den Konsumenten und Sie werden feststellen, dass die Implementierung eines Konsumenten vergleichsweise einfach ist. Dies ist hauptsächlich auf die Dienste zurückzuführen, die der KCL anbietet.

In dieser Anwendung konzentrieren Sie sich auf die Implementierung einer Datensatzprozessor-Klasse, die einzelne Datensätze verarbeiten kann. Sie müssen sich keine Gedanken darüber machen, wie die Datensätze aus Kinesis Data Streams abgerufen werden. Der KCL ruft die Datensätze ab und ruft den Datensatzprozessor auf, wenn neue Datensätze verfügbar sind. Sie müssen sich zudem keine Gedanken über die Anzahl der vorhandenen Shards und Konsumenten-Instances machen. Wenn der Stream skaliert wird, müssen Sie Ihre Anwendung nicht neu schreiben, damit mehr als ein Shard oder eine Konsumenten-Instance verwaltet werden können.

Der Begriff Checkpointing bedeutet, dass der Punkt im Stream bis hin zu den Datensätzen aufgezeichnet wird, die bisher verbraucht und verarbeitet wurden. Wenn die Anwendung abstürzt, wird der Stream von diesem Punkt aus gelesen und nicht vom Anfang des Streams. Das Checkpointing sowie zugehörige Entwurfsmuster und bewährte Methoden sind nicht Gegenstand dieses Kapitels. Es ist jedoch etwas, das Ihnen in Produktionsumgebungen begegnen könnte.

Wie Sie unter erfahren haben, verwenden die `put` Operationen in den Kinesis Data Streams API einen Partitionsschlüssel als Eingabe. Kinesis Data Streams verwendet einen Partitionsschlüssel zum

Aufteilen von Datensätzen auf mehrere Shards (wenn mehr als ein Shard im Stream vorhanden ist). Derselbe Partitionsschlüssel leitet immer an denselben Shard weiter. Dadurch kann der Konsument, der einen bestimmten Shard verarbeitet, davon ausgehen, dass Datensätze mit demselben Partitionsschlüssel nur an ihn gesendet werden und Datensätze mit diesem Partitionsschlüssel nicht bei einem anderen Konsumenten landen. Deshalb kann der Worker eines Konsumenten alle Datensätze mit demselben Partitionsschlüssel aggregieren, ohne zu befürchten, dass erforderliche Daten fehlen.

In dieser Anwendung ist die Verarbeitung von Datensätzen durch den Benutzer nicht intensiv, sodass Sie einen Shard verwenden und die Verarbeitung im selben Thread wie der KCL Thread durchführen können. In der Praxis sollten Sie allerdings zunächst die Anzahl der Shards skalieren. Gelegentlich kann es vorkommen, dass Sie die Verarbeitung an einen anderen Thread übergeben oder einen Thread-Pool nutzen möchten, wenn eine intensive Datensatzverarbeitung ansteht. Auf diese Weise KCL können neue Datensätze schneller abgerufen werden, während die anderen Threads die Datensätze parallel verarbeiten können. Multithread-Design ist nicht trivial und sollte mit fortgeschrittenen Techniken angegangen werden. Daher ist die Erhöhung der Anzahl Ihrer Shards in der Regel die effektivste Methode zur Skalierung.

Nächste Schritte

(Optional) Erweitern Sie die Zahl der Verbraucher

(Optional) Erweitern Sie die Zahl der Verbraucher

Möglicherweise ist die Anwendung Tutorial: Verarbeiten Sie Aktiendaten in Echtzeit mit KPL und KCL 1.x bereits ausreichend für Ihre Zwecke. Dieser optionale Abschnitt zeigt, wie Sie den Konsumentencode erweitern können, um einem komplexeren Szenario gerecht zu werden.

Wenn Sie minütlich über die größten Verkaufsaufträge informiert werden möchten, können Sie die `StockStats`-Klasse an drei Stellen bearbeiten.

So erweitern Sie den Konsumenten

1. Fügen Sie neue Instance-Variablen hinzu:

```
// 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. Fügen Sie folgenden Code zu hinz `addStockTrade`:

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

3. Ändern Sie die `toString`-Methode, um die zusätzlichen Informationen zu drucken:

```
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);
}
```

Wenn Sie den Konsumenten jetzt ausführen (führen Sie auch den Produzenten), sollten Sie eine Ausgabe ähnlich der folgenden sehen:

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

Nächste Schritte

Bereinigen von -Ressourcen

Bereinigen von -Ressourcen

Da Sie für die Nutzung des Kinesis Data Streams zahlen, sollten Sie diesen ebenso wie die entsprechende Amazon-DynamoDB-Tabelle löschen, wenn sie nicht mehr benötigt werden. Für

einen aktiven Stream fallen auch dann nominale Gebühren an, wenn Sie keine Datensätze senden oder abrufen. Der Grund hierfür ist, dass ein aktiver Stream durch kontinuierlichen „Horchen“ darauf, ob neue Datensätze oder Anforderungen zum Abrufen von Datensätzen eingehen, Ressourcen verbraucht.

So löschen Sie den Stream und die Tabelle

1. Fahren Sie alle laufenden Produzenten und Konsumenten herunter.
2. Öffnen Sie die Kinesis-Konsole unter https://console.aws.amazon.com/kinesis.
3. Wählen Sie den Stream aus, den Sie für diese Anwendung erstellt haben (`StockTradeStream`).
4. Wählen Sie Delete Stream (Stream löschen) aus.
5. Öffnen Sie die DynamoDB-Konsole unter. https://console.aws.amazon.com/dynamodb/
6. Löschen Sie die `StockTradesProcessor`-Tabelle.

Übersicht

Die Verarbeitung einer großen Datenmenge nahezu in Echtzeit erfordert weder das Schreiben von kompliziertem Code noch die Entwicklung einer riesigen Infrastruktur. Es ist genauso einfach wie die Schreiblogik, eine kleine Datenmenge zu verarbeiten (wie Schreiben`processRecord(Record)`), aber Kinesis Data Streams zu verwenden, um so zu skalieren, dass es für eine große Menge an Streaming-Daten funktioniert. Sie müssen sich keine Gedanken über die Skalierung der Verarbeitung machen, da Kinesis Data Streams das für Sie übernimmt. Sie müssen lediglich Ihre Streaming-Datensätze an Kinesis Data Streams senden und eine Logik für die Verarbeitung neu empfangener Datensätze schreiben.

Nachfolgend einige mögliche Erweiterungen für diese Anwendung.

Shard-übergreifende Aggregation

> Derzeit erhalten Sie Statistiken, die aus der Aggregation von Datensätzen resultieren, die von einem einzelnen Auftragnehmer eines einzelnen Shards empfangen werden. (Ein Shard kann jeweils nur von einem Auftragnehmer in einer einzelnen Anwendung verarbeitet werden.) Möglicherweise möchten Sie, wenn Sie eine Skalierung durchführen und mehr als einen Shard haben, eine Shard-übergreifende Aggregation vornehmen. Dies kann mit einer Pipeline-Architektur realisiert werden, bei der die Ausgabe eines jeden Auftragnehmers in einen anderen Stream mit einem einzelnen Shard eingespeist wird, der von einem Auftragnehmer verarbeitet

wird, der die Ausgaben der ersten Phase aggregiert. Da die Daten aus der ersten Phase begrenzt sind (eine Ausgabe pro Minute pro Shard), können sie problemlos von nur einem Shard verarbeitet werden.

Skalierung

Wenn der Stream skaliert wird, damit mehr Shards zur Verfügung stehen (da viele Produzenten Daten senden), geschieht dies dadurch, dass mehr Worker hinzugefügt werden. Sie können die Worker in EC2 Amazon-Instances ausführen und Auto Scaling Scaling-Gruppen verwenden.

Verwenden Sie Konnektoren zu Amazon S3/DynamoDB/Amazon Redshift/Storm

Da ein Stream kontinuierlich verarbeitet wird, kann seine Ausgabe an andere Ziele gesendet werden. AWS bietet Konnektoren zur Integration von Kinesis Data Streams mit anderen AWS Diensten und Tools von Drittanbietern.

Nächste Schritte

- Weitere Informationen zur Verwendung von Kinesis Data Streams API Streams-Vorgängen finden Sie unter Entwickeln Sie Produzenten, die Amazon Kinesis Data Streams verwenden, API mit dem AWS SDK for JavaEntwickeln Sie benutzerdefinierte Verbraucher mit gemeinsamem Durchsatz mithilfe der AWS SDK for Java, undKinesis-Datenstreams erstellen und verwalten.
- Weitere Informationen zur Kinesis Client Library finden Sie unter Entwickeln Sie KCL 1.x-Verbraucher.
- Weitere Informationen zur Optimierung Ihrer Anwendung finden Sie unter Optimieren Sie die Verbraucher von Amazon Kinesis Data Streams.

Tutorial: Analysieren Sie Aktiendaten in Echtzeit mit Amazon Managed Service für Apache Flink

Im Szenario dieses Tutorials werden Wertpapierdaten in einen Datenstrom geschrieben. Zudem wird eine einfache Amazon Managed Service für Apache Flink-Anwendung erstellt, die Berechnungen mit dem Stream durchführt. Sie lernen, wie Sie einen Stream von Datensätzen an Kinesis Data Streams senden und eine Anwendung implementieren, die die Datensätze nahezu in Echtzeit verarbeitet und verarbeitet.

Mit Amazon Managed Service für Apache Flink können Sie Java oder Scala verwenden, um Streaming-Daten zu verarbeiten und zu analysieren. Mit diesem Service können Sie Java- oder

Scala-Code für Streaming-Quellen erstellen und ausführen, um Zeitreihenanalysen durchzuführen, Echtzeit-Dashboards zu füttern und Echtzeit-Metriken zu erstellen.

Sie können Flink-Anwendungen in Managed Service für Apache Flink mithilfe von Open-Source-Bibliotheken erstellen, die auf Apache Flink basieren. Apache Flink ist ein beliebtes Framework und eine verteilte Engine zum Verarbeiten von Datenströmen.

> **Important**
>
> Nachdem Sie zwei Datenstreams und eine Anwendung erstellt haben, fallen für Ihr Konto geringe Gebühren für Kinesis Data Streams und Managed Service für die Nutzung von Apache Flink an, da diese nicht für das AWS kostenlose Kontingent in Frage kommen. Wenn Sie mit dieser Anwendung fertig sind, löschen Sie Ihre AWS Ressourcen, damit keine Gebühren mehr anfallen.

Der Code greift nicht auf tatsächliche Wertpapierdaten zu, sondern simuliert nur deren Strom. Dazu werden zufällige Wertpapierdaten erzeugt. Wenn Sie Zugriff auf einen Echtzeit-Stream von Wertpapierdaten haben, möchten Sie vermutlich nützliche, zeitnahe Statistiken aus den Stream-Daten erzeugen. Sie können beispielsweise eine Zeitfensteranalyse durchführen, um festzustellen, welche Aktie in den letzten 5 Minuten am häufigsten erworben wurde. Oder Sie möchten im Falle eines zu großen Verkaufsauftrags (d. h. zu viele Anteile) benachrichtigt werden. Der Code in diesem Tutorial kann erweitert werden, um solche Funktionen bereitzustellen.

Bei den gezeigten Beispielen wird die Region USA West (Oregon) verwendet. Sie funktionieren aber auch für alle anderen AWS -Regionen, die Managed Service für Apache Flink unterstützen.

Aufgaben

- Voraussetzungen für das Abschließen der Übungen
- Richten Sie ein AWS Konto ein und erstellen Sie einen Administratorbenutzer
- Richten Sie das AWS Command Line Interface (AWS CLI) ein
- Erstellen Sie eine Managed Service for Apache Flink-Anwendung und führen Sie sie aus

Voraussetzungen für das Abschließen der Übungen

Zur Durchführung der Schritte in dieser Anleitung benötigen Sie Folgendes:

- Java-Entwicklungskit (JDK) Version 8. Stellen Sie die `JAVA_HOME` Umgebungsvariable so ein, dass sie auf Ihren JDK Installationsort verweist.
- Wir empfehlen die Verwendung einer Entwicklungsumgebung (wie Eclipse Java Neon oder IntelliJ Idea), um Ihre Anwendung zu entwickeln und zu kompilieren.
- Git-Client. Installieren Sie den Git-Client, wenn Sie dies noch nicht getan haben.
- Apache Maven-Compiler-Plugin. Maven muss sich in Ihrem Arbeitspfad befinden. Zum Testen Ihrer Apache Maven-Installation geben Sie Folgendes ein:

```
$ mvn -version
```

Um zu beginnen, gehen Sie zu Richten Sie ein AWS Konto ein und erstellen Sie einen Administratorbenutzer.

Richten Sie ein AWS Konto ein und erstellen Sie einen Administratorbenutzer

Bevor Sie Amazon Managed Service für Apache Flink zum ersten Mal verwenden, führen Sie die folgenden Aufgaben aus:

1. Melden Sie sich an für AWS
2. Erstellen eines IAM-Benutzers

Melden Sie sich an für AWS

Wenn Sie sich für Amazon Web Services (AWS) registrieren, wird Ihr AWS Konto automatisch für alle Dienste angemeldet AWS, einschließlich Amazon Managed Service für Apache Flink. Berechnet werden Ihnen aber nur die Services, die Sie nutzen.

Mit Managed Service für Apache Flink zahlen Sie nur für die Ressourcen, die Sie wirklich nutzen. Wenn Sie ein neuer AWS -Kunde sind, können Sie kostenlos mit Managed Service für Apache Flink beginnen. Weitere Informationen finden Sie unter Kostenloses Kontingent für AWS.

Wenn Sie bereits ein AWS Konto haben, fahren Sie mit der nächsten Aufgabe fort. Wenn Sie noch kein AWS -Konto haben, führen Sie die folgenden Schritte aus, um ein Konto zu erstellen.

Um ein AWS Konto zu erstellen

1. Öffnen Sie https://portal.aws.amazon.com/billing/die Anmeldung.
2. Folgen Sie den Online-Anweisungen.

   Bei der Anmeldung müssen Sie auch einen Telefonanruf entgegennehmen und einen Verifizierungscode über die Telefontasten eingeben.

   Wenn Sie sich für eine anmelden AWS-Konto, Root-Benutzer des AWS-Kontoswird eine erstellt. Der Root-Benutzer hat Zugriff auf alle AWS -Services und Ressourcen des Kontos. Als bewährte Sicherheitsmethode weisen Sie einem Administratorbenutzer Administratorzugriff zu und verwenden Sie nur den Root-Benutzer, um Aufgaben auszuführen, die Root-Benutzerzugriff erfordern.

Notieren Sie sich Ihre AWS Konto-ID, da Sie sie für die nächste Aufgabe benötigen.

Erstellen eines IAM-Benutzers

Dienste in AWS, wie Amazon Managed Service für Apache Flink, erfordern, dass Sie beim Zugriff auf sie Anmeldeinformationen angeben. Der Service kann feststellen, ob Sie über die Berechtigung für den Zugriff auf die Ressourcen im Besitz dieses Service verfügen. Das AWS Management Console erfordert, dass Sie Ihr Passwort eingeben.

Sie können Zugangsschlüssel für Ihr AWS Konto erstellen, um auf die AWS Command Line Interface (AWS CLI) oder zuzugreifenAPI. Wir empfehlen jedoch nicht, dass Sie AWS mit den Anmeldeinformationen für Ihr AWS Konto darauf zugreifen. Stattdessen empfehlen wir Ihnen, AWS Identity and Access Management (IAM) zu verwenden. Erstellen Sie einen IAM Benutzer, fügen Sie den Benutzer einer IAM Gruppe mit Administratorberechtigungen hinzu und gewähren Sie dann dem IAM Benutzer, den Sie erstellt haben, Administratorberechtigungen. Sie können dann AWS mit einem speziellen Benutzer URL und den Anmeldeinformationen dieses IAM Benutzers darauf zugreifen.

Wenn Sie sich für registriert haben AWS, aber noch keinen IAM Benutzer für sich selbst erstellt haben, können Sie über die IAM Konsole einen erstellen.

Für die Erste-Schritte-Übungen in diesem Handbuch wird davon ausgegangen, dass Sie einen Benutzer (`adminuser`) mit Administratorrechten haben. Befolgen Sie die Schritte zum Einrichten des `adminuser` in Ihrem Konto.

So erstellen Sie eine Gruppe für Administratoren:

1. Melden Sie sich bei der an AWS Management Console und öffnen Sie die IAM Konsole unter https://console.aws.amazon.com/iam/.
2. Wählen Sie im Navigationsbereich Groups (Gruppen) und dann Create New Group (Neue Gruppe erstellen) aus.
3. Geben Sie für Group Name (Gruppenname) einen Namen für die Gruppe ein, z. B. **`Administrators`**. Wählen Sie dann Next Step (Nächster Schritt) aus.
4. Wählen Sie in der Liste der Richtlinien das Kontrollkästchen neben der AdministratorAccessRichtlinie aus. Über das Menü Filter (Filtern) und das Feld Search (Suchen) können Sie die Liste filtern.
5. Wählen Sie Next Step (Nächster Schritt) und anschließend Create Group (Gruppe erstellen) aus.

Ihre neue Gruppe wird unter Group Name aufgeführt.

Um einen IAM Benutzer für sich selbst zu erstellen, fügen Sie ihn der Administratorgruppe hinzu und erstellen Sie ein Passwort

1. Wählen Sie im Navigationsbereich Users (Benutzer) und dann Add User (Benutzer hinzufügen) aus.
2. Geben Sie im Feld Benutzername einen Benutzernamen ein.
3. Wählen Sie die beiden Optionen Programmgesteuerter Zugriff und Zugriff auf die AWS - Managementkonsole aus.
4. Wählen Sie Weiter: Berechtigungen aus.
5. Aktivieren Sie das Kontrollkästchen neben der Administratorengruppe. Wählen Sie dann Next: Review aus.
6. Wählen Sie Create user (Benutzer erstellen) aus.

Um sich als neuer IAM Benutzer anzumelden

1. Melden Sie sich von der ab AWS Management Console.
2. Verwenden Sie das folgende URL Format, um sich bei der Konsole anzumelden:

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

Das Tool *aws_account_number* ist Ihre AWS Konto-ID ohne Bindestriche. Wenn Ihre AWS Konto-ID beispielsweise 1234-5678-9012 lautet, ersetzen Sie *aws_account_number* **123456789012**mit. Informationen dazu, wie Sie Ihre Kontonummer finden, finden Sie im IAMBenutzerhandbuch unter Ihre AWS Konto-ID und deren Alias.

3. Geben Sie den IAM Benutzernamen und das Passwort ein, die Sie gerade erstellt haben. Wenn Sie angemeldet sind, wird die Navigationsleiste angezeigt *your_user_name* @ *your_aws_account_id*.

> **Note**
>
> Wenn Sie nicht möchten, dass URL die Anmeldeseite Ihre AWS Konto-ID enthält, können Sie einen Kontoalias erstellen.

So erstellen oder entfernen Sie einen Konto-Alias

1. Öffnen Sie die IAM Konsole unter. https://console.aws.amazon.com/iam/
2. Wählen Sie im Navigationsbereich Dashboard aus.
3. Suchen Sie den IAM Anmeldelink des Benutzers.
4. Um einen Alias zu erstellen, klicken Sie auf Anpassen. Geben Sie den gewünschten Namen für den Alias ein und wählen Sie Yes, Create (Ja, erstellen) aus.
5. Um den Alias zu löschen, wählen Sie Customize (Anpassen) und dann Yes, Delete (Ja, löschen). Bei der Anmeldung wird URL wieder Ihre AWS Konto-ID verwendet.

Verwenden Sie Folgendes, um sich anzumelden, nachdem Sie einen Kontoalias erstellt haben: URL

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

Um den Anmeldelink für IAM Benutzer für Ihr Konto zu überprüfen, öffnen Sie die IAM Konsole und suchen Sie im Dashboard unter Anmeldelink für IAM Benutzer nach.

Weitere Informationen zu IAM finden Sie im Folgenden:

- AWS Identity and Access Management (IAM)
- Erste Schritte

- IAM-Benutzerhandbuch

Nächster Schritt

Richten Sie das AWS Command Line Interface (AWS CLI) ein

Richten Sie das AWS Command Line Interface (AWS CLI) ein

In diesem Schritt laden Sie den herunter und konfigurieren ihn für AWS CLI die Verwendung mit Amazon Managed Service für Apache Flink.

> **Note**
>
> Bei allen Erste-Schritte-Übungen in diesem Handbuch wird davon ausgegangen, dass Sie in Ihrem Konto Administrator-Anmeldeinformationen (`adminuser`) verwenden, um die Operationen auszuführen.

> **Note**
>
> Wenn Sie den bereits AWS CLI installiert haben, müssen Sie möglicherweise ein Upgrade durchführen, um die neuesten Funktionen zu erhalten. Weitere Informationen finden Sie im AWS Command Line Interface Benutzerhandbuch unter Installation der AWS Befehlszeilenschnittstelle. Führen Sie den folgenden Befehl aus AWS CLI, um die Version von zu überprüfen:
>
> ```
> aws --version
> ```
>
> Für die Übungen in diesem Tutorial ist die folgende AWS CLI Version oder höher erforderlich:
>
> ```
> aws-cli/1.16.63
> ```

Um das einzurichten AWS CLI

1. Herunterladen und Konfigurieren von AWS CLI. Eine Anleitung finden Sie unter den folgenden Themen im AWS Command Line Interface -Benutzerhandbuch:

- Installieren des AWS Command Line Interface
- Konfigurieren von AWS CLI

2. Fügen Sie der AWS CLI Konfigurationsdatei ein benanntes Profil für den Administratorbenutzer hinzu. Sie verwenden dieses Profil, wenn Sie die AWS CLI Befehle ausführen. Weitere Informationen zu benannten Profilen finden Sie unter Benannte Profile im AWS Command Line Interface Benutzerhandbuch.

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

   Eine Liste der verfügbaren AWS Regionen finden Sie unter AWS Regionen und Endpunkte in der Allgemeine Amazon Web Services-Referenz.

3. Überprüfen Sie die Einrichtung, indem Sie die folgenden Hilfebefehle in die Befehlszeile eingeben:

```
aws help
```

Nachdem Sie ein AWS Konto eingerichtet haben AWS CLI, können Sie die nächste Übung ausprobieren, in der Sie eine Beispielanwendung konfigurieren und das end-to-end Setup testen.

Nächster Schritt

Erstellen Sie eine Managed Service for Apache Flink-Anwendung und führen Sie sie aus

Erstellen Sie eine Managed Service for Apache Flink-Anwendung und führen Sie sie aus

In dieser Übung erstellen Sie eine Anwendung von Managed Service für Apache Flink mit Datenströmen als Quelle und Senke.

Dieser Abschnitt enthält die folgenden Schritte:

- Erstellen Sie zwei Amazon Kinesis Kinesis-Datenstreams
- Schreiben Sie Beispieldatensätze in den Eingabe-Stream
- Laden Sie den Apache Flink-Streaming-Java-Code herunter und untersuchen Sie ihn

- Kompilieren Sie den Anwendungscode
- Laden Sie den Apache Flink-Streaming-Java-Code hoch
- Erstellen Sie die Anwendung Managed Service for Apache Flink und führen Sie sie aus

Erstellen Sie zwei Amazon Kinesis Kinesis-Datenstreams

Bevor Sie für diese Übung einen Amazon Managed Service für Apache Flink erstellen, erstellen Sie zwei Kinesis-Datenstreams (`ExampleInputStream`und`ExampleOutputStream`). Ihre Anwendung verwendet diese Streams für die Quell- und Ziel-Streams der Anwendung.

Sie können diese Streams mithilfe der Amazon-Kinesis-Konsole oder des folgenden AWS CLI -Befehls erstellen. Detaillierte Konsolenanweisungen finden Sie unter Erstellen und Aktualisieren von Daten-Streams.

So erstellen Sie die Daten-Streams (AWS CLI)

1. Verwenden Sie den folgenden Amazon Kinesis `create-stream` AWS CLI Kinesis-Befehl, um den ersten Stream (`ExampleInputStream`) zu erstellen.

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

2. Um den zweiten Stream zu erstellen, den die Anwendung zum Schreiben der Ausgabe verwendet, führen Sie denselben Befehl aus und ändern den Stream-Namen in `ExampleOutputStream`.

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

Schreiben Sie Beispieldatensätze in den Eingabe-Stream

In diesem Abschnitt verwenden Sie ein Python-Skript zum Schreiben von Datensätzen in den Stream für die zu verarbeitende Anwendung.

> **Note**
>
> Dieser Abschnitt erfordert AWS SDK for Python (Boto).

1. Erstellen Sie eine Datei `stock.py` mit dem folgenden Inhalt:

```
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. Im weiteren Verlauf des Tutorials führen Sie das `stock.py`-Skript zum Senden von Daten an die Anwendung aus.

```
$ python stock.py
```

Laden Sie den Apache Flink-Streaming-Java-Code herunter und untersuchen Sie ihn

Der Java-Anwendungscode für diese Beispiele ist verfügbar unter GitHub. Zum Herunterladen des Anwendungscodes gehen Sie wie folgt vor:

1. Klonen Sie das Remote-Repository mit dem folgenden Befehl:

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

2. Navigieren Sie zum `GettingStarted` Verzeichnis .

Der Anwendungscode befindet sich in den Dateien `CloudWatchLogSink.java` und `CustomSinkStreamingJob.java`. Beachten Sie Folgendes zum Anwendungscode:

- Die Anwendung verwendet eine Kinesis-Quelle zum Lesen aus dem Quell-Stream. Der folgende Codeausschnitt erstellt die Kinesis-Senke:

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

Kompilieren Sie den Anwendungscode

In diesem Abschnitt verwenden Sie den Apache Maven-Compiler zum Erstellen des Java-Codes für die Anwendung. Hinweise zur Installation von Apache Maven und dem Java Development Kit (JDK) finden Sie unterVoraussetzungen für das Abschließen der Übungen.

Ihre Java-Anwendung erfordert die folgenden Komponenten:

- Eine Projektobjektmodell (pom.xml)-Datei. Diese Datei enthält Informationen zur Konfiguration und zu den Abhängigkeiten der Anwendung, einschließlich der Amazon Managed Service for Apache Flink-Bibliotheken.
- Eine `main`-Methode, die die Logik der Anwendung enthält.

> **Note**
>
> Um den Kinesis-Konnektor für die folgende Anwendung zu verwenden, müssen Sie den Quellcode für den Konnektor herunterladen und ihn wie in der Apache Flink-Dokumentation beschrieben erstellen.

So erstellen und kompilieren Sie den Anwendungscode

1. Erstellen Sie eine Java/Maven-Anwendung in Ihrer Entwicklungsumgebung. Weitere Informationen zum Erstellen einer Anwendung finden Sie in der Dokumentation für Ihre Entwicklungsumgebung:
   - Erstellen Sie Ihr erstes Java-Projekt (Eclipse Java Neon)
   - Erstellen, Ausführen und Packen Ihrer ersten Java-Anwendung (IntelliJ Idea)
2. Verwenden Sie den folgenden Code für eine Datei mit dem Namen `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");
    }
}
```

Beachten Sie die folgenden Informationen zum vorherigen Codebeispiel:

- Diese Datei enthält die `main`-Methode, die die Funktionalität der Anwendung definiert.
- Ihre Anwendung erstellt Quell- und Senkenkonnektoren für den Zugriff auf externe Ressourcen, indem ein `StreamExecutionEnvironment`-Objekt verwendet wird.
- Die Anwendung erstellt Quell- und Senkenkonnektoren mit statischen Eigenschaften. Zum Verwenden dynamischer Anwendungseigenschaften verwenden Sie die Methoden `createSourceFromApplicationProperties` und `createSinkFromApplicationProperties`, um die Konnektoren zu erstellen. Diese Methoden lesen die Eigenschaften der Anwendung zum Konfigurieren der Konnektoren.

3. Um Ihren Anwendungscode zu verwenden, kompilieren Sie ihn und packen ihn in eine JAR Datei. Sie können Ihren Code auf zwei Arten kompilieren und packen:

- Verwenden Sie das Befehlszeilen-Maven-Tool. Erstellen Sie Ihre JAR Datei, indem Sie den folgenden Befehl in dem Verzeichnis ausführen, das die `pom.xml` Datei enthält:

```
mvn package
```

- Verwenden Sie Ihre Entwicklungsumgebung. Weitere Informationen finden Sie in der Dokumentation Ihrer Entwicklungsumgebung.

Sie können Ihr Paket entweder als JAR Datei hochladen oder Sie können Ihr Paket komprimieren und als ZIP Datei hochladen. Wenn Sie Ihre Anwendung mit dem erstellen AWS CLI, geben Sie Ihren Codeinhaltstyp (JARoderZIP) an.

4. Wenn während der Erstellung Fehler aufgetreten sind, überprüfen Sie, ob Ihre `JAVA_HOME`-Umgebungsvariable richtig eingestellt ist.

Wenn die Anwendung erfolgreich kompiliert wurde, wird die folgende Datei erstellt:

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

Laden Sie den Apache Flink-Streaming-Java-Code hoch

In diesem Abschnitt erstellen Sie einen Amazon Simple Storage Service (Amazon S3)-Bucket und laden Ihren Anwendungscode hoch.

So laden Sie den Anwendungscode hoch

1. Öffnen Sie die Amazon S3 S3-Konsole unter https://console.aws.amazon.com/s3/.
2. Wählen Sie Bucket erstellen aus.
3. Geben Sie **`ka-app-code-`*`<username>`*** im Feld Bucket-Name ein. Fügen Sie dem Bucket-Namen ein Suffix hinzu, wie z. B. Ihren Benutzernamen, damit er global eindeutig ist. Wählen Sie Weiter.
4. Lassen Sie im Schritt Optionen konfigurieren die Einstellungen unverändert und klicken Sie auf Weiter.
5. Lassen Sie im Schritt Berechtigungen festlegen die Einstellungen unverändert und klicken Sie auf Weiter.
6. Wählen Sie Bucket erstellen aus.

7. Wählen Sie in der Amazon S3 S3-Konsole die Option ka-app-code -**<username>**Bucket und wählen Sie Upload.
8. Klicken Sie im Schritt Auswählen von Dateien auf Hinzufügen von Dateien. Navigieren Sie zu der `java-getting-started-1.0.jar`Datei, die Sie im vorherigen Schritt erstellt haben. Wählen Sie Weiter.
9. Lassen Sie im Schritt Berechtigungen festlegen die Einstellungen unverändert. Wählen Sie Weiter.
10. Lassen Sie im Schritt Eigenschaften festlegen die Einstellungen unverändert. Klicken Sie auf Hochladen.

Ihr Anwendungscode ist jetzt in einem Amazon-S3-Bucket gespeichert, in dem Ihre Anwendung darauf zugreifen kann.

Erstellen Sie die Anwendung Managed Service for Apache Flink und führen Sie sie aus

Sie können eine Anwendung von Managed Service für Apache Flink entweder über die Konsole oder AWS CLI erstellen und ausführen.

> **Note**
>
> Wenn Sie die Anwendung mithilfe der Konsole erstellen, werden Ihre AWS Identity and Access Management (IAM) und Amazon CloudWatch Logs-Ressourcen für Sie erstellt. Wenn Sie die Anwendung mithilfe von erstellen AWS CLI, erstellen Sie diese Ressourcen separat.

Themen

- Erstellen Sie die Anwendung und führen Sie sie aus (Konsole)
- Erstellen Sie die Anwendung und führen Sie sie aus (AWS CLI)

Erstellen Sie die Anwendung und führen Sie sie aus (Konsole)

Befolgen Sie diese Schritte, um die Anwendung über die Konsole zu erstellen, zu konfigurieren, zu aktualisieren und auszuführen.

Erstellen der Anwendung

1. Öffnen Sie die Kinesis-Konsole unter https://console.aws.amazon.com/kinesis.

2. Wählen Sie auf dem Amazon-Kinesis-Dashboard die Option Create analytics application (Analyseanwendung erstellen) aus.
3. Geben Sie auf der Seite Kinesis Analytics – Anwendung erstellen die Anwendungsdetails wie folgt an:
   - Geben Sie als Anwendungsname ein **`MyApplication`**.
   - Geben Sie für Beschreibung den Text **`My java test app`** ein.
   - Wählen Sie für Runtime (Laufzeit) die Option Apache Flink 1.6 aus.
4. Wählen Sie für Zugriffsberechtigungen die Option Rolle erstellen/aktualisieren aus. IAM `kinesis-analytics-MyApplication-us-west-2`
5. Wählen Sie Create application aus.

> **Note**
>
> Wenn Sie mithilfe der Konsole eine Amazon Managed Service for Apache Flink-Anwendung erstellen, haben Sie die Möglichkeit, eine IAM Rolle und Richtlinie für Ihre Anwendung erstellen zu lassen. Ihre Anwendung verwendet diese Rolle und Richtlinie für den Zugriff auf ihre abhängigen Ressourcen. Diese IAM Ressourcen werden anhand Ihres Anwendungsnamens und Ihrer Region wie folgt benannt:
>
> - Richtlinie: `kinesis-analytics-service-`*`MyApplication`*`-`*`us-west-2`*
> - Rolle: `kinesis-analytics-`*`MyApplication`*`-`*`us-west-2`*

Bearbeiten Sie die IAM Richtlinie

Bearbeiten Sie die IAM Richtlinie, um Berechtigungen für den Zugriff auf die Kinesis-Datenstreams hinzuzufügen.

1. Öffnen Sie die IAM Konsole unter. https://console.aws.amazon.com/iam/
2. Wählen Sie Policies (Richtlinien). Wählen Sie die **`kinesis-analytics-service-MyApplication-us-west-2`**-Richtlinie aus, die die Konsole im vorherigen Abschnitt für Sie erstellt hat.
3. Wählen Sie auf der Seite Summary (Übersicht) die Option Edit policy (Richtlinie bearbeiten) aus. Wählen Sie die JSONRegisterkarte.

4. Fügen Sie den markierten Abschnitt der folgenden Beispielrichtlinie der Richtlinie hinzu. Ersetzen Sie das Beispielkonto IDs (*012345678901*) durch Ihre Konto-ID.

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

Konfigurieren Sie die Anwendung

1. Wählen Sie auf der MyApplicationSeite Configure aus.
2. Klicken Sie auf der Seite Configure application (Anwendung konfigurieren) auf die Option Code location (Codespeicherort):
   - Geben Sie für Amazon-S3-Bucket **`ka-app-code-<username>`** ein.
   - Geben Sie als Pfad zum Amazon-S3-Objekt den Wert **`java-getting-started-1.0.jar`** ein.
3. Wählen Sie unter Zugriff auf Anwendungsressourcen für Zugriffsberechtigungen die Option IAMRolle erstellen/aktualisieren aus`kinesis-analytics-MyApplication-us-west-2`.
4. Geben Sie unter Eigenschaften für Gruppen-ID den Text **`ProducerConfigProperties`** ein.
5. Geben Sie die folgenden Eigenschaften und Werte der Anwendung ein:

Schlüssel	Wert
flink.inputstream.initpos	**LATEST**
aws:region	**us-west-2**
AggregationEnabled	**false**

6. Stellen Sie unter Überwachung sicher, dass die Ebene der Überwachungsmetriken auf Anwendung eingestellt ist.
7. Wählen Sie für die CloudWatch Protokollierung das Kontrollkästchen Aktivieren aus.
8. Wählen Sie Aktualisieren.

> **Note**
>
> Wenn Sie die CloudWatch Protokollierung aktivieren möchten, erstellt Managed Service for Apache Flink eine Protokollgruppe und einen Protokollstream für Sie. Die Namen dieser Ressourcen lauten wie folgt:
>
> - Protokollgruppe: `/aws/kinesis-analytics/MyApplication`
> - Protokollstream: `kinesis-analytics-log-stream`

Führen Sie die Anwendung aus.

1. Wählen Sie auf der MyApplicationSeite die Option Ausführen aus. Bestätigen Sie die Aktion.
2. Wenn die Anwendung ausgeführt wird, aktualisieren Sie die Seite. Die Konsole zeigt den Application graph (Anwendungs-Graph) an.

Beenden Sie die Anwendung

Wählen Sie auf der MyApplicationSeite Stopp. Bestätigen Sie die Aktion.

Aktualisieren der Anwendung

Mithilfe der Konsole können Sie Anwendungseinstellungen wie Anwendungseigenschaften, Überwachungseinstellungen und den Speicherort oder den Dateinamen der Anwendung

aktualisierenJAR. Sie können die Anwendung auch JAR aus dem Amazon S3 S3-Bucket neu laden, wenn Sie den Anwendungscode aktualisieren müssen.

Wählen Sie auf der MyApplicationSeite Configure aus. Aktualisieren Sie die Anwendungseinstellungen und klicken Sie auf Aktualisieren.

Erstellen Sie die Anwendung und führen Sie sie aus (AWS CLI)

In diesem Abschnitt verwenden Sie die, AWS CLI um die Anwendung Managed Service for Apache Flink zu erstellen und auszuführen. Managed Service for Apache Flink verwendet den `kinesisanalyticsv2` AWS CLI Befehl, um Managed Service for Apache Flink-Anwendungen zu erstellen und mit ihnen zu interagieren.

Erstellen einer Berechtigungsrichtlinie

Zuerst erstellen Sie eine Berechtigungsrichtlinie mit zwei Anweisungen: eine, die Berechtigungen für die `read`-Aktion auf den Quell-Stream zulässt, und eine andere, die Berechtigungen für die `write`-Aktionen auf den Senken-Stream zulässt. Anschließend fügen Sie die Richtlinie einer IAM Rolle hinzu (die Sie im nächsten Abschnitt erstellen). Wenn Managed Service für Apache Flink also die Rolle übernimmt, verfügt der Service über die erforderlichen Berechtigungen zum Lesen aus dem Quell-Stream und zum Schreiben in den Senken-Stream.

Verwenden Sie den folgenden Code zum Erstellen der `KAReadSourceStreamWriteSinkStream`-Berechtigungsrichtlinie. Ersetzen Sie *`username`* durch den Benutzernamen, den Sie verwendet haben, um den Amazon-S3-Bucket zum Speichern des Anwendungscodes zu erstellen. Ersetzen Sie die Konto-ID in den Amazon-Ressourcennamen (ARNs) (*`012345678901`*) durch Ihre Konto-ID.

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

step-by-step Anweisungen zum Erstellen einer Berechtigungsrichtlinie finden Sie unter Tutorial: Erstellen und Anhängen Ihrer ersten vom Kunden verwalteten Richtlinie im IAMBenutzerhandbuch.

> **Note**
>
> Um auf andere AWS Dienste zuzugreifen, können Sie die verwenden AWS SDK for Java. Managed Service for Apache Flink setzt die für den Dienst erforderlichen Anmeldeinformationen automatisch SDK auf die Anmeldeinformationen der IAM Dienstausführungsrolle, die Ihrer Anwendung zugeordnet ist. Es sind keine weiteren Schritte erforderlich.

Erstellen Sie eine IAM-Rolle.

In diesem Abschnitt erstellen Sie eine IAM Rolle, die Managed Service for Apache Flink übernehmen kann, um einen Quellstream zu lesen und in den Sink-Stream zu schreiben.

Managed Service für Apache Flink kann ohne Berechtigungen nicht auf Ihren Stream zugreifen. Sie gewähren diese Berechtigungen über eine IAM Rolle. Jeder IAM Rolle sind zwei Richtlinien zugeordnet. Die Vertrauensrichtlinie erteilt Managed Service für Apache Flink die Berechtigung zum Übernehmen der Rolle und die Berechtigungsrichtlinie bestimmt, was Managed Service für Apache Flink nach Annahme der Rolle tun kann.

Sie können die Berechtigungsrichtlinie, die Sie im vorherigen Abschnitt erstellt haben, dieser Rolle anfügen.

So erstellen Sie eine IAM-Rolle

1. Öffnen Sie die IAM Konsole unter https://console.aws.amazon.com/iam/.
2. Wählen Sie im Navigationsbereich Roles (Rollen) und Create Role (Rolle erstellen) aus.
3. Wählen Sie unter Typ der vertrauenswürdigen Entität auswählen die Option AWS -Service aus. Wählen Sie unter Choose the service that will use this role (Wählen Sie den Service aus, der diese Rolle verwendet) die Option Kinesis aus. Wählen Sie unter Select your use case (Wählen Sie Ihren Anwendungsfall aus) die Option Kinesis Analytics aus.

   Wählen Sie Weiter: Berechtigungen aus.
4. Wählen Sie auf der Seite Attach permissions policies (Berechtigungsrichtlinien hinzufügen) Next: Review (Weiter: Überprüfen) aus. Sie fügen Berechtigungsrichtlinien an, nachdem Sie die Rolle erstellt haben.
5. Geben Sie auf der Seite Create role (Rolle erstellen) den Text **`KA-stream-rw-role`** für Role name (Rollenname) ein. Wählen Sie Rolle erstellen.

   Jetzt haben Sie eine neue IAM Rolle namens erstellt`KA-stream-rw-role`. Im nächsten Schritt aktualisieren Sie die Vertrauens- und Berechtigungsrichtlinien für die Rolle.
6. Fügen Sie die Berechtigungsrichtlinie der Rolle an.

   > **Note**
   >
   > Für diese Übung übernimmt Managed Service für Apache Flink diese Rolle sowohl für das Lesen von Daten aus einem Kinesis-Datenstrom (Quelle) als auch zum Schreiben der Ausgabedaten in einen anderen Kinesis-Datenstrom. Daher fügen Sie die Richtlinie an, die Sie im vorherigen Schritt erstellt haben, the section called “Erstellen einer Berechtigungsrichtlinie”.

   a. Wählen Sie auf der Seite Summary (Übersicht) die Registerkarte Permissions (Berechtigungen) aus.
   b. Wählen Sie Attach Policies (Richtlinien anfügen) aus.
   c. Geben Sie im Suchfeld **`KAReadSourceStreamWriteSinkStream`** (die Richtlinie, die Sie im vorhergehenden Abschnitt erstellt haben) ein.
   d. Wählen Sie die KAReadInputStreamWriteOutputStreamRichtlinie aus und klicken Sie auf Richtlinie anhängen.

Sie haben nun die Service-Ausführungsrolle erstellt, die Ihre Anwendung für den Zugriff auf Ressourcen verwendet. Notieren Sie sich die ARN der neuen Rolle.

step-by-step Anweisungen zum Erstellen einer Rolle finden Sie unter Erstellen einer IAM Rolle (Konsole) im IAMBenutzerhandbuch.

Erstellen Sie die Anwendung Managed Service für Apache Flink

1. Speichern Sie den folgenden JSON Code in einer Datei mit dem Namen`create_request.json`. Ersetzen Sie die Beispielrolle ARN durch die ARN für die Rolle, die Sie zuvor erstellt haben. Ersetzen Sie das ARN Bucket-Suffix (*`username`*) durch das Suffix, das Sie im vorherigen Abschnitt ausgewählt haben. Ersetzen Sie die beispielhafte Konto-ID (*`012345678901`*) in der Service-Ausführungsrolle mit Ihrer Konto-ID.

```
{
    "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. Führen Sie die `CreateApplication`-Aktion mit der vorherigen Anforderung zum Erstellen der Anwendung aus:

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

Die Anwendung wird nun erstellt. Sie starten die Anwendung im nächsten Schritt.

Starten der Anwendung

In diesem Abschnitt verwenden Sie die `StartApplication`-Aktion, um die Anwendung zu starten.

So starten Sie die Anwendung

1. Speichern Sie den folgenden JSON Code in einer Datei mit dem Namen. `start_request.json`

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

2. Führen Sie die `StartApplication`-Aktion mit der vorherigen Anforderung zum Starten der Anwendung aus:

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

Die Anwendung wird jetzt ausgeführt. Sie können die Kennzahlen Managed Service for Apache Flink auf der CloudWatch Amazon-Konsole überprüfen, um sicherzustellen, dass die Anwendung funktioniert.

Stoppen der Anwendung

In diesem Abschnitt verwenden Sie die `StopApplication`-Aktion, um die Anwendung zu stoppen.

So stoppen Sie die Anwendung

1. Speichern Sie den folgenden JSON Code in einer Datei mit dem Namen`stop_request.json`.

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

2. Führen Sie die `StopApplication`-Aktion mit der folgenden Anforderung zum Stoppen der Anwendung aus:

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

Die Anwendung wird nun gestoppt.

Tutorial: Verwendung AWS Lambda mit Amazon Kinesis Data Streams

In dieser Anleitung erstellen Sie eine Lambda-Funktion, um Ereignisse aus einem Kinesis-Datenstrom zu nutzen. In diesem Beispielszenario schreibt eine benutzerdefinierte Anwendung Datensätze in einen Kinesis-Datenstrom. AWS Lambda fragt dann diesen Datenstrom ab und ruft, wenn er neue Datensätze erkennt, Ihre Lambda-Funktion auf. AWS Lambda führt dann die Lambda-Funktion aus, indem es die Ausführungsrolle annimmt, die Sie bei der Erstellung der Lambda-Funktion angegeben haben.

Eine ausführliche schrittweise Anleitung finden Sie unter Tutorial: Verwenden von AWS Lambda mit Amazon Kinesis.

> **Note**
>
> In diesem Tutorial wird davon ausgegangen, dass Sie über einige Kenntnisse der grundlegenden Lambda-Operationen und der AWS Lambda Konsole verfügen. Falls Sie dies

noch nicht getan haben, folgen Sie den Anweisungen unter Erste Schritte mit AWS Lambda, um Ihre erste Lambda-Funktion zu erstellen.

Verwenden Sie die AWS Streaming-Datenlösung für Amazon Kinesis

Die AWS Streaming-Datenlösung für Amazon Kinesis konfiguriert automatisch die AWS Services, die für die einfache Erfassung, Speicherung, Verarbeitung und Bereitstellung von Streaming-Daten erforderlich sind. Die Lösung bietet mehrere Optionen zur Lösung von Anwendungsfällen für Streaming-Daten, die mehrere AWS Dienste nutzen, darunter Kinesis Data Streams AWS Lambda, Amazon API Gateway und Amazon Managed Service für Apache Flink.

Jede Lösung enthält die folgenden Komponenten:

- Ein AWS CloudFormation Paket zur Bereitstellung des vollständigen Beispiels.
- Ein CloudWatch Dashboard zur Anzeige von Anwendungsmetriken.
- CloudWatch Alarme zu den relevantesten Anwendungsmetriken.
- Alle erforderlichen IAM Rollen und Richtlinien.

Die Lösung finden Sie hier: Streaming-Datenlösung für Amazon Kinesis

Kinesis-Datenstreams erstellen und verwalten

Amazon Kinesis Data Streams übernimmt eine große Menge von Daten in Echtzeit, speichert diese dauerhaft und stellt sie für den Verbrauch zur Verfügung. Die von Kinesis Data Streams gespeicherte Dateneinheit wird als Datensatz bezeichnet. Ein Datenstream repräsentiert eine Gruppe von Datensätzen. Die Datensätze in einem Daten-Stream werden auf Shards verteilt.

Ein Shard hat eine Sequenz von Datensätzen in einem Stream. Sie dient als Basisdurchsatzeinheit eines Kinesis Data Streams. Ein Shard unterstützt 1 Mbit/s und 1 000 Datensätze pro Sekunde für Schreibvorgänge und 2 Mbit/s für Lesevorgänge sowohl bei Bedarf als auch im Modus mit bereitgestellter Kapazität. Die Shard-Limits sorgen für eine vorhersehbare Leistung und erleichtern so die Entwicklung und den Betrieb eines äußerst zuverlässigen Datenstreaming-Workflows.

In diesem Abschnitt erfahren Sie, wie Sie den Kapazitätsmodus für den Stream festlegen und wie Sie einen Stream entweder mit AWS Management Console oder APIs erstellen. Anschließend können Sie weitere Aktionen für den Stream ausführen.

Themen

Wählen Sie den Datenstream-Kapazitätsmodus

In den folgenden Themen wird erklärt, wie Sie den richtigen Kapazitätsmodus für Ihre Anwendung auswählen und bei Bedarf zwischen den Kapazitätsmodi wechseln können.

Themen

Was ist ein Datenstream-Kapazitätsmodus?

Ein Kapazitätsmodus bestimmt, wie die Kapazität eines Datenstroms verwaltet wird und wie Ihnen die Nutzung Ihres Datenstroms in Rechnung gestellt wird. In Amazon Kinesis Data Streams können Sie zwischen einem On-Demand-Modus und einem bereitgestellten Modus für Ihre Datenströme wählen.

- On-Demand – Datenströme mit einem On-Demand-Modus erfordern keine Kapazitätsplanung und werden automatisch skaliert, sodass sie einen Schreib- und Lesedurchsatz von mehreren Gigabyte pro Minute verarbeiten können. Im On-Demand-Modus verwaltet Kinesis Data Streams die Shards automatisch, um den erforderlichen Durchsatz bereitzustellen.
- Bereitgestellt – Für Datenströme mit einem Bereitstellungsmodus müssen Sie die Anzahl der Shards für den Datenstrom angeben. Die Gesamtkapazität eines Datenstroms ist die Summe der Kapazitäten der einzelnen Shards. Sie können die Anzahl der Shards in einem Datenstrom bei Bedarf erhöhen oder verringern.

Sie können Kinesis Data Streams verwenden `PutRecord` und `PutRecords` APIs Daten sowohl im On-Demand-Modus als auch im Bereitstellungsmodus in Ihre Datenstreams schreiben. Zum Abrufen von Daten unterstützen beide Kapazitätsmodi Standardverbraucher, die den verwenden, `GetRecords` API und Enhanced Fan-Out (EFO) -Verbraucher, die den verwenden. `SubscribeToShard` API

Alle Funktionen von Kinesis Data Streams, einschließlich Aufbewahrungsmodus, Verschlüsselung, Überwachungsmetriken und andere, werden sowohl für den On-Demand-Modus als auch für den Bereitstellungsmodus unterstützt. Kinesis Data Streams bietet eine hohe Beständigkeit und Verfügbarkeit sowohl im On-Demand-Modus als auch im Bereitstellungsmodus.

Funktionen und Anwendungsfälle des On-Demand-Modus

Datenströme im On-Demand-Modus erfordern keine Kapazitätsplanung und werden automatisch skaliert, sodass sie einen Schreib- und Lesedurchsatz von mehreren Gigabyte pro Minute verarbeiten können. Der On-Demand-Modus vereinfacht die Aufnahme und Speicherung großer Datenmengen bei niedriger Latenz, da er die Bereitstellung und Verwaltung von Servern, Speicher oder Durchsatz

überflüssig macht. Sie können Milliarden von Datensätzen pro Tag aufnehmen, ohne dass der Betrieb beeinträchtigt wird.

Der On-Demand-Modus ist ideal, um den Anforderungen eines stark variablen und unvorhersehbaren Anwendungsverkehrs gerecht zu werden. Sie müssen diese Workloads nicht mehr für Spitzenkapazitäten bereitstellen, was bei geringer Auslastung zu höheren Kosten führen kann. Der On-Demand-Modus eignet sich für Workloads mit unvorhersehbaren und stark variablen Datenverkehrsmustern.

Im On-Demand-Kapazitätsmodus zahlen Sie pro GB geschriebener und gelesener Daten aus Ihren Datenströmen. Sie müssen nicht angeben, wie viel Lese- und Schreibdurchsatz Sie von Ihrer Anwendung erwarten. Kinesis Data Streams passt sich sofort an Ihre Workloads an, wenn diese ansteigen oder sinken. Weitere Informationen finden Sie unter Amazon Kinesis Data Streams – Preise.

Ein Datenstrom im On-Demand-Modus ermöglicht bis zu doppelt so viel wie den Spitzenschreibdurchsatz, der in den letzten 30 Tagen beobachtet wurde. Wenn der Schreibdurchsatz Ihres Datenstroms einen neuen Höchstwert erreicht, skaliert Kinesis Data Streams die Kapazität des Datenstroms automatisch. Wenn Ihr Datenstrom beispielsweise einen Schreibdurchsatz hat, der zwischen 10 Mbit/s und 40 Mbit/s variiert, stellt Kinesis Data Streams sicher, dass Sie problemlos auf das Doppelte des bisherigen Spitzendurchsatzes, also 80 Mbit/s, hochfahren können. Wenn derselbe Datenstrom einen neuen Spitzendurchsatz von 50 Mbit/s erreicht, stellt Kinesis Data Streams sicher, dass genügend Kapazität vorhanden ist, um einen Schreibdurchsatz von 100 Mbit/s aufzunehmen. Eine Drosselung des Schreibvorgangs kann jedoch auftreten, wenn Ihr Datenverkehr innerhalb von 15 Minuten auf mehr als das Doppelte des vorherigen Spitzenwerts ansteigt. Sie müssen diese gedrosselten Anfragen erneut versuchen.

Die aggregierte Lesekapazität eines Datenstroms im On-Demand-Modus steigt proportional zum Schreibdurchsatz. Auf diese Weise wird sichergestellt, dass Verbraucheranwendungen immer über einen ausreichenden Lesedurchsatz verfügen, um eingehende Daten in Echtzeit zu verarbeiten. Sie erhalten mindestens den doppelten Schreibdurchsatz im Vergleich zu Lesedaten mit `GetRecords`API. Wir empfehlen, dass Sie eine Verbraucheranwendung zusammen mit der verwenden `GetRecord`API, damit sie genügend Speicherplatz hat, um catch, wenn die Anwendung nach einem Ausfall wiederhergestellt werden muss. Es wird empfohlen, die Funktion Erweitertes Rundsenden von Kinesis Data Streams für Szenarien zu verwenden, in denen mehr als eine Verbraucheranwendung hinzugefügt werden muss. Enhanced Fan-Out unterstützt das Hinzufügen von bis zu 20 Verbraucheranwendungen zu einem Datenstrom mithilfe von `SubscribeToShard`API, wobei jede Verbraucheranwendung über einen eigenen Durchsatz verfügt.

Behandelt Ausnahmen beim Lese- und Schreibdurchsatz

Im On-Demand-Kapazitätsmodus (wie im Modus „Bereitgestellte Kapazität“) müssen Sie für jeden Datensatz einen Partitionsschlüssel angeben, um Daten in Ihren Datenstrom zu schreiben. Kinesis Data Streams verwendet Ihre Partitionsschlüssel, um Daten auf Shards zu verteilen. Kinesis Data Streams überwacht den Verkehr für jeden Shard. Wenn der eingehende Verkehr 500 KB/s pro Shard überschreitet, wird der Shard innerhalb von 15 Minuten aufgeteilt. Die Hash-Schlüsselwerte des übergeordneten Shards werden gleichmäßig auf die untergeordneten Shards verteilt.

Wenn der eingehende Datenverkehr das Doppelte des vorherigen Spitzenwertes übersteigt, kann es für etwa 15 Minuten zu Lese- oder Schreibausfällen kommen, selbst wenn die Daten gleichmäßig auf die Shards verteilt sind. Wir empfehlen, dass Sie alle derartigen Anfragen erneut versuchen, damit alle Datensätze ordnungsgemäß in Kinesis Data Streams gespeichert werden.

Es kann zu Lese- und Schreibausnahmen kommen, wenn Sie einen Partitionsschlüssel verwenden, der zu einer ungleichmäßigen Datenverteilung führt, und die einem bestimmten Shard zugewiesenen Datensätze dessen Grenzwerte überschreiten. Im On-Demand-Modus passt sich der Datenstrom automatisch an ungleichmäßige Datenverteilungsmuster an, es sei denn, ein einzelner Partitionsschlüssel überschreitet den Durchsatz von 1 Mbit/s und 1 000 Datensätze pro Sekunde eines Shards.

Im On-Demand-Modus teilt Kinesis Data Streams die Shards gleichmäßig auf, wenn ein Anstieg des Datenverkehrs festgestellt wird. Hash-Schlüssel, die einen höheren Anteil des eingehenden Datenverkehrs an einen bestimmten Shard weiterleiten, werden jedoch nicht erkannt und isoliert. Wenn Sie stark ungleichmäßige Partitionsschlüssel verwenden, erhalten Sie möglicherweise weiterhin Schreibausnahmen. Für solche Anwendungsfälle empfehlen wir, den Modus für bereitgestellte Kapazität zu verwenden, der granulare Shard-Splits unterstützt.

Funktionen und Anwendungsfälle im Bereitstellungsmodus

Im Bereitstellungsmodus können Sie, nachdem Sie den Datenstream erstellt haben, Ihre Shard-Kapazität mithilfe von oder dynamisch nach oben oder unten skalieren. AWS Management Console UpdateShardCountAPI Sie können Aktualisierungen vornehmen, während eine Produzenten- oder Konsumentenanwendung von Kinesis Data Streams Daten in den Stream schreibt oder aus dem Stream liest.

Der Bereitstellungsmodus eignet sich für vorhersehbaren Datenverkehr mit leicht zu prognostizierenden Kapazitätsanforderungen. Sie können den Bereitstellungsmodus verwenden, wenn Sie eine genaue Kontrolle darüber haben möchten, wie Daten auf Shards verteilt werden.

Mit einem Bereitstellungsmodus müssen Sie die Anzahl der Shards für den Datenstrom angeben. Zum Festlegen der Größe eines Datenstroms mit dem Bereitstellungsmodus benötigen Sie die folgenden Eingabewerte:

- Die durchschnittliche Größe des Datensatzes, der in den Stream geschrieben wird, in Kilobyte (KB), gerundet auf das nächste ganze Kilobyte (`average_data_size_in_KB`).
- Die Anzahl der Datensätze, die pro Sekunde in den Stream geschrieben bzw. daraus gelesen werden (`records_per_second`).
- Die Anzahl der Verbraucher, die Anwendungen von Kinesis Data Streams sind, die gleichzeitig und unabhängig voneinander Daten aus dem Stream konsumieren (`number_of_consumers`).
- Die Bandbreite für eingehende Schreiboperationen in KB (`incoming_write_bandwidth_in_KB`). Diese ist gleich `average_data_size_in_KB` multipliziert mit `records_per_second`.
- Die Bandbreite für ausgehende Leseoperationen in KB (`outgoing_read_bandwidth_in_KB`). Diese ist gleich `incoming_write_bandwidth_in_KB` multipliziert mit `number_of_consumers`.

Sie können die Anzahl der Shards (`number_of_shards`), die der Stream benötigt, mit den Eingabewerte in der folgenden Formel berechnen.

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

Im Bereitstellungsmodus kann es immer noch zu Ausnahmen beim Lese- und Schreibdurchsatz kommen, wenn Sie Ihren Datenstrom nicht für den Spitzendurchsatz konfigurieren. In diesem Fall müssen Sie Ihren Datenstrom manuell skalieren, um den Datenverkehr zu bewältigen.

Es kann auch zu Lese- und Schreibausnahmen kommen, wenn Sie einen Partitionsschlüssel verwenden, der zu einer ungleichmäßigen Datenverteilung führt, und die einem Shard zugewiesenen Datensätze dessen Grenzwerte überschreiten. Um dieses Problem im Bereitstellungsmodus zu beheben, identifizieren Sie solche Shards und teilen Sie sie manuell auf, um Ihren Datenverkehr besser zu bewältigen. Weitere Informationen finden Sie unter Resharding eines Streams.

Wechseln Sie zwischen den Kapazitätsmodi

Sie können den Kapazitätsmodus Ihres Datenstroms von On-Demand-Modus auf Bereitgestellt oder von Bereitgestellt auf On-Demand-Modus umschalten. Für jeden Datenstrom in Ihrem AWS -Konto

können Sie innerhalb von 24 Stunden zweimal zwischen den Modi „On-Demand-Kapazität“ und „Bereitgestellte Kapazität“ wechseln.

Das Umschalten zwischen den Kapazitätsmodi eines Datenstroms verursacht keine Unterbrechungen Ihrer Anwendungen, die diesen Datenstrom verwenden. Sie können weiterhin in diesen Datenstrom schreiben und aus ihm lesen. Wenn Sie zwischen den Kapazitätsmodi wechseln, entweder von On-Demand-Modus zu Bereitgestellt oder von Bereitgestellt zu On-Demand-Modus, wird der Status des Streams auf Aktualisierung gesetzt. Sie müssen warten, bis der Status des Datenstroms den Status Aktiv erreicht hat, bevor Sie seine Eigenschaften erneut ändern können.

Wenn Sie vom Bereitstellungs- zum On-Demand-Kapazitätsmodus wechseln, behält Ihr Datenstrom zunächst die Shard-Anzahl bei, die er vor der Umstellung hatte. Ab diesem Zeitpunkt überwacht Kinesis Data Streams Ihren Datenverkehr und skaliert die Anzahl der Shards dieses On-Demand-Datenstroms in Abhängigkeit von Ihrem Schreibdurchsatz.

Wenn Sie vom On-Demand-Modus in den Bereitstellungsmodus wechseln, behält Ihr Datenstrom zunächst auch die Shard-Anzahl bei, die er vor der Umstellung hatte. Ab diesem Zeitpunkt sind Sie jedoch dafür verantwortlich, die Shard-Anzahl dieses Datenstroms zu überwachen und anzupassen, um Ihrem Schreibdurchsatz gerecht zu werden.

Erstellen Sie einen Stream mit dem AWS Management Console

Sie können einen Stream mit der Kinesis Data Streams-Konsole, den Kinesis Data Streams API oder dem AWS Command Line Interface ()AWS CLI erstellen.

So erstellen Sie einen Datenstrom mit der Konsole

1. Melden Sie sich bei der an AWS Management Console und öffnen Sie die Kinesis-Konsole unter https://console.aws.amazon.com/kinesis.
2. Erweitern Sie in der Navigationsleiste die Regionsauswahl und wählen Sie eine Region aus.
3. Klicken Sie auf Create data stream (Daten-Stream erstellen).
4. Geben Sie auf der Seite Kinesis-Stream erstellen einen Namen für Ihren Datenstrom ein und wählen Sie dann entweder den Kapazitätsmodus On-Demand oder Bereitgestellt. Der Modus On-Demand ist standardmäßig ausgewählt. Weitere Informationen finden Sie unter Wählen Sie den Datenstream-Kapazitätsmodus.

Im Modus On-Demand können Sie dann Kinesis-Stream erstellen wählen, um Ihren Datenstrom zu erstellen. Im Modus Bereitgestellt müssen Sie dann die Anzahl der benötigten Shards angeben und dann Kinesis-Stream erstellen auswählen.

Auf der Seite Kinesis streams (Kinesis-Streams) wird für den Wert Status des Streams Creating (Erstellen) angezeigt, während der Stream erstellt wird. Sobald der Stream verwendet werden kann, ändert sich der Wert von Status in Active (Aktiv).

5. Wählen Sie den Namen des Streams aus. Auf der Seite Stream Details (Stream-Details) wird eine Zusammenfassung der Stream-Konfiguration zusammen mit Überwachungsinformationen angezeigt.

So erstellen Sie einen Stream mit den Kinesis Data Streams API

- Informationen zum Erstellen eines Streams mithilfe der Kinesis Data Streams finden Sie API unterErstellen Sie einen Stream mit dem APIs.

Um einen Stream mit dem zu erstellen AWS CLI

- Informationen zum Erstellen eines Streams mit dem AWS CLI finden Sie im Befehl create-stream.

Erstellen Sie einen Stream mit dem APIs

Führen Sie die folgenden Schritte aus, um Ihren Kinesis-Datenstrom zu erstellen.

Den Kinesis Data Streams Streams-Client erstellen

Bevor Sie mit Kinesis-Datenströmen arbeiten können, müssen Sie ein Client-Objekt erstellen. Der folgende Java-Code instanziiert einen Client-Builder und verwendet ihn zum Festlegen der Region, der Anmeldeinformationen und der Client-Konfiguration. Anschließend erstellt er ein Client-Objekt.

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

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

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

Weitere Informationen finden Sie unter Regionen und Endpunkte der Kinesis Data Streams im Allgemeine AWS-Referenz.

Erstellen Sie den Stream

Nachdem Sie Ihren Kinesis Data Streams Streams-Client erstellt haben, können Sie einen Stream mithilfe der Konsole oder programmgesteuert erstellen. Um einen Stream programmgesteuert zu erstellen, instanziieren Sie ein `CreateStreamRequest` Objekt und geben Sie einen Namen für den Stream an. Wenn Sie den Bereitstellungsmodus verwenden möchten, geben Sie die Anzahl der Shards an, die der Stream verwenden soll.

- On-Demand-Modus:

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

- Bereitgestellt:

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

Der Streamname bezeichnet den Stream. Der Name ist auf das AWS Konto beschränkt, das von der Anwendung verwendet wird. Er ist auch nach Region beschränkt. Das heißt, zwei Streams in zwei verschiedenen AWS Konten können denselben Namen haben, und zwei Streams in demselben AWS Konto, aber in zwei verschiedenen Regionen können denselben Namen haben, aber nicht zwei Streams auf demselben Konto und in derselben Region.

Der Durchsatz des Streams hängt von der Anzahl der Shards ab. Für einen höheren bereitgestellten Durchsatz benötigen Sie mehr Shards. Mehr Shards erhöhen auch die Kosten, die für den Stream AWS anfallen. Weitere Informationen zum Berechnen einer ausreichenden Anzahl von Shards für Ihre Anwendung finden Sie unter Wählen Sie den Datenstream-Kapazitätsmodus.

Nachdem Sie das `createStreamRequest` Objekt konfiguriert haben, erstellen Sie einen Stream, indem Sie die `createStream` Methode auf dem Client aufrufen. Warten Sie nach dem Aufrufen von `createStream`, bis der Stream den Status `ACTIVE` erreicht hat, bevor Sie eine Operation auf dem Stream ausführen. Rufen Sie die `describeStream`-Methode auf, um den Status des

Streams zu überprüfen. Allerdings löst `describeStream` eine Ausnahme aus, wenn der Stream nicht existiert. Aus diesem Grund müssen Sie den Aufruf `describeStream` in einen `try/catch`-Block einschließen.

```
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" );
}
```

Aktualisieren Sie einen Stream

Sie können die Details eines Streams mit der Kinesis Data Streams-Konsole, den Kinesis Data Streams API oder dem aktualisieren. AWS CLI

> **Note**
>
> Sie können die serverseitige Verschlüsselung für vorhandene Streams oder für vor kurzem erstellte Streams aktivieren.

Verwenden der Konsole

So aktualisieren Sie einen Daten-Stream mit der Konsole

1. Öffnen Sie die Amazon Kinesis Kinesis-Konsole unter https://console.aws.amazon.com/kinesis/.
2. Erweitern Sie in der Navigationsleiste die Regionsauswahl und wählen Sie eine Region aus.
3. Wählen Sie den Namen des Streams in der Liste. Auf der Seite Stream Details (Stream-Details) werden eine Zusammenfassung der Stream-Konfiguration und Überwachungsinformationen angezeigt.
4. Wählen Sie auf der Registerkarte Konfiguration die Option Kapazitätsmodus bearbeiten aus, um für einen Datenstrom zwischen den Modi On-Demand-Kapazität und Bereitgestellte Kapazität zu wechseln. Weitere Informationen finden Sie unter Wählen Sie den Datenstream-Kapazitätsmodus.

   > **Important**
   >
   > Für jeden Datenstream in Ihrem AWS Konto können Sie innerhalb von 24 Stunden zweimal zwischen dem On-Demand-Modus und dem Bereitstellungsmodus wechseln.

5. Um für einen Datenstrom im Bereitstellungsmodus die Anzahl der Shards zu bearbeiten, wählen Sie auf der Registerkarte Konfiguration die Option Bereitgestellte Shards bearbeiten aus und geben Sie dann eine neue Shard-Anzahl ein.
6. Wählen Sie zum Aktivieren der serverseitigen Verschlüsselung von Datensätzen die Option Edit (Bearbeiten) im Abschnitt Server-side encryption (Serverseitige Verschlüsselung) aus. Wählen Sie einen KMS Schlüssel, der als Hauptschlüssel für die Verschlüsselung verwendet werden soll, oder verwenden Sie den Standard-Masterschlüssel aws/kinesis, der von Kinesis verwaltet wird. Wenn Sie die Verschlüsselung für einen Stream aktivieren und Ihren eigenen AWS KMS Masterschlüssel verwenden, stellen Sie sicher, dass Ihre Producer- und Consumer-Anwendungen Zugriff auf den von Ihnen verwendeten AWS KMS Masterschlüssel haben. Informationen zum Zuweisen von Berechtigungen zu Anwendungen für den Zugriff auf einen

benutzergenerierten AWS KMS -Schlüssel finden Sie unter the section called “Berechtigungen zur Verwendung von benutzergenerierten Schlüsseln KMS”.

7. Zum Bearbeiten des Datenaufbewahrungszeitraums wählen Sie die Option Edit (Bearbeiten) im Abschnitt Data retention period (Datenaufbewahrungszeitraum) aus und geben dann den neuen Zeitraum ein.
8. Wenn Sie benutzerdefinierte Metriken in Ihrem Konto aktiviert haben, wählen Sie die Option Edit (Bearbeiten) im Abschnitt Shard level metrics (Metriken auf Shard-Ebene) aus und geben anschließend Metriken für Ihren Stream an. Weitere Informationen finden Sie unter the section called “Überwachen Sie den Kinesis Data Streams Streams-Dienst mit CloudWatch”.

Verwenden Sie den API

Informationen zum Aktualisieren von Stream-Details mithilfe von finden Sie unter den folgenden Methoden: API

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

Verwenden Sie die AWS CLI

Informationen zum Aktualisieren eines Streams mithilfe von finden Sie in der CLIKinesis-Referenz. AWS CLI

Auflisten von Streams

Streams sind auf das AWS Konto beschränkt, das mit den AWS Anmeldeinformationen verknüpft ist, die zur Instanziierung des Kinesis Data Streams Streams-Clients verwendet wurden, sowie auf die für den Client angegebene Region. In einem AWS -Konto könnten viele Streams gleichzeitig aktiv sein.

Sie können Ihre Streams in der Konsole von Kinesis Data Streams oder programmgesteuert auflisten. Der Code in diesem Abschnitt zeigt, wie Sie alle Streams für Ihr Konto auflisten. AWS

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

In diesem Codebeispiel wird zuerst eine neue Instance von `ListStreamsRequest` erstellt und die zugehörige `setLimit`-Methode aufgerufen, um festzulegen, dass maximal 20-Streams für die einzelnen Aufrufe von `listStreams` zurückgegeben werden sollen. Wenn Sie keinen Wert für `setLimit` festlegen, gibt Kinesis Data Streams eine Anzahl von Streams kleiner oder gleich der Anzahl im Konto zurück. Der Code übergibt `listStreamsRequest` anschließend an die `listStreams`-Methode des Clients. Der Rückgabewert `listStreams` wird in einem `ListStreamsResult`-Objekt gespeichert. Der Code ruft die `getStreamNames`-Methode auf diesem Objekt auf und speichert den zurückgegebenen Stream-Namen in der Liste `streamNames`. Beachten Sie, dass Kinesis Data Streams möglicherweise weniger Streams zurückgibt als durch das angegebene Limit festgelegt, selbst wenn es mehr Streams als die im Konto und in der Region gibt. Um sicherzustellen, dass alle Streams abgerufen werden, verwenden Sie die `getHasMoreStreams`-Methode, wie im nächsten Codebeispiel beschrieben.

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

Dieser Code ruft die `getHasMoreStreams`-Methode für `listStreamsRequest` auf, um zu überprüfen, ob zusätzlich zu den im ersten Aufruf von `listStreams` zurückgegebenen Streams weitere verfügbar sind. Wenn dies der Fall ist, ruft der Code die `setExclusiveStartStreamName`-Methode mit dem Namen des letzten Streams auf, der im vorherigen Aufruf von `listStreams` zurückgegeben wurde. Die `setExclusiveStartStreamName`-Methode bewirkt, dass der nächste Aufruf von `listStreams` nach diesem Stream beginnt. Die Gruppe der von diesem Aufruf zurückgegebenen Stream-Namen wird dann zur Liste `streamNames` hinzugefügt. Dieser Vorgang wird so lange fortgesetzt, bis alle Stream-Namen in der Liste erfasst wurden.

Die von `listStreams` zurückgegebenen Streams können einen der folgenden Status aufweisen:

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

Sie können den Status eines Streams mit der `describeStream`-Methode überprüfen, wie im vorherigen Abschnitt Erstellen Sie einen Stream mit dem APIs dargestellt.

Shards auflisten

Ein Datenstrom kann einen oder mehrere Shards aufweisen. Die empfohlene Methode zum Auflisten oder Abrufen der Shards aus einem Datenstream ist die Verwendung von. ListShardsAPI Das folgende Beispiel zeigt, wie Sie eine Liste der Shards in einem Datenstrom erhalten. Eine vollständige Beschreibung der in diesem Beispiel verwendeten Hauptoperation und aller Parameter, die Sie für die Operation festlegen können, finden Sie unter. 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());
        }
    }
}
```

Um das vorherige Codebeispiel auszuführen, können Sie eine POM Datei wie die folgende verwenden.

```
<?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>
```

Mit dem `ListShards` API können Sie den ShardFilterParameter verwenden, um die Antwort von herauszufilternAPI. Sie können jeweils nur einen Filter angeben.

Wenn Sie den `ShardFilter` Parameter beim Aufrufen von verwenden ListShardsAPI, `Type` ist das die erforderliche Eigenschaft und muss angegeben werden. Wenn Sie die Typen `AT_TRIM_HORIZON`, `FROM_TRIM_HORIZON`, oder `AT_LATEST` angeben, müssen Sie weder die optionalen Eigenschaften `ShardId` noch `Timestamp` angeben.

Wenn Sie den `AFTER_SHARD_ID`-Typ angeben, müssen Sie auch den Wert für die optionale `ShardId`-Eigenschaft angeben. Die `ShardId` Eigenschaft ist in ihrer Funktionalität identisch mit dem `ExclusiveStartShardId` Parameter von. ListShards API Wenn eine Eigenschaft `ShardId` angegeben ist, enthält die Antwort die Shards, beginnend mit dem Shard, dessen ID unmittelbar auf das von Ihnen angegebene `ShardId` folgt.

Wenn Sie den Typ `AT_TIMESTAMP` oder `FROM_TIMESTAMP_ID` angeben, müssen Sie auch den Wert für die optionale `Timestamp`-Eigenschaft angeben. Wenn Sie den Typ `AT_TIMESTAMP` angeben, werden alle Shards zurückgegeben, die zum angegebenen Zeitstempel geöffnet waren. Wenn Sie den `FROM_TIMESTAMP` Typ angeben, werden alle Shards, beginnend mit dem angegebenen Zeitstempel bis, zurückgegebenTIP.

Important

`DescribeStreamSummary`und `ListShard` APIs bieten eine skalierbarere Methode zum Abrufen von Informationen über Ihre Datenströme. Insbesondere die Kontingente für DescribeStream API können zu Drosselungen führen. Weitere Informationen finden Sie unter Kontingente und -Einschränkungen. Beachten Sie auch, dass `DescribeStream` Kontingente von allen Anwendungen gemeinsam genutzt werden, die mit allen Datenströmen in Ihrem AWS Konto interagieren. Die Kontingente für sind dagegen spezifisch für einen einzelnen Datenstrom. ListShards API Sie werden also nicht nur höher TPS ListShards API, sondern die Aktion skaliert auch besser, je mehr Datenströme Sie erstellen.
Wir empfehlen Ihnen, all Ihre Produzenten und Verbraucher DescribeStream API zu migrieren, die das aufrufen, stattdessen das DescribeStreamSummary und das ListShard APIs aufrufen. Um diese Hersteller und Verbraucher zu identifizieren, empfehlen wir, Athena zu verwenden, um CloudTrail Protokolle als Benutzeragenten für die Anrufe zu analysieren KPL und KCL sie bei den API Anrufen zu erfassen.

```
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
```

Wir empfehlen außerdem, die AWS Lambda- und Amazon Firehose-Integrationen mit Kinesis Data Streams, die das aufrufen, neu zu konfigurieren, sodass die `DescribeStream` API Integrationen stattdessen und aufrufen. `DescribeStreamSummary` `ListShards` Insbesondere für AWS Lambda müssen Sie Ihre Ereignisquellenzuordnung aktualisieren. Für Amazon Firehose müssen die entsprechenden IAM Berechtigungen aktualisiert werden, sodass sie die `ListShards` IAM Berechtigung enthalten.

Löschen Sie einen Stream

Sie können einen Stream mit der Konsole von Kinesis Data Streams oder programmgesteuert löschen. Um einen Stream programmgesteuert zu löschen, verwenden Sie `DeleteStreamRequest`, wie im folgenden Code gezeigt.

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

Schließen Sie alle auf dem Stream ausgeführten Anwendungen, bevor Sie ihn löschen. Wenn eine Anwendung versucht, sich auf einem gelöschten Stream zu öffnen, empfängt sie `ResourceNotFound`-Ausnahmen. Wenn Sie darüber hinaus anschließend einen neuen Stream erstellen, der denselben Namen wie Ihr vorheriger Stream hat, und Anwendungen, die auf dem vorherigen Stream ausgeführt wurden, weiterhin ausgeführt werden, versuchen diese Anwendungen möglicherweise, mit dem neuen Stream zu interagieren, so als wäre es der vorherige Stream – was zu einem unerwarteten Verhalten führen könnte.

Einen Stream erneut teilen

> **Important**
>
> Sie können Ihren Stream mit dem neu teilen. UpdateShardCountAPI Andernfalls können Sie auch weiterhin Teilungen und Zusammenführungen ausführen, wie hier beschrieben.

Amazon Kinesis Data Streams unterstützt das Resharding, das Ihnen ermöglicht, die Anzahl der Shards in Ihrem Stream zur Anpassung an Änderungen an der Durchflussrate der Daten im Stream anzupassen. Resharding ist eine erweiterte Operation. Wenn Sie noch keine Erfahrung mit Kinesis Data Streams haben, machen Sie sich zuerst mit allen anderen Aspekten von Kinesis Data Streams vertraut, bevor Sie zu diesem Thema zurückkehren.

Derzeit gibt es zwei Arten von Resharding-Vorgängen: Shard-Teilungen und Shard-Zusammenführungen. Bei einer Shard-Teilung wird ein einzelner Shard in zwei Shards geteilt. Bei einer Shard-Zusammenführung werden zwei Shards in einem einzelnen Shard zusammengeführt. Resharding erfolgt stets paarweise, d. h. dass in einem einzelnen Vorgang keine Teilungen in mehr als zwei Shards möglich sind und dass in einem einzelnen Vorgang keine Zusammenführungen von mehr als zwei Shards möglich sind. Der Shard oder das Shard-Paar, auf dem der Resharding-Vorgang ausgeführt wird, wird als übergeordneter Shard bezeichnet. Der Shard oder das Shard-Paar, der/das aus dem Resharding-Vorgang resultiert, wird als untergeordneter Shard bezeichnet.

Durch Teilungen erhöht sich die Anzahl der Shards in Ihrem Stream und damit die Datenkapazität des Streams. Da Gebühren pro Shard anfallen, erhöhen Teilungen die Kosten für Ihren Stream. Gleichermaßen verringern Zusammenführungen die Anzahl der Shards in Ihrem Stream und damit die Datenkapazität – und Kosten – des Streams.

Das Resharding wird üblicherweise von einer administrativen Anwendung ausgeführt.
Diese unterscheiden sich von den Produzentenanwendungen (zum Senden) und den Konsumentenanwendungen (zum Abrufen). Eine solche Verwaltungsanwendung überwacht die Gesamtleistung des Streams auf der Grundlage von Metriken, die von Amazon bereitgestellt werden, CloudWatch oder auf der Grundlage von Metriken, die von Herstellern und Verbrauchern gesammelt wurden. Die Verwaltungsanwendung benötigt außerdem ein breiteres Spektrum an IAM Berechtigungen als die der Verbraucher oder Produzenten, da die Verbraucher und Produzenten in der Regel keinen Zugriff auf die für das Resharding APIs verwendeten Daten benötigen sollten. Weitere Informationen zu IAM Berechtigungen für Kinesis Data Streams finden Sie unterSteuern des Zugriffs auf Amazon Kinesis Data Streams Streams-Ressourcen mithilfe IAM.

Weitere Informationen zu Resharding finden Sie unter Wie ändere ich die Anzahl der offenen Shards in Kinesis Data Streams?

Themen

Entscheiden Sie sich für eine Strategie für das Resharding

Der Zweck des Reshardings in Amazon Kinesis Data Streams besteht darin, Ihrem Stream die Anpassung an Änderungen in der Durchflussrate von Daten zu ermöglichen. Sie teilen Shards, um die Kapazität (und Kosten) Ihres Streams zu erhöhen. Sie führen Shards zusammen, um die Kosten (und Kapazität) Ihres Streams zu verringern.

Ein Ansatz für das Resharding besteht darin, jeden Shard im Stream zu teilen – wodurch sich die Kapazität des Streams verdoppeln würde. Dadurch kann jedoch mehr zusätzliche Kapazität bereitgestellt werden, als Sie tatsächlich benötigen, was unnötige Kosten verursachen würde.

Sie können darüber hinaus Metriken verwenden, um zu ermitteln, welche Ihre heißen oder kalten Shards sind, d. h. Shards, die viel mehr bzw. viel weniger Daten empfangen als erwartet. Sie könnten dann selektiv die heißen Shards teilen, um die Kapazität für die Hash-Schlüssel zu erhöhen, die auf solche Shards abzielen. Gleichermaßen könnten Sie kalte Shards zusammenführen, um deren ungenutzte Kapazitäten besser zu nutzen.

Sie können einige Leistungsdaten für Ihren Stream aus den CloudWatch Amazon-Metriken abrufen, die Kinesis Data Streams veröffentlicht. Sie können jedoch auch eigene Metriken für Ihre Streams erfassen. Ein Ansatz wäre die Protokollierung der Hash-Schlüsselwerte, die von den Partitionsschlüsseln für Ihre Datensätze generiert wurden. Beachten Sie, dass Sie den Partitionsschlüssel zu dem Zeitpunkt festlegen, an dem Sie den Datensatz zu dem Stream hinzufügen.

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

Kinesis Data Streams verwendet MD5, um den Hash-Schlüssel aus dem Partitionsschlüssel zu berechnen. Da Sie den Partitionsschlüssel für den Datensatz angeben, könnten Sie ihn verwenden, MD5 um den Hash-Schlüsselwert für diesen Datensatz zu berechnen und zu protokollieren.

Sie könnten auch die IDs Shards protokollieren, denen Ihre Datensätze zugewiesen sind. Die Shard-ID ist verfügbar über die `getShardId`-Methode des `putRecordResults`-Objekts, das von der `putRecords`-Methode zurückgegeben wurde, und des `putRecordResult`-Objekts, das von der `putRecord`-Methode zurückgegeben wurde.

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

Anhand des Shards IDs und der Hash-Schlüsselwerte können Sie feststellen, welche Shards und Hash-Schlüssel den meisten oder den wenigsten Traffic erhalten. Anschließend können Sie mithilfe von Resharding mehr oder weniger Kapazitäten bereitstellen, wie für diese Schlüssel angemessen.

Teilen Sie einen Shard

Um einen Shard in Amazon Kinesis Data Streams zu teilen, müssen Sie festlegen, wie Hash-Schlüsselwerte aus dem übergeordneten Shard auf die untergeordneten Shards umverteilt werden sollten. Wenn Sie einen Datensatz zu einem Stream hinzufügen, wird dieser basierend auf einem Hash-Schlüsselwert einem Shard zugeordnet. Der Hash-Schlüsselwert ist der MD5Hash des Partitionsschlüssels, den Sie für den Datensatz angeben, wenn Sie den Datensatz dem Stream hinzufügen. Datensätze mit demselben Partitionsschlüssel haben auch denselben Hash-Schlüsselwert.

Die möglichen Hash-Schlüsselwerte für einen bestimmten Shard bilden eine Reihe von geordneten fortlaufenden nicht-negativen Ganzzahlen. Diese Reihe möglicher Hash-Schlüsselwerte ergibt sich wie folgt:

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

Wenn Sie die Shard teilen, legen Sie einen Wert in diesem Bereich fest. Dieser Hash-Schlüsselwert und alle höheren Hash-Schlüsselwerte werden auf eine der untergeordneten Shards verteilt. Alle niedrigeren Hash-Schlüsselwerte werden an die andere untergeordnete Shard verteilt.

Der folgende Code zeigt einen Shard -Teilungsvorgang, bei dem die Hash-Schlüssel gleichmäßig zwischen den untergeordneten Shards verteilt werden, wobei der übergeordnete Shard im Wesentlichen in zwei Hälften geteilt wird. Dies ist nur eine Möglichkeit, den übergeordneten Shard

zu teilen. Sie könnten beispielsweise den Shard so teilen, dass das untere Drittel des Schlüssels aus dem übergeordneten Shard an einen untergeordneten Shard und die zwei oberen Drittel der Schlüssel an einen anderen untergeordneten Shard übergehen. Bei vielen Anwendungen ist jedoch das Teilen von Shards in zwei Hälften ein effektiver Ansatz.

Der Code setzt voraus, dass `myStreamName` den Namen Ihres Streams enthält und dass die Objektvariable `shard` den zu teilenden Shard enthält. Beginnen Sie mit der Instanziierung eines neuen `splitShardRequest`-Objekts und dem Festlegen des Stream-Namens und der Shard-ID.

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

Ermitteln Sie den Hash-Schlüsselwert, der genau zwischen dem niedrigsten und dem höchsten Wert im Shard liegt. Dies ist der Hash-Startschlüsselwert für den untergeordneten Shard, der die obere Hälfte der Hash-Schlüssel aus dem übergeordneten Shard enthält. Geben Sie diesen Wert in der `setNewStartingHashKey`-Methode an. Sie müssen nur diesen Wert angeben. Kinesis Data Streams verteilt automatisch die Hash-Schlüssel unter diesem Wert an den anderen untergeordneten Shard, der bei dieser Teilung erstellt wird. Der letzte Schritt besteht darin, die `splitShard`-Methode auf dem Kinesis-Data-Streams-Client aufzurufen.

```
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);
```

Der erste Schritt nach diesem Verfahren wird unter Warten Sie, bis ein Stream wieder aktiv wird veranschaulicht.

Zwei Shards zusammenführen

Bei einem Shard-Zusammenführungsvorgang werden zwei festgelegte Shards in einem einzelnen Shard kombiniert. Nach der Zusammenfassung empfängt der einzelne untergeordnete Shard Daten für alle Hash-Schlüsselwerte, die in den zwei übergeordneten Shards enthalten sind.

Shard-Nachbarschaft

Um zwei Shards zusammenführen zu können, müssen die Shards benachbart sein. Zwei Shards gelten als benachbart, wenn der Verband der Hash-Schlüsselbereiche für die beiden Shards eine fortlaufende Reihe ohne Lücken bildet. Wenn Sie zum Beispiel über zwei Shards verfügen, von denen einer einen Hash-Schlüsselbereich von 276 bis 381 und der andere einen Hash-Schlüsselbereich von 382 bis 454 aufweist, dann könnten Sie diese beiden Shards zu einem einzigen Shard zusammenführen, der über einen Hash-Schlüsselbereich von 276 bis 454 verfügen würde.

Wenn Sie in einem anderen Beispiel über zwei Shards verfügen, von denen einer einen Hash-Schlüsselbereich von 276 bis 381 und der andere einen Hash-Schlüsselbereich von 455 bis 560 aufweist, dann könnten Sie diese beiden Shards nicht zusammenführen, weil ein oder mehr weitere Shards zwischen diesen beiden liegen würden, die den Bereich von 382 bis 454 abdecken.

Der Satz aller `OPEN` Shards in einem Stream — als Gruppe — umfasst immer den gesamten Bereich der Hash-Schlüsselwerte. MD5 Weitere Informationen zu Shard-Status – wie beispielsweise `CLOSED` – finden Sie unter Denken Sie an Datenrouting, Datenpersistenz und den Shard-Status nach einem Reshard.

Zum Identifizieren von Shards, die zum Zusammenführen infrage kommen, sollten Sie alle Shards im Status `CLOSED` herausfiltern. Shards mit dem Status `OPEN` – also nicht `CLOSED` – weisen die Sequenzendzahl `null` auf. Sie können die Sequenzendzahl für einen Shard folgendermaßen testen:

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

Nachdem die geschlossenen Shards herausgefiltert wurden, sortieren Sie die verbleibenden Shards nach dem höchsten Hash-Schlüsselwert, der von den Shards jeweils unterstützt wird. Sie können diesen Wert folgendermaßen abrufen:

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

Wenn zwei Shards in dieser gefilterten, sortierten Liste benachbart sind, können sie zusammengeführt werden.

Code für den Zusammenführungsvorgang

Mit dem folgenden Code lassen sich zwei Shards zusammenführen. Der Code setzt voraus, dass `myStreamName` den Namen Ihres Streams enthält und dass die Objektvariablen `shard1` und `shard2` die zwei benachbarten zusammenzuführenden Shards enthalten.

Beginnen Sie für den Zusammenführungsvorgang mit der Instanziierung eines neuen `mergeShardsRequest`-Objekts. Geben Sie den Stream-Namen mit der `setStreamName`-Methode an. Geben Sie anschließend die zwei zusammenzuführenden Shards mithilfe der Methoden `setShardToMerge` und `setAdjacentShardToMerge` an. Rufen Sie anschließend die Methode `mergeShards` auf dem Client von Kinesis Data Streams aus, um den Vorgang auszuführen.

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

Der erste Schritt nach diesem Verfahren wird unter Warten Sie, bis ein Stream wieder aktiv wird veranschaulicht.

Schließen Sie die Resharding-Aktion ab

Nach allen Resharding-Vorgängen in Amazon Kinesis Data Streams und vor dem Fortsetzen der normalen Datensatzverarbeitung sind weitere Verfahren und Überlegungen erforderlich. In den folgenden Abschnitten werden diese beschrieben.

Themen

- Warten Sie, bis ein Stream wieder aktiv wird
- Denken Sie an Datenrouting, Datenpersistenz und den Shard-Status nach einem Reshard

Warten Sie, bis ein Stream wieder aktiv wird

Nachdem Sie einen Resharding-Vorgang (entweder `splitShard` oder) aufgerufen habenmergeShards, müssen Sie warten, bis der Stream wieder aktiv wird. Der zu verwendende Code entspricht dem Code beim Warten, bis ein Stream nach dem Erstellen eines Streams aktiviert wird. Dieser Code sieht wie folgt aus:

```
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" );
}
```

Denken Sie an Datenrouting, Datenpersistenz und den Shard-Status nach einem Reshard

Kinesis Data Streams ist ein Datenstreaming-Dienst in Echtzeit. Ihre Anwendungen sollten davon ausgehen, dass Daten kontinuierlich durch die Shards in Ihrem Stream fließen. Bei einem Resharding werden Datensätze, die an die übergeordneten Shards geleitet wurden, basierend auf den Hash-Schlüsselwerten, die den Partitionsschlüsseln zu den Datensätzen zugeordnet werden, zu den untergeordneten Shards umgeleitet. Datensätze, die sich vor dem Resharding in den übergeordneten Shards befunden haben, verbleiben jedoch in diesen Shards. Die übergeordneten Shards verschwinden nicht, wenn der Reshard auftritt. Sie bleiben zusammen mit den Daten erhalten, die

sie vor dem Resharding enthielten. Auf die Datensätze in den übergeordneten Shards kann über die `getRecords` Operationen `getShardIteratorund` in den Kinesis Data Streams API oder über die Kinesis Client Library zugegriffen werden.

> **Note**
>
> Datensätze können ab dem Zeitpunkt, an dem Sie zum Stream hinzugefügt wurden, bis zum aktuellen Aufbewahrungszeitraum aufgerufen werden. Dies gilt unabhängig von Änderungen an den Shards in dem Stream in diesem Zeitraum. Weitere Informationen zum Aufbewahrungszeitraum eines Streams finden Sie unter Ändern Sie den Aufbewahrungszeitraum für Daten.

Bei einem Resharding-Vorgang geht ein übergeordneter Shard vom Status `OPEN` in den Status `CLOSED` und anschließend in den Status `EXPIRED` über.

- OPEN: Vor einer Reshard-Operation befindet sich ein übergeordneter Shard im `OPEN` Status, was bedeutet, dass Datensätze sowohl zum Shard hinzugefügt als auch aus dem Shard abgerufen werden können.
- CLOSED: Nach einer Reshard-Operation wechselt der übergeordnete Shard in einen Status. `CLOSED` Das bedeutet, dass Datensätze sind mehr zum Shard hinzugefügt werden. Die Datensätze, die zu diesem Shard hinzugefügt worden wären, werden nun stattdessen zu einem untergeordneten Shard hinzugefügt. Datensätze können jedoch weiterhin für einen begrenzten Zeitraum von dem Shard abgerufen werden.
- EXPIRED: Nach Ablauf der Aufbewahrungsfrist des Streams sind alle Datensätze im übergeordneten Shard abgelaufen und es kann nicht mehr darauf zugegriffen werden. Zu diesem Zeitpunkt geht der Shard selbst in den Status `EXPIRED` über. Aufrufe an `getStreamDescription().getShards` zum Aufzählen der Shards im Stream berücksichtigen keine Shards mit dem Status `EXPIRED` in der Liste der zurückgegebenen Shards. Weitere Informationen zum Aufbewahrungszeitraum eines Streams finden Sie unter Ändern Sie den Aufbewahrungszeitraum für Daten.

Wenn sich der Stream nach einem Resharding-Vorgang wieder im Status `ACTIVE` befindet, können Sie sofort mit dem Auslesen von Daten aus den untergeordneten Shards beginnen. Die übergeordneten Shards, die nach dem Reshard verbleiben, können jedoch immer noch Daten enthalten, die Sie noch nicht gelesen haben und die dem Stream vor dem Reshard hinzugefügt wurden. Wenn Sie Daten aus den untergeordneten Shards auslesen, bevor alle Daten aus den

übergeordneten Shards ausgelesen wurden, lesen Sie die Daten für einen bestimmten Hash-Schlüssel möglicherweise nicht in der durch die Sequenznummern der Datensätze festgelegten Reihenfolge. Unter der Annahme, dass die Reihenfolge der Daten wichtig ist, sollten Sie daher nach einem Resharding Daten aus den übergeordneten Shards immer so lange weiter auslesen, bis sie ausgeschöpft sind. Erst dann sollten Sie mit dem Auslesen von Daten aus den untergeordneten Shards beginnen. Wenn `getRecordsResult.getNextShardIterator` den Wert `null` zurückgibt, bedeutet dies, dass alle Daten im übergeordneten Shard ausgelesen wurden. Wenn Sie Daten mit der Kinesis Client Library lesen, erhalten Sie die Daten nach einem Reshard möglicherweise nicht in der richtigen Reihenfolge.

Ändern Sie den Aufbewahrungszeitraum für Daten

Amazon Kinesis Data Streams unterstützt Änderungen des Zeitraums der Datensatzaufbewahrung für einen Datenstrom. Ein Kinesis-Datenstrom ist eine sortierte Folge von Datensätzen, in die in Echtzeit geschrieben und aus denen in Echtzeit gelesen werden kann. Datensätze werden deshalb vorübergehend in Shards in Ihrem Stream gespeichert. Der Zeitraum ab dem Hinzufügen eines Datensatzes bis zu dem Zeitpunkt, an dem er nicht mehr verfügbar ist, wird als Aufbewahrungszeitraum bezeichnet. Ein Kinesis-Datenstrom speichert standardmäßig Datensätze von 24 Stunden bis zu 8 760 Stunden (365 Tage).

Sie können den Aufbewahrungszeitraum über die Kinesis Data Streams Streams-Konsole oder mithilfe der DecreaseStreamRetentionPeriodOperationen IncreaseStreamRetentionPeriodund den Vorgängen aktualisieren. Mit der Konsole von Kinesis Data Streams können Sie den Aufbewahrungszeitraum mehrerer Datenströme gleichzeitig bearbeiten. Sie können die Aufbewahrungsdauer mithilfe der IncreaseStreamRetentionPeriodOperation oder der Kinesis Data Streams Streams-Konsole auf maximal 8760 Stunden (365 Tage) erhöhen. Sie können die Aufbewahrungsdauer mithilfe der DecreaseStreamRetentionPeriodOperation oder der Kinesis Data Streams Streams-Konsole auf mindestens 24 Stunden reduzieren. Die Anforderungssyntax für beide Operationen enthält den Stream-Namen und den Aufbewahrungszeitraum in Stunden. Schließlich können Sie die aktuelle Aufbewahrungsdauer eines Streams überprüfen, indem Sie den DescribeStreamVorgang aufrufen.

Das folgende Beispiel zeigt die Änderung des Aufbewahrungszeitraums unter Verwendung der AWS CLI.

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

Kinesis Data Streams hört innerhalb von einigen Minuten nach einer Erhöhung des Aufbewahrungszeitraums damit auf, Datensätze zum alten Aufbewahrungszeitraum unzugänglich zu machen. Beispielsweise bedeutet eine Änderung des Aufbewahrungszeitraums von 24 Stunden auf 48 Stunden, dass Datensätze, die 23 Stunden und 55 Minuten vorher zum Stream hinzugefügt wurden, nach 24 Stunden weiterhin verfügbar sind.

Kinesis Data Streams macht bei einer Verringerung des Aufbewahrungszeitraums beinahe sofort die Datensätze unzugänglich, die vor dem neuen Aufbewahrungszeitraum liegen. Seien Sie daher sehr vorsichtig, wenn Sie den DecreaseStreamRetentionPeriodVorgang aufrufen.

Legen Ihren Zeitraum der Datenaufbewahrung so fest, dass im Fall von Problemen sichergestellt ist, dass Ihre Konsumenten Daten auslesen können, bevor diese auslaufen. Sie sollten alle Möglichkeiten sorgfältig prüfen, wie beispielsweise ein Problem mit Ihrer Datensatzverarbeitungslogik oder einer nachgelagerten Abhängigkeit, die für einen langen Zeitraum inaktiv ist. Der Aufbewahrungszeitraum kann als Sicherheitsnetz betrachtet werden, durch das Ihre Datenkonsumenten mehr Zeit für die Wiederherstellung haben. Die Aufbewahrungsfristen API ermöglichen es Ihnen, dies proaktiv einzurichten oder reaktiv auf betriebliche Ereignisse zu reagieren.

Für Streams mit einem Aufbewahrungszeitraum von mehr als 24 Stunden fallen zusätzliche Gebühren an. Weitere Informationen finden Sie unter Preise für Amazon Kinesis Daten-Streams.

Kennzeichnen Sie Ihre Streams in Amazon Kinesis Data Streams

Sie können den Streams, die Sie in Amazon Kinesis Data Streams erstellen, eigene Metadaten in Form von Tags zuweisen. Ein Tag ist ein Schlüssel-Wert-Paar, das Sie für einen Stream definieren. Die Verwendung von Tags ist eine einfache und dennoch leistungsstarke Methode, um AWS Ressourcen zu verwalten und Daten, einschließlich Rechnungsdaten, zu organisieren.

Inhalt

Lesen Sie die Grundlagen von Stichwörtern

Sie verwenden die Kinesis Data Streams-Konsole oder Kinesis Data Streams AWS CLI, API um die folgenden Aufgaben auszuführen:

- Hinzufügen von Tags zu einem Stream
- Auflisten der Tags für Ihre Streams
- Entfernen von Tags von einem Stream

Sie können Tags verwenden, um Ihre Streams zu kategorisieren. Sie können Streams beispielsweise nach Zweck, Inhaber oder Umgebung kategorisieren. Da Sie für jeden Tag den Schlüssel und Wert definieren, können Sie eine auf benutzerdefinierte Reihe von Kategorien anlegen, die Ihren jeweiligen Anforderungen gerecht wird. Sie könnten zum Beispiel eine Reihe von Tags definieren, mit der Sie Streams nach Inhaber und zugehöriger Anwendung nachverfolgen können. Im Folgenden finden Sie einige Beispiele für Tags:

- Projekt: Projektname
- Inhaber: Name
- Zweck: Belastungstest
- Anwendung: Anwendungsname
- Umgebung: Produktion

Verfolgen Sie die Kosten mithilfe von Tagging

Sie können Tags verwenden, um Ihre AWS Kosten zu kategorisieren und nachzuverfolgen. Wenn Sie Tags auf Ihre AWS Ressourcen, einschließlich Streams, anwenden, enthält Ihr AWS Kostenzuordnungsbericht die Nutzung und die Kosten, die nach Stichwörtern zusammengefasst sind. Sie können Tags anwenden, die geschäftliche Kategorien (wie Kostenstellen, Anwendungsnamen oder Eigentümer) darstellen, um die Kosten für mehrere Services zu organisieren. Weitere Informationen finden Sie unter Verwenden von Kostenzuordnungs-Tags für benutzerdefinierte Fakturierungsberichte im AWS Billing -Benutzerhandbuch.

Verstehen Sie die Einschränkungen von Tags

Für Tags gelten die folgenden Einschränkungen.

Grundlegende Einschränkungen

- Die maximale Anzahl an Tags pro Ressource (Stream) beträgt 50.
- Bei Tag-Schlüsseln und -Werten wird zwischen Groß- und Kleinschreibung unterschieden.
- Sie können Tags für einen gelöschten Stream nicht ändern oder bearbeiten.

Einschränkungen für Tag-Schlüssel

- Jeder Tag-Schlüssel muss einmalig sein. Wenn Sie einen Tag mit einem Schlüssel hinzufügen, der bereits verwendet wird, wird das vorhandene Schlüssel-Wert-Paar durch den neuen Tag überschrieben.
- Sie können einen Tag-Schlüssel nicht mit beginnen`aws:`, da dieses Präfix für die Verwendung durch reserviert ist AWS. AWS erstellt in Ihrem Namen Tags, die mit diesem Präfix beginnen, aber Sie können sie nicht bearbeiten oder löschen.
- Tag-Schlüssel müssen zwischen 1 und 128 Unicode-Zeichen lang sein.
- Tag-Schlüssel müssen die folgenden Zeichen enthalten: Unicode-Zeichen, Ziffern, Leerzeichen sowie die folgenden Sonderzeichen: `_ . / = + - @`.

Einschränkungen für den Tag-Wert

- Tag-Werte müssen zwischen 0 und 255 Unicode-Zeichen lang sein.
- Tag-Werte können leer sein. Ansonsten müssen sie die folgenden Zeichen enthalten: Unicode-Zeichen, Ziffern, Leerzeichen und eines der folgenden Sonderzeichen: `_ . / = + - @`.

Taggen Sie Streams mit der Kinesis Data Streams Streams-Konsole

Sie können Tags mithilfe der Kinesis-Data-Streams-Konsole hinzufügen, auflisten und entfernen.

Anzeigen der Tags für einen Stream

1. Öffnen Sie die Kinesis-Data-Streams-Konsole. Erweitern Sie in der Navigationsleiste die Regionsauswahl und wählen Sie eine Region aus.
2. Wählen Sie auf der Seite Stream List (Stream-Liste) einen Stream aus.
3. Wählen Sie auf der Seite „Stream-Details“ den Tab „Tags“ aus.

Hinzufügen eines Tags zu einem Stream

1. Öffnen Sie die Kinesis-Data-Streams-Konsole. Erweitern Sie in der Navigationsleiste die Regionsauswahl und wählen Sie eine Region aus.
2. Wählen Sie auf der Seite Stream List (Stream-Liste) einen Stream aus.
3. Wählen Sie auf der Seite „Stream-Details“ den Tab „Tags“ aus.
4. Geben Sie den Tag-Schlüssel im Feld Schlüssel an, geben Sie optional einen Tag-Wert im Feld Wert an und wählen Sie dann Tag hinzufügen.

   Wenn die Schaltfläche Add Tag (Tag hinzufügen) nicht aktiviert ist, entspricht entweder der angegebene Tag-Schlüssel oder Tag-Wert nicht den Tag-Einschränkungen. Weitere Informationen finden Sie unter Verstehen Sie die Einschränkungen von Tags.
5. Um Ihr neues Tag in der Liste auf der Registerkarte „Tags“ anzuzeigen, wählen Sie das Aktualisierungssymbol.

Entfernen eines Tags von einem Stream

1. Öffnen Sie die Kinesis-Data-Streams-Konsole. Erweitern Sie in der Navigationsleiste die Regionsauswahl und wählen Sie eine Region aus.
2. Wählen Sie auf der Seite Stream List (Stream-Liste) einen Stream aus.
3. Wählen Sie auf der Seite „Stream-Details“ den Tab „Tags“ und anschließend das Symbol „Entfernen“ für das Tag aus.
4. Wählen Sie im Dialogfeld „Tag löschen“ die Option Ja, Löschen aus.

Taggen Sie Streams mit dem AWS CLI

Sie können Tags mithilfe der AWS CLI hinzufügen, auflisten und entfernen. Beispiele finden Sie in der folgenden Dokumentation.

add-tags-to-stream

Fügt Tags für den angegebenen Stream hinzu oder aktualisiert diese.

list-tags-for-stream

Listet die Tags für den angegebenen Stream auf.

remove-tags-from-stream

Entfernt Tags von dem angegebenen Stream.

Taggen von Streams mithilfe der Kinesis Data Streams API

Mit den Kinesis Data Streams API können Sie Tags hinzufügen, auflisten und entfernen. Beispiele finden Sie in der folgenden Dokumentation:

AddTagsToStream

Fügt Tags für den angegebenen Stream hinzu oder aktualisiert diese.

ListTagsForStream

Listet die Tags für den angegebenen Stream auf.

RemoveTagsFromStream

Entfernt Tags von dem angegebenen Stream.

Daten in Amazon Kinesis Data Streams schreiben

Ein Produzent ist eine Anwendung, die Daten in Amazon Kinesis Data Streams schreibt. Sie können Producer für Kinesis Data Streams mithilfe der AWS SDK for Java und der Kinesis Producer Library () KPL erstellen.

Wenn Sie Kinesis Data Streams zum ersten Mal verwenden, sollten Sie sich zunächst mit den Konzepten und der Terminologie in Was ist Amazon Kinesis Data Streams? und Verwenden Sie die AWS CLI , um Amazon Kinesis Data Streams Streams-Operationen durchzuführen vertraut machen.

> **Important**
>
> Kinesis Data Streams unterstützt Änderungen des Zeitraums der Datensatzaufbewahrung für einen Datenstrom. Weitere Informationen finden Sie unter Ändern Sie den Aufbewahrungszeitraum für Daten.

Zur Übergabe von Daten an einen Stream müssen Sie den Namen des Streams, einen Partitionsschlüssel und den Daten-Blob, der zum Stream hinzugefügt wird, angeben. Der Partitionsschlüssel wird verwendet, um zu ermitteln, zu welchem Shard im Stream der Datensatz hinzugefügt wird.

Alle Daten im Shard werden an denselben Worker gesendet, der den Shard verarbeitet. Welchen Partitionsschlüssel Sie verwenden, hängt von Ihrer Anwendungslogik ab. Die Anzahl der Partitionsschlüssel sollte in der Regel viel größer sein als die Anzahl der Shards. Der Grund hierfür ist, dass über den Partitionsschlüssel festgelegt wird, wie ein Datensatz einem bestimmten Shard zugeordnet wird. Wenn Sie genügend Partitionsschlüssel haben, können die Daten gleichmäßig auf die Shards in einem Stream verteilt werden.

Themen

- Problembehandlung Amazon Kinesis Data Streams Streams-Produzenten
- Optimieren Sie Kinesis Data Streams Streams-Produzenten

Entwickeln Sie Produzenten mithilfe der Amazon Kinesis Producer Library () KPL

Ein Produzent von Amazon Kinesis Data Streams ist eine Anwendung, die Datensätze an einen Kinesis-Datenstrom übergibt (dies wird auch als Datenerfassung bezeichnet). Die Kinesis Producer Library (KPL) vereinfacht die Entwicklung von Producer-Anwendungen und ermöglicht es Entwicklern, einen hohen Schreibdurchsatz in einen Kinesis-Datenstream zu erreichen.

Sie können das KPL mit Amazon überwachen CloudWatch. Weitere Informationen finden Sie unter Überwachen Sie die Kinesis Producer Library mit Amazon CloudWatch.

Themen

- Informieren Sie sich über die Rolle des KPL
- Machen Sie sich die Vorteile der Verwendung von bewusst KPL
- Verstehen Sie, wann Sie das nicht verwenden sollten KPL
- KPL installieren
- Umstellung auf Amazon Trust Services (ATS) -Zertifikate für KPL
- KPL Unterstützte Plattformen
- KPL Die wichtigsten Konzepte
- Integrieren Sie den KPL mit dem Herstellercode
- Schreiben Sie in Ihren Kinesis-Datenstream mit dem KPL
- Kinesis Producer Library konfigurieren
- Implementieren Sie die Deaggregation für Verbraucher
- Verwenden Sie die Firehose KPL mit Amazon Data
- Verwenden Sie das KPL mit der Schemaregistrierung AWS Glue
- Konfigurieren Sie die KPL Proxy-Konfiguration

> **Note**
>
> Es wird empfohlen, auf die neueste KPL Version zu aktualisieren. KPLwird regelmäßig mit neueren Versionen aktualisiert, die die neuesten Abhängigkeits- und Sicherheitspatches, Bugfixes und abwärtskompatible neue Funktionen enthalten. Weitere Informationen finden Sie unter /releases/. https://github.com/awslabs/ amazon-kinesis-producer

Informieren Sie sich über die Rolle des KPL

Die KPL ist eine easy-to-use hochgradig konfigurierbare Bibliothek, mit der Sie in einen Kinesis-Datenstrom schreiben können. Es fungiert als Vermittler zwischen Ihrem Producer-Anwendungscode und den Kinesis Data Streams API Streams-Aktionen. Der KPL führt die folgenden Hauptaufgaben aus:

- Schreiben in einen oder mehrere Kinesis-Datenströme mit einem automatischen und konfigurierbaren Wiederholungsmechanismus
- Sammeln von Datensätzen und Verwenden von `PutRecords`, um pro Anforderung mehrere Datensätze für mehrere Shards zu schreiben
- Aggregieren von Benutzerdatensätzen zur Erhöhung der Nutzlast und Verbesserung des Durchsatzes
- Lässt sich nahtlos in die Kinesis Client Library (KCL) integrieren, um gebündelte Datensätze auf dem Endverbraucher zu deaggregieren
- Übermittelt in Ihrem Namen CloudWatch Amazon-Metriken, um Einblick in die Leistung der Produzenten zu erhalten

Beachten Sie, dass KPL sich der von den Kinesis Data Streams unterscheidetAPI, die in der AWS SDKsverfügbar sind. Kinesis Data Streams API unterstützt Sie bei der Verwaltung vieler Aspekte von Kinesis Data Streams (einschließlich der Erstellung von Streams, Resharding sowie dem Einfügen und Abrufen von Datensätzen) und KPL bietet gleichzeitig eine Abstraktionsebene speziell für die Datenaufnahme. Informationen zu den Kinesis Data Streams API finden Sie in der Amazon Kinesis Kinesis-Referenz API.

Machen Sie sich die Vorteile der Verwendung von bewusst KPL

In der folgenden Liste sind einige der wichtigsten Vorteile aufgeführt, die sich aus der Verwendung von Kinesis Data Streams Streams-Produzenten KPL für die Entwicklung ergeben.

Der KPL kann entweder in synchronen oder asynchronen Anwendungsfällen verwendet werden. Wir empfehlen, die höhere Leistung der asynchronen Schnittstelle zu verwenden, sofern keine besonderen Gründe für die Nutzung des synchronen Verhaltens vorliegen. Weitere Informationen zu diesen beiden Anwendungsfällen sowie einen Beispiel-Code finden Sie unter Schreiben Sie in Ihren Kinesis-Datenstream mit dem KPL.

Leistungsvorteile

KPLSie können beim Aufbau von Hochleistungsproduzenten helfen. Stellen Sie sich eine Situation vor, in der Ihre EC2 Amazon-Instances als Proxy für die Erfassung von 100-Byte-Ereignissen von Hunderten oder Tausenden von Geräten mit geringem Stromverbrauch und das Schreiben von Datensätzen in einen Kinesis-Datenstrom dienen. Diese EC2 Instances müssen jeweils Tausende von Ereignissen pro Sekunde in Ihren Datenstrom schreiben. Um den nötigen Durchsatz zu erreichen, müssen Produzenten eine komplexe Logik, beispielsweise Stapelverarbeitung oder Multithreading, implementieren und auf Konsumentenseite eine Wiederholung der Logik sowie die Disaggregation der Datensätze sicherstellen. Der KPL führt all diese Aufgaben für Sie aus.

Anwenderfreundlichkeit auf Konsumentenseite

Für Entwickler auf Kundenseite, die KCL in Java verwenden, KPL lässt sich der ohne zusätzlichen Aufwand integrieren. Wenn der einen aggregierten Kinesis Data Streams Streams-Datensatz KCL abruft, der aus mehreren KPL Benutzerdatensätzen besteht, ruft er automatisch den auf, KPL um die einzelnen Benutzerdatensätze zu extrahieren, bevor sie an den Benutzer zurückgegeben werden.

Für Entwickler auf Kundenseite, die den API Vorgang nicht, KCL sondern `GetRecords` direkt verwenden, steht eine KPL Java-Bibliothek zur Verfügung, mit der die einzelnen Benutzerdatensätze extrahiert werden können, bevor sie an den Benutzer zurückgegeben werden.

Überwachen von Produzenten

Sie können Ihre Kinesis Data Streams Streams-Produzenten mit Amazon und dem KPL sammeln, überwachen CloudWatch und analysieren. Der KPL sendet in Ihrem Namen Durchsatz, Fehler und andere Messwerte an und ist für die Überwachung CloudWatch auf Stream-, Shard- oder Producer-Ebene konfigurierbar.

Asynchrone Architektur

Da sie Datensätze puffern KPL können, bevor sie an Kinesis Data Streams gesendet werden, zwingt sie die aufrufende Anwendung nicht, zu blockieren und auf eine Bestätigung zu warten, dass der Datensatz auf dem Server angekommen ist, bevor die Ausführung fortgesetzt wird.

Ein Aufruf zum Einfügen eines Datensatzes KPL erfolgt immer sofort und wartet nicht darauf, dass der Datensatz gesendet oder eine Antwort vom Server empfangen wird. Stattdessen wird ein `Future`-Objekt erstellt, das zu einem späteren Zeitpunkt das Ergebnis der Sendung des Datensatzes an Kinesis Data Streams empfängt. Dies ist dasselbe Verhalten wie bei asynchronen Clients in der AWS SDK.

Verstehen Sie, wann Sie das nicht verwenden sollten KPL

Dies KPL kann zu einer zusätzlichen Verarbeitungsverzögerung von bis zu `RecordMaxBufferedTime` innerhalb der Bibliothek führen (vom Benutzer konfigurierbar). Größere Werte von `RecordMaxBufferedTime` führen zu einer schnelleren Verpackung und einer besseren Leistung. Anwendungen, die diese zusätzliche Verzögerung nicht tolerieren können, müssen die AWS SDK möglicherweise direkt verwenden. Weitere Informationen zur Verwendung von AWS SDK mit Kinesis Data Streams finden Sie unterEntwickeln Sie Produzenten, die Amazon Kinesis Data Streams verwenden, API mit dem AWS SDK for Java. Weitere Informationen über `RecordMaxBufferedTime` und andere vom Benutzer konfigurierbare Eigenschaften von finden Sie KPL unter. Kinesis Producer Library konfigurieren

KPL installieren

Amazon stellt vorgefertigte Binärdateien der C++ Kinesis Producer Library (KPL) für macOS, Windows und aktuelle Linux-Distributionen bereit (Einzelheiten zu den unterstützten Plattformen finden Sie im nächsten Abschnitt). Diese Binärdateien sind Teil der JAR-Dateien von Java und werden automatisch aufgerufen und verwendet, wenn Sie das Paket mit Maven installieren. Verwenden Sie die folgenden Maven-SuchlinksKCL, um die neuesten Versionen von KPL und zu finden:

- KPL
- KCL

Die Linux-Binärdateien wurden mit der GNU Compiler Collection (GCC) kompiliert und statisch gegen libstdc++ unter Linux gelinkt. Sie werden voraussichtlich von allen 64-Bit-Linux-Distributionen unterstützt, in denen glibc in der Version 2.5 oder höher enthalten ist.

Benutzer älterer Linux-Distributionen können sie KPL mithilfe der Bauanweisungen erstellen, die zusammen mit dem Quellcode auf bereitgestellt werden. GitHub Informationen zum Herunterladen KPL von GitHub finden Sie in der Kinesis Producer Library.

Umstellung auf Amazon Trust Services (ATS) -Zertifikate für KPL

Am 9. Februar 2018 um 9:00 Uhr PST installierte Amazon Kinesis Data Streams ATS Zertifikate. Um weiterhin Datensätze mit der Kinesis Producer Library (KPL) in Kinesis Data Streams schreiben zu können, müssen Sie Ihre Installation auf Version 0.12.6 KPL oder höher aktualisieren. Diese Änderung wirkt sich auf alle Regionen aus. AWS

Informationen zur Umstellung auf ATS eine eigene Zertifizierungsstelle finden Sie unter So bereiten AWS Sie sich auf die Umstellung auf eine eigene Zertifizierungsstelle vor.

Falls Sie Probleme haben und technischen Support benötigen, erstellen Sie einen Fall im AWS Support Center.

KPL Unterstützte Plattformen

Die Kinesis Producer Library (KPL) ist in C++ geschrieben und wird als untergeordneter Prozess des Hauptbenutzerprozesses ausgeführt. Vorkompilierte native 64-Bit-Binärdateien sind in der Java-Version enthalten und werden vom Java-Wrapper verwaltet.

Das Java-Paket kann auf folgenden Betriebssystemen ausgeführt werden, ohne dass zusätzliche Bibliotheken installiert werden müssen:

- Linux-Distributionen mit Kernel 2.6.18 (September 2006) und höher
- Apple OS X 10.9 und höher
- Windows Server 2008 und höher

> **Important**
>
> Windows Server 2008 und höher wird für alle KPL Versionen bis Version 0.14.0 unterstützt. Die Windows-Plattform wird ab KPL Version 0.14.0 oder höher NOT unterstützt.

Beachten Sie, dass das nur 64-Bit KPL ist.

Quellcode

Wenn die in der KPL Installation bereitgestellten Binärdateien für Ihre Umgebung nicht ausreichen, KPL wird der Kern von als C++-Modul geschrieben. Der Quellcode für das C++-Modul und die Java-Schnittstelle sind unter der Amazon Public License veröffentlicht und GitHub in der Kinesis Producer Library verfügbar. Obwohl die auf jeder Plattform verwendet werden KPL können, für die ein aktueller

standardkonformer C++-Compiler verfügbar ist, unterstützt Amazon offiziell keine Plattform, die nicht auf der Liste der unterstützten Plattformen steht. JRE

KPL Die wichtigsten Konzepte

Die folgenden Abschnitte enthalten Konzepte und Terminologie, die erforderlich sind, um die Kinesis Producer Library zu verstehen und von ihr zu profitieren (KPL).

Themen

Datensätze

In diesem Handbuch unterscheiden wir zwischen KPLBenutzerdatensätzen und Kinesis Data Streams Streams-Datensätzen. Wenn wir den Begriff Datensatz ohne Qualifier verwenden, beziehen wir uns auf einen KPLBenutzerdatensatz. Wenn wir uns auf einen Datensatz von Kinesis Data Streams beziehen, sagen wir ausdrücklich Datensatz von Kinesis Data Streams.

Ein KPL Benutzerdatensatz ist ein Datenblock, der für den Benutzer eine besondere Bedeutung hat. Beispiele hierfür sind ein JSON Blob, der ein Benutzeroberflächenereignis auf einer Website darstellt, oder ein Protokolleintrag von einem Webserver.

Ein Kinesis Data Streams Streams-Datensatz ist eine Instanz der `Record` Datenstruktur, die vom Kinesis Data Streams Streams-Dienst definiert wird. API Er enthält eine Sequenznummer, einen Partitionsschlüssel und einen Daten-Blob.

Stapelverarbeitung

Unter Stapelverarbeitung versteht man das Durchführen einer einzelnen Aktion für mehrere Elemente gleichzeitig (im Gegensatz zur Ausführung ein und derselben Aktion für jedes Element separat).

In diesem Kontext steht „Element“ für einen Datensatz, der durch die Aktion an Kinesis Data Streams gesendet wird. In einer Situation ohne Batchverarbeitung würden Sie jeden Datensatz in einem separaten Kinesis Data Streams-Datensatz platzieren und eine HTTP Anfrage stellen, um ihn an Kinesis Data Streams zu senden. Bei der Batchverarbeitung kann jede HTTP Anfrage mehrere Datensätze enthalten, anstatt nur einen.

Der KPL unterstützt zwei Arten der Batchverarbeitung:

- Aggregation – Speichern mehrerer Datensätze in einem einzelnen Datensatz von Kinesis Data Streams.
- Erfassung — Verwenden Sie den API Vorgang`PutRecords`, um mehrere Kinesis Data Streams Streams-Datensätze an einen oder mehrere Shards in Ihrem Kinesis-Datenstream zu senden.

Die beiden KPL Batching-Arten sind so konzipiert, dass sie nebeneinander existieren und unabhängig voneinander ein- oder ausgeschaltet werden können. Standardmäßig sind beide aktiviert.

Aggregation

Aggregation bezieht sich auf die Speicherung mehrerer Datensätze in einem Datensatz von Kinesis Data Streams. Durch die Aggregation können Kunden die Anzahl der pro API Anruf gesendeten Datensätze erhöhen, wodurch der Produktionsdurchsatz effektiv erhöht wird.

Shards von Kinesis Data Streams unterstützen bis zu 1 000 Datensätze von Kinesis Data Streams pro Sekunde bzw. einen Durchsatz von 1 MB. Das Limit für Datensätze pro Sekunde von Kinesis Data Streams bindet Kunden mit Datensätzen kleiner als 1 KB. Durch die Datensatzaggregation können Kunden mehrere Datensätze in einem einzelnen Datensatz von Kinesis Data Streams zusammenfassen. So können Kunden Ihren Durchsatz pro Shard verbessern.

Stellen Sie sich vor, es gibt einen Shard in der Region us-east-1, der derzeit mit einer konstanten Rate von 1.000 Datensätzen pro Sekunde ausgeführt wird. Jeder Datensatz besteht aus 512 Bytes. Mit KPL der Aggregation können Sie 1.000 Datensätze in nur 10 Kinesis Data Streams Streams-Datensätze packen und die Anzahl RPS auf 10 reduzieren (bei jeweils 50 KB).

Sammlung

Erfassung bezieht sich auf das Stapeln mehrerer Kinesis Data Streams Streams-Datensätze und deren Senden in einer einzigen HTTP Anforderung mit einem Aufruf des API Vorgangs`PutRecords`, anstatt jeden Kinesis Data Streams Streams-Datensatz in einer eigenen Anfrage zu senden. HTTP

Dies erhöht den Durchsatz im Vergleich zur Verwendung ohne Erfassung, da dadurch der Aufwand für viele separate Anfragen reduziert wird. HTTP `PutRecords` wurde speziell für diesen Zweck entwickelt.

Eine Sammlung unterscheidet sich von der Aggregation dahingehend, dass mit Gruppen von Datensätzen von Kinesis Data Streams gearbeitet wird. Die gesammelten Datensätze von Kinesis

Data Streams können trotzdem mehrere Datensätze eines Benutzers enthalten. Die Beziehung lässt sich wie folgt visualisieren:

```
record 0 --|
record 1   |        [ Aggregation ]
    ...    |--> Amazon Kinesis record 0 --|
    ...    |                              |
record A --|                              |
                                           |
    ...                  ...               |
                                           |
record K --|                              |
record L   |                              |      [ Collection ]
    ...    |--> Amazon Kinesis record C --|--> PutRecords Request
    ...    |                              |
record S --|                              |
                                           |
    ...                  ...               |
                                           |
record AA--|                              |
record BB  |                              |
    ...    |--> Amazon Kinesis record M --|
    ...    |
record ZZ--|
```

Integrieren Sie den KPL mit dem Herstellercode

Die Kinesis Producer Library (KPL) wird in einem separaten Prozess ausgeführt und kommuniziert mit Ihrem übergeordneten Benutzerprozess über. IPC Diese Architektur wird gelegentlich auch als Microservice bezeichnet und aus zwei Gründen verwendet:

1) Ihr Benutzerprozess stürzt nicht ab, auch wenn er abstürzt KPL

Ihr Prozess könnte Aufgaben enthalten, die nichts mit Kinesis Data Streams zu tun haben, und kann möglicherweise auch dann weiterlaufen, wenn der KPL Prozess abstürzt. Es ist auch möglich, dass Ihr übergeordneter Benutzerprozess den neu startet KPL und wieder voll funktionsfähig ist (diese Funktionalität ist in den offiziellen Wrappern enthalten).

Ein Beispiel hierfür ist ein Webserver, der Metriken an Kinesis Data Streams sendet. Der Server kann auch noch Seiten bereitstellen, wenn die Komponente Kinesis Data Streams nicht mehr betriebsfähig

ist. Ein Absturz des gesamten Servers aufgrund eines Fehlers in der KPL würde daher zu einem unnötigen Ausfall führen.

2) Es können beliebige Clients unterstützt werden

Es gibt immer Kunden, die Sprachen verwenden, die nicht offiziell unterstützt werden. Diese Kunden sollten das auch einfach nutzen können. KPL

Matrix zur empfohlenen Verwendung

In der folgenden Verwendungsmatrix sind die empfohlenen Einstellungen für verschiedene Benutzer aufgeführt und Sie werden darüber informiert, ob und wie Sie die verwenden sollten. KPL Beachten Sie, dass bei aktivierter Aggregation auch eine Disaggregation verwendet werden muss, um die Datensätze auf der Konsumentenseite zu extrahieren.

Sprache auf Produzentenseite	Sprache auf Konsumentenseite	KCLVersion	Prüfpunkt-Logik	Können Sie die KPL verwenden?	Einschränkungen
Alles außer Java	*	*	*	Nein	N/A
Java	Java	Verwendet Java SDK direkt	N/A	Ja	Bei aktivierter Aggregation muss nach `GetRecords` -Aufrufen die bereitgestellte Disaggregationsbibliothek verwendet werden.
Java	Alles außer Java	Verwendet SDK direkt	N/A	Ja	Aggregation muss deaktiviert werden.

Sprache auf Produzent enseite	Sprache auf Konsument enseite	KCLVersion	Prüfpunkt-Logik	Können Sie die KPL verwenden?	Einschränkungen
Java	Java	1.3.x	N/A	Ja	Aggregation muss deaktiviert werden.
Java	Java	1.4.x	Prüfpunkt-Aufruf ohne Argumente	Ja	None
Java	Java	1.4.x	Prüfpunkt-Aufruf mit expliziter Sequenznummer	Ja	Entweder muss die Aggregation deaktiviert oder der Code so geändert werden, dass erweiterte Sequenznummern für das Setzen von Prüfpunkten verwendet werden.
Java	Alles außer Java	1.3.x + mehrsprachiger Daemon + sprachspezifischer Wrapper	N/A	Ja	Aggregation muss deaktiviert werden.

Schreiben Sie in Ihren Kinesis-Datenstream mit dem KPL

In den folgenden Abschnitten wird Beispielcode in der Entwicklung vom einfachsten Producer zum vollständig asynchronen Code dargestellt.

Barebones-Produzentencode

Der folgende Code reicht für die Entwicklung eines einfachen Produzenten aus. Die Benutzerdatensätze der Kinesis Producer Library (KPL) werden im Hintergrund verarbeitet.

```
// 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 ...
```

Reagieren Sie synchron auf Ergebnisse

Im vorherigen Beispiel überprüfte der Code nicht, ob die KPL Benutzeraufzeichnungen erfolgreich waren. Der KPL führt alle Wiederholungsversuche durch, die erforderlich sind, um Fehler zu berücksichtigen. Wenn Sie jedoch die Ergebnisse prüfen möchten, können Sie sich diese, wie im folgenden Beispiel gezeigt, mit den `Future`-Objekten ansehen, die von `addUserRecord` zurückgegeben werden (vorheriges Beispiel aus Kontextgründen erneut aufgeführt).

```
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
        }
    }
}
```

Reagieren Sie asynchron auf Ergebnisse

Im vorherigen Beispiel wird `get()` auf einem `Future`-Objekt aufgerufen, wodurch die Ausführung blockiert wird. Wenn Sie die Ausführung nicht blockieren möchten, können Sie einen asynchronen Callback nutzen, wie im folgenden Beispiel gezeigt:

```
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);
}
```

Kinesis Producer Library konfigurieren

Die Standardeinstellungen reichen für die meisten Anwendungsfälle aus. Bei Bedarf können Sie aber auch das Verhalten von `KinesisProducer` an Ihre Bedürfnisse anpassen. Dazu kann beispielsweise eine Instance der `KinesisProducerConfiguration`-Klasse an den `KinesisProducer`-Konstruktor übergeben werden:

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

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

Sie können zudem eine Konfiguration aus einer Eigenschaftendatei laden:

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

Sie können jeden Pfad und jeden Dateinamen ersetzen, auf den der Benutzerprozess Zugriff hat. Sie können weitere SET-Methoden auf der so erstellten `KinesisProducerConfiguration`-Instance aufrufen, um die Konfiguration anzupassen.

In der Eigenschaftendatei sollten Parameter anhand ihrer Namen in angegeben PascalCase werden. Die Namen stimmen mit denen überein, die in den SET-Methoden der `KinesisProducerConfiguration`-Klasse verwendet werden. Beispielsweise:

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

Weitere Informationen zur Verwendung von Konfigurationsparametern und zu Wertgrenzen finden Sie in der Beispieldatei mit den Konfigurationseigenschaften unter GitHub.

Beachten Sie, dass nach dem Initialisieren von `KinesisProducer` eine Änderung der verwendeten `KinesisProducerConfiguration`-Instance keine weiteren Auswirkungen mehr hat. `KinesisProducer` unterstützt derzeit keine dynamische Neukonfiguration.

Implementieren Sie die Deaggregation für Verbraucher

Ab Version 1.4.0 KCL unterstützt der die automatische Deaggregation von Benutzerdatensätzen. KPL Anwendungscode für Verbraucher, der mit früheren Versionen von geschrieben wurde, KCL wird nach dem Update von ohne Änderungen kompiliert. KCL Wenn die KPL Aggregation jedoch auf der Produzentenseite verwendet wird, gibt es eine Feinheit, die Checkpoints beinhaltet: Alle Unterdatensätze innerhalb eines aggregierten Datensatzes haben dieselbe Sequenznummer, sodass zusätzliche Daten mit dem Checkpoint gespeichert werden müssen, wenn Sie zwischen Unterdatensätzen unterscheiden müssen. Diese zusätzlichen Daten werden als Teilsequenznummer bezeichnet.

Optionen

- Migrieren Sie von früheren Versionen von KCL
- KPLVerwenden Sie Erweiterungen für die Deaggregation KCL
- GetRecordsDirekt verwenden

Migrieren Sie von früheren Versionen von KCL

Sie müssen Ihre vorhandenen Aufrufe zum Setzen eines Prüfpunkts in Verbindung mit einer Aggregation nicht ändern. Sie können nach wie vor alle Datensätze erfolgreich abrufen, die in Kinesis Data Streams gespeichert sind. Das bietet KCL jetzt zwei neue Checkpoint-Operationen zur Unterstützung bestimmter Anwendungsfälle, die im Folgenden beschrieben werden.

Falls Ihr vorhandener Code für die KCL vorherige KPL Unterstützung geschrieben wurde und Ihre Checkpoint-Operation ohne Argumente aufgerufen wird, entspricht dies dem Checkpoint der Sequenznummer des letzten KPL Benutzerdatensatzes im Batch. Wird Ihre Prüfpunkt-Operation mit einer Sequenznummernzeichenfolge aufgerufen, entspricht dies dem Checkpointing der angegebenen Sequenznummer des Stapels zusammen mit der impliziten Teilsequenznummer 0 (null).

Das Aufrufen der neuen KCL Checkpoint-Operation `checkpoint()` ohne Argumente entspricht semantisch dem Checkpointing der Sequenznummer des letzten `Record` Aufrufs im Batch zusammen mit der impliziten Teilsequenznummer 0 (Null).

Der Aufruf der neuen KCL Checkpoint-Operation `checkpoint(Record record)` entspricht semantisch dem Checkpointing der Sequenznummer `Record` des angegebenen Objekts zusammen mit der impliziten Teilsequenznummer 0 (Null). Handelt es sich beim `Record`-Aufruf um einen

`UserRecord`, wird ein Checkpointing der Sequenznummer und Teilsequenznummer von `UserRecord` durchgeführt.

Durch das Aufrufen der neuen KCL Checkpoint-Operation `checkpoint(String sequenceNumber, long subSequenceNumber)` wird die angegebene Sequenznummer zusammen mit der angegebenen Teilsequenznummer explizit überprüft.

In jedem dieser Fälle KCL können sie, nachdem der Checkpoint in der Amazon DynamoDB DynamoDB-Checkpoint-Tabelle gespeichert wurde, das Abrufen von Datensätzen korrekt fortsetzen, selbst wenn die Anwendung abstürzt und neu gestartet wird. Sind mehr Datensätze in der Sequenz enthalten, beginnt das Abrufen mit dem Datensatz, dem die nächste Teilsequenznummer zugeordnet wurde, innerhalb des Datensatzes mit der Sequenznummer, für die zuletzt ein Prüfpunkt gesetzt wurde. Enthält der letzte Prüfpunkt die allerletzte Teilsequenznummer des vorherigen Sequenznummerndatensatzes, beginnt das Abrufen mit dem Datensatz, dem die nächst folgende Sequenznummer zugeordnet ist.

Im nächsten Abschnitt wird das Sequenz- und Teilsequenz-Checkpointing für Konsumenten erläutert, bei denen ein Überspringen und Duplizieren von Datensätzen vermieden werden muss. Wenn das Überspringen (oder Duplizieren) bei einem Stopp und Neustart der Datensatzverarbeitung Ihres Konsumenten keine Rolle spielt, können Sie Ihren vorhandenen Code ohne Änderungen ausführen.

KPLVerwenden Sie Erweiterungen für die Deaggregation KCL

KPLBei der Deaggregation kann es zu einer Überprüfung von Teilsequenzen kommen. Um die Verwendung von Subsequenz-Checkpointing zu erleichtern, wurde der folgenden `UserRecord` Klasse hinzugefügt: 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 */
    }
}
```

Diese Klasse wird nun anstelle von `Record` verwendet. Sie führt nicht zu Fehlern im vorhandenen Code, da es sich um eine Subklasse von `Record` handelt. Die `UserRecord`-Klasse repräsentiert sowohl tatsächlich untergeordnete Datensätze als auch standardmäßige, nicht aggregierte Datensätze. Nicht-aggregierte Datensätze sind aggregierte Datensätze mit genau einem untergeordneten Datensatz.

Darüber hinaus wurden zwei neue Operationen zu `IRecordProcessorCheckpointer` hinzugefügt:

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

Führen Sie die folgende Konvertierung durch, um mit dem Checkpointing einer Teilsequenznummer zu beginnen: Ändern Sie folgenden Formularcode:

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

Neue Formularcode:

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

Wir empfehlen für das Checkpointing der Teilsequenz das `checkpoint(Record record)`-Formular. Wenn Sie jedoch bereits `sequenceNumbers` in Zeichenfolgen für das Checkpointing gespeichert haben, sollten Sie nun auch `subSequenceNumber` speichern, wie im folgenden Beispiel gezeigt:

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

Die Umwandlung von `Record` nach ist `UserRecord` immer erfolgreich, da die Implementierung immer verwendet. `UserRecord` Wenn Sie keine arithmetischen Operationen für die Sequenznummern durchführen müssen, ist dieser Ansatz nicht zu empfehlen.

Bei der Verarbeitung von KPL Benutzerdatensätzen KCL schreibt der die Teilsequenznummer als zusätzliches Feld für jede Zeile in Amazon DynamoDB. Frühere Versionen von wurden zum Abrufen von Datensätzen KCL bei der `AFTER_SEQUENCE_NUMBER` Wiederaufnahme von

Checkpoints verwendet. Die aktuelle Version KCL mit KPL Unterstützung verwendet stattdessen. `AT_SEQUENCE_NUMBER` Wenn der Datensatz bei der Sequenznummer abgerufen wird, bei der ein Prüfpunkt gesetzt wurde, wird die Teilsequenznummer, für die ein Checkpointing durchgeführt wurde, geprüft und untergeordnete Datensätze gegebenenfalls ausgelassen (möglicherweise alle, wenn beim letzten Datensatz ein Prüfpunkt gesetzt wurde). Nochmals: Nicht aggregierte Datensätze können als aggregierte Datensätze mit einem einzelnen untergeordneten Datensatz betrachtet werden, sodass derselbe Algorithmus sowohl für aggregierte als auch für nicht aggregierte Datensätze funktioniert.

GetRecordsDirekt verwenden

Sie können sich auch dafür entscheiden, den Vorgang nicht zu verwendenKCL, sondern den API Vorgang `GetRecords` direkt aufzurufen, um Kinesis Data Streams Streams-Datensätze abzurufen. Um diese abgerufenen Datensätze in Ihre ursprünglichen KPL Benutzerdatensätze zu entpacken, rufen Sie eine der folgenden statischen Operationen in auf: `UserRecord.java`

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

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

Die erste Operation verwendet den Standardwert `0` (null) für `startingHashKey` und den Standardwert `2^128 -1` für `endingHashKey`.

Bei jeder dieser Operationen wird die angegebene Liste von Kinesis Data Streams Streams-Datensätzen in eine Liste von KPL Benutzerdatensätzen deaggregiert. Alle KPL Benutzerdatensätze, deren expliziter Hashschlüssel oder Partitionsschlüssel außerhalb des Bereichs von `startingHashKey` (einschließlich) und (einschließlich) liegt, werden aus der `endingHashKey` zurückgegebenen Datensatzliste gelöscht.

Verwenden Sie die Firehose KPL mit Amazon Data

Wenn Sie die Kinesis Producer Library (KPL) verwenden, um Daten in einen Kinesis-Datenstream zu schreiben, können Sie die Datensätze, die Sie in diesen Kinesis-Datenstream schreiben, mithilfe der Aggregation kombinieren. Wenn Sie diesen Datenstrom dann als Quelle für Ihren Firehose-Lieferstream verwenden, deaggregiert Firehose die Datensätze, bevor sie an das Ziel geliefert werden. Wenn Sie Ihren Delivery Stream so konfigurieren, dass er die Daten transformiert, deaggregiert Firehose die Datensätze, bevor es sie zustellt. AWS Lambda Weitere Informationen finden Sie unter Schreiben zu Amazon Firehose mithilfe von Kinesis Data Streams.

Verwenden Sie das KPL mit der Schemaregistrierung AWS Glue

Sie können Ihre Kinesis-Datenströme in die AWS Glue Schema Registry integrieren. Mit der AWS Glue Schema Registry können Sie Schemas zentral erkennen, steuern und weiterentwickeln und gleichzeitig sicherstellen, dass die erstellten Daten kontinuierlich anhand eines registrierten Schemas validiert werden. Ein Schema definiert die Struktur und das Format eines Datensatzes. Ein Schema ist eine versionierte Spezifikation für zuverlässige Datenveröffentlichung, -nutzung oder -speicherung. Mit der AWS Glue Schema Registry können Sie die end-to-end Datenqualität und Datenverwaltung in Ihren Streaming-Anwendungen verbessern. Weitere Informationen finden Sie unter AWS Glue Schema Registry. Eine Möglichkeit, diese Integration einzurichten, sind die Bibliotheken KPL und Kinesis Client Library (KCL) in Java.

> **Important**
>
> Derzeit wird die Integration von Kinesis Data Streams und AWS Glue Schema-Registry nur für Kinesis-Datenstreams unterstützt, die in Java implementierte KPL Producer verwenden. Mehrsprachige Unterstützung wird nicht bereitgestellt.

Detaillierte Anweisungen zum Einrichten der Integration von Kinesis Data Streams mit Schema Registry mithilfe von finden Sie im Abschnitt „Interaktion mit Daten mithilfe der KPL KCL /-Bibliotheken“ unter Anwendungsfall: Integration von Amazon Kinesis Data Streams mit der AWS Glue Schema Registry. KPL

Konfigurieren Sie die KPL Proxy-Konfiguration

Für Anwendungen, die keine direkte Verbindung zum Internet herstellen können, unterstützen alle AWS SDK Clients die Verwendung von HTTP HTTPS Proxys. In einer typischen Unternehmensumgebung muss der gesamte ausgehende Netzwerkverkehr über Proxyserver geleitet werden. Wenn Ihre Anwendung Kinesis Producer Library (KPL) zum Sammeln und Senden von Daten AWS in eine Umgebung verwendet, die Proxyserver verwendet, benötigt Ihre Anwendung eine KPL Proxykonfiguration. KPList eine hochwertige Bibliothek, die auf der AWS Kinesis SDK aufbaut. Sie ist in einen systemeigenen Prozess und einen Wrapper aufgeteilt. Der native Prozess führt alle Aufgaben der Verarbeitung und des Sendens von Datensätzen aus, während der Wrapper den nativen Prozess verwaltet und mit ihm kommuniziert. Weitere Informationen finden Sie unter Implementieren effizienter und zuverlässiger Produzenten mit der Amazon Kinesis Producer Library.

Der Wrapper ist in Java geschrieben und der native Prozess ist in C++ unter Verwendung von SDK Kinesis geschrieben. KPLVersion 0.14.7 und höher unterstützt jetzt die Proxykonfiguration im Java-Wrapper, der alle Proxykonfigurationen an den nativen Prozess übergeben kann. Weitere Informationen finden Sie unter /releases/tag/v0.14.7. https://github.com/awslabs/ amazon-kinesis-producer

Sie können den folgenden Code verwenden, um Proxykonfigurationen zu Ihren Anwendungen hinzuzufügen. KPL

```
KinesisProducerConfiguration configuration = new KinesisProducerConfiguration();
// Next 4 lines used to configure proxy
configuration.setProxyHost("10.0.0.0"); // required
configuration.setProxyPort(3128); // default port is set to 443
configuration.setProxyUserName("username"); // no default
configuration.setProxyPassword("password"); // no default

KinesisProducer kinesisProducer = new KinesisProducer(configuration);
```

Entwickeln Sie Produzenten, die Amazon Kinesis Data Streams verwenden, API mit dem AWS SDK for Java

Sie können Producer entwickeln, die Amazon Kinesis Data Streams API mit dem AWS SDK für Java verwenden. Wenn Sie Kinesis Data Streams zum ersten Mal verwenden, sollten Sie sich zunächst mit den Konzepten und der Terminologie in Was ist Amazon Kinesis Data Streams? und Verwenden Sie die AWS CLI , um Amazon Kinesis Data Streams Streams-Operationen durchzuführen vertraut machen.

In diesen Beispielen werden die Kinesis Data Streams beschrieben API und es wird AWS SDKfor Java verwendet, um Daten zu einem Stream hinzuzufügen (abzulegen). In den meisten Anwendungsfällen sollten Sie jedoch die Kinesis Data Streams KPL Streams-Bibliothek bevorzugen. Weitere Informationen finden Sie unter Entwickeln Sie Produzenten mithilfe der Amazon Kinesis Producer Library () KPL.

Der Java-Beispielcode in diesem Kapitel zeigt, wie grundlegende Kinesis Data Streams API Streams-Operationen ausgeführt werden. Er ist logisch nach Operationstypen unterteilt. Diese Beispiele stellen keinen produktionsbereiten Code dar, d. h. es werden nicht alle möglichen Ausnahmen geprüft

und es werden nicht alle möglichen Sicherheits- oder Leistungsüberlegungen berücksichtigt. Sie können die Kinesis Data Streams auch API mit anderen Programmiersprachen aufrufen. Weitere Informationen zu allen verfügbaren AWS SDKs Produkten finden Sie unter Start Developing with Amazon Web Services.

Für jede Aufgabe gibt es Voraussetzungen. So können Sie beispielsweise erst dann Daten zu einem Stream hinzufügen, wenn Sie einen erstellt haben. Deshalb müssen Sie einen Client anlegen. Weitere Informationen finden Sie unter Kinesis-Datenstreams erstellen und verwalten.

Themen

- Daten zu einem Stream hinzufügen
- Interagieren Sie mit Daten mithilfe der AWS Glue Schema Registry

Daten zu einem Stream hinzufügen

Sobald ein Stream eingerichtet wurde, können Sie Daten in Form von Datensätzen hinzufügen. Ein Datensatz ist eine Datenstruktur, die die zu verarbeitenden Daten in Form eines Daten-Blobs enthält. Nach dem Speichern der Daten im Datensatz prüft, interpretiert und ändert Kinesis Data Streams die Daten nicht. Jeder Datensatz verfügt zudem über eine zugeordnete Sequenznummer und einen Partitionsschlüssel.

Es gibt zwei verschiedene Operationen in den Kinesis Data StreamsAPI, die Daten zu einem Stream hinzufügen, `PutRecords`und `PutRecord`. Der `PutRecords` Vorgang sendet pro HTTP Anfrage mehrere Datensätze an Ihren Stream, und der einzelne `PutRecord` Vorgang sendet Datensätze nacheinander an Ihren Stream (für jeden Datensatz ist eine separate HTTP Anforderung erforderlich). Für die meisten Anwendungen sollten Sie `PutRecords` bevorzugen, da diese Operation pro Datenproduzent einen höheren Durchsatz erzielt. Weitere Informationen zu diesen Operationen finden Sie unten in separaten Unterabschnitten.

Themen

- Fügen Sie mehrere Datensätze hinzu mit PutRecords
- Fügen Sie einen einzelnen Datensatz hinzu mit PutRecord

Denken Sie immer daran, dass Ihre Quellanwendung dem Stream mithilfe der Kinesis Data Streams Daten hinzufügtAPI, höchstwahrscheinlich eine oder mehrere Verbraucheranwendungen gleichzeitig Daten aus dem Stream verarbeiten. Informationen darüber, wie Verbraucher Daten mithilfe der Kinesis Data Streams abrufenAPI, finden Sie unterDaten aus einem Stream abrufen.

> **Important**
>
> Ändern Sie den Aufbewahrungszeitraum für Daten

Fügen Sie mehrere Datensätze hinzu mit PutRecords

Die `PutRecords`-Operation sendet in einer einzelnen Anforderung mehrere Datensätze an Kinesis Data Streams. Mit `PutRecords` erzielen Produzenten einen höheren Durchsatz, wenn sie Daten an ihren Kinesis-Datenstrom senden. Jede `PutRecords`-Anfrage kann bis zu 500 Datensätze unterstützen. Die maximale Größe jedes Datensatzes in der Anforderung beträgt 1 MB bis zu einem Limit von 5 MB für die gesamte Anforderung einschließlich Partitionsschlüsseln. Wie bei der unten beschriebenen einzelnen `PutRecord`-Operation nutzt `PutRecords` Sequenznummern und Partitionsschlüssel. Der `PutRecord`-Parameter `SequenceNumberForOrdering` ist jedoch nicht in einem `PutRecords`-Aufruf enthalten. Die `PutRecords`-Operation versucht, alle Datensätze in der natürlichen Reihenfolge der Anforderung zu verarbeiten.

Jeder Datensatz hat eine eindeutige Sequenznummer. Die Sequenznummer wird von Kinesis Data Streams zugewiesen, nachdem Sie `client.putRecords` aufgerufen haben, um die Datensätze zum Stream hinzuzufügen. Sequenznummern für denselben Partitionsschlüssel steigen in der Regel im Laufe der Zeit; je länger die Zeitdauer zwischen `PutRecords`-Anforderungen ist, desto größer wird die Sequenznummer.

> **Note**
>
> Sequenznummern können nicht als Index für Datensätze innerhalb desselben Streams verwendet werden. Nutzen Sie für die logische Unterteilung von Datensätzen Partitionsschlüssel oder erstellen Sie für jeden Datensatz einen separaten Stream.

Eine `PutRecords`-Anforderung kann Datensätze mit unterschiedlichen Partitionsschlüsseln enthalten. Der Anforderung gilt für einen Stream. Jede Anforderung kann eine beliebige Kombination aus Partitionsschlüsseln und Datenschlüsseln enthalten, sofern die Grenzwerte der Anforderung nicht überschritten werden. Anforderungen mit vielen verschiedenen Partitionsschlüsseln an Streams mit vielen verschiedenen Shards sind im Allgemeinen schneller als Anforderungen mit wenigen Partitionsschlüsseln an Streams mit nur wenigen Shards. Die Anzahl der Partitionsschlüssel sollte viel größer sein als die Anzahl der Shards, um Latenzzeiten zu verkürzen und einen maximalen Durchsatz zu erzielen.

PutRecordsBeispiel

Der folgende Code erstellt 100 Datensätze mit sequenziellen Partitionsschlüsseln und übergibt diese an einen Stream namens `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);
```

Die `PutRecords`-Antwort umfasst ein Array von `Records`-Antworten. Jeder Eintrag im Antwort-Array korreliert direkt mit einem Eintrag im Anforderungs-Array und zwar in natürlicher Reihenfolge von oben nach unten wie in der Anforderung und der Antwort. Das `Records`-Antwort-Array enthält stets die gleiche Anzahl an Datensätzen wie das Anforderungs-Array.

Behandeln Sie Fehler bei der Verwendung PutRecords

Standardmäßig führt ein Fehler bei einzelnen Datensätzen innerhalb einer Anforderung nicht zur Beendigung der Verarbeitung nachfolgender Datensätze in einer `PutRecords`-Anforderung. Das bedeutet, dass ein `Records`-Antwort-Array sowohl erfolgreich verarbeitete als auch nicht erfolgreich

verarbeitete Datensätze enthält. Sie müssen nicht erfolgreich verarbeitete Datensätze erkennen und in einen nachfolgenden Aufruf aufnehmen.

Erfolgreich verarbeitete Datensätze enthalten `SequenceNumber`- und `ShardID`-Werte, nicht erfolgreiche verarbeitete Datensätze enthalten `ErrorCode`-und `ErrorMessage`-Werte. Der `ErrorCode`-Parameter gibt den Fehlertyp an. Folgende Werte sind möglich: `ProvisionedThroughputExceededException` oder `InternalFailure`. `ErrorMessage` enthält weitere Informationen zur `ProvisionedThroughputExceededException`-Ausnahme, einschließlich Konto-ID, Stream-Name und Shard-ID des gedrosselten Datensatzes. Im folgenden Beispiel umfasst eine `PutRecords`-Anforderung drei Datensätze. Der zweite Datensatz schlägt fehl und wird in der Antwort angegeben.

Example PutRecords Syntax anfordern

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

Example PutRecords Syntax der Antwort

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

Datensätze, die nicht erfolgreich verarbeitet wurden, können in nachfolgende `PutRecords`-Anforderungen aufgenommen werden. Überprüfen Sie zuerst den Parameter `FailedRecordCount` im `putRecordsResult`, um zu bestätigen, ob Datensätze in der Anforderung fehlgeschlagen sind. Wenn dies der Fall ist, darf jeder `putRecordsEntry`, dessen `ErrorCode` nicht `null` ist, nicht in der nächsten Anforderung hinzugefügt werden. Ein Beispiel für diese Art von Handler finden Sie in folgendem Code:

Example PutRecords Fehlerhandler

```
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);
}
```

Fügen Sie einen einzelnen Datensatz hinzu mit PutRecord

Jeder Aufruf von `PutRecord` wird auf einem einzelnen Datensatz ausgeführt. Bevorzugen Sie die `PutRecords`-Operation, die in Fügen Sie mehrere Datensätze hinzu mit PutRecords beschrieben ist, es sei denn, Ihre Anwendung muss stets einzelne Datensätze per Anforderung senden oder es liegen andere Gründe vor, die gegen den Einsatz von `PutRecords` sprechen.

Jeder Datensatz hat eine eindeutige Sequenznummer. Die Sequenznummer wird von Kinesis Data Streams zugewiesen, nachdem Sie `client.putRecord` aufgerufen haben, um den Datensatz zum Stream hinzuzufügen. Sequenznummern für denselben Partitionsschlüssel steigen in der Regel im Laufe der Zeit; je länger die Zeitdauer zwischen `PutRecord`-Anforderungen ist, desto größer wird die Sequenznummer.

Wenn nacheinander viele PUT-Operationen durchgeführt werden, steigen die zurückgegebenen Sequenznummern nicht zwangsläufig an, da Kinesis Data Streams diese als gleichzeitig interpretiert. Um steigende Sequenznummern für denselben Partitionsschlüssel sicherzustellen, sollten Sie den `SequenceNumberForOrdering`-Parameter so verwenden, wie es im PutRecordBeispiel-Codebeispiel gezeigt wird.

Unabhängig davon, ob Sie `SequenceNumberForOrdering` verwenden, werden Datensätze, die Kinesis Data Streams über einen `GetRecords`-Aufruf empfängt, streng nach Sequenznummern sortiert.

> **Note**
>
> Sequenznummern können nicht als Index für Datensätze innerhalb desselben Streams verwendet werden. Nutzen Sie für die logische Unterteilung von Datensätzen Partitionsschlüssel oder erstellen Sie für jeden Datensatz einen separaten Stream.

Ein Partitionsschlüssel wird verwendet, um Daten in einem Stream zu gruppieren. Ein Datensatz wird einem Shard innerhalb des Streams basierend auf dessen Partitionsschlüssel zugewiesen. Insbesondere Kinesis Data Streams verwendet den Partitionsschlüssel als Eingabe für eine Hash-Funktion, die den Partitionsschlüssel (und die zugehörigen Daten) einem bestimmten Shard zuordnet.

Aufgrund dieses Hashing-Mechanismus werden alle Datensätze mit demselben Partitionsschlüssel zum selben Shard im Stream zugeordnet. Wenn jedoch die Anzahl der Partitionsschlüssel die Anzahl der Shards übersteigt, enthalten einige Shards zwangsläufig Datensätze mit unterschiedlichen Partitionsschlüsseln. Vom Design-Standpunkt aus sollte die Anzahl der Shards (die über die `setShardCount`-Methode von `CreateStreamRequest` angegeben wird) wesentlich geringer sein als die Anzahl der eindeutigen Partitionsschlüssel und das Datenvolumen für einen einzelnen Partitionsschlüssel sollte wesentlich geringer sein, als die Shard-Kapazität, um sicherzustellen, dass alle Shards gut ausgelastet sind.

PutRecordBeispiel

Der folgende Code erstellt 10 Datensätze, die auf zwei Partitionsschlüssel verteilt werden, und fügt diese einem Stream namens `myStreamName` hinzu.

```
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();
}
```

Beim zuvor beschriebenen Codebeispiel wird `setSequenceNumberForOrdering` eingesetzt, um eine aufsteigende Reihenfolge innerhalb der einzelnen Partitionsschlüssel zu gewährleisten. Für eine effektive Verwendung des Parameters setzen Sie die `SequenceNumberForOrdering` des aktuellen Datensatzes (Datensatz n) auf die Sequenznummer des vorhergehenden Datensatzes (Datensatz n-1). Rufen Sie `getSequenceNumber` auf dem Ergebnis von `putRecord` auf, um die Sequenznummer des Datensatzes zu erhalten, der zum Stream hinzugefügt wurde.

Der `SequenceNumberForOrdering`-Parameter stellt strikt aufsteigende Sequenznummern für denselben Partitionsschlüssel sicher. `SequenceNumberForOrdering` stellt keine Datensatzsortierung bei mehreren Partitionsschlüsseln bereit.

Interagieren Sie mit Daten mithilfe der AWS Glue Schema Registry

Sie können Ihre Kinesis-Datenströme in die AWS Glue Schema Registry integrieren. Mit der AWS Glue Schema Registry können Sie Schemas zentral erkennen, steuern und weiterentwickeln und gleichzeitig sicherstellen, dass die erstellten Daten kontinuierlich anhand eines registrierten Schemas validiert werden. Ein Schema definiert die Struktur und das Format eines Datensatzes. Ein Schema ist eine versionierte Spezifikation für zuverlässige Datenveröffentlichung, -nutzung oder -speicherung. Mit der AWS Glue Schema Registry können Sie die end-to-end Datenqualität und die Datenverwaltung in Ihren Streaming-Anwendungen verbessern. Weitere Informationen finden Sie unter AWS Glue Schema Registry. Eine Möglichkeit, diese Integration einzurichten, sind die `PutRecords` in AWS Java APIs SDK verfügbaren `PutRecord` Kinesis Data Streams.

Detaillierte Anweisungen zur Einrichtung der Integration von Kinesis Data Streams mit der Schemaregistry mithilfe von PutRecords und PutRecord Kinesis Data Streams APIs finden Sie im Abschnitt „Interaktion mit Daten mithilfe der Kinesis Data StreamsAPIs“ unter Anwendungsfall: Integration von Amazon Kinesis Data Streams mit der AWS Glue Schema Registry.

Schreiben Sie mit Kinesis Agent in Amazon Kinesis Data Streams

Kinesis Agent ist eine eigenständige Java-Anwendung, mit der Sie Daten einfach erfassen und an Kinesis Data Streams senden können. Der Agent überwacht kontinuierlich eine Reihe von Dateien und sendet neue Daten an Ihren Stream. Der Agent übernimmt die Dateirotation, das Checkpointing und die Wiederholung bei Fehlern. Er sorgt für eine zuverlässige, zeitgerechte und einfache Bereitstellung Ihrer Daten. Außerdem werden CloudWatch Amazon-Metriken ausgegeben, damit Sie den Streaming-Prozess besser überwachen und Fehler beheben können.

Standardmäßig werden Datensätze aus den einzelnen Dateien anhand des Zeilenumbruchzeichens (`'\n'`) analysiert. Der Agent kann jedoch auch für die Analyse mehrzeiliger Datensätze konfiguriert werden (siehe Geben Sie die Einstellungen für die Agentenkonfiguration an).

Sie können den Agenten in Linux-Serverumgebungen installieren, beispielsweise auf Webservern, Protokollservern und Datenbankservern. Nach der Installation des Agenten konfigurieren Sie diesen, indem Sie die zu überwachenden Dateien sowie den Stream für die Daten angeben. Nachdem der Agent konfiguriert wurde, sammelt er permanent Daten aus den Dateien und sendet sie zuverlässig an den Stream.

Themen

Erfüllen Sie die Voraussetzungen für Kinesis Agent

- Ihr Betriebssystem muss entweder Amazon Linux AMI mit Version 2015.09 oder höher oder Red Hat Enterprise Linux Version 7 oder höher sein.
- Wenn Sie Amazon verwendenEC2, um Ihren Agenten auszuführen, starten Sie Ihre EC2 Instance.
- Verwalten Sie Ihre AWS Anmeldeinformationen mit einer der folgenden Methoden:
  - Geben Sie beim Starten Ihrer EC2 Instance eine IAM Rolle an.
  - Geben Sie bei der Konfiguration des Agenten AWS Anmeldeinformationen an (siehe awsAccessKeyID und awsSecretAccessSchlüssel).
  - Bearbeiten Sie`/etc/sysconfig/aws-kinesis-agent`, um Ihre Region und Ihre AWS Zugriffsschlüssel anzugeben.
  - Wenn sich Ihre EC2 Instance in einem anderen AWS Konto befindet, erstellen Sie eine IAM Rolle, um Zugriff auf den Kinesis Data Streams Streams-Dienst zu gewähren, und geben Sie diese Rolle bei der Konfiguration des Agenten an (siehe assumeRoleARNund assumeRoleExternalID). Verwenden Sie eine der vorherigen Methoden, um die AWS Anmeldeinformationen eines Benutzers im anderen Konto anzugeben, der berechtigt ist, diese Rolle anzunehmen.
- Die IAM Rolle oder die AWS Anmeldeinformationen, die Sie angeben, müssen berechtigt sein, den Kinesis Data Streams PutRecordsStreams-Vorgang auszuführen, damit der Agent Daten an Ihren Stream senden kann. Wenn Sie die CloudWatch Überwachung für den Agenten aktivieren, ist auch eine Genehmigung zur Durchführung des CloudWatch PutMetricDataVorgangs erforderlich. Weitere Informationen finden Sie unter Steuern des Zugriffs auf Amazon Kinesis Data Streams

Streams-Ressourcen mithilfe IAMÜberwachen Sie den Zustand des Kinesis Data Streams Streams-Agenten mit Amazon CloudWatch, und CloudWatch Zugriffskontrolle.

Laden Sie den Agenten herunter und installieren Sie ihn

Stellen Sie zunächst eine Verbindung mit Ihrer Instance her. Weitere Informationen finden Sie unter Connect to Your Instance im EC2Amazon-Benutzerhandbuch. Wenn Sie Probleme mit der Verbindung haben, finden Sie weitere Informationen unter Problembehandlung beim Herstellen einer Verbindung zu Ihrer Instance im EC2Amazon-Benutzerhandbuch.

So richten Sie den Agenten mit Amazon Linux ein AMI

Verwenden Sie den folgenden Befehl zum Herunterladen und Installieren des Agenten:

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

So richten Sie den Agenten mit Red Hat Enterprise Linux ein

Verwenden Sie den folgenden Befehl zum Herunterladen und Installieren des Agenten:

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

Um den Agenten einzurichten mit GitHub

1. Laden Sie den Agenten von amazon-kinesis-agentawlabs/ herunter.
2. Installieren Sie den Agenten, indem Sie zum Download-Verzeichnis navigieren und den folgenden Befehl ausführen:

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

So richten Sie den Agenten in einem Docker-Container ein

Kinesis Agent kann auch in einem Container über die amazonlinux-Containerbasis ausgeführt werden. Verwenden Sie die folgende Docker-Datei und führen Sie dann `docker build` aus.

```
FROM amazonlinux
```

```
RUN yum install -y aws-kinesis-agent which findutils
COPY agent.json /etc/aws-kinesis/agent.json

CMD ["start-aws-kinesis-agent"]
```

Konfigurieren und starten Sie den Agenten

So konfigurieren und starten Sie den Agenten

1. Öffnen und bearbeiten Sie die Konfigurationsdatei (als Superuser, wenn Sie standardmäßige Dateizugriffsberechtigungen nutzen): `/etc/aws-kinesis/agent.json`

   Geben Sie in der Konfigurationsdatei die Dateien (`"filePattern"`) an, aus denen der Agent Daten sammelt, sowie den Namen des Streams (`"kinesisStream"`, an den der Agent die Daten sendet. Beachten Sie, dass der Dateiname ein Muster ist und der Agent Dateirotationen erkennt. Sie können nur einmal pro Sekunde Dateien rotieren oder neue Dateien erstellen. Der Agent verwendet den Zeitstempel der Dateierstellung, um die Dateien zu ermitteln, die gesammelt und zum Stream hinzugefügt werden. Wenn häufiger als einmal pro Sekunde Dateien rotiert oder neue Dateien erstellt werden, kann der Agent nicht mehr ordnungsgemäß zwischen den Dateien unterscheiden.

   ```
   {
      "flows": [
           {
               "filePattern": "/tmp/app.log*",
               "kinesisStream": "yourkinesisstream"
           }
      ]
   }
   ```

2. Starten Sie den Agenten manuell:

   ```
   sudo service aws-kinesis-agent start
   ```

3. (Optional) Konfigurieren Sie den Agenten so, dass er beim Startup des Systems gestartet wird:

   ```
   sudo chkconfig aws-kinesis-agent on
   ```

Der Agent wird jetzt als Systemdienst im Hintergrund ausgeführt. Er überwacht kontinuierlich die angegebenen Dateien und sendet Daten an den angegebenen Stream. Die Agentenaktivität wird in `/var/log/aws-kinesis-agent/aws-kinesis-agent.log` protokolliert.

Geben Sie die Einstellungen für die Agentenkonfiguration an

Der Agent unterstützt die beiden obligatorischen Konfigurationseinstellungen `filePattern` und `kinesisStream` sowie optionale Konfigurationseinstellungen für zusätzliche Funktionen. Sie können sowohl die obligatorische als auch die optionale Konfiguration in `/etc/aws-kinesis/agent.json` angeben.

Wenn Sie die Konfigurationsdatei ändern, müssen Sie den Agenten mit den folgenden Befehlen anhalten und starten:

```
sudo service aws-kinesis-agent stop
sudo service aws-kinesis-agent start
```

Alternativ können Sie auch den folgenden Befehl nutzen:

```
sudo service aws-kinesis-agent restart
```

Im Folgenden finden Sie die allgemeinen Konfigurationseinstellungen.

Konfigurationseinstellung	Beschreibung
`assumeRoleARN`	Die ARN Rolle, die der Benutzer übernehmen soll. Weitere Informationen finden Sie im IAMBenutzerhandbuch unter AWS Kontenübergreifendes Delegieren des Zugriffs mithilfe von IAM Rollen.
`assumeRoleExternalId`	Eine optionale Kennung, die festlegt, wer die Rolle übernehmen kann. Weitere Informationen finden Sie im IAMBenutzerhandbuch unter So verwenden Sie eine externe ID.
`awsAccessKeyId`	AWS Zugriffsschlüssel-ID, die die Standardanmeldedaten überschreibt. Diese Einstellung hat Vorrang vor allen anderen Anbietern von Anmeldeinformationen.

Konfigurationseinstellung	Beschreibung
`awsSecretAccessKey`	AWS geheimer Schlüssel, der die Standardanmeldedaten überschreibt. Diese Einstellung hat Vorrang vor allen anderen Anbietern von Anmeldeinformationen.
`cloudwatch.emitMetrics`	Ermöglicht dem Agenten, Metriken auszusenden, CloudWatch sofern diese Einstellung gesetzt ist (true). Standard: true
`cloudwatch.endpoint`	Der regionale Endpunkt für CloudWatch. Standard: `monitoring.us-east-1.amazonaws.com`
`kinesis.endpoint`	Der regionale Endpunkt für Kinesis Data Streams. Standard: `kinesis.us-east-1.amazonaws.com`

Im Folgenden finden Sie die Konfigurationseinstellungen für den Ablauf.

Konfigurationseinstellung	Beschreibung
`dataProcessingOptions`	Die Liste der Verarbeitungsoptionen, die auf jeden analysierten Datensatz angewendet werden, ehe ein Datensatz an den Stream gesendet wird. Die Verarbeitungsoptionen werden in der angegebenen Reihenfolge ausgeführt. Weitere Informationen finden Sie unter Verwenden Sie den Agenten zur Vorverarbeitung von Daten.
`kinesisStream`	[Erforderlich] Der Name des Streams.
`filePattern`	[Erforderlich] Das Verzeichnis- und Dateimuster, das abgeglichen werden muss, damit es vom Agenten abgerufen werden kann. Für alle Dateien, die mit diesem Muster übereinstimmen, müssen Leseberechtigungen für `aws-kinesis-agent-user` vergeben werden. Für

Konfigurationseinstellung	Beschreibung
	das die Dateien enthaltende Verzeichnis müssen Lese- und Ausführberechtigungen für `aws-kinesis-agent-user` vergeben werden.
`initialPosition`	Die Position, an der mit der Analyse der Datei begonnen wurde. Gültige Werte sind `START_OF_FILE` und `END_OF_FILE` . Standard: `END_OF_FILE`
`maxBufferAgeMillis`	Die maximale Zeit (in Millisekunden), während der durch den Agenten Daten gepuffert werden, bevor sie an den Stream gesendet werden. Wertebereich: 1.000 bis 900.000 (1 Sekunde bis 15 Minuten) Standard: 60.000 (1 Minute)
`maxBufferSizeBytes`	Die maximale Größe (in Bytes), bis zu der durch den Agenten Daten gepuffert werden, bevor sie an den Stream gesendet werden. Wertebereich: 1 bis 4.194.304 (4 MB) Standard: 4.194.304 (4 MB)
`maxBufferSizeRecords`	Die maximale Anzahl der Datensätze, für die der Agent Daten puffert, ehe diese an den Stream gesendet werden. Wertebereich: 1 bis 500 Standard: 500
`minTimeBetweenFilePollsMillis`	Das Zeitintervall (in Millisekunden), in dem der Agent die überwachten Dateien auf neue Daten abfragt und analysiert. Wertbereich: 1 oder höher Standard: 100

Konfigurationseinstellung	Beschreibung
`multiLineStartPattern`	Das Muster für die Identifizierung des Datensatzbeginns. Ein Datensatz besteht aus einer Zeile, die mit dem angegebenen Muster übereinstimmt, und allen folgenden Zeilen, die nicht dem Muster entsprechen. Gültige Werte sind reguläre Ausdrücke. Standardmäßig wird jede neue Zeile in den Protokolldateien als einziger Datensatz analysiert.
`partitionKeyOption`	Die Methode zum Erstellen des Partitionsschlüssels. Gültige Werte sind `RANDOM` (zufällig generierte Ganzzahl) und `DETERMINISTIC` (ein aus den Daten berechneter Hash-Wert). Standard: `RANDOM`
`skipHeaderLines`	Die Anzahl der Zeilen, die der Agent überspringt, ehe mit der Analyse der überwachten Dateien begonnen wird. Wertbereich: 0 oder höher Standard: 0 (null)
`truncatedRecordTerminator`	Die Zeichenfolge, mit der der Agent einen analysierten Datensatz kürzt, wenn die Datensatzgröße das zulässige Datensatz-Limit von Kinesis Data Streams überschreitet. (1,000 KB) Standard: `'\n'` (Zeilenumbruch)

Überwachen Sie mehrere Dateiverzeichnisse und schreiben Sie in mehrere Streams

Wenn Sie mehrere Ablaufkonfigurationseinstellungen angeben, können Sie den Agenten so konfigurieren, dass er mehrere Dateiverzeichnisse überwacht und Daten an verschiedene Streams sendet. Im folgenden Konfigurationsbeispiel überwacht der Agent zwei Dateiverzeichnisse und sendet Daten an einen Kinesis-Stream bzw. einen Firehose-Lieferstream. Beachten Sie, dass Sie unterschiedliche Endpunkte für Kinesis Data Streams und Firehose angeben können, sodass sich Ihr Kinesis-Stream und Ihr Firehose-Lieferstream nicht in derselben Region befinden müssen.

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

Ausführlichere Informationen zur Verwendung des Agenten mit Firehose finden Sie unter Writing to Amazon Data Firehose with Kinesis Agent.

Verwenden Sie den Agenten zur Vorverarbeitung von Daten

Der Agent kann die Datensätze vorverarbeiten, die aus den überwachten Dateien analysiert wurden, ehe diese an Ihren Stream gesendet werden. Sie können dieses Feature aktivieren, indem Sie Ihrem Dateifluss die Konfigurationseinstellung `dataProcessingOptions` hinzufügen. Sie können eine oder mehrere Verarbeitungsoptionen hinzufügen. Diese werden in der angegebenen Reihenfolge ausgeführt.

Der Agent unterstützt die folgenden aufgelisteten Verarbeitungsoptionen. Der Agent ist ein Open-Source-Tool, sodass Sie dessen Verarbeitungsoptionen optimieren und erweitern können. Sie können den Agenten von Kinesis Agent herunterladen.

Verarbeitungsoptionen

`SINGLELINE`

Konvertiert einen mehrzeiligen Datensatz in einen einzeiligen Datensatz, indem Zeilenumbruchzeichen sowie vorangestellte und folgende Leerzeichen entfernt werden.

```
{
    "optionName": "SINGLELINE"
}
```

`CSVTOJSON`

Konvertiert einen Datensatz von einem durch Trennzeichen getrennten Format in ein Format. JSON

```
{
    "optionName": "CSVTOJSON",
    "customFieldNames": [ "field1", "field2", ... ],
    "delimiter": "yourdelimiter"
}
```

`customFieldNames`

[Erforderlich] Die Feldnamen, die als Schlüssel in jedem JSON Schlüssel-Wert-Paar verwendet werden. Wenn Sie beispielsweise `["f1", "f2"]` angeben, wird der Datensatz „v1, v2“ in `{"f1":"v1","f2":"v2"}` konvertiert.

`delimiter`

Die Zeichenfolge, die als Trennzeichen im Datensatz verwendet wird. Standardmäßig wird ein Komma (,) verwendet.

`LOGTOJSON`

Konvertiert einen Datensatz von einem Protokollformat in ein JSON Format. Folgende Protokollformate werden unterstützt: Apache Common Log, Apache Combined Log, Apache Error Log und RFC3164 Syslog.

```
{
    "optionName": "LOGTOJSON",
    "logFormat": "logformat",
    "matchPattern": "yourregexpattern",
    "customFieldNames": [ "field1", "field2", … ]
}
```

`logFormat`

[Erforderlich] Das Format des Protokolleintrags. Folgende Werte sind möglich:

- `COMMONAPACHELOG` – Das Apache-Common-Log-Format. Jeder Protokolleintrag weist standardmäßig das folgende Muster auf: „`%{host} %{ident} %{authuser} [%{datetime}] \"%{request}\" %{response} %{bytes}`“.

- `COMBINEDAPACHELOG` – Das Apache-Combined-Log-Format. Jeder Protokolleintrag weist standardmäßig das folgende Muster auf: „`%{host} %{ident} %{authuser} [%{datetime}] \"%{request}\" %{response} %{bytes} %{referrer} %{agent}`“.
- `APACHEERRORLOG` – Das Apache-Error-Log-Format. Jeder Protokolleintrag weist standardmäßig das folgende Muster auf: „`[%{timestamp}] [%{module}: %{severity}] [pid %{processid}:tid %{threadid}] [client: %{client}] %{message}`“.
- `SYSLOG`— Das RFC3164 Syslog-Format. Jeder Protokolleintrag weist standardmäßig das folgende Muster auf: „`%{timestamp} %{hostname} %{program}[%{processid}]: %{message}`“.

`matchPattern`

Das reguläre Ausdrucksmuster, mit dem Werte aus den Protokolleinträgen extrahiert werden. Diese Einstellung wird verwendet, wenn Ihr Protokolleintrag nicht eines der vordefinierten Protokollformate aufweist. Bei dieser Einstellung müssen Sie auch `customFieldNames` angeben.

`customFieldNames`

Die benutzerdefinierten Feldnamen, die als Schlüssel in jedem JSON Schlüssel-Wert-Paar verwendet werden. Mit dieser Einstellung können Sie Feldnamen für Werte definieren, die aus `matchPattern` extrahiert wurden, oder die Standardfeldnamen von vordefinierten Protokollformaten überschreiben.

Example : LOGTOJSON Konfiguration

Hier ist ein Beispiel für eine `LOGTOJSON` Konfiguration für einen Apache Common Log-Eintrag, der in ein JSON Format umgewandelt wurde:

```
{
    "optionName": "LOGTOJSON",
    "logFormat": "COMMONAPACHELOG"
}
```

Vor der Konvertierung:

```
64.242.88.10 - - [07/Mar/2004:16:10:02 -0800] "GET /mailman/listinfo/hsdivision
 HTTP/1.1" 200 6291
```

Nach der Konvertierung:

```
{"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 : LOGTOJSON Konfiguration mit benutzerdefinierten Feldern

Im Folgenden ein weiteres Beispiel einer LOGTOJSON-Konfiguration:

```
{
    "optionName": "LOGTOJSON",
    "logFormat": "COMMONAPACHELOG",
    "customFieldNames": ["f1", "f2", "f3", "f4", "f5", "f6", "f7"]
}
```

Mit dieser Konfigurationseinstellung wird derselbe Apache Common Log-Eintrag aus dem vorherigen Beispiel wie folgt in JSON das Format konvertiert:

```
{"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 : Konvertieren eines Apache Common Log-Eintrags

Die folgende Flow-Konfiguration konvertiert einen Apache Common Log-Eintrag in einen einzeiligen Datensatz im JSON Format:

```
{
    "flows": [
        {
            "filePattern": "/tmp/app.log*",
            "kinesisStream": "my-stream",
            "dataProcessingOptions": [
                {
                    "optionName": "LOGTOJSON",
                    "logFormat": "COMMONAPACHELOG"
                }
```

```
            ]
        }
    ]
}
```

Example : Konvertieren mehrzeiliger Datensätze

Bei der folgenden Ablaufkonfiguration werden mehrzeilige Datensätze analysiert, deren erste Zeile mit „`[SEQUENCE=`“ beginnt. Jeder Datensatz wird in einen einzeiligen Datensatz konvertiert. Anschließend werden Werte aus dem Datensatz basierend auf einem Tabulatortrennzeichen extrahiert. Extrahierte Werte werden bestimmten `customFieldNames` Werten zugeordnet, sodass ein einzeiliger Datensatz im JSON Format entsteht.

```
{
    "flows": [
        {
            "filePattern": "/tmp/app.log*",
            "kinesisStream": "my-stream",
            "multiLineStartPattern": "\\[SEQUENCE=",
            "dataProcessingOptions": [
                {
                    "optionName": "SINGLELINE"
                },
                {
                    "optionName": "CSVTOJSON",
                    "customFieldNames": [ "field1", "field2", "field3" ],
                    "delimiter": "\\t"
                }
            ]
        }
    ]
}
```

Example : LOGTOJSON Konfiguration mit Match Pattern

Hier ist ein Beispiel für eine `LOGTOJSON` Konfiguration für einen Apache Common Log-Eintrag, der in ein JSON Format konvertiert wurde, wobei das letzte Feld (Byte) weggelassen wurde:

```
{
    "optionName": "LOGTOJSON",
    "logFormat": "COMMONAPACHELOG",
```

```
    "matchPattern": "^([\\d.]+) (\\S+) (\\S+) \\[([\\w:/]+\\s[+\\-]\\d{4})\\] \"(.
+?)\" (\\d{3})",
    "customFieldNames": ["host", "ident", "authuser", "datetime", "request",
 "response"]
}
```

Vor der Konvertierung:

```
123.45.67.89 - - [27/Oct/2000:09:27:09 -0400] "GET /java/javaResources.html HTTP/1.0"
 200
```

Nach der Konvertierung:

```
{"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"}
```

Verwenden Sie CLI Agentenbefehle

Automatisches Startup des Agenten beim Systemstart:

```
sudo chkconfig aws-kinesis-agent on
```

Prüfen des Status des Agenten:

```
sudo service aws-kinesis-agent status
```

Beenden des Agenten:

```
sudo service aws-kinesis-agent stop
```

Auslesen der Protokolldatei des Agenten von diesem Speicherort:

```
/var/log/aws-kinesis-agent/aws-kinesis-agent.log
```

Deinstallieren des Agenten:

```
sudo yum remove aws-kinesis-agent
```

FAQ

Gibt es einen Kinesis Agent für Windows?

Kinesis Agent für Windows ist eine andere Software als Kinesis Agent für Linux-Plattformen.

Warum verlangsamt sich Kinesis Agent und/oder **`RecordSendErrors`** nimmt zu?

Dies ist normalerweise auf die Drosselung durch Kinesis zurückzuführen. Überprüfen Sie die `WriteProvisionedThroughputExceeded` Metrik für Kinesis Data Streams oder die `ThrottledRecords` Metrik für Firehose Delivery Streams. Jede Erhöhung dieser Metriken von 0 zeigt an, dass die Stream-Grenzwerte erhöht werden müssen. Weitere Informationen finden Sie unter Kinesis-Datenstrom-Grenzwerte und Amazon Firehose Delivery Streams.

Sobald Sie die Drosselung ausgeschlossen haben, überprüfen Sie, ob der Kinesis Agent so konfiguriert ist, dass er eine große Menge kleiner Dateien durchsucht. Es gibt eine Verzögerung, wenn der Kinesis Agent eine neue Datei überwacht, daher sollte der Kinesis-Agent eine kleine Menge größerer Dateien überwachen. Versuchen Sie, Ihre Protokolldateien in größeren Dateien zusammenzufassen.

Warum erhalte ich **`java.lang.OutOfMemoryError`** -Ausnahmen?

Kinesis Agent verfügt nicht über genügend Arbeitsspeicher, um seinen aktuellen Workload zu bewältigen. Versuchen Sie, `JAVA_START_HEAP` und `JAVA_MAX_HEAP` in `/usr/bin/start-aws-kinesis-agent` zu erhöhen und den Agenten neu zu starten.

Warum erhalte ich **`IllegalStateException : connection pool shut down`**-Ausnahmen?

Kinesis Agent verfügt nicht über genügend Verbindungen, um seinen aktuellen Workload zu bewältigen. Versuchen Sie, `maxConnections` und `maxSendingThreads` in den allgemeinen Konfigurationseinstellungen des Agenten unter `/etc/aws-kinesis/agent.json` zu erhöhen. Der Standardwert für diese Felder ist das 12-fache der verfügbaren Laufzeitprozessoren. Weitere Informationen zu den Einstellungen für erweiterte Agentenkonfigurationen finden Sie unter AgentConfiguration.java.

Wie kann ich ein anderes Problem mit Kinesis Agent beheben?

`DEBUG`-Level-Protokolle können in `/etc/aws-kinesis/log4j.xml` aktiviert werden.

Wie sollte ich Kinesis Agent konfigurieren?

Je kleiner das `maxBufferSizeBytes`, desto häufiger sendet der Kinesis Agent Daten. Dies kann nützlich sein, da es die Lieferzeit von Datensätzen verkürzt, aber es erhöht auch die Anfragen pro Sekunde an Kinesis.

Warum sendet Kinesis Agent doppelte Datensätze?

Dies ist auf eine Fehlkonfiguration bei der Dateiüberwachung zurückzuführen. Stellen Sie sicher, dass jedes `fileFlow's filePattern` nur einer Datei entspricht. Dies kann auch auftreten, wenn der verwendete `logrotate`-Modus im `copytruncate`-Modus ist. Versuchen Sie, den Modus auf den Standard- oder Erstellungsmodus zu ändern, um Duplikate zu vermeiden. Weitere Informationen zum Umgang mit doppelten Datensätzen finden Sie unter Umgang mit doppelten Datensätzen.

Schreiben Sie mithilfe anderer AWS Dienste in Kinesis Data Streams

Die folgenden AWS Services können direkt in Amazon Kinesis Data Streams integriert werden, um Daten in Kinesis-Datenstreams zu schreiben. Überprüfen Sie die Informationen für jeden Service, an dem Sie interessiert sind, und verweisen Sie auf die bereitgestellten Referenzen.

Themen

- Schreiben Sie in Kinesis Data Streams mit AWS Amplify
- Schreiben Sie mit Amazon Aurora in Kinesis Data Streams
- Schreiben Sie mit Amazon in Kinesis Data Streams CloudFront
- Schreiben Sie mithilfe von Amazon CloudWatch Logs in Kinesis Data Streams
- Schreiben Sie mit Amazon Connect in Kinesis Data Streams
- Schreiben Sie in Kinesis Data Streams mit AWS Database Migration Service
- Schreiben Sie mit Amazon DynamoDB in Kinesis Data Streams
- Schreiben Sie mit Amazon in Kinesis Data Streams EventBridge
- Schreiben Sie in Kinesis Data Streams mit AWS IoT Core
- Schreiben Sie mit Amazon Relational Database Service in Kinesis Data Streams
- Schreiben Sie in Kinesis Data Streams Pinpoint usingAmazon

- Schreiben Sie mithilfe der Amazon Quantum Ledger Database (Amazon) in Kinesis Data Streams QLDB

Schreiben Sie in Kinesis Data Streams mit AWS Amplify

Sie können Amazon Kinesis Data Streams verwenden, um Daten aus Ihren mit AWS Amplify erstellten mobilen Anwendungen für die Echtzeitverarbeitung zu streamen. Sie können dann Echtzeit-Dashboards erstellen, Ausnahmen erfassen, Warnungen generieren, Empfehlungen erzeugen und in Echtzeit andere geschäftliche oder betriebliche Entscheidungen treffen. Sie können Daten auch an andere Dienste wie Amazon Simple Storage Service, Amazon DynamoDB und Amazon Redshift senden.

Weitere Informationen finden Sie unter Verwenden von Amazon Kinesis im AWS Amplify Developer Center.

Schreiben Sie mit Amazon Aurora in Kinesis Data Streams

Sie können Amazon Kinesis Data Streams verwenden, um Aktivitäten auf Ihren Amazon Aurora-DB-Clustern zu überwachen. Mithilfe von Database Activity Streams überträgt Ihr Aurora-DB-Cluster Aktivitäten in Echtzeit an einen Amazon Kinesis Data Streams. Anschließend können Sie Anwendungen für das Compliance-Management entwickeln, die diese Aktivitäten nutzen, prüfen und Warnmeldungen generieren. Sie können auch Amazon Firehose verwenden, um die Daten zu speichern.

Weitere Informationen finden Sie unter Datenbankaktivitätsstreams im Entwicklerhandbuch zu Amazon Aurora.

Schreiben Sie mit Amazon in Kinesis Data Streams CloudFront

Sie können Amazon Kinesis Data Streams mit CloudFront Echtzeitprotokollen verwenden und Informationen über Anfragen an eine Distribution in Echtzeit abrufen. Sie können dann Ihren eigenen Kinesis Data Stream Consumer erstellen oder Amazon Data Firehose verwenden, um die Protokolldaten an Amazon S3, Amazon Redshift, Amazon OpenSearch Service oder einen Protokollverarbeitungsservice eines Drittanbieters zu senden.

Weitere Informationen finden Sie unter Echtzeitprotokolle im Amazon CloudFront Developer Guide.

Schreiben Sie mithilfe von Amazon CloudWatch Logs in Kinesis Data Streams

Sie können CloudWatch Abonnements verwenden, um Zugriff auf einen Echtzeit-Feed mit Protokollereignissen von Amazon CloudWatch Logs zu erhalten und diesen zur Verarbeitung, Analyse und zum Laden in andere Systeme an einen Kinesis-Datenstream senden zu lassen.

Weitere Informationen finden Sie unter Echtzeitverarbeitung von Protokolldaten mit Abonnements im Amazon CloudWatch Logs-Benutzerhandbuch.

Schreiben Sie mit Amazon Connect in Kinesis Data Streams

Sie können Kinesis Data Streams verwenden, um Kontaktdatensätze und Agentenereignisse in Echtzeit aus Ihrer Amazon-Connect-Instance zu exportieren. Sie können auch Datenstreaming aus Amazon Connect Connect-Kundenprofilen aktivieren, um automatisch Updates für einen Kinesis-Datenstream über die Erstellung neuer Profile oder Änderungen an bestehenden Profilen zu erhalten.

Anschließend können Sie Verbraucheranwendungen erstellen, um die Daten in Echtzeit zu verarbeiten und zu analysieren. Mithilfe von Kontaktdatensätzen und Kundenprofildaten können Sie beispielsweise Ihre Quellsystemdaten, wie CRMs z. B. Marketingautomatisierungstools, auf up-to-date dem neuesten Stand halten. Mithilfe der Ereignisdaten der Agenten können Sie Dashboards erstellen, die Agenteninformationen und Ereignisse anzeigen und benutzerdefinierte Benachrichtigungen über bestimmte Agentenaktivitäten auslösen.

Weitere Informationen finden Sie unter Datenstreaming für Ihre Instance, Echtzeit-Export einrichten und Agenten-Event-Streams im Amazon-Connect-Administratorhandbuch.

Schreiben Sie in Kinesis Data Streams mit AWS Database Migration Service

Sie können AWS Database Migration Service es verwenden, um Daten in einen Kinesis-Datenstream zu migrieren. Anschließend können Sie Verbraucheranwendungen erstellen, die die Datensätze in Echtzeit verarbeiten. Sie können Daten auch problemlos nachgelagert an andere Services wie Amazon Simple Storage Service, Amazon DynamoDB und Amazon Redshift senden

Weitere Informationen finden Sie unter Nutzung von Kinesis Data Streams im Benutzerhandbuch für AWS Database Migration Service .

Schreiben Sie mit Amazon DynamoDB in Kinesis Data Streams

Sie können Amazon Kinesis Data Streams verwenden, um Änderungen an Amazon DynamoDB zu erfassen. Kinesis Data Streams erfasst Änderungen auf Elementebene in jeder DynamoDB-Tabelle und repliziert sie in einen Kinesis Data Stream. Ihre Verbraucheranwendungen können auf diesen Stream zugreifen, um Änderungen auf Artikelebene in Echtzeit anzuzeigen und diese Änderungen anschließend weiterzuleiten oder auf der Grundlage des Inhalts Maßnahmen zu ergreifen.

Weitere Informationen finden Sie unter wie Kinesis Data Streams mit DynamoDB funktionieren im Amazon-DynamoDB-Entwicklerhandbuch.

Schreiben Sie mit Amazon in Kinesis Data Streams EventBridge

Mit Kinesis Data Streams können Sie AWS API Anrufereignisse EventBridge an einen Stream senden, Verbraucheranwendungen erstellen und große Datenmengen verarbeiten. Sie können Kinesis Data Streams auch als Ziel in EventBridge Pipes verwenden und Datensätze nach optionaler Filterung und Anreicherung als Stream aus einer der verfügbaren Quellen bereitstellen.

Weitere Informationen finden Sie unter Senden von Ereignissen an einen Amazon Kinesis Kinesis-Stream und EventBridge Pipes im EventBridge Amazon-Benutzerhandbuch.

Schreiben Sie in Kinesis Data Streams mit AWS IoT Core

Mithilfe von IoT-Regelaktionen können Sie Daten in Echtzeit aus MQTT Nachrichten in AWS AWS IoT Core schreiben. Anschließend können Sie Anwendungen erstellen, die die Daten verarbeiten, ihren Inhalt analysieren und Warnmeldungen generieren und sie an Analyseanwendungen oder andere AWS -Services weiterleiten.

Weitere Informationen finden Sie unter Kinesis Data Streams im AWS IoT Core-Entwicklerhandbuch.

Schreiben Sie mit Amazon Relational Database Service in Kinesis Data Streams

Sie können Amazon Kinesis Data Streams verwenden, um Aktivitäten auf Ihren RDS Amazon-Instances zu überwachen. Mithilfe von Database Activity Streams RDS überträgt Amazon Aktivitäten in Echtzeit in einen Kinesis-Datenstream. Anschließend können Sie Anwendungen für das Compliance-Management entwickeln, die diese Aktivitäten nutzen, prüfen und Warnmeldungen generieren. Sie können auch Amazon Data Firehose verwenden, um die Daten zu speichern.

Weitere Informationen finden Sie unter Database Activity Streams im Amazon RDS Developer Guide.

Schreiben Sie in Kinesis Data Streams Pinpoint usingAmazon

Sie können Amazon Pinpoint so einrichten, dass Ereignisdaten an Amazon Kinesis Data Streams gesendet werden. Amazon Pinpoint kann Ereignisdaten für Kampagnen, Reisen sowie Transaktions-E-Mails und -Nachrichten senden. SMS Sie können die Daten dann in Analyseanwendungen aufnehmen oder Ihre eigenen Verbraucheranwendungen erstellen, die auf der Grundlage des Inhalts der Ereignisse Maßnahmen ergreifen.

Weitere Informationen finden Sie unter Streaming-Events im Entwicklerhandbuch zu Amazon Pinpoint.

Schreiben Sie mithilfe der Amazon Quantum Ledger Database (Amazon) in Kinesis Data Streams QLDB

Sie können in Amazon einen Stream erstellenQLDB, der jede Dokumentrevision erfasst, die in Ihr Journal übernommen wurde, und diese Daten in Echtzeit an Amazon Kinesis Data Streams übermittelt. Ein QLDB Stream ist ein kontinuierlicher Datenfluss vom Journal Ihres Ledgers zu einer Kinesis-Datenstream-Ressource. Anschließend können Sie die Kinesis-Streaming-Plattform oder die Kinesis-Client-Bibliothek verwenden, um Ihren Stream zu nutzen, die Datensätze zu verarbeiten und den Dateninhalt zu analysieren. Ein QLDB Stream schreibt Ihre Daten in drei Arten von Datensätzen in Kinesis Data Streams: `controlblock summary`, und`revision details`.

Weitere Informationen finden Sie unter Streams im Amazon QLDB Developer Guide.

Schreiben Sie mithilfe von Integrationen von Drittanbietern in Kinesis Data Streams

Sie können Daten mit einer der folgenden Drittanbieteroptionen, die in Kinesis Data Streams integriert sind, in Kinesis Data Streams schreiben. Wählen Sie die Option aus, über die Sie mehr erfahren möchten, und finden Sie Ressourcen und Links zu relevanter Dokumentation.

Themen

- Apache Flink
- Fluentd
- Debezium
- Oracle GoldenGate
- Kafka Connect

- Adobe Experience
- Striim

Apache Flink

Apache Flink ist ein Framework und eine verteilte Verarbeitungs-Engine für statusbehaftete Berechnungen über unbegrenzte und begrenzte Datenströme. Weitere Informationen zum Schreiben von Apache Flink in Kinesis Data Streams finden Sie unter Amazon Kinesis Data Streams Connector.

Fluentd

Fluentd ist ein Open-Source-Datensammler für eine einheitliche Protokollierungsebene. Weitere Informationen zum Schreiben von Fluentd in Kinesis Data Streams. Weitere Informationen finden Sie unter Stream-Verarbeitung mit Kinesis.

Debezium

Debezium ist eine verteilte Open-Source-Plattform für die Erfassung von Änderungsdaten. Weitere Informationen zum Schreiben von Debezium in Kinesis Data Streams finden Sie unter Meine SQL Datenänderungen an Amazon Kinesis streamen.

Oracle GoldenGate

Oracle GoldenGate ist ein Softwareprodukt, mit dem Sie Daten von einer Datenbank in eine andere replizieren, filtern und transformieren können. Weitere Informationen zum Schreiben von Oracle GoldenGate in Kinesis Data Streams finden Sie unter Datenreplikation nach Kinesis Data Stream mit Oracle. GoldenGate

Kafka Connect

Kafka Connect ist ein Tool für skalierbares und zuverlässiges Streamen von Daten zwischen Apache Kafka und anderen Systemen. Weitere Informationen zum Schreiben von Apache Kafka in Kinesis Data Streams finden Sie unter der Kinesis-Kafka-Konnektor.

Adobe Experience

Die Adobe Experience Platform ermöglicht es Unternehmen, Kundendaten aus jedem System zu zentralisieren und zu standardisieren. Anschließend werden Datenwissenschaft und Machine

Learning angewendet, um das Design und die Bereitstellung umfassender, personalisierter Erlebnisse erheblich zu verbessern. Weitere Informationen zum Schreiben von Daten von der Adobe Experience Platform in Kinesis Data Streams finden Sie unter So erstellen Sie eine Amazon-Kinesis-Verbindung.

Striim

Striim ist eine vollständige end-to-end In-Memory-Plattform zum Sammeln, Filtern, Transformieren, Anreichern, Aggregieren, Analysieren und Bereitstellen von Daten in Echtzeit. Weitere Informationen zum Schreiben von Daten von Striim in Kinesis Data Streams finden Sie im Kinesis Writer.

Problembehandlung Amazon Kinesis Data Streams Streams-Produzenten

Die folgenden Themen bieten Lösungen für häufig auftretende Probleme mit Amazon Kinesis Data Streams Streams-Herstellern:

- Meine Producer-Anwendung schreibt langsamer als erwartet
- Ich erhalte eine Fehlermeldung mit der Genehmigung eines nicht autorisierten KMS Master-Keys
- Beheben Sie andere häufig auftretende Probleme von Herstellern

Meine Producer-Anwendung schreibt langsamer als erwartet

Die häufigsten Gründe dafür, dass der Schreibdurchsatz langsamer als erwartet ist, sind:

- Die Servicelimits wurden überschritten
- Ich möchte meinen Producer optimieren

Die Servicelimits wurden überschritten

Um herauszufinden, ob die Dienstlimits überschritten werden, überprüfen Sie, ob Ihr Producer Durchsatzausnahmen für den Service ausgibt, und überprüfen Sie, welche API Operationen gedrosselt werden. Beachten Sie, dass es je nach Aufruf verschiedene Limits gibt, siehe Kontingente und -Einschränkungen. Beispielsweise gelten zusätzlich zu den bekannten Limits für Schreib- und Lesevorgänge auf Shard-Ebene folgenden Limits auf Stream-Ebene:

- CreateStream

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

Die Operationen `CreateStreamDeleteStream`, `ListStreams`, `GetShardIterator` und `MergeShards` dürfen maximal 5 mal pro Sekunde aufgerufen werden. Die `DescribeStream`-Operation ist auf 10 Aufrufe pro Sekunde begrenzt. Die `DescribeStreamSummary`-Operation ist auf 20 Aufrufe pro Sekunde begrenzt.

Wenn diese Aufrufe nicht das Problem sind, stellen Sie sicher, dass Sie einen Partitionsschlüssel ausgewählt haben, mit dem Sie die PUT-Operationen gleichmäßig auf alle Shards verteilen können, und dass Sie keinen Partitionsschlüssel haben, der die Service Limits ausschöpft, wenn der Rest dies nicht tut. Dazu müssen Sie den Spitzendurchsatz messen und die Anzahl der Shards im Stream berücksichtigen. Weitere Informationen zum Verwalten von Streams finden Sie unter Kinesis-Datenstreams erstellen und verwalten.

> **Tip**
>
> Denken Sie daran, bei Berechnungen zur Durchsatzdrosselung auf das nächste Kilobyte aufzurunden, wenn Sie die Operation mit einem Datensatz verwenden PutRecord, während bei der Operation mit mehreren Datensätzen die Gesamtsumme der Datensätze in PutRecordsjedem Aufruf abgerundet wird. Eine `PutRecords`-Anforderung mit 600 Datensätzen und einer Größe von 1,1 KB wird beispielsweise nicht gedrosselt.

Ich möchte meinen Producer optimieren

Bevor Sie mit der Optimierung Ihres Producers beginnen, müssen Sie die folgenden wichtigen Aufgaben erledigen. Bestimmen Sie zunächst den gewünschten Spitzenwert hinsichtlich Datensatzgröße und Datensätze pro Sekunde. Schließen Sie dann die Stream-Kapazität als Begrenzungsfaktor aus (Die Servicelimits wurden überschritten). Nutzen Sie die folgenden Tipps zur Fehlerbehebung und Optimierungsanleitungen für die beiden häufigsten Produzententypen, wenn Sie die Stream-Kapazität als Begrenzungsfaktor ausgeschlossen haben.

Großer Produzent

Ein großer Hersteller läuft normalerweise von einem lokalen Server oder einer EC2 Amazon-Instance aus. Für Kunden, die einen höheren Durchsatz eines großen Produzenten benötigen, ist in der Regel die Latenz pro Datensatz entscheidend. Zu den Strategien für den Umgang mit Latenz gehören die folgenden: Wenn der Kunde Datensätze per Mikrobatch/Puffer speichern kann, verwenden Sie die Kinesis Producer Library (die über eine erweiterte Aggregationslogik verfügt), die Operation mit mehreren Datensätzen oder aggregieren Sie Datensätze zu einer größeren Datei PutRecords, bevor Sie die Operation mit einem Datensatz verwenden. PutRecord Wenn Sie keine Stapel bilden und keine Pufferung verwenden können, nutzen Sie mehrere Threads, um gleichzeitig in Service Kinesis Data Streams schreiben zu können. Zu diesen AWS SDK for Java und anderen SDKs gehören asynchrone Clients, die dies mit sehr wenig Code bewerkstelligen können.

Kleiner Produzent

Eine kleiner Produzent ist in der Regel eine mobile App, ein IoT-Gerät oder ein Webclient. Wenn es sich um eine mobile App handelt, empfehlen wir, den `PutRecords` Vorgang oder den Kinesis Recorder auf dem AWS Handy SDKs zu verwenden. Weitere Informationen finden Sie in den Handbüchern „ AWS Mobile SDK for Android Erste Schritte“ und „ AWS Mobile SDK for iOS Erste Schritte“. Mobile Anwendungen müssen Verbindungsunterbrechungen bewältigen und benötigen eine Art Stapeleingabe, beispielsweise `PutRecords`. Wenn Sie aus irgendeinem Grund keine Stapel bilden können, lesen Sie sich die oben stehenden Informationen zu großen Produzenten durch. Ist der Produzent ein Browser, ist die Menge der generierten Dateien in der Regel sehr klein. Allerdings platzieren Sie PUT-Operationen im kritischen Pfad der Anwendung. Dies wird von uns nicht empfohlen.

Ich erhalte eine Fehlermeldung mit der Genehmigung eines nicht autorisierten KMS Master-Keys

Dieser Fehler tritt auf, wenn eine Producer-Anwendung ohne Berechtigungen für den KMS Masterschlüssel in einen verschlüsselten Stream schreibt. Informationen zum Zuweisen von Berechtigungen für den Zugriff auf einen KMS Schlüssel einer Anwendung finden Sie unter Verwenden von Schlüsselrichtlinien in AWS KMS und Verwenden von IAM Richtlinien mit AWS KMS.

Beheben Sie andere häufig auftretende Probleme von Herstellern

- Warum gibt mein Kinesis-Datenstrom einen 500 Internal Server Error zurück?
- Wie behebe ich Timeout-Fehler beim Schreiben von Flink in Kinesis Data Streams?

- Wie behebe ich Drosselungsfehler in Kinesis Data Streams?
- Warum drosselt mein Kinesis-Datenstrom?
- Wie kann ich Datensätze mit dem in einen Kinesis-Datenstream einfügen? KPL

Optimieren Sie Kinesis Data Streams Streams-Produzenten

Sie können Ihre Amazon Kinesis Data Streams Streams-Producer je nach dem spezifischen Verhalten, das Sie beobachten, weiter optimieren. Sehen Sie sich die folgenden Themen an, um Lösungen zu finden.

Themen

- Passen Sie KPL Wiederholungsversuche und Verhalten bei der Ratenbegrenzung an
- Wenden Sie bewährte Methoden auf die Aggregation an KPL

Passen Sie KPL Wiederholungsversuche und Verhalten bei der Ratenbegrenzung an

Wenn Sie Benutzerdatensätze der Kinesis Producer Library (KPL) mithilfe dieses KPL `addUserRecord()` Vorgangs hinzufügen, erhält ein Datensatz einen Zeitstempel und wird einem Puffer hinzugefügt, dessen Frist durch den `RecordMaxBufferedTime` Konfigurationsparameter festgelegt wird. Diese Kombination aus Zeitstempel und Frist legt die Pufferpriorität fest. Datensätze werden aufgrund der folgenden Kriterien aus dem Puffer entfernt:

- Puffer-Priorität
- Aggregierungskonfiguration
- Sammlungskonfiguration

Die Parameter der Aggregierungskonfiguration und der Sammlungskonfiguration, die das Pufferverhalten beeinflussen, sind:

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

Die gelöschten Datensätze werden dann mithilfe eines Aufrufs des Kinesis Data Streams-Vorgangs als Amazon Kinesis Data Streams Streams-Aufzeichnungen an Ihren Kinesis-Datenstream gesendet. API `PutRecords` Die Operation `PutRecords` sendet Anfragen an Ihre Stream, die gelegentlich vollständige oder teilweise Fehlschläge zeigen. Datensätze, die fehlschlagen, werden automatisch wieder dem Puffer hinzugefügt. KPL Die neue Frist wird auf der Grundlage des kleineren dieser beiden Werte festgelegt:

- Die Hälfte der aktuellen `RecordMaxBufferedTime`-Konfiguration
- Der time-to-live Wert des Datensatzes

Mit dieser Strategie können erneut versuchte KPL Benutzerdatensätze in nachfolgende Kinesis Data Streams API Streams-Aufrufe aufgenommen werden, um den Durchsatz zu verbessern und die Komplexität zu reduzieren, während gleichzeitig der Wert des Kinesis Data Streams Streams-Datensatzes durchgesetzt wird. time-to-live Es gibt keinen Backoff-Algorithmus, was dies zu einer relativ aggressiven Wiederholungsstrategie macht. Spamming aufgrund übermäßiger Wiederholungen wird durch die Quotengrenze bewirkt, die im nächsten Abschnitt erläutert wird.

Ratenbegrenzung

Das KPL beinhaltet eine Funktion zur Ratenbegrenzung, die den Durchsatz pro Shard begrenzt, der von einem einzigen Hersteller gesendet wird. Die Quotenbegrenzung wird durch einen Token-Bucket-Algorithmus mit separaten Buckets für Datensätze von Kinesis Data Streams und Bytes implementiert. Jeder erfolgreiche Schreibvorgang in einen Kinesis-Datenstrom fügt jedem Bucket ein Token (oder mehrere Token) hinzu, bis ein bestimmter Grenzwert erreicht ist. Dieser Grenzwert ist konfigurierbar, ist aber standardmäßig um 50 % höher als das tatsächliche Shard-Limit eingestellt, um die Shard-Sättigung durch einen einzelnen Produzenten zu ermöglichen.

Sie können diesen Grenzwert herabsetzen, um Spamming durch übermäßig viele Wiederholungsversuche zu reduzieren. Die bewährte Methode besteht jedoch darin, dass jeder Produzent aggressiv einen maximalen Durchsatz durch Wiederholungen zu erzielen versucht und mit der daraus resultierenden, als übermäßig angesehenen Drosselung so umzugehen, dass die Kapazität des Streams erweitert und eine geeignete Partitionsschlüsselstrategie implementiert wird.

Wenden Sie bewährte Methoden auf die Aggregation an KPL

Das Sequenznummernschema der resultierenden Amazon Kinesis Data Streams Streams-Datensätze bleibt zwar gleich, aber die Aggregation führt dazu, dass die Indizierung von Benutzerdatensätzen der Kinesis Producer Library (KPL), die in einem aggregierten Kinesis Data

Streams Streams-Datensatz enthalten sind, bei 0 (Null) beginnt. Solange Sie sich jedoch nicht auf Sequenznummern verlassen, um Ihre KPL Benutzerdatensätze eindeutig zu identifizieren, kann Ihr Code dies ignorieren, da die Aggregation (Ihrer Benutzerdatensätze zu einem Kinesis Data Streams Streams-Datensatz) und nachfolgende Deaggregation (eines Kinesis Data Streams-Datensatzes) in Ihren KPL KPLBenutzerdatensätze) erledigt das automatisch für Sie. Dies gilt unabhängig davon, ob Ihr Verbraucher das KCL oder das verwendet AWS SDK. Um diese Aggregationsfunktion nutzen zu können, müssen Sie den Java-Teil von KPL in Ihren Build übernehmen, falls Ihr Consumer mit dem in der API AWS SDK bereitgestellten geschrieben wurde.

Wenn Sie beabsichtigen, Sequenznummern als eindeutige Identifikatoren für Ihre KPL Benutzerdatensätze zu verwenden, empfehlen wir Ihnen, die unter `public int hashCode()` und `public boolean equals(Object obj)` für den Vergleich Ihrer Benutzerdatensätze vorgesehenen `Record` `UserRecord` vertragsgemäßen Verfahren zu verwenden. KPL Wenn Sie die Teilsequenznummer Ihres KPL Benutzerdatensatzes überprüfen möchten, können Sie ihn außerdem in eine `UserRecord` Instanz umwandeln und deren Teilsequenznummer abrufen.

Weitere Informationen finden Sie unter Implementieren Sie die Deaggregation für Verbraucher.

Daten aus Amazon Kinesis Data Streams lesen

Ein Konsument ist eine Anwendung, die alle Daten aus einem Kinesis-Datenstrom verarbeitet. Wenn ein Konsument erweiterte Rundsendungen verwendet, erhält er ein eigenes Kontingent von 2 MB/s für den Lesedurchsatz, sodass mehrere Konsumenten parallel Daten aus demselben Stream lesen können, ohne den Lesedurchsatz von anderen Konsumenten abziehen zu müssen. Informationen zum Verwenden der Shard-Funktionen für erweiterte Rundsendungen erhalten Sie unter Entwickeln Sie maßgeschneiderte Verbraucher mit dediziertem Durchsatz (verbesserter Fan-Out).

Standardmäßig stellen Shards in einem Stream 2 MB/s Lesedurchsatz pro Shard bereit. Dieser Durchsatz gilt insgesamt für alle Konsumenten, die aus dem gegebenen Shard lesen. Der Standardwert von 2 MB/s Lesedurchsatz pro Shard ist also ein fester Wert, auch wenn mehrere Konsumenten aus dem betreffenden Shard lesen. Informationen zur Nutzung dieses Standarddurchsatzes von Shards finden Sie unter Entwickeln Sie maßgeschneiderte Verbraucher mit gemeinsamem Durchsatz.

Die folgende Tabelle vergleicht den Standard-Durchsatz mit den erweiterten Rundsendungen. Die Verzögerung der Nachrichtenweiterleitung ist definiert als die Zeit in Millisekunden, die eine Nutzlast, die mithilfe des Payload-Dispatchings APIs (like PutRecord und PutRecords) gesendet wurde, benötigt, um die Nutzlast verbrauchende Anwendung (like und) zu erreichen. APIs GetRecords SubscribeToShard

Merkmale	Nicht registrierte Verbraucher ohne erweitertes Rundsenden	Registrierte Verbraucher mit erweitertem Rundsenden
Shard-Lesedurchsatz	Fester Gesamtwert von 2 MB/s pro Shard. Wenn mehrere Konsumenten aus demselben Shard lesen, teilen sie sich diesen Durchsatz. Die Summe des Lesedurchsatzes der einzelnen Konsumenten eines Shards beträgt maximal 2 MB/s.	Der Wert erhöht sich nur, wenn Konsumenten für erweiterte Rundsendungen registriert werden. Jeder für erweiterte Rundsendungen registrierte Konsument erhält einen eigenen Lesedurchsatz von bis zu 2 MB/s pro Shard, den er nicht mit anderen Konsumenten teilen muss.

Merkmale	Nicht registrierte Verbraucher ohne erweitertes Rundsenden	Registrierte Verbraucher mit erweitertem Rundsenden
Verzögerung der Nachrichtenverbreitung	Ein Durchschnitt von etwa 200 ms, wenn Sie einen Verbraucher haben, der aus dem Stream liest. Dieser Durchschnitt steigt bis zu 1000 ms an, wenn Sie fünf Verbraucher haben.	In der Regel durchschnittlich 70 ms, unabhängig davon, ob Sie einen oder fünf Verbraucher haben.
Kosten	N/A	Es gibt Datenabrufkosten und Konsumenten-Shard-Stundensätze. Weitere Informationen finden Sie unter Preise für Amazon Kinesis Daten-Streams.
Datensatzbereitstellungsmodell	HTTPZiehen GetRecordsSie das Modell vor, indem Sie es verwenden.	Kinesis Data Streams überträgt Ihnen die Datensätze über HTTP /2 mithilfe von. SubscribeToShard

Themen

- Verwenden Sie den Data Viewer in der Kinesis-Konsole
- Fragen Sie Ihre Datenströme in der Kinesis-Konsole ab
- Entwickeln Sie Verbraucher mit AWS Lambda
- Entwickeln Sie Kunden mithilfe von Amazon Managed Service für Apache Flink
- Entwickeln Sie Verbraucher mithilfe von Amazon Data Firehose
- Verwenden Sie die Kinesis Client Library
- Entwickeln Sie maßgeschneiderte Verbraucher mit gemeinsamem Durchsatz
- Entwickeln Sie maßgeschneiderte Verbraucher mit dediziertem Durchsatz (verbesserter Fan-Out)
- Migrieren Sie Verbraucher von KCL 1.x auf KCL 2.x
- Daten aus Kinesis Data Streams mithilfe anderer AWS Dienste lesen
- Lesen Sie mithilfe von Integrationen von Drittanbietern aus Kinesis Data Streams
- Problembehandlung bei Kinesis Data Streams Streams-Verbrauchern
- Optimieren Sie die Verbraucher von Amazon Kinesis Data Streams

Verwenden Sie den Data Viewer in der Kinesis-Konsole

Mit dem Data Viewer in der Kinesis Management Console können Sie Datensätze innerhalb des angegebenen Shards Ihres Datenstroms anzeigen, ohne eine Verbraucheranwendung entwickeln zu müssen. Gehen Sie folgendermaßen vor, um den Data Viewer zu verwenden:

1. Melden Sie sich bei der an AWS Management Console und öffnen Sie die Kinesis-Konsole unter https://console.aws.amazon.com/kinesis.
2. Wählen Sie den aktiven Datenstrom aus, dessen Datensätze Sie mit dem Data Viewer anzeigen möchten, und wählen Sie dann die Registerkarte Data Viewer.
3. Wählen Sie auf der Registerkarte Data Viewer für den ausgewählten aktiven Datenstrom den Shard aus, dessen Datensätze Sie anzeigen möchten, wählen Sie die Startposition aus und klicken Sie dann auf Datensätze abrufen. Sie können die Startposition auf einen der folgenden Werte stellen:

   - Bei Sequenznummer: Zeigt Datensätze von der Position an, die durch die im Feld Sequenznummer angegebene Sequenznummer gekennzeichnet ist.
   - Nach Sequenznummer: Zeigt Datensätze direkt nach der Position an, die durch die im Feld Sequenznummer angegebene Sequenznummer gekennzeichnet ist.
   - Bei Zeitstempel: Zeigt Datensätze von der Position an, die durch den im Zeitstempelfeld angegebenen Zeitstempel gekennzeichnet ist.
   - Horizont kürzen: Zeigt Datensätze am letzten nicht getrimmten Datensatz im Shard an, der der älteste Datensatz im Shard ist.
   - Aktuell: Zeigt Datensätze direkt nach dem neuesten Datensatz im Shard an, sodass Sie immer die neuesten Daten im Shard lesen.

     Die generierten Datensätze, die mit der angegebenen Shard-ID und Startposition übereinstimmen, werden dann in einer Datensatztabelle in der Konsole angezeigt. Es werden maximal 50 Datensätze gleichzeitig angezeigt. Um die nächste Gruppe von Datensätzen anzuzeigen, klicken Sie auf die Schaltfläche Weiter.
4. Klicken Sie auf einen einzelnen Datensatz, um die Payload des Datensatzes als Rohdaten oder als JSON Format in einem separaten Fenster anzuzeigen.

Beachten Sie, dass, wenn Sie in der Datenanzeige auf „Datensätze abrufen“ oder „Weiter“ klicken, das GetRecordsAPIaufgerufen wird. Dies gilt für die GetRecordsAPIObergrenze von 5 Transaktionen pro Sekunde.

Fragen Sie Ihre Datenströme in der Kinesis-Konsole ab

Auf der Registerkarte Data Analytics in der Kinesis Data Streams Console können Sie Ihre Datenströme mithilfe von SQL abfragen. Gehen Sie wie folgt vor, um diese Funktion zu nutzen:

1. Melden Sie sich bei der an AWS Management Console und öffnen Sie die Kinesis-Konsole unter https://console.aws.amazon.com/kinesis.
2. Wählen Sie den aktiven Datenstream aus, mit dem Sie Abfragen durchführen möchten, SQL und wählen Sie dann die Registerkarte Datenanalyse.
3. Auf der Registerkarte Datenanalyse können Sie Datenströme mit einem Managed Apache Flink Studio-Notebook überprüfen und visualisieren. Mit Apache Zeppelin können Sie SQL Ad-hoc-Abfragen durchführen, um Ihren Datenstrom zu überprüfen und die Ergebnisse innerhalb von Sekunden anzuzeigen. Wählen Sie auf der Registerkarte Datenanalyse die Option Ich stimme zu und wählen Sie dann Notizbuch erstellen aus, um ein Notizbuch zu erstellen.
4. Nachdem das Notizbuch erstellt wurde, wählen Sie In Apache Zeppelin öffnen. Dadurch wird Ihr Notizbuch in einem neuen Tab geöffnet. Ein Notizbuch ist eine interaktive Oberfläche, über die Sie Ihre SQL Fragen stellen können. Wählen Sie die Notiz aus, die den Namen Ihres Streams enthält.
5. Sie sehen eine Notiz mit einer `SELECT` Beispielabfrage zur Ausgabe der Daten im Stream, der bereits läuft. Auf diese Weise können Sie das Schema für Ihren Datenstream anzeigen.
6. Wählen Sie auf der Registerkarte Datenanalyse die Option Beispielabfragen anzeigen aus, wenn Sie andere Abfragen ausprobieren möchten, wie z. B. sich öffnende oder verschiebbare Fenster. Kopieren Sie die Abfrage, passen Sie sie an Ihr Datenstromschema an, und führen Sie sie dann in einem neuen Absatz in Ihrer Zeppelin-Notiz aus.

Entwickeln Sie Verbraucher mit AWS Lambda

Sie können eine AWS Lambda Funktion verwenden, um Datensätze in einem Datenstrom zu verarbeiten. AWS Lambda ist ein Rechendienst, mit dem Sie Code ausführen können, ohne Server bereitstellen oder verwalten zu müssen. Es führt Ihren Code nur bei Bedarf aus und skaliert automatisch – von einigen Anfragen pro Tag bis zu Tausenden pro Sekunde. Sie zahlen nur für die tatsächlich aufgewendete Zeit. Es werden keine Gebühren berechnet, solange Ihr Code nicht ausgeführt wird. Mit AWS Lambda können Sie Code für praktisch jede Art von Anwendung oder Back-End-Dienst ausführen, und das alles ohne Verwaltungsaufwand. Der Service führt Ihren Code auf einer hoch verfügbaren Datenverarbeitungsinfrastruktur aus and erledigt die

gesamte Administration der Datenverarbeitungsressourcen, einschließlich der Server- und Betriebssystemwartung, Kapazitätsbereitstellung und automatischen Skalierung sowie Code-Überwachung und -Protokollierung. Weitere Informationen finden Sie unter Verwenden AWS Lambda mit Amazon Kinesis.

Informationen zur Fehlerbehebung finden Sie unter Warum kann der Kinesis-Data-Streams-Trigger meine Lambda-Funktion nicht aufrufen?

Entwickeln Sie Kunden mithilfe von Amazon Managed Service für Apache Flink

Sie können eine Amazon Managed Service for Apache Flink-Anwendung verwenden, um Daten in einem Kinesis-Stream mit SQL Java oder Scala zu verarbeiten und zu analysieren. Anwendungen des Managed Service für Apache Flink kann Daten mithilfe von Referenzquellen anreichern, Daten im Laufe der Zeit aggregieren oder Machine Learning verwenden, um Datenanomalien zu finden. Anschließend können Sie die Analyseergebnisse in einen anderen Kinesis-Stream, einen Firehose-Lieferstream oder eine Lambda-Funktion schreiben. Weitere Informationen finden Sie im Managed Service for Apache Flink Developer Guide for SQL Applications oder im Managed Service for Apache Flink Developer Guide for Flink Applications.

Entwickeln Sie Verbraucher mithilfe von Amazon Data Firehose

Sie können einen Firehose verwenden, um Datensätze aus einem Kinesis-Stream zu lesen und zu verarbeiten. Firehose ist ein vollständig verwalteter Service für die Bereitstellung von Echtzeit-Streaming-Daten an Ziele wie Amazon S3, Amazon Redshift, Amazon OpenSearch Service und Splunk. Firehose unterstützt auch alle benutzerdefinierten HTTP Endpunkte oder HTTP Endpunkte, die unterstützten Drittanbietern gehören, darunter Datadog, MongoDB und New Relic. Sie können Firehose auch so konfigurieren, dass Ihre Datensätze transformiert und das Datensatzformat konvertiert wird, bevor Ihre Daten an das Ziel gesendet werden. Weitere Informationen finden Sie unter Writing to Firehose Using Kinesis Data Streams.

Verwenden Sie die Kinesis Client Library

Eine der Methoden zur Entwicklung benutzerdefinierter Verbraucheranwendungen, die Daten aus KDS Datenströmen verarbeiten können, ist die Verwendung der Kinesis Client Library (KCL).

Themen

- Was ist die Kinesis Client Library?
- KCLverfügbare Versionen
- KCL-Konzepte
- Verwenden Sie eine Leasetabelle, um nachzuverfolgen, welche Shards von der Consumer-Anwendung verarbeitet wurden KCL
- Verarbeiten Sie mehrere Datenströme mit derselben KCL 2.x für Java-Consumer-Anwendung
- Verwenden Sie das KCL mit der Schemaregistrierung AWS Glue

> **Note**
>
> Sowohl für KCL 1.x als auch für KCL 2.x wird empfohlen, je nach Nutzungsszenario ein Upgrade auf die neueste Version KCL KCL 1.x oder 2.x durchzuführen. Sowohl KCL 1.x als auch KCL 2.x werden regelmäßig mit neueren Versionen aktualisiert, die die neuesten Abhängigkeits- und Sicherheitspatches, Bugfixes und abwärtskompatible neue Funktionen enthalten. Weitere Informationen finden Sie unter /releases. https://github.com/awslabs/amazon-kinesis-client

Was ist die Kinesis Client Library?

KCLhilft Ihnen dabei, Daten aus einem Kinesis-Datenstrom zu nutzen und zu verarbeiten, indem es sich um viele der komplexen Aufgaben kümmert, die mit verteilter Datenverarbeitung verbunden sind. Dazu gehören der Lastenausgleich über mehrere Anwendungs-Instances hinweg, die Reaktion auf Ausfälle von Verbraucheranwendungs-Instances, die Überprüfung verarbeiteter Datensätze und die Reaktion auf Resharding. Der KCL kümmert sich um all diese Unteraufgaben, sodass Sie sich darauf konzentrieren können, Ihre benutzerdefinierte Logik für die Datensatzverarbeitung zu schreiben.

Das KCL unterscheidet sich von den Kinesis Data StreamsAPIs, die in der AWS SDKs verfügbar sind. Die Kinesis Data Streams APIs helfen Ihnen bei der Verwaltung vieler Aspekte von Kinesis Data Streams, darunter das Erstellen von Streams, das Resharding sowie das Einfügen und Abrufen von Datensätzen. Das KCL bietet eine Abstraktionsebene für all diese Unteraufgaben, sodass Sie sich auf die benutzerdefinierte Datenverarbeitungslogik Ihrer Verbraucheranwendung konzentrieren können. Informationen zu den Kinesis Data Streams API finden Sie in der Amazon Kinesis Kinesis-Referenz API.

> **Important**
>
> Das KCL ist eine Java-Bibliothek. Support für andere Sprachen als Java wird über eine mehrsprachige Schnittstelle namens bereitgestellt. MultiLangDaemon Dieser Daemon basiert auf Java und wird im Hintergrund ausgeführt, wenn Sie eine andere KCL Sprache als Java verwenden. Wenn Sie beispielsweise das KCL für Python installieren und Ihre Verbraucheranwendung vollständig in Python schreiben, müssen Sie trotzdem Java auf Ihrem System installieren, da MultiLangDaemon Darüber hinaus MultiLangDaemon verfügt es über einige Standardeinstellungen, die Sie möglicherweise an Ihren Anwendungsfall anpassen müssen, z. B. an die AWS Region, mit der eine Verbindung hergestellt wird. Weitere Informationen dazu finden Sie MultiLangDaemon GitHub unter KCL MultiLangDaemon Projekt.

Der KCL fungiert als Vermittler zwischen Ihrer Datensatzverarbeitungslogik und Kinesis Data Streams. Die KCL führt die folgenden Aufgaben aus:

- Stellt eine Verbindung mit dem Datenstrom her
- Listet die Shards innerhalb des Datenstroms auf
- Nutzt Leases, um Shard-Verbindungen mit seinen Mitarbeitern zu koordinieren
- Instanziiert einen Datensatzverarbeiter für jeden Shard, der verwaltet wird
- Ruft Datensätze aus dem Datenstrom ab
- Überträgt per Push Datensätze an den entsprechenden Datensatzverarbeiter
- Verwendet Checkpoints für verarbeitete Datensätze
- Gleicht zwischen Shard-Worker-Zuordnungen (Leases) aus, wenn sich die Anzahl der Worker-Instances ändert oder wenn der Datenstrom erneut geteilt wird (Shards werden aufgeteilt oder zusammengeführt)

KCLverfügbare Versionen

Derzeit können Sie eine der folgenden unterstützten Versionen von verwendenKCL, um Ihre benutzerdefinierten Verbraucheranwendungen zu erstellen:

- KCL1.x

  Weitere Informationen finden Sie unter Entwickeln Sie KCL 1.x-Verbraucher.

- KCL2.x

  Weitere Informationen finden Sie unter Develop KCL 2.x-Verbraucher.

Sie können entweder KCL 1.x oder KCL 2.x verwenden, um Verbraucheranwendungen zu erstellen, die einen gemeinsamen Durchsatz verwenden. Weitere Informationen finden Sie unter Entwickeln Sie benutzerdefinierte Verbraucher mit gemeinsamem Durchsatz mithilfe von KCL.

Um Verbraucheranwendungen zu erstellen, die einen dedizierten Durchsatz verwenden (erweiterte Fan-Out-Verbraucher), können Sie nur 2.x verwenden. KCL Weitere Informationen finden Sie unter Entwickeln Sie maßgeschneiderte Verbraucher mit dediziertem Durchsatz (verbesserter Fan-Out).

Informationen zu den Unterschieden zwischen KCL 1.x und KCL 2.x sowie Anweisungen zur Migration von KCL 1.x zu 2.x finden Sie unter. KCL Migrieren Sie Verbraucher von KCL 1.x auf KCL 2.x

KCL-Konzepte

- KCLAnwendung für Privatanwender — eine Anwendung, die speziell für das Lesen KCL und Verarbeiten von Datensätzen aus Datenströmen entwickelt wurde.
- Anwendungsinstanz für KCL Privatanwender — Anwendungen für Privatanwender sind in der Regel verteilt, wobei eine oder mehrere Anwendungsinstanzen gleichzeitig ausgeführt werden, um Fehler zu koordinieren und einen dynamischen Lastenausgleich bei der Verarbeitung von Datensätzen vorzunehmen.
- Worker — eine übergeordnete Klasse, die eine Anwendungsinstanz für KCL Privatanwender verwendet, um mit der Datenverarbeitung zu beginnen.

  > **Important**
  >
  > Jede Instanz einer KCL Verbraucheranwendung hat einen Worker.

  Der Worker initialisiert und überwacht verschiedene Aufgaben, darunter das Synchronisieren von Shard- und Leasing-Informationen, das Verfolgen von Shard-Zuweisungen und das Verarbeiten von Daten aus den Shards. Ein Worker stellt KCL die Konfigurationsinformationen für die Verbraucheranwendung bereit, z. B. den Namen des Datenstroms, dessen Datensätze diese KCL Verbraucheranwendung verarbeiten wird, und die AWS Anmeldeinformationen, die für den Zugriff auf diesen Datenstrom erforderlich sind. Der Worker startet außerdem die spezifische Instanz der

KCL Verbraucheranwendung, um Datensätze aus dem Datenstrom an die Datensatzprozessoren weiterzuleiten.

> **Important**
>
> In KCL 1.x heißt diese Klasse Worker. Weitere Informationen (dies sind die KCL Java-Repositorys) finden https://github.com/awslabs/ amazon-kinesis-client Sie unter /blob/v1.x/ src/main/java/com/amazonaws/services/Kinesis/ClientLibrary/lib/Worker/Worker.java. In 2.x heißt diese Klasse Scheduler. KCL Der Zweck von Scheduler in KCL 2.x ist identisch mit dem Zweck von Worker in 1.x. KCL Weitere Informationen zur Scheduler-Klasse in KCL 2.x finden Sie unter /blob/master/ /src/main/java/software/amazon/kinesis/coordinator/ Scheduler.java. https://github.com/awslabs/ amazon-kinesis-client amazon-kinesis-client

- Lease – Daten, die die Bindung zwischen einem Worker und einem Shard definieren. Verteilte KCL Verbraucheranwendungen nutzen Leases, um die Verarbeitung von Datensätzen auf eine ganze Flotte von Mitarbeitern zu verteilen. Zu einem bestimmten Zeitpunkt ist jede Datenmenge durch einen durch die Variable identifizierten Leasingvertrag an einen bestimmten Mitarbeiter gebunden. leaseKey

  Standardmäßig kann ein Arbeitnehmer einen oder mehrere Mietverträge (abhängig vom Wert der Variablen maxLeasesForWorker) gleichzeitig abschließen.

  > **Important**
  >
  > Jeder Worker versucht, alle verfügbaren Leases für alle verfügbaren Shards in einem Datenstrom zu halten. Aber es kann nur jeweils ein Worker jeden Lease zu einem bestimmten Zeitpunkt erfolgreich halten.

  Wenn Sie beispielsweise eine Konsumentenanwendungs-Instance A mit Worker A haben, die einen Datenstrom mit 4 Shards verarbeitet, kann Worker A Leases für die Shards 1, 2, 3 und 4 gleichzeitig halten. Wenn Sie jedoch zwei Konsumentenanwendungs-Instances haben: A und B mit Worker A und Worker B und diese Instances einen Datenstrom mit 4 Shards verarbeiten, können Worker A und Worker B nicht gleichzeitig den Lease für Shard 1 halten. Ein Worker hält den Lease für einen bestimmten Shard, bis er bereit ist, die Verarbeitung der Datensätze dieses Shards zu beenden, oder bis er ausfällt. Wenn ein Worker den Lease beendet, nimmt ein anderer Worker den Lease auf und hält ihn.

Weitere Informationen (dies sind die KCL Java-Repositorys) finden Sie unter https://github.com/awslabs/amazon-kinesis-client/blob/v1.x/src/main/java/com/amazonaws/services/kinesis/leases/impl/Lease.java für 1.x und /blob/master/ /src/main/java/software/amazon/kinesis/leases/Lease.java für 2.x. KCL https://github.com/awslabs/ amazon-kinesis-client amazon-kinesis-client KCL

- Leasing-Tabelle — eine einzigartige Amazon DynamoDB-Tabelle, die verwendet wird, um die Shards in einem KDS Datenstrom zu verfolgen, die von den Mitarbeitern der Verbraucheranwendung geleast und verarbeitet werden. KCL Die Leasing-Tabelle muss (innerhalb eines Workers und für alle Mitarbeiter) mit den neuesten Shard-Informationen aus dem Datenstrom synchronisiert bleiben, während die Consumer-Anwendung ausgeführt wirdKCL. Weitere Informationen finden Sie unter Verwenden Sie eine Leasetabelle, um nachzuverfolgen, welche Shards von der Consumer-Anwendung verarbeitet wurden KCL.
- Record Processor — die Logik, die definiert, wie Ihre KCL Verbraucheranwendung die Daten verarbeitet, die sie aus den Datenströmen erhält. Zur Laufzeit instanziiert eine KCL Consumer-Anwendungsinstanz einen Worker, und dieser Worker instanziiert einen Datensatzprozessor für jeden Shard, für den er eine Lease hält.

Verwenden Sie eine Leasetabelle, um nachzuverfolgen, welche Shards von der Consumer-Anwendung verarbeitet wurden KCL

Themen

- Was ist eine Leasingtabelle
- Durchsatz
- So wird eine Leasing-Tabelle mit den Shards in einem Kinesis-Datenstrom synchronisiert

Was ist eine Leasingtabelle

KCLVerwendet für jede Amazon Kinesis Data Streams Streams-Anwendung eine eindeutige Leasetabelle (gespeichert in einer Amazon DynamoDB-Tabelle), um den Überblick über die Shards in einem KDS Datenstream zu behalten, die von den Mitarbeitern der Verbraucheranwendung geleast und verarbeitet werden. KCL

> **Important**
>
> KCLverwendet den Namen der Verbraucheranwendung, um den Namen der Leasetabelle zu erstellen, die diese Consumer-Anwendung verwendet. Daher muss jeder Consumer-Anwendungsname eindeutig sein.

Sie können die Leasetabelle mit der Amazon-DynamoDB-Konsole anzeigen, während die Konsumentenanwendung ausgeführt wird.

Wenn die Leasetabelle für Ihre KCL Verbraucheranwendung beim Start der Anwendung nicht vorhanden ist, erstellt einer der Mitarbeiter die Leasingtabelle für diese Anwendung.

> **Important**
>
> Ihr Konto wird neben den Kosten für Kinesis Data Streams mit den Kosten belastet, die für die DynamoDB-Tabelle anfallen.

Jede Zeile in der Leasetabelle stellt einen Shard dar, der von den Workern Ihrer Konsumentenanwendung verarbeitet wird. Wenn Ihre KCL Consumer-Anwendung nur einen Datenstrom verarbeitet`leaseKey`, dann ist der Hash-Schlüssel für die Leasing-Tabelle die Shard-ID. Wenn jaVerarbeiten Sie mehrere Datenströme mit derselben KCL 2.x für Java-Consumer-Anwendung, dann leaseKey sieht die Struktur von so aus:`account-id:StreamName:streamCreationTimestamp:ShardId`. z. B. `111111111:multiStreamTest-1:12345:shardId-000000000336`.

Neben der Shard-ID enthält jede Zeile noch folgende Daten:

- checkpoint: Die letzte Prüfpunkt-Sequenznummer des Shards. Dieser Wert ist für alle Shards im Datenstrom eindeutig.
- checkpointSubSequenceNummer: Bei Verwendung der Aggregationsfunktion der Kinesis Producer Library ist dies eine Erweiterung des Checkpoints, mit der einzelne Benutzerdatensätze innerhalb des Kinesis-Datensatzes verfolgt werden.
- leaseCounter: Wird für die Versionierung von Leasing-Verträgen verwendet, sodass Mitarbeiter erkennen können, dass ihr Leasing von einem anderen Mitarbeiter angenommen wurde.
- leaseKey: Eine eindeutige Kennung für einen Mietvertrag. Jeder Lease gilt für einen bestimmten Shard im Datenstrom und wird immer nur von einem Worker gehalten.

- leaseOwner: Der Arbeitnehmer, der diesen Mietvertrag innehat.
- ownerSwitchesSinceCheckpoint: Wie oft hat dieser Mietvertrag die Mitarbeiter gewechselt, seit ein Checkpoint das letzte Mal geschrieben wurde.
- parentShardId: Wird verwendet, um sicherzustellen, dass der übergeordnete Shard vollständig verarbeitet ist, bevor die Verarbeitung der untergeordneten Shards beginnt. So wird sichergestellt, dass Datensätze in der gleichen Reihenfolge verarbeitet werden, in der sie in den Stream eingespeist wurden.
- hashrange: Wird von `PeriodicShardSyncManager` verwendet, um regelmäßige Synchronisierungen durchzuführen, um fehlende Shards in der Leasetabelle zu finden und bei Bedarf Leases für sie zu erstellen.

  > **Note**
  >
  > Diese Daten sind in der Leasetabelle für jeden Shard enthalten, der mit KCL 1.14 und 2.3 beginnt. KCL Weitere Hinweise zu `PeriodicShardSyncManager` und die periodische Synchronisierung zwischen Leases und Shards finden Sie unter So wird eine Leasing-Tabelle mit den Shards in einem Kinesis-Datenstrom synchronisiert.

- childshards: Wird von `LeaseCleanupManager` verwendet, um den Verarbeitungsstatus des untergeordneten Shards zu überprüfen und zu entscheiden, ob der übergeordnete Shard aus der Leasetabelle gelöscht werden kann.

  > **Note**
  >
  > Diese Daten sind in der Leasing-Tabelle für jeden Shard enthalten, der mit KCL 1.14 und 2.3 beginnt. KCL

- shardID: Die ID des Shards.

  > **Note**
  >
  > Diese Daten sind nur dann in der Leasetabelle enthalten, wenn Sie Verarbeiten Sie mehrere Datenströme mit derselben KCL 2.x für Java-Consumer-Anwendung sind. Dies wird nur in KCL 2.x für Java unterstützt, beginnend mit KCL 2.3 für Java und höher.

- Streamname Die Kennung des Datenstroms im folgenden Format: `account-id:StreamName:streamCreationTimestamp`.

> **Note**
>
> Diese Daten sind nur dann in der Leasetabelle enthalten, wenn Sie Verarbeiten Sie mehrere Datenströme mit derselben KCL 2.x für Java-Consumer-Anwendung sind. Dies wird nur in KCL 2.x für Java unterstützt, beginnend mit KCL 2.3 für Java und höher.

Durchsatz

Wenn Ihre Amazon Kinesis Data Streams-Anwendung Ausnahmen für den bereitgestellten Durchsatz erhält, sollten Sie den bereitgestellten Durchsatz für die DynamoDB-Tabelle erhöhen. Das KCL erstellt die Tabelle mit einem bereitgestellten Durchsatz von 10 Lesevorgängen pro Sekunde und 10 Schreibvorgängen pro Sekunde, was für Ihre Anwendung jedoch möglicherweise nicht ausreichend ist. Beispiel: Wenn Ihre Amazon Kinesis Data Streams-Anwendung häufig Prüfpunkte setzt oder einen Stream verarbeitet, der aus vielen Shards besteht, müssen Sie den Durchsatz möglicherweise erhöhen.

Weitere Informationen zum bereitgestellten Durchsatz in DynamoDB finden Sie unter Lese-/Schreibkapazitätsmodus und Arbeiten mit Tabellen und Daten im Amazon-DynamoDB-Entwicklerhandbuch.

So wird eine Leasing-Tabelle mit den Shards in einem Kinesis-Datenstrom synchronisiert

Mitarbeiter in KCL Verbraucheranwendungen verwenden Leases, um Shards aus einem bestimmten Datenstrom zu verarbeiten. Die Informationen darüber, welcher Worker zu einem bestimmten Zeitpunkt welchen Shard least, werden in einer Leasetabelle gespeichert. Die Leasetabelle muss mit den neuesten Shard-Informationen aus dem Datenstrom synchron bleiben, während die KCL Verbraucheranwendung ausgeführt wird. KCLsynchronisiert die Leasetabelle mit den Shard-Informationen, die vom Kinesis Data Streams Streams-Dienst während des Bootstrapings der Consumer-Anwendung (entweder wenn die Consumer-Anwendung initialisiert oder neu gestartet wird) und auch immer dann, wenn ein Shard, der gerade verarbeitet wird, ein Ende erreicht (Resharding), erhält. Mit anderen Worten, die Worker oder eine KCL Verbraucheranwendung werden mit dem Datenstrom synchronisiert, den sie beim ersten Bootstrap der Verbraucheranwendung und immer dann verarbeiten, wenn die Verbraucheranwendung auf ein Datenstrom-Reshard-Ereignis trifft.

Themen

- Synchronisation in KCL 1.0-1.13 und 2.0-2.2 KCL
- Synchronisation in KCL 2.x, beginnend mit 2.3 und höher KCL
- Synchronisation in KCL 1.x, beginnend mit KCL 1.14 und höher

Synchronisation in KCL 1.0-1.13 und 2.0-2.2 KCL

In KCL 1.0-1.13 und KCL 2.0-2.2 wird beim Bootstraping der Verbraucheranwendung und auch bei jedem Reshard-Ereignis für den Datenstrom die Leasetabelle mit den Shard-Informationen KCL synchronisiert, die vom Kinesis Data Streams Streams-Dienst abgerufen wurden, indem die oder die Erkennung aufgerufen wird. `ListShards` `DescribeStream` APIs In allen oben aufgeführten KCL Versionen führt jeder Worker einer KCL Consumer-Anwendung die folgenden Schritte aus, um den Lease-/Shard-Synchronisierungsprozess während des Bootstrappings der Consumer-Anwendung und bei jedem Stream-Reshard-Ereignis durchzuführen:

- Ruft alle Shards für Daten des Streams ab, der gerade verarbeitet wird
- Ruft alle Shard-Leases aus der Leasetabelle ab
- Filtert jeden offenen Shard heraus, für den es in der Leasetabelle keinen Lease gibt
- Iteriert über alle gefundenen offenen Shards und für jeden offenen Shard ohne offenes übergeordnetes Element:
  - Durchläuft den Hierarchiebaum über den Pfad der Vorgänger, um festzustellen, ob der Shard ein Nachfolger ist. Ein Shard wird als untergeordnetes Element betrachtet, wenn gerade ein Vorgänger-Shard verarbeitet wird (der Leaseeintrag für den Vorgänger-Shard ist in der Leasetabelle vorhanden) oder wenn ein Vorgänger-Shard verarbeitet werden soll (wenn die Anfangsposition beispielsweise `TRIM_HORIZON` oder `AT_TIMESTAMP` ist)
  - Wenn es sich bei dem offenen Shard im Kontext um ein abgeleitetes Objekt handelt, KCL überprüft es den Shard anhand seiner ursprünglichen Position und erstellt, falls erforderlich, Leasingverträge für seine übergeordneten Elemente

Synchronisation in KCL 2.x, beginnend mit 2.3 und höher KCL

Beginnend mit den neuesten unterstützten Versionen von KCL 2.x (KCL2.3) und höher unterstützt die Bibliothek jetzt die folgenden Änderungen am Synchronisationsprozess. Diese Änderungen an der Lease-/Shard-Synchronisierung reduzieren die Anzahl der API Aufrufe von KCL Verbraucheranwendungen an den Kinesis Data Streams Streams-Service erheblich und optimieren das Lease-Management in Ihrer Verbraucheranwendung. KCL

- Wenn die Leasetabelle beim Bootstraping der Anwendung leer ist, KCL verwendet sie die Filteroption `ListShard` API von (den `ShardFilter` optionalen Anforderungsparameter), um Leases nur für einen Snapshot von Shards abzurufen und zu erstellen, die zu dem durch den Parameter angegebenen Zeitpunkt geöffnet sind. `ShardFilter` Mit dem `ShardFilter` Parameter können Sie die Antwort von herausfiltern. `ListShards` API Die einzige erforderliche Eigenschaft des `ShardFilter`-Parameters ist `Type`. KCLverwendet die Eigenschaft `Type` filter und die folgenden gültigen Werte, um eine Momentaufnahme offener Shards zu identifizieren und zurückzugeben, für die möglicherweise neue Leases erforderlich sind:
  - `AT_TRIM_HORIZON` – Die Antwort umfasst alle Shards, die am `TRIM_HORIZON` geöffnet waren.
  - `AT_LATEST` – Die Antwort enthält nur die aktuell geöffneten Shards des Datenstroms.
  - `AT_TIMESTAMP` – Die Antwort umfasst alle Shards, deren Startzeitstempel kleiner oder gleich dem angegebenen Zeitstempel sind und deren Endzeitstempel größer oder gleich dem angegebenen Zeitstempel ist oder solche, die noch offen sind.

  `ShardFilter` wird verwendet, wenn Leases für eine leere Leasetabelle erstellt werden, um Leases für einen Snapshot von Shards zu initialisieren, die unter `RetrievalConfig#initialPositionInStreamExtended` angegeben sind.

  Mehr über `ShardFilter` erfahren Sie unter https://docs.aws.amazon.com/kinesis/latest/APIReference/API_ShardFilter.html.
- Anstatt dass alle Worker die Leasing-/Shard-Synchronisierung durchführen, um die Leasetabelle mit den neuesten Shards im Datenstrom auf dem neuesten Stand zu halten, führt ein einziger gewählter Worker-Leader die Lease/Shard-Synchronisierung durch.
- KCL2.3 verwendet den `ChildShards` Rückgabeparameter von `GetRecords` und, `SubscribeToShard` APIs um die Leasing-/Shard-Synchronisierung durchzuführen, die bei geschlossenen Shards stattfindet, sodass ein KCL Worker nur Leases `SHARD_END` für die untergeordneten Shards des Shards erstellen kann, dessen Verarbeitung abgeschlossen ist. Diese Optimierung der Leasing-/Shard-Synchronisierung verwendet bei der gemeinsamen Nutzung von Consumer-Anwendungen den Parameter von. `ChildShards` `GetRecords` API Für Verbraucheranwendungen mit dediziertem Durchsatz (erweiterter Fan-Out) verwendet diese Optimierung der Lease-/Shard-Synchronisierung den Parameter von. `ChildShards` `SubscribeToShard` API Weitere Informationen finden Sie unter GetRecords, und. SubscribeToShardsChildShard
- Mit den oben genannten Änderungen KCL geht das Verhalten von von weg von dem Modell, dass alle Mitarbeiter über alle vorhandenen Shards Bescheid wissen, hin zu dem Modell, dass die Arbeiter nur mehr über die untergeordneten Scherben der Shards lernen, die jeder Arbeiter besitzt.

Daher werden neben der Synchronisation, die beim Bootstraping von Verbraucheranwendungen und bei Reshard-Ereignissen stattfindet, KCL jetzt auch zusätzliche regelmäßige Shard-/Lease-Scans durchgeführt, um potenzielle Lücken in der Leasetabelle zu identifizieren (mit anderen Worten, um mehr über alle neuen Shards zu erfahren), um sicherzustellen, dass der gesamte Hash-Bereich des Datenstroms verarbeitet wird, und um bei Bedarf Leases für sie zu erstellen. `PeriodicShardSyncManager`ist die Komponente, die für die regelmäßige Ausführung von Lease-/Shard-Scans verantwortlich ist.

Weitere Informationen zu Version KCL 2.3 finden Sie unter https://github.com/awslabs/amazon-kinesis-client/blob/master/ `PeriodicShardSyncManager` amazon-kinesis-client /src/main/java/software/amazon/kinesis/leases/ .java #L201 -L213. LeaseManagementConfig

In 2.3 stehen neue Konfigurationsoptionen zur Konfiguration zur Verfügung in: KCL `PeriodicShardSyncManager` `LeaseManagementConfig`

Name	Standardwert	Beschreibung
leasesRecoveryAuditorExecutionFrequencyMillis	120 000 (2 Minuten)	Häufigkeit (in Millionen), mit der der Auditor in der Leasetabelle nach teilweisen Leases sucht. Wenn der Auditor eine Lücke in den Leases für einen Stream feststellt, würde er die Shard-Synchronisierung auf der Grundlage von `leasesRecoveryAuditorInconsistencyConfidenceThreshold` auslösen.

Name	Standardwert	Beschreibung
leasesRecoveryAuditorInconsistencyConfidenceThreshold	3	Konfidenzschwellenwert für die regelmäßige Auditortätigkeit, um festzustellen, ob die Leases für einen Datenstrom in der Leasetabelle inkonsistent sind. Wenn der Auditor für diesen Datenstrom mehrmals hintereinander dieselben Inkonsistenzen feststellt, würde er eine Shard-Synchronisierung auslösen.

Außerdem werden jetzt neue CloudWatch Messwerte ausgegeben, um den Zustand von zu überwachen`PeriodicShardSyncManager`. Weitere Informationen finden Sie unter PeriodicShardSyncManager.

- Einschließlich einer Optimierung auf `HierarchicalShardSyncer`, um nur Leases für eine Shard-Ebene zu erstellen.

Synchronisation in KCL 1.x, beginnend mit KCL 1.14 und höher

Beginnend mit den neuesten unterstützten Versionen von KCL 1.x (KCL1.14) und höher unterstützt die Bibliothek jetzt die folgenden Änderungen am Synchronisationsprozess. Diese

Änderungen an der Lease-/Shard-Synchronisierung reduzieren die Anzahl der API Aufrufe von KCL Verbraucheranwendungen an den Kinesis Data Streams Streams-Service erheblich und optimieren das Lease-Management in Ihrer Verbraucheranwendung. KCL

- Wenn die Leasetabelle beim Bootstraping der Anwendung leer ist, KCL verwendet sie die Filteroption `ListShard` API von (den `ShardFilter` optionalen Anforderungsparameter), um Leases nur für einen Snapshot von Shards abzurufen und zu erstellen, die zu dem durch den Parameter angegebenen Zeitpunkt geöffnet sind. `ShardFilter` Mit dem `ShardFilter` Parameter können Sie die Antwort von herausfiltern. `ListShards` API Die einzige erforderliche Eigenschaft des `ShardFilter`-Parameters ist `Type`. KCLverwendet die Eigenschaft `Type` filter und die folgenden gültigen Werte, um eine Momentaufnahme offener Shards zu identifizieren und zurückzugeben, für die möglicherweise neue Leases erforderlich sind:
  - `AT_TRIM_HORIZON` – Die Antwort umfasst alle Shards, die am `TRIM_HORIZON` geöffnet waren.
  - `AT_LATEST` – Die Antwort enthält nur die aktuell geöffneten Shards des Datenstroms.
  - `AT_TIMESTAMP` – Die Antwort umfasst alle Shards, deren Startzeitstempel kleiner oder gleich dem angegebenen Zeitstempel sind und deren Endzeitstempel größer oder gleich dem angegebenen Zeitstempel ist oder solche, die noch offen sind.

  `ShardFilter` wird verwendet, wenn Leases für eine leere Leasetabelle erstellt werden, um Leases für einen Snapshot von Shards zu initialisieren, die unter `KinesisClientLibConfiguration#initialPositionInStreamExtended` angegeben sind.

  Mehr über `ShardFilter` erfahren Sie unter https://docs.aws.amazon.com/kinesis/latest/APIReference/API_ShardFilter.html.
- Anstatt dass alle Worker die Leasing-/Shard-Synchronisierung durchführen, um die Leasetabelle mit den neuesten Shards im Datenstrom auf dem neuesten Stand zu halten, führt ein einziger gewählter Worker-Leader die Lease/Shard-Synchronisierung durch.
- KCL1.14 verwendet den `ChildShards` Rückgabeparameter von `GetRecords` und, `SubscribeToShard` APIs um die Leasing-/Shard-Synchronisierung durchzuführen, die bei geschlossenen Shards stattfindet, sodass ein KCL Worker nur Leases `SHARD_END` für die untergeordneten Shards des Shards erstellen kann, dessen Verarbeitung abgeschlossen ist. GetRecordsWeitere ChildShardInformationen finden Sie unter und.
- Mit den oben genannten Änderungen KCL geht das Verhalten von weg von dem Modell, dass alle Arbeitnehmer über alle vorhandenen Scherben Bescheid wissen, hin zu dem Modell, dass Arbeitnehmer nur über die untergeordneten Scherben der Scherben lernen, die jeder

Mitarbeiter besitzt. Daher werden neben der Synchronisation, die beim Bootstraping von Verbraucheranwendungen und bei Reshard-Ereignissen stattfindet, KCL jetzt auch zusätzliche regelmäßige Shard-/Lease-Scans durchgeführt, um potenzielle Lücken in der Leasetabelle zu identifizieren (mit anderen Worten, um mehr über alle neuen Shards zu erfahren), um sicherzustellen, dass der gesamte Hash-Bereich des Datenstroms verarbeitet wird, und um bei Bedarf Leases für sie zu erstellen. `PeriodicShardSyncManager`ist die Komponente, die für die regelmäßige Ausführung von Lease-/Shard-Scans verantwortlich ist.

Wenn `KinesisClientLibConfiguration#shardSyncStrategyType` auf `ShardSyncStrategyType.SHARD_END` gesetzt ist, wird `PeriodicShardSync` `leasesRecoveryAuditorInconsistencyConfidenceThreshold` verwendet, um den Schwellenwert für die Anzahl der aufeinanderfolgenden Scans mit Lücken in der Leasetabelle zu bestimmen, nach dem eine Shard-Synchronisierung erzwungen werden soll. Wenn `KinesisClientLibConfiguration#shardSyncStrategyType` auf `ShardSyncStrategyType.PERIODIC` gesetzt ist, wird `leasesRecoveryAuditorInconsistencyConfidenceThreshold` ignoriert.

Weitere Informationen zu Version KCL 1.14 finden Sie unter `PeriodicShardSyncManager` https://github.com/awslabs/amazon-kinesis-clientKinesisClientLibConfiguration/blob/v1.x/src/main/java/com/amazonaws/services/kinesis/clientlibrary/lib/worker/ .java #L987 -L999.

KCLIn 1.14 ist eine neue Konfigurationsoption verfügbar, die konfiguriert werden kann in: `PeriodicShardSyncManager` `LeaseManagementConfig`

Name	Standardwert	Beschreibung
leasesRecoveryAuditorInconsistencyConfidenceThreshold	3	Konfidenzschwellenwert für die regelmäßige Auditortätigkeit, um festzustellen, ob die Leases für einen Datenstrom in der Leasetabelle inkonsistent sind. Wenn der Auditor für diesen Datenstrom mehrmals hintereinander dieselben Inkonsistenzen feststellt, würde er eine Shard-Synchronisierung auslösen.

Neue CloudWatch Metriken werden jetzt auch ausgegeben, um den Zustand von zu überwachen. `PeriodicShardSyncManager` Weitere Informationen finden Sie unter PeriodicShardSyncManager.

- KCL1.14 unterstützt jetzt auch die verzögerte Leasingbereinigung. Leases werden asynchron von `LeaseCleanupManager` bei Erreichen von `SHARD_END` gelöscht, wenn ein Shard entweder die Aufbewahrungsfrist des Datenstroms überschritten hat oder wenn er aufgrund eines Resharding-Vorgangs geschlossen wurde.

Es stehen neue Konfigurationsoptionen zur Verfügung, um `LeaseCleanupManager` zu konfigurieren.

Name	Standardwert	Beschreibung
leaseCleanupIntervalMillis	1 Minute	Intervall, in dem der Lease-Cleanup-Thread ausgeführt werden soll.
completedLeaseCleanupIntervalMillis	5 Minuten	Intervall, in dem überprüft werden soll, ob ein Lease abgeschlossen ist oder nicht.
garbageLeaseCleanupIntervalMillis	30 Minuten	Intervall, in dem geprüft werden soll, ob ein Lease Datenmüll ist (d. h. über die Aufbewahrungsfrist des Datenstroms hinaus abgeschnitten wurde) oder nicht.

- Einschließlich einer Optimierung auf `KinesisShardSyncer`, um nur Leases für eine Shard-Ebene zu erstellen.

Verarbeiten Sie mehrere Datenströme mit derselben KCL 2.x für Java-Consumer-Anwendung

In diesem Abschnitt werden die folgenden Änderungen in KCL 2.x für Java beschrieben, mit denen Sie KCL Verbraucheranwendungen erstellen können, die mehr als einen Datenstrom gleichzeitig verarbeiten können.

> **Important**
>
> Die Multistream-Verarbeitung wird nur in KCL 2.x für Java unterstützt, beginnend mit KCL 2.3 für Java und höher.
> Die Multistream-Verarbeitung wird für alle anderen Sprachen NOT unterstützt, in denen KCL 2.x implementiert werden kann.
> Die Multistream-Verarbeitung wird in allen Versionen von 1.x NOT unterstützt. KCL

- MultistreamTracker Schnittstelle

  Um eine Verbraucheranwendung zu erstellen, die mehrere Streams gleichzeitig verarbeiten kann, müssen Sie eine neue Schnittstelle namens implementieren MultistreamTracker. Diese Schnittstelle umfasst die `streamConfigList` Methode, die die Liste der Datenströme und ihrer Konfigurationen zurückgibt, die von der KCL Verbraucheranwendung verarbeitet werden sollen. Beachten Sie, dass die Datenströme, die verarbeitet werden, während der Laufzeit der Verbraucheranwendung geändert werden können. `streamConfigList`wird regelmäßig von der aufgerufenKCL, um mehr über die Änderungen der zu verarbeitenden Datenströme zu erfahren.

  Die `streamConfigList` Methode füllt die StreamConfigListe auf.

```
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;
}
```

Beachten Sie, dass es sich bei den Feldern `StreamIdentifier` und `InitialPositionInStreamExtended` um Pflichtfelder handelt, während `consumerArn` optional ist. Sie müssen das `consumerArn` nur angeben, wenn Sie KCL 2.x verwenden, um eine erweiterte Fan-Out-Consumer-Anwendung zu implementieren.

Weitere Informationen dazu finden Sie unter https://github.com/awslabs/amazon-kinesis-client/blob/v2.5.8/ `StreamIdentifier` /src/main/java/software/amazon/kinesis/common/ .java #L129 amazon-kinesis-client. StreamIdentifier Um eine zu erstellen, empfehlen wir, eine Multistream-Instanz aus der und der zu erstellen, die in Version 2.5.0 und höher verfügbar ist. `StreamIdentifier` `streamArn` `streamCreationEpoch` Erstellen Sie in KCL Version 2.3 und Version 2.4, die dies nicht unterstützen`streamArm`, eine Multistream-Instance mithilfe des folgenden Formats. `account-id:StreamName:streamCreationTimestamp` Dieses Format ist veraltet und wird ab der nächsten Hauptversion nicht mehr unterstützt.

`MultistreamTracker` beinhaltet auch eine Strategie zum Löschen von Leases alter Streams in der Leasetabelle (`formerStreamsLeasesDeletionStrategy`). Beachten Sie, dass die Strategie CANNOT während der Laufzeit der Verbraucheranwendung geändert wird. Weitere Informationen finden Sie unter https://github.com/awslabs/amazon-kinesis-client/blob/0c5042dadf794fe988438436252a5a8fe70b6b0b/ amazon-kinesis-client /src/main/java/software/amazon/kinesis/processor/ .java FormerStreamsLeasesDeletionStrategy

- ConfigsBuilderist eine anwendungsweite Klasse, mit der Sie alle 2.x-Konfigurationseinstellungen angeben können, die beim Erstellen Ihrer Verbraucheranwendung verwendet werden sollen. KCL KCL `ConfigsBuilder`Die Klasse unterstützt jetzt die Schnittstelle. `MultistreamTracker` Sie können ConfigsBuilder entweder mit dem Namen des einen Datenstroms initialisieren, aus dem Datensätze abgerufen werden sollen:

```
/**
    * 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;
    }
```

Oder Sie können ConfigsBuilder mit initialisieren, `MultiStreamTracker` wenn Sie eine KCL Verbraucheranwendung implementieren möchten, die mehrere Streams gleichzeitig verarbeitet.

```
* 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;
    }
```

- Da die Multistream-Unterstützung für Ihre KCL Consumer-Anwendung implementiert ist, enthält jede Zeile der Leasetabelle der Anwendung jetzt die Shard-ID und den Stream-Namen der mehreren Datenströme, die diese Anwendung verarbeitet.
- Wenn die Multistream-Unterstützung für Ihre KCL Verbraucheranwendung implementiert ist, hat sie die leaseKey folgende Struktur:. `account-id:StreamName:streamCreationTimestamp:ShardId` z. B. `111111111:multiStreamTest-1:12345:shardId-000000000336`.

> **Important**
>
> Wenn Ihre bestehende KCL Verbraucheranwendung so konfiguriert ist, dass sie nur einen Datenstrom verarbeitet, ist der leaseKey (der Hash-Schlüssel für die Leasing-Tabelle) die Shard-ID. Wenn Sie diese bestehende KCL Verbraucheranwendung so umkonfigurieren, dass sie mehrere Datenströme verarbeitet, wird Ihre Leasing-Tabelle beschädigt, da bei Multistream-Unterstützung die leaseKey Struktur wie folgt aussehen muss:. `account-id:StreamName:StreamCreationTimestamp:ShardId`

Verwenden Sie das KCL mit der Schemaregistrierung AWS Glue

Sie können Ihre Kinesis-Datenströme in die AWS Glue Schema Registry integrieren. Mit der AWS Glue Schema Registry können Sie Schemas zentral erkennen, steuern und weiterentwickeln und gleichzeitig sicherstellen, dass die erstellten Daten kontinuierlich anhand eines registrierten Schemas validiert werden. Ein Schema definiert die Struktur und das Format eines Datensatzes. Ein Schema ist eine versionierte Spezifikation für zuverlässige Datenveröffentlichung, -nutzung oder -speicherung. Mit der AWS Glue Schema Registry können Sie die end-to-end Datenqualität und die Datenverwaltung in Ihren Streaming-Anwendungen verbessern. Weitere Informationen finden Sie unter AWS Glue Schema Registry. Eine Möglichkeit, diese Integration einzurichten, ist die KCL in Java.

> **Important**
>
> Derzeit wird die Integration von Kinesis Data Streams und AWS Glue Schema Registry nur für Kinesis-Datenstreams unterstützt, die in Java implementierte KCL 2.3-Consumer verwenden. Mehrsprachige Unterstützung wird nicht bereitgestellt. KCL1.0-Verbraucher werden nicht unterstützt. KCL2.x-Verbraucher vor KCL 2.3 werden nicht unterstützt.

Detaillierte Anweisungen zum Einrichten der Integration von Kinesis Data Streams mit Schema Registry mithilfe von finden Sie im Abschnitt „Interaktion mit Daten mithilfe der KPL KCL /-Bibliotheken“ unter Anwendungsfall: Integration von Amazon Kinesis Data Streams mit der AWS Glue Schema Registry. KCL

Entwickeln Sie maßgeschneiderte Verbraucher mit gemeinsamem Durchsatz

Wenn Sie keinen spezifischen Durchsatz beim Empfangen von Daten von Kinesis Data Streams und keine Verbreitungswerte für Leseoperationen von unter 200 ms benötigen, können Sie Konsumentenanwendungen wie in den folgenden Themen beschrieben erstellen. Sie können die Kinesis Client Library (KCL) oder die AWS SDK for Java verwenden.

Themen

- Entwickeln Sie benutzerdefinierte Verbraucher mit gemeinsamem Durchsatz mithilfe von KCL
- Entwickeln Sie benutzerdefinierte Verbraucher mit gemeinsamem Durchsatz mithilfe der AWS SDK for Java

Für Informationen zum Erstellen von Verbrauchern, die Datensätze aus Kinesis-Datenströmen mit dediziertem Durchsatz empfangen können, siehe Entwickeln Sie maßgeschneiderte Verbraucher mit dediziertem Durchsatz (verbesserter Fan-Out).

Entwickeln Sie benutzerdefinierte Verbraucher mit gemeinsamem Durchsatz mithilfe von KCL

Eine der Methoden zur Entwicklung einer benutzerdefinierten Verbraucheranwendung mit gemeinsamem Durchsatz ist die Verwendung der Kinesis Client Library (KCL).

Wählen Sie für die KCL Version, die Sie verwenden, eines der folgenden Themen aus.

Themen

- Entwickeln Sie KCL 1.x-Verbraucher
- Develop KCL 2.x-Verbraucher

Entwickeln Sie KCL 1.x-Verbraucher

Sie können mithilfe der Kinesis Client Library () KCL eine Verbraucheranwendung für Amazon Kinesis Data Streams entwickeln.

Weitere Informationen zu finden Sie KCL unter. Was ist die Kinesis Client Library?

Wählen Sie je nach der Option, die Sie verwenden möchten, aus den folgenden Themen.

Inhalt

- Entwickeln Sie einen Kinesis Client Library-Consumer in Java
- Entwickeln Sie einen Kinesis Client Library-Consumer in Node.js
- Entwickeln Sie einen Kinesis Client Library-Benutzer in. NET
- Entwickeln Sie einen Kinesis Client Library-Consumer in Python
- Entwickeln Sie einen Kinesis Client Library-Consumer in Ruby

Entwickeln Sie einen Kinesis Client Library-Consumer in Java

Sie können die Kinesis Client Library (KCL) verwenden, um Anwendungen zu erstellen, die Daten aus Ihren Kinesis-Datenströmen verarbeiten. Die Kinesis Client Library ist in mehreren Sprachen verfügbar. In diesem Thema wird Java behandelt. Die Javadoc-Referenz finden Sie im AWS Javadoc-Thema für Class. AmazonKinesisClient

Gehen Sie zur Kinesis Client Library (JavaKCL) GitHub, um Java herunterzuladen. Gehen Sie zur KCLSuchergebnisseite, um Java KCL auf Apache Maven zu finden. Um Beispielcode für eine KCL Java-Consumer-Anwendung von herunterzuladen GitHub, gehen Sie auf die Seite KCLfür das Java-Beispielprojekt unter GitHub.

Die Beispielanwendung verwendet Apache Commons Logging. Sie können die Konfiguration der Protokollierung in der statischen Methode `configure` ändern, die in der Datei

`AmazonKinesisApplicationSample.java` definiert ist. Weitere Informationen zur Verwendung von Apache Commons Logging mit Log4j und AWS Java-Anwendungen finden Sie unter Logging with Log4j im Developer Guide.AWS SDK for Java

Sie müssen die folgenden Aufgaben ausführen, wenn Sie eine KCL Verbraucheranwendung in Java implementieren:

Aufgaben

- Implementieren Sie die IRecordProcessor Methoden
- Implementieren Sie eine Klassenfabrik für die Schnittstelle IRecordProcessor
- Erstellen Sie einen Arbeiter
- Ändern Sie die Konfigurationseigenschaften
- Migrieren Sie zu Version 2 der Record Processor-Schnittstelle

Implementieren Sie die IRecordProcessor Methoden

Die unterstützt KCL derzeit zwei Versionen der `IRecordProcessor` Schnittstelle: Die ursprüngliche Schnittstelle ist mit der ersten Version von verfügbarKCL, und Version 2 ist ab Version 1.5.0 verfügbar. KCL Beide Schnittstellen werden vollständig unterstützt. Ihre Wahl hängt von den speziellen Anforderungen Ihres Anwendungsfalls ab. Um mehr über Unterschiede zu erfahren, betrachten Sie die lokal entwickelten Javadocs oder den Quellcode. In den folgenden Abschnitten wird die Mindestimplementierung für die ersten Schritte beschrieben.

IRecordProcessorVersionen

- Ursprüngliche Schnittstelle (Version 1)
- Aktualisierte Schnittstelle (Version 2)

Ursprüngliche Schnittstelle (Version 1)

Die ursprüngliche `IRecordProcessor` Schnittstelle (`package com.amazonaws.services.kinesis.clientlibrary.interfaces`) stellt die folgenden Datensatzverarbeitermethoden bereit, die Ihr Konsument implementieren muss. Das Beispiel stellt Implementierungen bereit, die Sie als Ausgangspunkt verwenden können (siehe `AmazonKinesisApplicationSampleRecordProcessor.java`).

```
public void initialize(String shardId)
```

```
public void processRecords(List<Record> records, IRecordProcessorCheckpointer
 checkpointer)
public void shutdown(IRecordProcessorCheckpointer checkpointer, ShutdownReason reason)
```

initialize

Der KCL ruft die `initialize` Methode auf, wenn der Datensatzprozessor instanziiert wird, und übergibt eine bestimmte Shard-ID als Parameter. Dieser Datensatzverarbeiter verarbeitet nur diese Shard und in der Regel ist dies auch umgekehrt der Fall (diese Shard wird nur durch diesen Datensatverarbeiter verarbeitet). Ihr Konsument sollte jedoch die Möglichkeit berücksichtigen, dass ein Datensatz mehr als einmal verarbeitet werden könnte. Kinesis Data Streams besitzt eine Semantik nach dem Grundsatz mindestens einmal. Das bedeutet, dass jeder Datensatz aus einer Shard mindestens einmal von einem Worker in Ihrem Konsumenten verarbeitet wird. Weitere Informationen zu Fällen, in denen eine bestimmte Shard möglicherweise durch mehr als einen Auftragnehmer verarbeitet wird, finden Sie unter Verwenden Sie Resharding, Skalierung und Parallelverarbeitung, um die Anzahl der Shards zu ändern.

```
public void initialize(String shardId)
```

processRecords

Der KCL ruft die `processRecords` Methode auf und übergibt eine Liste von Datensätzen aus dem durch die Methode angegebenen Shard. `initialize(shardId)` Der Datensatzverarbeiter verarbeitet die Daten in diesen Datensätzen entsprechend der Semantik des Konsumenten. Beispielsweise kann der Auftragnehmer eine Transformation für die Daten ausführen und das Ergebnis dann in einem Amazon Simple Storage Service (Amazon S3)-Bucket speichern.

```
public void processRecords(List<Record> records, IRecordProcessorCheckpointer
 checkpointer)
```

Zusätzlich zu den Daten selbst enthält der Datensatz auch eine Sequenznummer und einen Partitionsschlüssel. Der Auftragnehmer kann diese Werte beim Verarbeiten der Daten verwenden. Beispielsweise könnte der Auftragnehmer basierend auf dem Wert des Partitionsschlüssels den S3-Bucket wählen, in dem die Daten gespeichert werden sollen. Die Klasse `Record` stellt die folgenden Methoden bereit, die Zugriff auf die Daten des Datensatzes, die Sequenznummer und den Partitionsschlüssel bieten.

```
record.getData()
```

```
record.getSequenceNumber()
record.getPartitionKey()
```

In diesem Beispiel weist die private Methode `processRecordsWithRetries` Code auf, der zeigt, wie ein Auftragnehmer auf die Daten des Datensatzes, die Sequenznummer und den Partitionsschlüssel zugreifen kann.

Kinesis Data Streams erfordert, dass der Datensatzverarbeiter die Datensätze nachverfolgt, die bereits in einer Shard verarbeitet wurden. Der KCL kümmert sich für Sie um dieses Tracking, indem er einen Checkpointer (`IRecordProcessorCheckpointer`) an übergibt. `processRecords` Der Datensatzprozessor ruft die `checkpoint` Methode auf dieser Schnittstelle auf, um ihn darüber zu informieren, wie weit er bei KCL der Verarbeitung der Datensätze im Shard fortgeschritten ist. Wenn der Worker ausfällt, KCL verwendet er diese Informationen, um die Verarbeitung des Shards beim letzten bekannten verarbeiteten Datensatz neu zu starten.

Bei einem Split- oder Merge-Vorgang beginnt der KCL erst mit der Verarbeitung der neuen Shards, wenn die Prozessoren für die ursprünglichen Shards aufgerufen haben, `checkpoint` um zu signalisieren, dass die gesamte Verarbeitung der ursprünglichen Shards abgeschlossen ist.

Wenn Sie keinen Parameter übergeben, KCL wird davon ausgegangen, dass der Aufruf von `checkpoint` bedeutet, dass alle Datensätze verarbeitet wurden, bis zu dem letzten Datensatz, der an den Datensatzprozessor übergeben wurde. Daher sollte der Datensatzverarbeiter die Methode `checkpoint` erst aufrufen, wenn er alle Datensätze in der Liste, die ihm übergeben wurden, verarbeitet hat. Datensatzverarbeiter müssen `checkpoint` nicht bei jedem Aufruf von `processRecords` aufrufen. Ein Prozessor könnte beispielsweise `checkpoint` bei jedem dritten Aufruf von `processRecords` aufrufen. Sie können optional die exakte Sequenznummer eines Datensatzes als Parameter für `checkpoint` angeben. In diesem Fall KCL geht der davon aus, dass alle Datensätze nur bis zu diesem Datensatz verarbeitet wurden.

Im Beispiel zeigt die private Methode `checkpoint`, wie `IRecordProcessorCheckpointer.checkpoint` mithilfe der entsprechenden Ausnahmebehandlung und Wiederholungslogik aufgerufen wird.

Das KCL ist darauf angewiesen, alle Ausnahmen `processRecords` zu behandeln, die sich aus der Verarbeitung der Datensätze ergeben. Wenn eine Ausnahme ausgelöst wird`processRecords`, werden die Datensätze KCL übersprungen, die vor der Ausnahme übergeben wurden. Das heißt, diese Datensätze werden nicht erneut an den Datensatzprozessor gesendet, der die Ausnahme ausgelöst hat, oder an einen anderen Datensatzprozessor im Verbraucher.

shutdown

Die `shutdown` Methode wird entweder KCL aufgerufen, wenn die Verarbeitung beendet ist (der Grund für das Herunterfahren ist`TERMINATE`) oder wenn der Worker nicht mehr reagiert (der Grund für das Herunterfahren ist`ZOMBIE`).

```
public void shutdown(IRecordProcessorCheckpointer checkpointer, ShutdownReason reason)
```

Die Verarbeitung endet, wenn der Datensatzverarbeiter keine weiteren Datensätze aus der Shard erhält, entweder weil die Shard geteilt oder zusammengeführt wurde oder weil der Stream gelöscht wurde.

Der übergibt KCL auch eine `IRecordProcessorCheckpointer` Schnittstelle an`shutdown`. Wenn der Grund für das Herunterfahren `TERMINATE` ist, sollte der Datensatzverarbeiter alle Datensätze fertigstellen und dann die Methode `checkpoint` in seiner Schnittstelle aufrufen.

Aktualisierte Schnittstelle (Version 2)

Die aktualisierte `IRecordProcessor` Schnittstelle (`package com.amazonaws.services.kinesis.clientlibrary.interfaces.v2`) stellt die folgenden Datensatzverarbeitermethoden bereit, die Ihr Konsument implementieren muss:

```
void initialize(InitializationInput initializationInput)
void processRecords(ProcessRecordsInput processRecordsInput)
void shutdown(ShutdownInput shutdownInput)
```

Sie können auf alle Argumente aus der ursprünglichen Version der Schnittstelle über Get-Methoden für die Container-Objekte zugreifen. Um die Liste der Datensätze in `processRecords()` abzurufen, können Sie `processRecordsInput.getRecords()` verwenden.

Ab Version 2 dieser Schnittstelle (KCL1.5.0 und höher) sind zusätzlich zu den Eingaben der ursprünglichen Schnittstelle die folgenden neuen Eingaben verfügbar:

Startsequenznummer

Im `InitializationInput`-Objekt, das an die Operation `initialize()` übergeben wird, die Startsequenznummer, aus der Datensätze für die Datenverarbeiter-Instance bereitgestellt würden. Dies ist die Sequenznummer, die zuletzt durch die Datensatzverarbeiter-Instance überprüft wurde, die dieselbe Shard zuvor verarbeitet hat. Sie wird für den Fall angegeben, dass Ihre Anwendung diese Informationen benötigt.

Ausstehende Checkpoint-Sequenznummer

Im `InitializationInput`-Objekt, das an die Operation `initialize()` übergeben wird, die ausstehende Checkpoint-Sequenznummer (wenn vorhanden), die nicht übergeben werden konnte, bevor die vorherige Datensatzverarbeiter-Instance angehalten wurde.

Implementieren Sie eine Klassenfabrik für die Schnittstelle IRecordProcessor

Sie müssen darüber hinaus eine Factory für die Klasse implementieren, die die Datensatzverarbeitermethoden implementiert. Wenn der Konsument den Auftragnehmer instanziiert, übergibt er dieser Factory eine Referenz.

Im Beispiel wird die Factory-Klasse in der Datei `AmazonKinesisApplicationSampleRecordProcessorFactory.java` mithilfe der ursprünglichen Datensatzverarbeiter-Schnittstelle implementiert. Wenn Sie möchten, dass die Class Factory Datensatzverarbeiter mit Version 2 erstellt, verwenden Sie den Paketnamen `com.amazonaws.services.kinesis.clientlibrary.interfaces.v2`.

```
public class SampleRecordProcessorFactory implements IRecordProcessorFactory {
    /**
    * Constructor.
    */
    public SampleRecordProcessorFactory() {
        super();
    }
    /**
    * {@inheritDoc}
    */
    @Override
    public IRecordProcessor createProcessor() {
        return new SampleRecordProcessor();
    }
}
```

Erstellen Sie einen Arbeiter

Wie unter beschriebenImplementieren Sie die IRecordProcessor Methoden, stehen zwei Versionen der KCL Record Processor-Schnittstelle zur Auswahl, was sich darauf auswirkt, wie Sie einen Worker erstellen würden. Die ursprüngliche Datensatzverarbeiterschnittstelle verwendet die folgende Codestruktur, um einen Auftragnehmer zu erstellen:

```
final KinesisClientLibConfiguration config = new KinesisClientLibConfiguration(...)
final IRecordProcessorFactory recordProcessorFactory = new RecordProcessorFactory();
final Worker worker = new Worker(recordProcessorFactory, config);
```

Mit Version 2 der Datensatzverarbeiterschnittstelle können Sie `Worker.Builder` verwenden, um einen Auftragnehmer zu erstellen, ohne sich Gedanken über den Konstruktor und die Reihenfolge der Argumente zu machen. Die aktualisierte Datensatzverarbeiterschnittstelle verwendet die folgende Codestruktur, um einen Auftragnehmer zu erstellen:

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

Ändern Sie die Konfigurationseigenschaften

Das Beispiel zeigt Standardwerte für Konfigurationseigenschaften. Diese Konfigurationsdaten für den Auftragnehmer werden anschließend in einem `KinesisClientLibConfiguration`-Objekt konsolidiert. Dieses Objekt und eine Referenz auf die Class Factory für `IRecordProcessor` werden an den Aufruf übergeben, der den Auftragnehmer instanziiert. Sie können diese Eigenschaften mithilfe einer Java-Eigenschaftendatei (siehe `AmazonKinesisApplicationSample.java`) durch eigene Werte überschreiben.

Anwendungsname

Das KCL erfordert einen Anwendungsnamen, der für Ihre Anwendungen und für alle Amazon DynamoDB-Tabellen in derselben Region eindeutig ist. Sie verwendet den Wert der Anwendungsnamenkonfiguration auf folgende Arten:

- Für mit diesem Anwendungsnamen verknüpfte Auftragnehmer wird angenommen, dass sie gemeinsam im gleichen Stream arbeiten. Diese Auftragnehmer können auf mehrere Instances verteilt sein. Wenn Sie eine zusätzliche Instance desselben Anwendungscodes ausführen, jedoch mit einem anderen Anwendungsnamen, KCL behandelt das die zweite Instance als eine völlig separate Anwendung, die ebenfalls auf demselben Stream ausgeführt wird.
- Das KCL erstellt eine DynamoDB-Tabelle mit dem Anwendungsnamen und verwendet die Tabelle, um Statusinformationen (wie Checkpoints und Worker-Shard-Mapping) für die Anwendung zu

verwalten. Jede Anwendung verfügt über eine eigene DynamoDB-Tabelle. Weitere Informationen finden Sie unter Verwenden Sie eine Leasetabelle, um nachzuverfolgen, welche Shards von der Consumer-Anwendung verarbeitet wurden KCL.

Richten Sie Anmeldeinformationen ein

Sie müssen Ihre AWS Anmeldeinformationen einem der Anmeldeinformationsanbieter in der Kette der Standardanmeldedienstanbieter zur Verfügung stellen. Wenn Sie Ihren Consumer beispielsweise auf einer EC2 Instance ausführen, empfehlen wir, dass Sie die Instance mit einer IAM Rolle starten. AWS Anmeldeinformationen, die die mit dieser IAM Rolle verknüpften Berechtigungen widerspiegeln, werden den Anwendungen auf der Instance über deren Instance-Metadaten zur Verfügung gestellt. Dies ist die sicherste Methode, Anmeldeinformationen für einen Verbraucher zu verwalten, der auf einer EC2 Instance läuft.

Die Beispielanwendung versucht zunächst, IAM Anmeldeinformationen aus den Metadaten der Instanz abzurufen:

```
credentialsProvider = new InstanceProfileCredentialsProvider();
```

Wenn die Beispielanwendung keine Anmeldeinformationen aus den Instance-Metadaten abrufen kann, versucht sie, Anmeldeinformationen aus einer Eigenschaftendatei abzurufen:

```
credentialsProvider = new ClasspathPropertiesFileCredentialsProvider();
```

Weitere Informationen zu Instance-Metadaten finden Sie unter Instance-Metadaten im EC2Amazon-Benutzerhandbuch.

Verwenden Sie die Worker-ID für mehrere Instances

Derselbe Initialisierungscode erstellt unter Verwendung des Namens des lokalen Computers und Anfügung eines global eindeutigen Bezeichners eine ID für den Auftragnehmer, `workerId`, wie im folgenden Codeauszug gezeigt. Dieser Ansatz unterstützt das Szenario mit mehreren Instances der Konsumentenanwendung, die auf einem einzigen Computer ausgeführt werden.

```
String workerId = InetAddress.getLocalHost().getCanonicalHostName() + ":" +
 UUID.randomUUID();
```

Migrieren Sie zu Version 2 der Record Processor-Schnittstelle

Wenn Sie Code migrieren möchten, der die ursprüngliche Schnittstelle verwendet, sind zusätzlich zu den zuvor beschriebenen Schritten die folgenden Schritte erforderlich:

1. Ändern der Datensatzverarbeiterklasse, um Version 2 der Datensatzverarbeiterschnittstelle zu importieren:

   ```
   import com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor;
   ```

2. Ändern der Referenzen zu Eingaben, um `get`-Methoden für die Container-Objekte zu verwenden. In der Operation `shutdown()` ändern Sie beispielsweise „`checkpointer`“ in „`shutdownInput.getCheckpointer()`“.
3. Ändern der Datensatzverarbeiter-Factory, um Version 2 der Datensatzverarbeiter-Factory zu importieren:

   ```
   import
    com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessorFactory;
   ```

4. Ändern der Konstruktion für den Auftragnehmer, um `Worker.Builder` zu verwenden. Beispielsweise:

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

Entwickeln Sie einen Kinesis Client Library-Consumer in Node.js

Sie können die Kinesis Client Library (KCL) verwenden, um Anwendungen zu erstellen, die Daten aus Ihren Kinesis-Datenströmen verarbeiten. Die Kinesis Client Library ist in mehreren Sprachen verfügbar. In diesem Thema wird Node.js behandelt.

Die KCL ist eine Java-Bibliothek. Unterstützung für andere Sprachen als Java wird über eine mehrsprachige Schnittstelle bereitgestellt, die als. MultiLangDaemon Dieser Daemon basiert auf Java und wird im Hintergrund ausgeführt, wenn Sie eine andere KCL Sprache als Java verwenden. Wenn Sie also KCL for Node.js installieren und Ihre Consumer-App vollständig in Node.js schreiben, muss Java aufgrund von trotzdem auf Ihrem System installiert sein. MultiLangDaemon Darüber hinaus MultiLangDaemon verfügt es über einige Standardeinstellungen, die Sie möglicherweise

an Ihren Anwendungsfall anpassen müssen, z. B. an die AWS Region, mit der eine Verbindung hergestellt wird. Weitere Informationen dazu finden Sie MultiLangDaemon auf GitHub der KCL MultiLangDaemon Projektseite.

Gehen Sie zur Kinesis Client Library (Node.js) GitHub, um die Datei Node.js KCL von herunterzuladen.

Downloads von Beispiel-Code

In Node.js sind zwei Codebeispiele verfügbar: KCL

- basic-sample

  Wird in den folgenden Abschnitten verwendet, um die Grundlagen der Erstellung einer KCL Verbraucheranwendung in Node.js zu veranschaulichen.
- click-stream-sample

  Etwas komplexere und verwendet ein reales Szenario, nachdem Sie sich mit dem grundlegenden Beispiel-Code vertraut gemacht haben. Dieses Beispiel wird hier nicht behandelt, enthält aber eine README Datei mit weiteren Informationen.

Bei der Implementierung einer KCL Verbraucheranwendung in Node.js müssen Sie die folgenden Aufgaben ausführen:

Aufgaben

- Implementieren Sie den Record Processor
- Ändern Sie die Konfigurationseigenschaften

Implementieren Sie den Record Processor

Der einfachste Benutzer, der KCL for Node.js verwendet, muss eine `recordProcessor` Funktion implementieren, die wiederum die Funktionen `initialize`, `processRecords` und enthält`shutdown`. Das Beispiel zeigt eine Implementierung, die Sie als Ausgangspunkt verwenden können (siehe `sample_kcl_app.js`).

```
function recordProcessor() {
  // return an object that implements initialize, processRecords and shutdown
 functions.}
```

initialize

Der KCL ruft die `initialize` Funktion auf, wenn der Recordprozessor gestartet wird. Dieser Datensatzverarbeiter verarbeitet nur die Shard-ID, die als `initializeInput.shardId` übergeben wird. In der Regel ist dies auch umgekehrt der Fall (diese Shard wird nur durch diesen Datensatverarbeiter verarbeitet). Ihr Konsument sollte jedoch die Möglichkeit berücksichtigen, dass ein Datensatz mehr als einmal verarbeitet werden könnte. Das liegt daran, dass Kinesis Data Streams eine Semantik nach dem Grundsatz mindestens einmal hat. Das bedeutet, dass jeder Datensatz aus einer Shard mindestens einmal von einem Worker in Ihrem Konsumenten verarbeitet wird. Weitere Informationen zu Fällen, in denen eine bestimmte Shard möglicherweise durch mehr als einen Auftragnehmer verarbeitet wird, finden Sie unter Verwenden Sie Resharding, Skalierung und Parallelverarbeitung, um die Anzahl der Shards zu ändern.

```
initialize: function(initializeInput, completeCallback)
```

processRecords

Der KCL ruft diese Funktion mit einer Eingabe auf, die eine Liste von Datensätzen aus dem für die `initialize` Funktion angegebenen Shard enthält. Der von Ihnen implementierte Datensatzverarbeiter verarbeitet die Daten in diesen Datensätzen entsprechend der Semantik Ihres Konsumenten. Beispielsweise kann der Auftragnehmer eine Transformation für die Daten ausführen und das Ergebnis dann in einem Amazon Simple Storage Service (Amazon S3)-Bucket speichern.

```
processRecords: function(processRecordsInput, completeCallback)
```

Zusätzlich zu den Daten enthält der Datensatz auch eine Sequenznummer und einen Partitionsschlüssel, die der Auftragnehmer beim Verarbeiten der Daten verwenden kann. Beispielsweise könnte der Auftragnehmer basierend auf dem Wert des Partitionsschlüssels den S3-Bucket wählen, in dem die Daten gespeichert werden sollen. Das `record`-Anmeldeverzeichnis stellt die folgenden Schlüssel-Wert-Paare für den Zugriff auf die Daten des Datensatzes, die Sequenznummer und den Partitionsschlüssel bereit:

```
record.data
record.sequenceNumber
record.partitionKey
```

Beachten Sie, dass die Daten Base64-kodiert sind.

Im einfachen Beispiel weist die Funktion `processRecords` Code auf, der zeigt, wie ein Auftragnehmer auf die Daten des Datensatzes, die Sequenznummer und den Partitionsschlüssel zugreifen kann.

Kinesis Data Streams erfordert, dass der Datensatzverarbeiter die Datensätze nachverfolgt, die bereits in einer Shard verarbeitet wurden. Der KCL kümmert sich um dieses Tracking für ein `checkpointer` Objekt, das als `processRecordsInput.checkpointer` übergeben wird. Ihr Datensatzprozessor ruft die `checkpointer.checkpoint` Funktion auf, um darüber zu informieren, KCL wie weit er bei der Verarbeitung der Datensätze im Shard fortgeschritten ist. Falls der Worker ausfällt, KCL verwendet er diese Informationen, wenn Sie die Verarbeitung des Shards erneut starten, sodass die Verarbeitung mit dem letzten bekannten verarbeiteten Datensatz fortgesetzt wird.

Bei einem Split- oder Merge-Vorgang beginnt der erst KCL mit der Verarbeitung der neuen Shards, wenn die Prozessoren für die ursprünglichen Shards aufgerufen haben, `checkpoint` um zu signalisieren, dass die gesamte Verarbeitung der ursprünglichen Shards abgeschlossen ist.

Wenn Sie die Sequenznummer nicht an die `checkpoint` Funktion übergeben, KCL wird davon ausgegangen, dass der Aufruf von `checkpoint` bedeutet, dass alle Datensätze verarbeitet wurden, bis zu dem letzten Datensatz, der an den Datensatzprozessor übergeben wurde. Daher sollte der Datensatzverarbeiter die Methode `checkpoint` erst aufrufen, wenn er alle Datensätze in der Liste, die ihm übergeben wurden, verarbeitet hat. Datensatzverarbeiter müssen `checkpoint` nicht bei jedem Aufruf von `processRecords` aufrufen. Ein Verarbeiter könnte beispielsweise `checkpoint` bei jedem dritten Aufruf oder beim Eintritt eines Ereignisses außerhalb Ihres Datensatzverarbeiters aufrufen, beispielsweise eines von Ihnen implementierten Verifizierung-/Validierungs-Service.

Sie können optional die exakte Sequenznummer eines Datensatzes als Parameter für `checkpoint` angeben. In diesem Fall KCL wird davon ausgegangen, dass alle Datensätze nur bis zu diesem Datensatz verarbeitet wurden.

Die einfache Beispielanwendung zeigt den einfachsten möglichen Aufruf der Funktion `checkpointer.checkpoint`. Sie können weitere Checkpoint-Logik hinzufügen, die Sie an diesem Punkt in der Funktion für Ihren Konsumenten benötigen.

shutdown

Der KCL ruft die `shutdown` Funktion entweder auf, wenn die Verarbeitung beendet `shutdownInput.reason` ist `TERMINATE` oder wenn der Worker nicht mehr reagiert (`shutdownInput.reasonistZOMBIE`).

```
shutdown: function(shutdownInput, completeCallback)
```

Die Verarbeitung endet, wenn der Datensatzverarbeiter keine weiteren Datensätze aus der Shard erhält, entweder weil die Shard geteilt oder zusammengeführt wurde oder weil der Stream gelöscht wurde.

Der übergibt KCL auch ein `shutdownInput.checkpointer` Objekt an`shutdown`. Wenn der Grund für das Herunterfahren `TERMINATE` ist, sollten Sie sicherstellen, dass der Datensatzverarbeiter die Verarbeitung aller Datensätze fertiggestellt hat, und dann die Funktion `checkpoint` in seiner Schnittstelle aufrufen.

Ändern Sie die Konfigurationseigenschaften

Das Beispiel zeigt Standardwerte für die Konfigurationseigenschaften. Sie können diese Eigenschaften mit eigenen Werten überschreiben (siehe `sample.properties` im einfachen Beispiel).

Anwendungsname

Das KCL erfordert eine Anwendung, die für Ihre Anwendungen und für Amazon DynamoDB-Tabellen in derselben Region einzigartig ist. Sie verwendet den Wert der Anwendungsnamenkonfiguration auf folgende Arten:

- Für mit diesem Anwendungsnamen verknüpfte Auftragnehmer wird angenommen, dass sie gemeinsam im gleichen Stream arbeiten. Diese Auftragnehmer können auf mehrere Instances verteilt sein. Wenn Sie eine zusätzliche Instance desselben Anwendungscodes ausführen, jedoch mit einem anderen Anwendungsnamen, KCL behandelt das die zweite Instance als eine völlig separate Anwendung, die ebenfalls auf demselben Stream ausgeführt wird.
- Das KCL erstellt eine DynamoDB-Tabelle mit dem Anwendungsnamen und verwendet die Tabelle, um Statusinformationen (wie Checkpoints und Worker-Shard-Mapping) für die Anwendung zu verwalten. Jede Anwendung verfügt über eine eigene DynamoDB-Tabelle. Weitere Informationen finden Sie unter Verwenden Sie eine Leasetabelle, um nachzuverfolgen, welche Shards von der Consumer-Anwendung verarbeitet wurden KCL.

Richten Sie Anmeldeinformationen ein

Sie müssen Ihre AWS Anmeldeinformationen einem der Anmeldeinformationsanbieter in der Kette der Standardanmeldedienstanbieter zur Verfügung stellen. Sie können die Eigenschaft

`AWSCredentialsProvider` verwenden, um einen Anmeldeinformationsanbieter einzurichten. Die `sample.properties`-Datei muss Anmeldeinformationen einem der Anmeldeinformationsanbieter in der Anmeldeinformationsanbieter-Standardkette bereitstellen. Wenn Sie Ihren Consumer auf einer EC2 Amazon-Instance ausführen, empfehlen wir Ihnen, die Instance mit einer IAM Rolle zu konfigurieren. AWS Anmeldeinformationen, die die mit dieser IAM Rolle verknüpften Berechtigungen widerspiegeln, werden Anwendungen auf der Instance über ihre Instance-Metadaten zur Verfügung gestellt. Dies ist die sicherste Methode zur Verwaltung von Anmeldeinformationen für eine Verbraucheranwendung, die auf einer EC2 Instance ausgeführt wird.

Im folgenden Beispiel wird die Verarbeitung eines Kinesis-Datenstroms konfiguriertKCL, der `kclnodejssample` mithilfe des in bereitgestellten Datensatzprozessors benannt wird: `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
```

Entwickeln Sie einen Kinesis Client Library-Benutzer in. NET

Sie können die Kinesis Client Library (KCL) verwenden, um Anwendungen zu erstellen, die Daten aus Ihren Kinesis-Datenströmen verarbeiten. Die Kinesis Client Library ist in mehreren Sprachen verfügbar. In diesem Thema wird beschrieben. NET.

Die KCL ist eine Java-Bibliothek. Unterstützung für andere Sprachen als Java wird über eine mehrsprachige Schnittstelle bereitgestellt, die MultiLangDaemonals. Dieser Daemon basiert auf Java und wird im Hintergrund ausgeführt, wenn Sie eine andere KCL Sprache als Java verwenden. Daher, wenn Sie das für installieren. KCL NETund schreiben Sie Ihre Verbraucher-App vollständig hinein. NET, Sie müssen immer noch Java auf Ihrem System installiert haben, weil MultiLangDaemon. Darüber hinaus MultiLangDaemon verfügt es über einige Standardeinstellungen, die Sie möglicherweise an Ihren Anwendungsfall anpassen müssen, z. B. an die AWS Region, mit der eine Verbindung hergestellt wird. Weitere Informationen dazu finden Sie MultiLangDaemon auf GitHub der KCL MultiLangDaemon Projektseite.

Zum Herunterladen der. NETKCLvon GitHub, gehe zur Kinesis Client Library (. NET). Um den Beispielcode für einen herunterzuladen. NETKCLVerbraucheranwendung, gehen Sie zum KCLfür. NETBeispiel für eine Verbraucherprojektseite auf GitHub.

Sie müssen die folgenden Aufgaben ausführen, wenn Sie eine KCL Verbraucheranwendung in implementieren. NET:

Aufgaben

- Implementieren Sie die IRecordProcessor Klassenmethoden
- Ändern Sie die Konfigurationseigenschaften

Implementieren Sie die IRecordProcessor Klassenmethoden

Der Konsument muss die folgenden Methoden für `IRecordProcessor` implementieren. Der Konsument im Beispiel stellt Implementierungen bereit, die Sie als Ausgangspunkt verwenden können (siehe die `SampleRecordProcessor`-Klasse in `SampleConsumer/AmazonKinesisSampleConsumer.cs`).

```
public void Initialize(InitializationInput input)
public void ProcessRecords(ProcessRecordsInput input)
public void Shutdown(ShutdownInput input)
```

Initialisieren

Der KCL ruft diese Methode auf, wenn der Datensatzprozessor instanziiert wird, und übergibt eine bestimmte Shard-ID im `input` Parameter (). `input.ShardId` Dieser Datensatzverarbeiter verarbeitet nur diese Shard und in der Regel ist dies auch umgekehrt der Fall (diese Shard wird nur durch diesen Datensatverarbeiter verarbeitet). Ihr Konsument sollte jedoch die Möglichkeit berücksichtigen, dass ein Datensatz mehr als einmal verarbeitet werden könnte. Das liegt daran, dass Kinesis Data Streams eine Semantik nach dem Grundsatz mindestens einmal hat. Das bedeutet, dass jeder Datensatz aus einer Shard mindestens einmal von einem Worker in Ihrem Konsumenten verarbeitet wird. Weitere Informationen zu Fällen, in denen eine bestimmte Shard möglicherweise durch mehr als einen Auftragnehmer verarbeitet wird, finden Sie unter Verwenden Sie Resharding, Skalierung und Parallelverarbeitung, um die Anzahl der Shards zu ändern.

```
public void Initialize(InitializationInput input)
```

ProcessRecords

Der KCL ruft diese Methode auf und übergibt eine Liste von Datensätzen im `input` Parameter (`input.Records`) aus dem durch die Methode angegebenen Shard. `Initialize` Der von Ihnen implementierte Datensatzverarbeiter verarbeitet die Daten in diesen Datensätzen entsprechend der Semantik Ihres Konsumenten. Beispielsweise kann der Auftragnehmer eine Transformation für die Daten ausführen und das Ergebnis dann in einem Amazon Simple Storage Service (Amazon S3)-Bucket speichern.

```
public void ProcessRecords(ProcessRecordsInput input)
```

Zusätzlich zu den Daten selbst enthält der Datensatz auch eine Sequenznummer und einen Partitionsschlüssel. Der Auftragnehmer kann diese Werte beim Verarbeiten der Daten verwenden. Beispielsweise könnte der Auftragnehmer basierend auf dem Wert des Partitionsschlüssels den S3-Bucket wählen, in dem die Daten gespeichert werden sollen. Die Klasse `Record` stellt die folgenden Methoden bereit, die Zugriff auf die Daten des Datensatzes, die Sequenznummer und den Partitionsschlüssel bieten:

```
byte[] Record.Data
string Record.SequenceNumber
string Record.PartitionKey
```

Im Beispiel weist die Methode `ProcessRecordsWithRetries` Code auf, der zeigt, wie ein Auftragnehmer auf die Daten des Datensatzes, die Sequenznummer und den Partitionsschlüssel zugreifen kann.

Kinesis Data Streams erfordert, dass der Datensatzverarbeiter die Datensätze nachverfolgt, die bereits in einer Shard verarbeitet wurden. Der KCL kümmert sich für Sie um dieses Tracking, indem er ein `Checkpointer` Objekt an `ProcessRecords` (`input.Checkpointer`) übergibt. Der Datensatzprozessor ruft die `Checkpointer.Checkpoint` Methode auf, um ihn darüber zu informieren, wie weit er bei der Verarbeitung der Datensätze im Shard fortgeschritten ist. KCL Wenn der Worker ausfällt, KCL verwendet er diese Informationen, um die Verarbeitung des Shards beim letzten bekannten verarbeiteten Datensatz neu zu starten.

Bei einem Split- oder Merge-Vorgang beginnt der erst KCL mit der Verarbeitung der neuen Shards, wenn die Prozessoren für die ursprünglichen Shards aufgerufen haben, `Checkpointer.Checkpoint` um zu signalisieren, dass die gesamte Verarbeitung der ursprünglichen Shards abgeschlossen ist.

Wenn Sie keinen Parameter übergeben, KCL wird davon ausgegangen, dass der Aufruf von `Checkpointer.Checkpoint` bedeutet, dass alle Datensätze verarbeitet wurden, bis zu dem letzten

Datensatz, der an den Datensatzprozessor übergeben wurde. Daher sollte der Datensatzverarbeiter die Methode `Checkpointer.Checkpoint` erst aufrufen, wenn er alle Datensätze in der Liste, die ihm übergeben wurden, verarbeitet hat. Datensatzverarbeiter müssen `Checkpointer.Checkpoint` nicht bei jedem Aufruf von `ProcessRecords` aufrufen. Ein Prozessor könnte beispielsweise `Checkpointer.Checkpoint` bei jedem dritten oder vierten Aufruf aufrufen. Sie können optional die exakte Sequenznummer eines Datensatzes als Parameter für `Checkpointer.Checkpoint` angeben. In diesem Fall KCL geht der davon aus, dass Datensätze nur bis zu diesem Datensatz verarbeitet wurden.

Im Beispiel zeigt die private Methode `Checkpoint(Checkpointer checkpointer)`, wie die `Checkpointer.Checkpoint`-Methode mithilfe der entsprechenden Ausnahmebehandlung und Wiederholungslogik aufgerufen wird.

Das KCL für. NETbehandelt Ausnahmen anders als andere KCL Sprachbibliotheken, da sie keine Ausnahmen behandelt, die sich aus der Verarbeitung der Datensätze ergeben. Alle nicht abgefangenen Ausnahmen vom Benutzer-Code bringen das Programm zum Absturz.

Herunterfahren

Sie KCL ruft die `Shutdown` Methode entweder auf, wenn die Verarbeitung beendet ist (der Grund für das Herunterfahren ist`TERMINATE`) oder wenn der Worker nicht mehr reagiert (der `input.Reason` Shutdown-Wert ist`ZOMBIE`).

```
public void Shutdown(ShutdownInput input)
```

Die Verarbeitung endet, wenn der Datensatzverarbeiter keine weiteren Datensätze aus der Shard erhält, weil die Shard geteilt oder zusammengeführt wurde oder der Stream gelöscht wurde.

Der übergibt KCL auch ein `Checkpointer` Objekt an`shutdown`. Wenn der Grund für das Herunterfahren `TERMINATE` ist, sollte der Datensatzverarbeiter alle Datensätze fertigstellen und dann die Methode `checkpoint` in seiner Schnittstelle aufrufen.

Ändern Sie die Konfigurationseigenschaften

Der Beispielkonsument zeigt Standardwerte für die Konfigurationseigenschaften. Sie können diese Eigenschaften mit eigenen Werten überschreiben (siehe `SampleConsumer/kcl.properties`).

Anwendungsname

Das KCL erfordert eine Anwendung, die für Ihre Anwendungen und für Amazon DynamoDB-Tabellen in derselben Region einzigartig ist. Sie verwendet den Wert der Anwendungsnamenkonfiguration auf folgende Arten:

- Für mit diesem Anwendungsnamen verknüpfte Auftragnehmer wird angenommen, dass sie gemeinsam im gleichen Stream arbeiten. Diese Auftragnehmer können auf mehrere Instances verteilt sein. Wenn Sie eine zusätzliche Instance desselben Anwendungscodes ausführen, jedoch mit einem anderen Anwendungsnamen, KCL behandelt das die zweite Instance als eine völlig separate Anwendung, die ebenfalls auf demselben Stream ausgeführt wird.
- Das KCL erstellt eine DynamoDB-Tabelle mit dem Anwendungsnamen und verwendet die Tabelle, um Statusinformationen (wie Checkpoints und Worker-Shard-Mapping) für die Anwendung zu verwalten. Jede Anwendung verfügt über eine eigene DynamoDB-Tabelle. Weitere Informationen finden Sie unter Verwenden Sie eine Leasetabelle, um nachzuverfolgen, welche Shards von der Consumer-Anwendung verarbeitet wurden KCL.

Richten Sie Anmeldeinformationen ein

Sie müssen Ihre AWS Anmeldeinformationen einem der Anmeldeinformationsanbieter in der Kette der Standardanmeldedienstanbieter zur Verfügung stellen. Sie können die Eigenschaft `AWSCredentialsProvider` verwenden, um einen Anmeldeinformationsanbieter einzurichten. Die sample.properties muss Ihre Anmeldeinformationen einem der Anmeldeinformationsanbieter in der Anmeldeinformationsanbieter-Standardkette bereitstellen. Wenn Sie Ihre Consumer-Anwendung auf einer EC2 Instance ausführen, empfehlen wir Ihnen, die Instance mit einer IAM Rolle zu konfigurieren. AWS Anmeldeinformationen, die die mit dieser IAM Rolle verknüpften Berechtigungen widerspiegeln, werden den Anwendungen auf der Instance über deren Instance-Metadaten zur Verfügung gestellt. Dies ist die sicherste Methode, Anmeldeinformationen für einen Verbraucher zu verwalten, der auf einer EC2 Instance läuft.

Die Eigenschaftendatei des Beispiels ist so konfiguriertKCL, dass ein Kinesis-Datenstrom namens „Wörter“ mithilfe des mitgelieferten Aufzeichnungsprozessors verarbeitet wird. `AmazonKinesisSampleConsumer.cs`

Entwickeln Sie einen Kinesis Client Library-Consumer in Python

Sie können die Kinesis Client Library (KCL) verwenden, um Anwendungen zu erstellen, die Daten aus Ihren Kinesis-Datenströmen verarbeiten. Die Kinesis Client Library ist in mehreren Sprachen verfügbar. In diesem Thema wird Python behandelt.

Die KCL ist eine Java-Bibliothek. Unterstützung für andere Sprachen als Java wird über eine mehrsprachige Schnittstelle bereitgestellt, die als. MultiLangDaemon Dieser Daemon basiert auf Java und wird im Hintergrund ausgeführt, wenn Sie eine andere KCL Sprache als Java verwenden. Wenn Sie also das KCL für Python installieren und Ihre Consumer-App vollständig in Python schreiben, müssen Sie aufgrund der trotzdem Java auf Ihrem System installiert haben MultiLangDaemon. Darüber hinaus MultiLangDaemon verfügt es über einige Standardeinstellungen, die Sie möglicherweise an Ihren Anwendungsfall anpassen müssen, z. B. an die AWS Region, mit der eine Verbindung hergestellt wird. Weitere Informationen dazu finden Sie MultiLangDaemon auf GitHub der KCL MultiLangDaemon Projektseite.

Um Python KCL von herunterzuladen GitHub, gehen Sie zur Kinesis Client Library (Python). Um Beispielcode für eine KCL Python-Consumer-Anwendung herunterzuladen, gehen Sie auf die Seite KCLfür das Python-Beispielprojekt unter GitHub.

Sie müssen die folgenden Aufgaben ausführen, wenn Sie eine KCL Verbraucheranwendung in Python implementieren:

Aufgaben

- Implementieren Sie die RecordProcessor Klassenmethoden
- Ändern Sie die Konfigurationseigenschaften

Implementieren Sie die RecordProcessor Klassenmethoden

Die `RecordProcess`-Klasse muss die `RecordProcessorBase` erweitern, um die folgenden Methoden zu implementieren. Das Beispiel stellt Implementierungen bereit, die Sie als Ausgangspunkt verwenden können (siehe `sample_kclpy_app.py`).

```
def initialize(self, shard_id)
def process_records(self, records, checkpointer)
def shutdown(self, checkpointer, reason)
```

initialize

Der KCL ruft die `initialize` Methode auf, wenn der Recordprozessor instanziiert wird, und übergibt eine bestimmte Shard-ID als Parameter. Dieser Datensatzverarbeiter verarbeitet nur diese Shard und in der Regel ist dies auch umgekehrt der Fall (diese Shard wird nur durch diesen Datensatverarbeiter verarbeitet). Ihr Konsument sollte jedoch die Möglichkeit berücksichtigen, dass ein Datensatz mehr als einmal verarbeitet werden könnte. Das liegt daran, dass Kinesis Data

Streams eine Semantik nach dem Grundsatz mindestens einmal hat. Das bedeutet, dass jeder Datensatz aus einer Shard mindestens einmal von einem Worker in Ihrem Konsumenten verarbeitet wird. Weitere Informationen zu Fällen, in denen eine bestimmte Shard möglicherweise durch mehr als einen Auftragnehmer verarbeitet wird, finden Sie unter Verwenden Sie Resharding, Skalierung und Parallelverarbeitung, um die Anzahl der Shards zu ändern.

```
def initialize(self, shard_id)
```

process_records

Er KCL ruft diese Methode auf und übergibt eine Liste von Datensätzen aus dem durch die Methode angegebenen Shard. `initialize` Der von Ihnen implementierte Datensatzverarbeiter verarbeitet die Daten in diesen Datensätzen entsprechend der Semantik Ihres Konsumenten. Beispielsweise kann der Auftragnehmer eine Transformation für die Daten ausführen und das Ergebnis dann in einem Amazon Simple Storage Service (Amazon S3)-Bucket speichern.

```
def process_records(self, records, checkpointer)
```

Zusätzlich zu den Daten selbst enthält der Datensatz auch eine Sequenznummer und einen Partitionsschlüssel. Der Auftragnehmer kann diese Werte beim Verarbeiten der Daten verwenden. Beispielsweise könnte der Auftragnehmer basierend auf dem Wert des Partitionsschlüssels den S3-Bucket wählen, in dem die Daten gespeichert werden sollen. Das `record`-Anmeldeverzeichnis stellt die folgenden Schlüssel-Wert-Paare für den Zugriff auf die Daten des Datensatzes, die Sequenznummer und den Partitionsschlüssel bereit:

```
record.get('data')
record.get('sequenceNumber')
record.get('partitionKey')
```

Beachten Sie, dass die Daten Base64-kodiert sind.

Im Beispiel weist die Methode `process_records` Code auf, der zeigt, wie ein Auftragnehmer auf die Daten des Datensatzes, die Sequenznummer und den Partitionsschlüssel zugreifen kann.

Kinesis Data Streams erfordert, dass der Datensatzverarbeiter die Datensätze nachverfolgt, die bereits in einer Shard verarbeitet wurden. Der KCL kümmert sich für Sie um dieses Tracking, indem er ein `Checkpointer` Objekt an `process_records` übergibt. Der Datensatzprozessor ruft die `checkpoint` Methode für dieses Objekt auf, um ihn darüber zu informieren, wie weit er

bei der Verarbeitung der Datensätze im Shard fortgeschritten ist. KCL Wenn der Worker ausfällt, KCL verwendet er diese Informationen, um die Verarbeitung des Shards beim letzten bekannten verarbeiteten Datensatz neu zu starten.

Bei einem Split- oder Merge-Vorgang beginnt der erst KCL mit der Verarbeitung der neuen Shards, wenn die Prozessoren für die ursprünglichen Shards aufgerufen haben, `checkpoint` um zu signalisieren, dass die gesamte Verarbeitung der ursprünglichen Shards abgeschlossen ist.

Wenn Sie keinen Parameter übergeben, KCL wird davon ausgegangen, dass der Aufruf von `checkpoint` bedeutet, dass alle Datensätze verarbeitet wurden, bis zu dem letzten Datensatz, der an den Datensatzprozessor übergeben wurde. Daher sollte der Datensatzverarbeiter die Methode `checkpoint` erst aufrufen, wenn er alle Datensätze in der Liste, die ihm übergeben wurden, verarbeitet hat. Datensatzverarbeiter müssen `checkpoint` nicht bei jedem Aufruf von `process_records` aufrufen. Ein Prozessor könnte beispielsweise `checkpoint` bei jedem dritten Aufruf aufrufen. Sie können optional die exakte Sequenznummer eines Datensatzes als Parameter für `checkpoint` angeben. In diesem Fall KCL geht der davon aus, dass alle Datensätze nur bis zu diesem Datensatz verarbeitet wurden.

Im Beispiel zeigt die private Methode `checkpoint`, wie die `Checkpointer.checkpoint`-Methode mithilfe der entsprechenden Ausnahmebehandlung und Wiederholungslogik aufgerufen wird.

Das KCL ist darauf angewiesen, alle Ausnahmen `process_records` zu behandeln, die sich aus der Verarbeitung der Datensätze ergeben. Wenn eine Ausnahme ausgelöst wird`process_records`, werden die Datensätze KCL übersprungen, an die `process_records` vor der Ausnahme übergeben wurden. Das heißt, diese Datensätze werden nicht erneut an den Datensatzprozessor gesendet, der die Ausnahme ausgelöst hat, oder an einen anderen Datensatzprozessor im Verbraucher.

shutdown

Die `shutdown` Methode wird entweder KCL aufgerufen, wenn die Verarbeitung beendet ist (der Grund für das Herunterfahren ist`TERMINATE`) oder wenn der Worker nicht mehr reagiert (der Shutdown `reason` ist`ZOMBIE`).

```
def shutdown(self, checkpointer, reason)
```

Die Verarbeitung endet, wenn der Datensatzverarbeiter keine weiteren Datensätze aus der Shard erhält, entweder weil die Shard geteilt oder zusammengeführt wurde oder weil der Stream gelöscht wurde.

Der übergibt KCL auch ein `Checkpointer` Objekt an`shutdown`. Wenn der `reason` für das Herunterfahren `TERMINATE` ist, sollte der Datensatzverarbeiter alle Datensätze fertigstellen und dann die Methode `checkpoint` in seiner Schnittstelle aufrufen.

Ändern Sie die Konfigurationseigenschaften

Das Beispiel zeigt Standardwerte für die Konfigurationseigenschaften. Sie können diese Eigenschaften mit eigenen Werten überschreiben (siehe `sample.properties`).

Anwendungsname

Das KCL erfordert einen Anwendungsnamen, der für Ihre Anwendungen und für Amazon DynamoDB-Tabellen in derselben Region eindeutig ist. Sie verwendet den Wert der Anwendungsnamenkonfiguration auf folgende Arten:

- Für mit diesem Anwendungsnamen verknüpfte Auftragnehmer wird angenommen, dass sie gemeinsam im gleichen Stream arbeiten. Diese Auftragnehmer können auf mehrere Instances verteilt sein. Wenn Sie eine zusätzliche Instance desselben Anwendungscodes ausführen, jedoch mit einem anderen Anwendungsnamen, KCL behandelt das die zweite Instance als eine völlig separate Anwendung, die ebenfalls auf demselben Stream ausgeführt wird.
- Das KCL erstellt eine DynamoDB-Tabelle mit dem Anwendungsnamen und verwendet die Tabelle, um Statusinformationen (wie Checkpoints und Worker-Shard-Mapping) für die Anwendung zu verwalten. Jede Anwendung verfügt über eine eigene DynamoDB-Tabelle. Weitere Informationen finden Sie unter Verwenden Sie eine Leasetabelle, um nachzuverfolgen, welche Shards von der Consumer-Anwendung verarbeitet wurden KCL.

Richten Sie Anmeldeinformationen ein

Sie müssen Ihre AWS Anmeldeinformationen einem der Anmeldeinformationsanbieter in der Kette der Standardanmeldedienstanbieter zur Verfügung stellen. Sie können die Eigenschaft `AWSCredentialsProvider` verwenden, um einen Anmeldeinformationsanbieter einzurichten. Die sample.properties muss Ihre Anmeldeinformationen einem der Anmeldeinformationsanbieter in der Anmeldeinformationsanbieter-Standardkette bereitstellen. Wenn Sie Ihre Consumer-Anwendung auf einer EC2 Amazon-Instance ausführen, empfehlen wir Ihnen, die Instance mit einer IAM Rolle zu konfigurieren. AWS Anmeldeinformationen, die die mit dieser IAM Rolle verknüpften Berechtigungen widerspiegeln, werden Anwendungen auf der Instance über ihre Instance-Metadaten zur Verfügung gestellt. Dies ist die sicherste Methode zur Verwaltung von Anmeldeinformationen für eine Verbraucheranwendung, die auf einer EC2 Instance ausgeführt wird.

Die Eigenschaftendatei des Beispiels ist so konfiguriertKCL, dass ein Kinesis-Datenstrom namens „Wörter“ mithilfe des mitgelieferten Aufzeichnungsprozessors verarbeitet wird. `sample_kclpy_app.py`

Entwickeln Sie einen Kinesis Client Library-Consumer in Ruby

Sie können die Kinesis Client Library (KCL) verwenden, um Anwendungen zu erstellen, die Daten aus Ihren Kinesis-Datenströmen verarbeiten. Die Kinesis Client Library ist in mehreren Sprachen verfügbar. In diesem Thema wird Ruby behandelt.

Die KCL ist eine Java-Bibliothek. Unterstützung für andere Sprachen als Java wird über eine mehrsprachige Schnittstelle bereitgestellt, die als. MultiLangDaemon Dieser Daemon basiert auf Java und wird im Hintergrund ausgeführt, wenn Sie eine andere KCL Sprache als Java verwenden. Wenn Sie also den KCL für Ruby installieren und Ihre Consumer-App vollständig in Ruby schreiben, muss Java aufgrund des trotzdem auf Ihrem System installiert sein. MultiLangDaemon Darüber hinaus MultiLangDaemon verfügt es über einige Standardeinstellungen, die Sie möglicherweise an Ihren Anwendungsfall anpassen müssen, z. B. an die AWS Region, mit der eine Verbindung hergestellt wird. Weitere Informationen dazu finden Sie MultiLangDaemon auf GitHub der KCL MultiLangDaemon Projektseite.

Um Ruby KCL von herunterzuladen GitHub, gehen Sie zur Kinesis Client Library (Ruby). Den Beispielcode für eine KCL Ruby-Consumer-Anwendung können Sie auf der Seite KCLfür das Ruby-Beispielprojekt unter GitHub herunterladen.

Weitere Informationen zur KCL Ruby-Supportbibliothek finden Sie in der KCLRuby Gems-Dokumentation.

Develop KCL 2.x-Verbraucher

In diesem Thema erfahren Sie, wie Sie Version 2.0 der Kinesis Client Library (KCL) verwenden.

Weitere Informationen zu finden Sie in der KCL Übersicht unter Developing Consumer Using the Kinesis Client Library 1.x.

Wählen Sie je nach der Option, die Sie verwenden möchten, aus den folgenden Themen.

Themen

- Entwickeln Sie einen Kinesis Client Library-Consumer in Java
- Entwickeln Sie einen Kinesis Client Library-Consumer in Python

Entwickeln Sie einen Kinesis Client Library-Consumer in Java

Der folgende Code zeigt eine Beispielimplementierung in Java für `ProcessorFactory` und `RecordProcessor`. Weitere Informationen zur Nutzung der Vorteile der erweiterten Rundsendefunktion finden Sie unter Verwenden von Verbrauchern mit erweitertem Rundsenden .

```
/*
 *  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);
            }
        }
    }

}
```

Entwickeln Sie einen Kinesis Client Library-Consumer in Python

Sie können die Kinesis Client Library (KCL) verwenden, um Anwendungen zu erstellen, die Daten aus Ihren Kinesis-Datenströmen verarbeiten. Die Kinesis Client Library ist in mehreren Sprachen verfügbar. In diesem Thema wird Python behandelt.

Die KCL ist eine Java-Bibliothek. Unterstützung für andere Sprachen als Java wird über eine mehrsprachige Schnittstelle bereitgestellt, die als. MultiLangDaemon Dieser Daemon basiert auf Java und wird im Hintergrund ausgeführt, wenn Sie eine andere KCL Sprache als Java verwenden. Wenn Sie also das KCL für Python installieren und Ihre Consumer-App vollständig in Python schreiben, müssen Sie aufgrund der trotzdem Java auf Ihrem System installiert haben MultiLangDaemon. Darüber hinaus MultiLangDaemon verfügt es über einige Standardeinstellungen, die Sie möglicherweise an Ihren Anwendungsfall anpassen müssen, z. B. an die AWS Region, mit der eine Verbindung hergestellt wird. Weitere Informationen dazu finden Sie MultiLangDaemon auf GitHub der KCL MultiLangDaemon Projektseite.

Um Python KCL von herunterzuladen GitHub, gehen Sie zur Kinesis Client Library (Python). Um Beispielcode für eine KCL Python-Consumer-Anwendung herunterzuladen, gehen Sie auf die Seite KCLfür das Python-Beispielprojekt unter GitHub.

Sie müssen die folgenden Aufgaben ausführen, wenn Sie eine KCL Verbraucheranwendung in Python implementieren:

Aufgaben

- Implementieren Sie die RecordProcessor Klassenmethoden
- Ändern Sie die Konfigurationseigenschaften

Implementieren Sie die RecordProcessor Klassenmethoden

Die `RecordProcess`-Klasse muss die `RecordProcessorBase`-Klasse erweitern, um die folgenden Methoden zu implementieren:

```
initialize
process_records
shutdown_requested
```

Dieses Beispiel stellt Implementierungen bereit, die Sie als Ausgangspunkt verwenden können.

```
#!/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()
```

Ändern Sie die Konfigurationseigenschaften

Das Beispiel zeigt Standardwerte für die Konfigurationseigenschaften, wie in dem folgenden Skript gezeigt. Sie können diese Eigenschaften mit eigenen Werten überschreiben.

```
# 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
```

Anwendungsname

Das KCL erfordert einen Anwendungsnamen, der für Ihre Anwendungen und für Amazon DynamoDB-Tabellen in derselben Region eindeutig ist. Sie verwendet den Wert der Anwendungsnamenkonfiguration auf folgende Arten:

- Für mit diesem Anwendungsnamen verknüpfte Auftragnehmer wird angenommen, dass sie gemeinsam im gleichen Stream arbeiten. Diese Worker können auf mehrere Instances verteilt sein. Wenn Sie eine zusätzliche Instance desselben Anwendungscodes ausführen, jedoch mit einem

anderen Anwendungsnamen, KCL behandelt das die zweite Instance als eine völlig separate Anwendung, die ebenfalls auf demselben Stream ausgeführt wird.

- Das KCL erstellt eine DynamoDB-Tabelle mit dem Anwendungsnamen und verwendet die Tabelle, um Statusinformationen (wie Checkpoints und Worker-Shard-Mapping) für die Anwendung zu verwalten. Jede Anwendung verfügt über eine eigene DynamoDB-Tabelle. Weitere Informationen finden Sie unter Verwenden Sie eine Leasetabelle, um nachzuverfolgen, welche Shards von der Consumer-Anwendung verarbeitet wurden KCL.

Anmeldeinformationen

Sie müssen Ihre AWS Anmeldeinformationen einem der Anmeldeinformationsanbieter in der Kette der Standardanmeldedienstanbieter zur Verfügung stellen. Sie können die Eigenschaft `AWSCredentialsProvider` verwenden, um einen Anmeldeinformationsanbieter einzurichten. Wenn Sie Ihre Consumer-Anwendung auf einer EC2 Amazon-Instance ausführen, empfehlen wir Ihnen, die Instance mit einer IAM Rolle zu konfigurieren. AWS Anmeldeinformationen, die die mit dieser IAM Rolle verknüpften Berechtigungen widerspiegeln, werden Anwendungen auf der Instance über deren Instance-Metadaten zur Verfügung gestellt. Dies ist die sicherste Methode zur Verwaltung von Anmeldeinformationen für eine Verbraucheranwendung, die auf einer EC2 Instance ausgeführt wird.

Entwickeln Sie benutzerdefinierte Verbraucher mit gemeinsamem Durchsatz mithilfe der AWS SDK for Java

Eine der Methoden für die Entwicklung benutzerdefinierter Kinesis Data Streams-Verbraucher, die durchgängig gemeinsam genutzt werden, ist die Verwendung von Amazon Kinesis Data Streams. APIs In diesem Abschnitt wird die Verwendung der Kinesis Data Streams APIs mit dem AWS SDK für Java beschrieben. Der Java-Beispielcode in diesem Abschnitt zeigt, wie grundlegende KDS API Operationen ausgeführt werden. Er ist logisch nach Operationstypen unterteilt.

Diese Beispiele stellen keinen produktionsbereiten Code dar. Sie überprüfen nicht alle möglichen Ausnahmen und es werden nicht alle möglichen Sicherheits- oder Leistungsüberlegungen berücksichtigt.

Sie können die Kinesis Data Streams APIs mit anderen Programmiersprachen aufrufen. Weitere Informationen zu allen verfügbaren AWS SDKs Produkten finden Sie unter Start Developing with Amazon Web Services.

> **Important**
>
> Die empfohlene Methode für die Entwicklung benutzerdefinierter Kinesis Data Streams Streams-Benutzer, die durchgehend gemeinsam genutzt werden, ist die Verwendung der Kinesis Client Library ()KCL. KCLhilft Ihnen dabei, Daten aus einem Kinesis-Datenstrom zu nutzen und zu verarbeiten, indem es sich um viele der komplexen Aufgaben kümmert, die mit verteilter Datenverarbeitung verbunden sind. Weitere Informationen finden Sie unter Entwicklung benutzerdefinierter Verbraucher mit gemeinsamem Durchsatz unter Verwendung von KCL.

Themen

Daten aus einem Stream abrufen

Die Kinesis Data Streams APIs umfassen die `getRecords` Methoden `getShardIterator` und, die Sie aufrufen können, um Datensätze aus einem Datenstream abzurufen. Dies ist das PULL-Modell, bei dem der Code Datensätze direkt aus den Shards des Datenstroms abruft.

> **Important**
>
> Wir empfehlen, dass Sie die von angebotene Unterstützung für den Record Processor verwendenKCL, um Datensätze aus Ihren Datenströmen abzurufen. Dies ist das PUSH-Modell, bei dem Sie den Code implementieren, der die Daten verarbeitet. Der KCL ruft Datensätze aus dem Datenstrom ab und übermittelt sie an Ihren Anwendungscode. Darüber hinaus KCL bietet das Funktionen für Failover, Wiederherstellung und Lastenausgleich. Weitere Informationen finden Sie unter Entwicklung benutzerdefinierter Verbraucher mit gemeinsamem Durchsatz unter Verwendung von KCL.

In einigen Fällen ziehen Sie es jedoch möglicherweise vor, die Kinesis Data Streams APIs zu verwenden. Dies ist beispielsweise der Fall, wenn Sie benutzerdefinierte Tools für das Überwachen und Debuggen Ihrer Datenströme implementieren.

> **Important**
>
> Kinesis Data Streams unterstützt Änderungen des Zeitraums der Datensatzaufbewahrung für einen Datenstrom. Weitere Informationen finden Sie unter Ändern Sie den Aufbewahrungszeitraum für Daten.

Verwenden Sie Shard-Iteratoren

Sie rufen Datensätze aus dem Stream pro Shard ab. Für jeden Shard und jeden Datensatzstapel, den Sie aus dem Shard abrufen, benötigen Sie einen Shard-Iterator. Der Shard-Iterator wird im `getRecordsRequest`-Objekt verwendet, um den Shard anzugeben, aus dem die Datensätze abgerufen werden. Der Typ, der dem Shard-Iterator zugeordnet ist, gibt die Stelle im Shard an, von der die Datensätze abgerufen werden sollen (weitere Informationen dazu finden Sie später in diesem Abschnitt). Bevor Sie mit dem Shard-Iterator arbeiten können, müssen Sie den Shard abrufen. Weitere Informationen finden Sie unter Shards auflisten.

Rufen Sie diesen ersten Shard-Iterator mit der `getShardIterator`-Methode ab. Rufen Sie Shard-Iteratoren für weitere Datensatzstapel mit der `getNextShardIterator` -Methode des `getRecordsResult`-Objekts ab, das von der `getRecords`-Methode zurückgegeben wird. Ein Shard-Iterator verliert seine Gültigkeit nach 5 Minuten. Wenn Sie einen Shard-Iterator verwenden, solange er gültig ist, erhalten Sie einen neuen. Jeder Shard-Iterator ist 5 Minuten lang gültig, selbst wenn er bereits verwendet wurde.

Instanziieren Sie `GetShardIteratorRequest`, um den ersten Shard-Iterator abzurufen. Übergeben Sie ihn an die `getShardIterator`-Methode. Geben Sie zum Konfigurieren der Anforderung den Stream und die Shard-ID an. Informationen darüber, wie Sie die Streams in Ihrem AWS Konto abrufen können, finden Sie unter. Auflisten von Streams Informationen zum Abrufen der Shards in einem Stream finden Sie unter Shards auflisten.

```
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();
```

Der Beispiel-Code gibt beim Abrufen des ersten Shard-Iterators `TRIM_HORIZON` als Iterator-Typ an. Dieser Iterator-Typ bedeutet, dass Datensätze zurückgegeben werden sollen. Und zwar beginnend mit dem ersten zum Shard hinzugefügten Datensatz und nicht mit dem letzten hinzugefügten Datensatz, auch als Spitze bezeichnet. Folgende Iterator-Typen werden unterstützt:

- `AT_SEQUENCE_NUMBER`
- `AFTER_SEQUENCE_NUMBER`
- `AT_TIMESTAMP`
- `TRIM_HORIZON`
- `LATEST`

Weitere Informationen finden Sie unter ShardIteratorType.

Bei einigen Iterator-Typen müssen Sie zusätzlich eine Sequenznummer angeben, z. B.:

```
getShardIteratorRequest.setShardIteratorType("AT_SEQUENCE_NUMBER");
getShardIteratorRequest.setStartingSequenceNumber(specialSequenceNumber);
```

Nachdem Sie einen Datensatz mit `getRecords` abgerufen haben, erhalten Sie die Sequenznummer für den Datensatz, indem Sie die `getSequenceNumber`-Methode des Datensatzes aufrufen.

```
record.getSequenceNumber()
```

Darüber hinaus erhält der Code, durch den Datensätze zum Stream hinzufügt werden, die Sequenznummer für einen hinzugefügten Datensatz, indem `getSequenceNumber` auf dem Ergebnis von `putRecord` aufgerufen wird.

```
lastSequenceNumber = putRecordResult.getSequenceNumber();
```

Sie können mit diesen Sequenznummern eine strenge aufsteigende Anordnung der Datensätze gewährleisten. Weitere Informationen finden Sie im Code-Beispiel unter PutRecordBeispiel.

Benutzen GetRecords

Instanziieren Sie nach Abruf des Shard-Iterators ein `GetRecordsRequest`-Objekt. Geben Sie den Iterator für die Anforderung mit der `setShardIterator`-Methode an.

Optional können Sie auch die Anzahl der abzurufenden Datensätze mithilfe der `setLimit`-Methode angeben. Die Anzahl der Datensätze, die `getRecords` zurückgibt, ist stets gleich oder kleiner als dieses Limit. Wenn Sie kein Limit angeben, gibt `getRecords` 10 MB abgerufener Datensätze zurück. Beim unten stehenden Beispiel-Code wird das Limit auf 25 Datensätze festgelegt.

Wenn keine Datensätze zurückgegeben werden, bedeutet dies, dass derzeit keine Datensätze von diesem Shard für die vom Shard-Iterator angegebene Sequenznummer verfügbar sind. Wenn dies der Fall ist, sollte Ihre Anwendung so lange warten, wie dies für die Datenquellen des Streams angemessen ist. Versuchen Sie dann erneut mit dem Shard-Iterator, der vom vorherigen Aufruf von `getRecords` zurückgegeben wurde, Daten aus dem Shard abzurufen.

Übergeben Sie `getRecordsRequest` an die `getRecords`-Methode und erfassen Sie die zurückgegebenen Werte als `getRecordsResult`-Objekt. Rufen Sie die `getRecords`-Methode auf dem `getRecordsResult`-Objekt auf, um die Datensätze abzurufen.

```
GetRecordsRequest getRecordsRequest = new GetRecordsRequest();
getRecordsRequest.setShardIterator(shardIterator);
getRecordsRequest.setLimit(25);

GetRecordsResult getRecordsResult = client.getRecords(getRecordsRequest);
List<Record> records = getRecordsResult.getRecords();
```

Zur Vorbereitung auf einen weiteren Aufruf von `getRecords` rufen Sie den nächsten Shard-Iterator von `getRecordsResult` ab.

```
shardIterator = getRecordsResult.getNextShardIterator();
```

Die besten Ergebnisse erzielen Sie, wenn Sie mindestens 1 Sekunde (1.000 Millisekunden) zwischen den Aufrufen von `getRecords` warten, um ein Überschreiten des Limits für die Aufrufe von `getRecords` zu vermeiden.

```
try {
  Thread.sleep(1000);
}
catch (InterruptedException e) {}
```

In der Regel sollten Sie `getRecords` in einer Schleife aufrufen, auch wenn Sie einen einzelnen Datensatz in einem Testszenario abrufen. Ein einzelner Aufruf von `getRecords` gibt möglicherweise eine leere Datensatzliste zurück, auch wenn der Shard mehrere Datensätze mit höheren Sequenznummern enthält. In diesem Fall verweist der zurückgegebene `NextShardIterator` zusammen mit der leeren Datensatzliste auf eine höhere Sequenznummer im Shard. Nachfolgende Aufrufe von `getRecords` führen dann zu einem erfolgreichen Abruf. Das folgende Beispiel zeigt die Verwendung einer Schleife.

Beispiel: getRecords

Das folgende Codebeispiel spiegelt die `getRecords`-Tipps in diesem Abschnitt wieder, einschließlich der Aufrufe in Schleifen.

```
// 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();
}
```

Bei der Nutzung der Kinesis Client Library müssen möglicherweise mehrere Aufrufe durchgeführt werden, ehe Daten zurückgegeben werden. Dieses Verhalten ist beabsichtigt und weist nicht auf ein Problem mit den KCL oder Ihren Daten hin.

Passen Sie sich an einen Reshard an

Wenn `getRecordsResult.getNextShardIterator` `null` zurückgibt, bedeutet dies, dass eine Aufteilung oder Zusammenführung des Shards stattgefunden hat, die diesen Shard betrifft. Dieser Shard befindet sich jetzt in einem `CLOSED`-Status und Sie haben alle verfügbaren Datensätze von diesem Shard gelesen.

In diesem Szenario können Sie `getRecordsResult.childShards` verwenden, um etwas über die neuen untergeordneten Shards des zu verarbeitenden Shards zu erfahren, die durch die Aufteilung oder Zusammenführung entstanden sind. Weitere Informationen finden Sie unter. ChildShard

Bei einer Teilung ist die `parentShardId` der beiden neuen Shards gleich der Shard-ID des zuvor verarbeiteten Shards. Der Wert von `adjacentParentShardId` für beide Shards ist `null`.

Bei einer Zusammenführung ist bei dem entstandenen einzelnen Datensatz die `parentShardId` identisch mit der Shard-ID eines übergeordneten Shards und die `adjacentParentShardId` ist gleich der Shard-ID des anderen übergeordneten Shards. Ihre Anwendung hat bereits alle Daten aus einem dieser Shards ausgelesen. Dies ist der Shard, für den `getRecordsResult.getNextShardIterator` `null` zurückgegeben hat. Wenn die Reihenfolge der Daten für Ihre Anwendung von Bedeutung ist, stellen Sie sicher, dass die Anwendung auch alle Daten des anderen übergeordneten Shards ausliest, ehe neue Daten aus dem durch die Zusammenführung entstandenen untergeordneten Shards ausgelesen werden.

Wenn Sie mehrere Prozessoren zum Abrufen von Daten aus dem Stream verwenden (beispielsweise einen Prozessor pro Shard), und es kommt zu einer Teilung oder Zusammenführung von Shards, sollten Sie die Anzahl der Prozessoren entsprechend anpassen.

Weitere Informationen zum Resharding, einschließlich einer Diskussion über Shard-Status, beispielsweise `CLOSED` finden Sie unter Einen Stream erneut teilen.

Interagieren Sie mit Daten mithilfe der AWS Glue Schema Registry

Sie können Ihre Kinesis-Datenströme in die AWS Glue Schema Registry integrieren. Mit der AWS Glue Schema Registry können Sie Schemas zentral erkennen, steuern und weiterentwickeln und gleichzeitig sicherstellen, dass die erstellten Daten kontinuierlich anhand eines registrierten Schemas validiert werden. Ein Schema definiert die Struktur und das Format eines Datensatzes. Ein Schema ist eine versionierte Spezifikation für zuverlässige Datenveröffentlichung, -nutzung oder -speicherung. Mit der AWS Glue Schema Registry können Sie die end-to-end Datenqualität und Datenverwaltung in Ihren Streaming-Anwendungen verbessern. Weitere Informationen finden Sie unter AWS Glue Schema Registry. Eine Möglichkeit, diese Integration einzurichten, sind die in AWS Java API SDK verfügbaren `GetRecords` Kinesis Data Streams.

Detaillierte Anweisungen zur Einrichtung der Integration von Kinesis Data Streams mit Schema Registry mithilfe der `GetRecords` Kinesis Data Streams APIs finden Sie im Abschnitt „Interaktion mit Daten mithilfe der Kinesis Data StreamsAPIs“ unter Anwendungsfall: Integration von Amazon Kinesis Data Streams mit der AWS Glue Schema Registry.

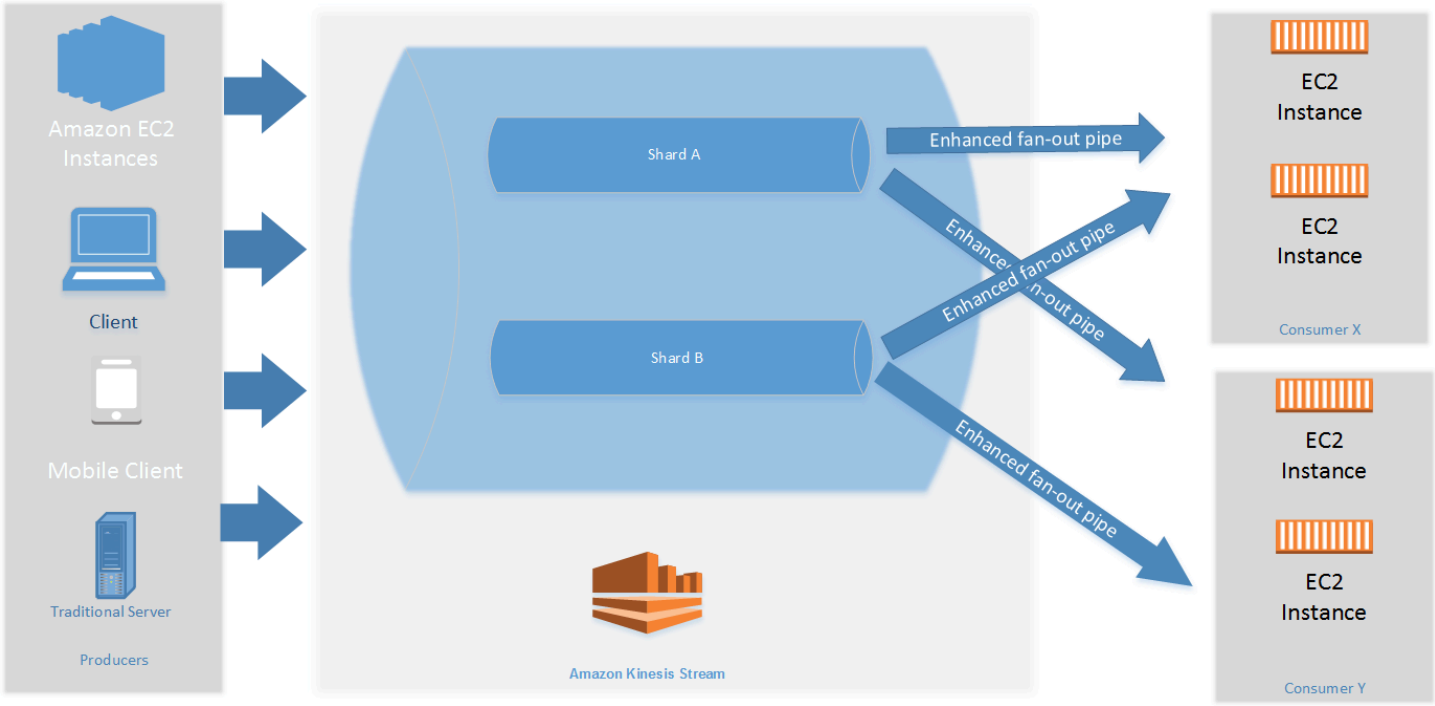
Entwickeln Sie maßgeschneiderte Verbraucher mit dediziertem Durchsatz (verbesserter Fan-Out)

In Amazon Kinesis Data Streams können Sie Verbraucher erstellen, die ein Feature namens erweiterte Rundsendungen verwenden. Mit dieser Funktion können Verbraucher Datensätze aus einem Stream empfangen, mit einem Durchsatz von bis zu 2 MB Daten pro Sekunde pro Shard. Dieser Durchsatz ist dediziert, d. h. dass Verbraucher, die ein erweitertes Rundsenden verwenden, nicht mit anderen Verbrauchern konkurrieren müssen, die Daten aus dem Stream empfangen. Kinesis Data Streams überträgt Datensätze aus dem Stream zu den Verbrauchern die ein erweitertes Rundsenden verwenden. Aus diesem Grund müssen diese Verbraucher keine Daten abfragen.

> **Important**
>
> Sie können bis zu zwanzig Nutzer pro Stream registrieren, um das erweiterte Rundsenden zu verwenden.

Das folgende Diagramm zeigt die Architektur für das erweiterte Rundsenden. Wenn Sie Version 2.0 oder höher der Amazon Kinesis Client Library (KCL) verwenden, um einen Consumer zu erstellen, KCL richtet der Consumer den Consumer so ein, dass er das erweiterte Fan-Out verwendet, um

Daten von allen Shards des Streams zu empfangen. Wenn Sie den verwendenAPI, um einen Consumer zu erstellen, der erweitertes Fan-Out verwendet, können Sie einzelne Shards abonnieren.



Das Diagramm zeigt Folgendes:

- Einen Stream mit zwei Shards.
- Zwei Verbraucher, die erweitertes Rundsenden verwenden, um Daten vom Stream zu empfangen: Consumer X und Consumer Y. Beide Verbraucher haben alle Shards und alle Datensätze im Stream abonniert. Wenn Sie Version 2.0 oder höher von verwenden, um einen Consumer KCL zu erstellen, abonniert der diesen Consumer KCL automatisch für alle Shards des Streams. Wenn Sie den dagegen verwenden, um einen Consumer API zu erstellen, können Sie einzelne Shards abonnieren.
- Pfeile stellen die Pipes für das erweiterte Rundsenden dar, die die Verbraucher verwenden, um Daten aus dem Stream zu erhalten. Eine Pipe für erweitertes Rundsenden liefert bis zu 2 MB/s Daten pro Shard, unabhängig von anderen Pipes oder der Gesamtzahl der Verbraucher.

Themen

- Entwickeln Sie mit 2.x erweiterte Fan-Out-Nutzer KCL
- Entwickeln Sie mit den Kinesis Data Streams erweiterte Fan-Out-Nutzer API
- Verwalten Sie erweiterte Fan-Out-Nutzer mit dem AWS Management Console

Entwickeln Sie mit 2.x erweiterte Fan-Out-Nutzer KCL

Verbraucher, die ein erweitertes Rundsenden in Amazon Kinesis Data Streams verwenden, können Datensätze aus einem Datenstrom mit einem dedizierten Durchsatz von bis zu 2 MB Daten pro Sekunde pro Shard empfangen. Diese Art Verbraucher muss nicht mit anderen Verbrauchern konkurrieren, die Daten aus dem Stream empfangen. Weitere Informationen finden Sie unter Entwickeln Sie maßgeschneiderte Verbraucher mit dediziertem Durchsatz (verbesserter Fan-Out).

Sie können Version 2.0 oder höher der Kinesis Client Library (KCL) verwenden, um Anwendungen zu entwickeln, die erweitertes Fan-Out verwenden, um Daten aus Streams zu empfangen. Die abonniert Ihre Anwendung KCL automatisch für alle Shards eines Streams und stellt sicher, dass Ihre Privatanwenderanwendung mit einem Durchsatz von 2 MB/s pro Shard lesen kann. Wenn Sie den verwenden möchten, KCL ohne den erweiterten Fan-Out zu aktivieren, finden Sie weitere Informationen unter Developing Consumer Using the Kinesis Client Library 2.0.

Themen

- Entwickeln Sie erweiterte Fan-Out-Consumer mit 2.x in Java KCL

Entwickeln Sie erweiterte Fan-Out-Consumer mit 2.x in Java KCL

Sie können Version 2.0 oder höher der Kinesis Client Library (KCL) verwenden, um Anwendungen in Amazon Kinesis Data Streams zu entwickeln, um Daten aus Streams mit erweitertem Fan-Out zu empfangen. Der folgende Code zeigt eine Beispielimplementierung in Java für `ProcessorFactory` und `RecordProcessor`.

Es wird empfohlen, dass Sie `KinesisClientUtil` zum Erstellen von `KinesisAsyncClient` und zum Konfigurieren von `maxConcurrency` in `KinesisAsyncClient` verwenden.

> **Important**
>
> Für den Amazon Kinesis Client kann sich die Latenz möglicherweise signifikant erhöhen, sofern Sie `KinesisAsyncClient` nicht für einen `maxConcurrency`-Wert konfigurieren, der hoch genug ist, um alle Leases plus zusätzliche Verwendungen von `KinesisAsyncClient` zu ermöglichen.

```
/*
 *  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);
            }
        }
    }

}
```

Entwickeln Sie mit den Kinesis Data Streams erweiterte Fan-Out-Nutzer API

Erweitertes Rundsenden ist ein Feature in Amazon Kinesis Data Streams, die es Verbrauchern ermöglicht, Datensätze aus einem Datenstrom mit einem dedizierten Durchsatz von bis zu 2 MB Daten pro Sekunde pro Shard zu empfangen. Ein Verbraucher, der ein erweitertes Rundsenden verwendet, muss nicht mit anderen Verbrauchern konkurrieren, die Daten aus dem Stream empfangen. Weitere Informationen finden Sie unter Entwickeln Sie maßgeschneiderte Verbraucher mit dediziertem Durchsatz (verbesserter Fan-Out).

Sie können API Operationen verwenden, um einen Consumer zu erstellen, der das erweiterte Fan-Out in Kinesis Data Streams verwendet.

Um einen Verbraucher mit erweitertem Fan-Out mithilfe der Kinesis Data Streams zu registrieren API

1. Rufen Sie an RegisterStreamConsumer, um Ihre Anwendung als Verbraucher zu registrieren, der den erweiterten Fan-Out verwendet. Kinesis Data Streams generiert einen Amazon-Ressourcennamen (ARN) für den Verbraucher und gibt ihn in der Antwort zurück.
2. Um mit dem Abhören eines bestimmten Shards zu beginnen, leiten Sie den Verbraucher ARN mit einem Anruf an weiter. SubscribeToShard Kinesis Data Streams beginnt dann, die Datensätze von diesem Shard in Form von Ereignissen des Typs SubscribeToShardEventüber eine HTTP /2-Verbindung an Sie weiterzuleiten. Die Verbindung bleibt für bis zu 5 Minuten offen. Rufen Sie SubscribeToSharderneut an, wenn Sie weiterhin Datensätze von dem Shard empfangen möchten`future`, nachdem der vom Anruf zurückgegebene Shard normal oder ausnahmsweise SubscribeToShardabgeschlossen wurde.

   > **Note**
   >
   > `SubscribeToShard`APIgibt außerdem die Liste der untergeordneten Shards des aktuellen Shards zurück, wenn das Ende des aktuellen Shards erreicht ist.

3. Rufen Sie an, um einen Verbraucher abzumelden, der den erweiterten Fanout verwendet. DeregisterStreamConsumer

Der folgende Code ist ein Beispiel dafür, wie Sie für Ihren Verbraucher ein Abonnement für einen Shard einrichten, das Abonnement regelmäßig erneuern und die Ereignisse verarbeiten können.

```
    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);
        }
    }
```

Wenn `event.ContinuationSequenceNumber` `null` zurückgibt, bedeutet dies, dass eine Aufteilung oder Zusammenführung des Shards stattgefunden hat, die diesen Shard betrifft. Dieser Shard befindet sich jetzt in einem `CLOSED`-Status und Sie haben alle verfügbaren Datensätze von diesem Shard gelesen. In diesem Szenario können Sie, wie im obigen Beispiel, `event.childShards` verwenden, um etwas über die neuen untergeordneten Shards des zu verarbeitenden Shards zu erfahren, die durch die Aufteilung oder Zusammenführung entstanden sind. Weitere Informationen finden Sie unter. ChildShard

Interagieren Sie mit Daten mithilfe der AWS Glue Schema Registry

Sie können Ihre Kinesis-Datenströme in die AWS Glue Schema Registry integrieren. Mit der AWS Glue Schema Registry können Sie Schemas zentral erkennen, steuern und weiterentwickeln und gleichzeitig sicherstellen, dass die erstellten Daten kontinuierlich anhand eines registrierten Schemas validiert werden. Ein Schema definiert die Struktur und das Format eines Datensatzes. Ein Schema ist eine versionierte Spezifikation für zuverlässige Datenveröffentlichung, -nutzung oder -speicherung. Mit der AWS Glue Schema Registry können Sie die end-to-end Datenqualität und Datenverwaltung in Ihren Streaming-Anwendungen verbessern. Weitere Informationen finden Sie unter AWS Glue Schema Registry. Eine Möglichkeit, diese Integration einzurichten, sind die in AWS Java API SDK verfügbaren `GetRecords` Kinesis Data Streams.

Detaillierte Anweisungen zur Einrichtung der Integration von Kinesis Data Streams mit Schema Registry mithilfe der `GetRecords` Kinesis Data Streams APIs finden Sie im Abschnitt „Interaktion mit Daten mithilfe der Kinesis Data StreamsAPIs“ unter Anwendungsfall: Integration von Amazon Kinesis Data Streams mit der AWS Glue Schema Registry.

Verwalten Sie erweiterte Fan-Out-Nutzer mit dem AWS Management Console

Verbraucher, die ein erweitertes Rundsenden in Amazon Kinesis Data Streams verwenden, können Datensätze aus einem Datenstrom mit einem dedizierten Durchsatz von bis zu 2 MB

Daten pro Sekunde pro Shard empfangen. Weitere Informationen finden Sie unter Entwickeln Sie maßgeschneiderte Verbraucher mit dediziertem Durchsatz (verbesserter Fan-Out).

Sie können den verwenden AWS Management Console , um eine Liste aller Verbraucher einzusehen, die für die Nutzung des erweiterten Fan-Outs mit einem bestimmten Stream registriert sind. Für jeden dieser Verbraucher können Sie Details wie StatusARN, Erstellungsdatum und Monitoring-Metriken einsehen.

Anzeige der Verbraucher, die registriert sind, um das erweiterte Rundsenden zu verwenden, ihres Status, ihres Erstellungsdatums ihrer Metriken in der Konsole

1. Melden Sie sich bei der an AWS Management Console und öffnen Sie die Kinesis-Konsole unter https://console.aws.amazon.com/kinesis.
2. Klicken Sie im Navigationsbereich auf Data Streams (Daten-Streams).
3. Wählen Sie einen Kinesis-Datenstrom aus, um dessen Details anzuzeigen.
4. Wählen Sie auf der Detailseite für den Stream die Registerkarte Enhanced fan-out (Erweitertes Rundsenden).
5. Wählen Sie einen Verbraucher, um seinen Namen, den Status und das Datum der Registrierung anzuzeigen.

Einen Verbraucher abmelden

1. Öffnen Sie die Kinesis-Konsole unter.https://console.aws.amazon.com/kinesis.
2. Klicken Sie im Navigationsbereich auf Data Streams (Daten-Streams).
3. Wählen Sie einen Kinesis-Datenstrom aus, um dessen Details anzuzeigen.
4. Wählen Sie auf der Detailseite für den Stream die Registerkarte Enhanced fan-out (Erweitertes Rundsenden).
5. Wählen Sie das Kontrollkästchen links neben dem Namen jedes Verbrauchers, den Sie abmelden möchten.
6. Wählen Sie Deregister consumer (Verbraucher abmelden).

Migrieren Sie Verbraucher von KCL 1.x auf KCL 2.x

In diesem Thema werden die Unterschiede zwischen den Versionen 1.x und 2.x der Kinesis Client Library () erklärt. KCL Außerdem erfahren Sie, wie Sie Ihren Kunden von Version 1.x auf Version

2.x von migrieren. KCL Nach der Migration des Clients werden Datensätze vom letzten Checkpoint-Speicherort verarbeitet.

Version 2.0 von KCL führt die folgenden Änderungen an der Benutzeroberfläche ein:

KCLÄnderungen an der Benutzeroberfläche

KCL1.x-Schnittstelle	KCL2.0-Schnittstelle
`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`	Umgewandelt in `software.amazon.kinesis.processor.ShardRecordProcessor`

Themen

- Migrieren Sie den Aufzeichnungsprozessor
- Migrieren Sie den Record Processor Factory
- Migrieren Sie den Worker
- Den Amazon Kinesis-Client konfigurieren
- Entfernung von Leerlaufzeiten
- Entfernung der Client-Konfiguration

Migrieren Sie den Aufzeichnungsprozessor

Das folgende Beispiel zeigt einen für KCL 1.x implementierten Datensatzprozessor:

```
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();
        }
    }
}
```

So migrieren Sie die Datensatzprozessorklasse

1. Ändern Sie die Schnittstellen von `com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor` und `com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IShutdownNotificat` folgendermaßen nach `software.amazon.kinesis.processor.ShardRecordProcessor`:

```
// 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. Aktualisieren Sie die `import`-Anweisungen für die Methoden `initialize` und `processRecords`.

```
// import com.amazonaws.services.kinesis.clientlibrary.types.InitializationInput;
import software.amazon.kinesis.lifecycle.events.InitializationInput;

//import com.amazonaws.services.kinesis.clientlibrary.types.ProcessRecordsInput;
import software.amazon.kinesis.lifecycle.events.ProcessRecordsInput;
```

3. Ersetzen Sie die Methode `shutdown` durch die folgenden neuen Methoden: `leaseLost`, `shardEnded` und `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();
        }
    }
```

Nachstehend finden Sie die aktualisierte Version der Datensatzprozessorklasse.

```
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();
        }
    }
}
```

Migrieren Sie den Record Processor Factory

Die Datensatzprozessor-Factory ist für das Erstellen von Prozessoren verantwortlich, wenn eine Lease erworben wird. Im Folgenden finden Sie ein Beispiel für eine KCL 1.x-Factory.

```
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();
    }
}
```

So migrieren Sie die Datensatzprozessor-Factory

1. Ändern Sie die implementierte Schnittstelle von `com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessorFa` folgendermaßen nach `software.amazon.kinesis.processor.ShardRecordProcessorFactory`:

```
// 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. Ändern Sie die Rückgabesignatur für `createProcessor`.

```
// public IRecordProcessor createProcessor() {
public ShardRecordProcessor shardRecordProcessor() {
```

Es folgt ein Beispiel für die Verwendung der Datensatzprozessor-Factory in 2.0:

```
package com.amazonaws.kcl;

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

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

Migrieren Sie den Worker

In Version 2.0 von ersetzt eine neue Klasse namens `Scheduler` die `Worker` Klasse. KCL Im Folgenden finden Sie ein Beispiel für einen KCL 1.x-Worker.

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

So migrieren Sie den Auftragnehmer

1. Ändern Sie die `import`-Anweisung für die `Worker`-Klasse, um Anweisungen für die Klassen `Scheduler` und `ConfigsBuilder` zu importieren.

   ```
   // import com.amazonaws.services.kinesis.clientlibrary.lib.worker.Worker;
   import software.amazon.kinesis.coordinator.Scheduler;
   import software.amazon.kinesis.common.ConfigsBuilder;
   ```

2. Erstellen Sie `ConfigsBuilder` und `Scheduler` wie im folgenden Beispiel gezeigt.

   Es wird empfohlen, dass Sie `KinesisClientUtil` zum Erstellen von `KinesisAsyncClient` und zum Konfigurieren von `maxConcurrency` in `KinesisAsyncClient` verwenden.

   > **Important**
   >
   > Für den Amazon Kinesis Client kann sich die Latenz möglicherweise signifikant erhöhen, sofern Sie `KinesisAsyncClient` nicht für einen `maxConcurrency`-Wert konfigurieren, der hoch genug ist, um alle Leases plus zusätzliche Verwendungen von `KinesisAsyncClient` zu ermöglichen.

   ```
   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()
    );
```

Den Amazon Kinesis-Client konfigurieren

Mit Version 2.0 der Kinesis Client Library wurde die Konfiguration des Clients aus einer einzelnen Konfigurationsklasse (`KinesisClientLibConfiguration`) in sechs Konfigurationsklassen verlagert. Die folgende Tabelle beschreibt die Migration.

Konfigurationsfelder und ihre neue Klassen

Originalfeld	Neue Konfigurationsklasse	Beschreibung
`applicationName`	`ConfigsBuilder`	Der Name für diese KCL Anwendung. Wird als Standard für `tableName` und `consumerName` verwendet.
`tableName`	`ConfigsBuilder`	Ermöglicht das Überschreiben des für die Lease-Tabelle von Amazon DynamoDB verwendeten Tabellennamens.
`streamName`	`ConfigsBuilder`	Der Name des Streams, dessen Datensätze diese Anwendung verarbeitet.
`kinesisEndpoint`	`ConfigsBuilder`	Diese Option wurde entfernt. Siehe "Entfernen von Client-Konfigurationen".
`dynamoDBEndpoint`	`ConfigsBuilder`	Diese Option wurde entfernt. Siehe "Entfernen von Client-Konfigurationen".
`initialPositionInStreamExtended`	`RetrievalConfig`	Der Ort auf dem Shard, von dem aus KCL Datensätze abgerufen werden, beginnend mit der ersten Ausführung der Anwendung.
`kinesisCredentialsProvider`	`ConfigsBuilder`	Diese Option wurde entfernt. Siehe "Entfernen von Client-Konfigurationen".
`dynamoDBCredentialsProvider`	`ConfigsBuilder`	Diese Option wurde entfernt. Siehe "Entfernen von Client-Konfigurationen".
`cloudWatchCredentialsProvider`	`ConfigsBuilder`	Diese Option wurde entfernt. Siehe "Entfernen von Client-Konfigurationen".

Originalfeld	Neue Konfigurationsklasse	Beschreibung
`failoverTimeMillis`	LeaseManagementConfig	Anzahl Millisekunden, nach deren Ablauf unterstellt werden kann, dass ein Lease-Eigentümer fehlgeschlagen ist.
`workerIdentifier`	`ConfigsBuilder`	Eine eindeutige Kennung, die die Instanziierung des Anwendungsprozessors repräsentiert. Dieser Wert muss eindeutig sein.
`shardSyncIntervalMillis`	LeaseManagementConfig	Die Zeit zwischen Shard-Synchronisierungsaufrufen.
`maxRecords`	`PollingConfig`	Ermöglicht das Einstellen der maximalen Anzahl an Datensätzen, die Kinesis zurückgibt.
`idleTimeBetweenReadsInMillis`	`CoordinatorConfig`	Diese Option wurde entfernt. Siehe "Entfernen der Leerlaufzeit".
`callProcessRecordsEvenForEmptyRecordList`	`ProcessorConfig`	Wenn diese Option aktiviert ist, wird der Datensatzprozessor aufgerufen, auch wenn Kinesis keine Datensätze bereitgestellt hat.
`parentShardPollIntervalMillis`	`CoordinatorConfig`	Gibt an, wie oft ein Datensatzprozessor abfragen soll, ob der übergeordnete Shard abgeschlossen wurde.
`cleanupLeasesUponShardCompletion`	`LeaseManagementConfig`	Wenn diese Option aktiviert ist, werden Leases entfernt, sobald die untergeordneten Leases die Verarbeitung gestartet haben.

Originalfeld	Neue Konfigurationsklasse	Beschreibung
`ignoreUnexpectedChildShards`	`LeaseManagementConfig`	Wenn diese Option aktiviert ist, werden untergeordnete Shards, die einen offenen Shard aufweisen, ignoriert. Dies gilt hauptsächlich für DynamoDB Streams.
`kinesisClientConfig`	`ConfigsBuilder`	Diese Option wurde entfernt. Siehe "Entfernen von Client-Konfigurationen".
`dynamoDBClientConfig`	`ConfigsBuilder`	Diese Option wurde entfernt. Siehe "Entfernen von Client-Konfigurationen".
`cloudWatchClientConfig`	`ConfigsBuilder`	Diese Option wurde entfernt. Siehe "Entfernen von Client-Konfigurationen".
`taskBackoffTimeMillis`	`LifecycleConfig`	Die Wartezeit beim erneuten Versuchen von fehlgeschlagenen Aufgaben.
`metricsBufferTimeMillis`	`MetricsConfig`	Steuert die Veröffentlichung von CloudWatch Metriken.
`metricsMaxQueueSize`	`MetricsConfig`	Steuert die Veröffentlichung von CloudWatch Metriken.
`metricsLevel`	`MetricsConfig`	Steuert die Veröffentlichung von CloudWatch Metriken.
`metricsEnabledDimensions`	`MetricsConfig`	Steuert die Veröffentlichung von CloudWatch Metriken.
`validateSequenceNumberBeforeCheckpointing`	`CheckpointConfig`	Diese Option wurde entfernt. Siehe "Checkpoint Sequence Number Validation".

Originalfeld	Neue Konfigurationsklasse	Beschreibung
`regionName`	`ConfigsBuilder`	Diese Option wurde entfernt. Siehe "Entfernen von Client-Konfigurationen".
`maxLeasesForWorker`	`LeaseManagementConfig`	Die maximale Anzahl der Leases, die eine einzelne Instance der Anwendung akzeptieren sollte.
`maxLeasesToStealAtOneTime`	`LeaseManagementConfig`	Die maximale Anzahl der Leases, die eine Anwendung zu einem gegebenen Zeitpunkt zu stehlen versuchen sollte.
`initialLeaseTableReadCapacity`	`LeaseManagementConfig`	Der DynamoDB-LesevorgangIOPs, der verwendet wird, wenn die Kinesis-Clientbibliothek eine neue DynamoDB-Leasetabelle erstellen muss.
`initialLeaseTableWriteCapacity`	`LeaseManagementConfig`	Der DynamoDB-LesevorgangIOPs, der verwendet wird, wenn die Kinesis-Clientbibliothek eine neue DynamoDB-Leasetabelle erstellen muss.
`initialPositionInStreamExtended`	LeaseManagementConfig	Die ursprüngliche Position im Stream, an der die Anwendung starten soll. Dieser Wert wird nur im Rahmen der Lease-Erstellung verwendet.
`skipShardSyncAtWorkerInitializationIfLeasesExist`	`CoordinatorConfig`	Synchronisieren der Shard-Daten deaktivieren, wenn die Lease-Tabelle Leases enthält. TODO KinesisEco: -438
`shardPrioritization`	`CoordinatorConfig`	Welche Shard-Priorisierung verwendet werden soll.
`shutdownGraceMillis`	N/A	Diese Option wurde entfernt. Siehe Umzüge MultiLang .

Originalfeld	Neue Konfigurationsklasse	Beschreibung
`timeoutInSeconds`	N/A	Diese Option wurde entfernt. Siehe MultiLang Umzüge.
`retryGetRecordsInSeconds`	`PollingConfig`	Konfiguriert die Verzögerung zwischen GetRecords Fehlversuchen.
`maxGetRecordsThreadPool`	`PollingConfig`	Die Thread-Pool-Größe, die für GetRecords verwendet wird.
`maxLeaseRenewalThreads`	`LeaseManagementConfig`	Steuert die Größe des Lease-Renewer-Thread-Pools. Dieser Pool muss größer sein, wenn die Anwendung mehr Leases annehmen kann.
`recordsFetcherFactory`	`PollingConfig`	Ermöglicht es, die Voreinstellung zu ersetzen, die zum Erstellen von Fetchers verwendet wird, die von Streams abrufen.
`logWarningForTaskAfterMillis`	`LifecycleConfig`	Wartezeit, bevor eine Warnung protokolliert wird, wenn eine Aufgabe nicht abgeschlossen wurde.
`listShardsBackoffTimeInMillis`	`RetrievalConfig`	Die Anzahl der zwischen Aufrufen von `ListShards` abzuwartenden Millisekunden, wenn es zu Fehlern kommt.
`maxListShardsRetryAttempts`	`RetrievalConfig`	Die maximale Anzahl Wiederholungsversuche durch `ListShards` , bevor abgebrochen wird.

Entfernung von Leerlaufzeiten

In der 1.x-Version von KCL `idleTimeBetweenReadsInMillis` entsprach das zwei Größen:

- Zeitraum zwischen Task-Versandprüfungen. Sie können diesen Zeitraum zwischen Tasks jetzt mit `CoordinatorConfig#shardConsumerDispatchPollIntervalMillis` konfigurieren.
- Zeit im Ruhemodus, wenn keine Datensätze von Kinesis Data Streams zurückgegeben wurden. In Version 2.0 werden Datensätze im Rahmen der verbesserten Rundsendung von den jeweiligen Abrufern im Push-Verfahren übermittelt. Aktivitäten auf dem Shard-Konsumenten treten nur auf, wenn eine Anforderung per Push ankommt.

Entfernung der Client-Konfiguration

In Version 2.0 erstellt der KCL keine Clients mehr. Der Benutzer muss einen gültigen Client bereitstellen. Mit dieser Änderung wurden alle Konfigurationsparameter für die Client-Erstellung entfernt. Wenn Sie diese Parameter benötigen, können Sie sie in den Clients einstellen, bevor die Clients für `ConfigsBuilder` bereitgestellt werden.

Entferntes Field	Äquivalente Konfiguration
`kinesisEndpoint`	Konfigurieren Sie den SDK `KinesisAsyncClient` mit dem bevorzugten Endpunkt:`KinesisAsyncClient.builder().endpointOverride(URI.create("https://<kinesis endpoint>")).build()` .
`dynamoDBEndpoint`	Konfigurieren Sie den SDK `DynamoDbAsyncClient` mit dem bevorzugten Endpunkt:`DynamoDbAsyncClient.builder().endpointOverride(URI.create("https://<dynamodb endpoint>")).build()` .
`kinesisClientConfig`	Konfigurieren Sie das SDK `KinesisAsyncClient` mit der erforderlichen Konfiguration:`KinesisAsyncClient.builder().overrideConfiguration(<your configuration>).build()` .
`dynamoDBClientConfig`	Konfigurieren Sie das SDK `DynamoDbAsyncClient` mit der erforderlichen Konfiguration:`DynamoDbAsyncClient.builder().overrideConfiguration(<your configuration>).build()` .
`cloudWatchClientConfig`	Konfigurieren Sie das SDK `CloudWatchAsyncClient` mit der erforderlichen Konfiguration:`CloudWatchAsyncClient.builder().overrideConfiguration(<your configuration>).build()` .

Entferntes Field	Äquivalente Konfiguration
`regionName`	Konfigurieren Sie das SDK mit der bevorzugten Region. Dies ist für alle SDK Kunden gleich. Zum Beispiel , ., `KinesisAsyncClient.builder( ).region(Region.US_WEST_2).build()` .

Daten aus Kinesis Data Streams mithilfe anderer AWS Dienste lesen

Die folgenden AWS Services können direkt in Amazon Kinesis Data Streams integriert werden, um Daten aus Kinesis-Datenströmen zu lesen. Überprüfen Sie die Informationen für jeden Service, an dem Sie interessiert sind, und verweisen Sie auf die bereitgestellten Referenzen.

Themen

- Daten aus Kinesis Data Streams mit Amazon lesen EMR
- Daten aus Kinesis Data Streams mit Amazon EventBridge Pipes lesen
- Daten aus Kinesis Data Streams lesen mit AWS Glue
- Daten aus Kinesis Data Streams mit Amazon Redshift lesen

Daten aus Kinesis Data Streams mit Amazon lesen EMR

EMRAmazon-Cluster können Kinesis-Streams direkt lesen und verarbeiten, indem sie vertraute Tools aus dem Hadoop-Ökosystem wie Hive, Pig MapReduce, Hadoop Streaming API und Cascading verwenden. Sie können auch Echtzeitdaten aus Kinesis Data Streams mit vorhandenen Daten auf Amazon S3, Amazon DynamoDB und HDFS in einem laufenden Cluster verknüpfen. Sie können die Daten für die Nachbearbeitung direkt von Amazon EMR in Amazon S3 oder DynamoDB laden.

Weitere Informationen finden Sie unter Amazon Kinesis im Amazon EMR Release Guide.

Daten aus Kinesis Data Streams mit Amazon EventBridge Pipes lesen

Amazon EventBridge Pipes unterstützt Kinesis-Datenstreams als Quelle. Amazon EventBridge Pipes unterstützt Sie bei der Erstellung von point-to-point Integrationen zwischen Veranstaltern und Verbrauchern mit optionalen Transformations-, Filter- und Anreicherungsschritten. Sie können EventBridge Pipes verwenden, um Datensätze in einem Kinesis-Datenstream zu empfangen und

diese Datensätze optional zu filtern oder zu verbessern, bevor Sie sie zur Verarbeitung an eines der verfügbaren Ziele senden, einschließlich Kinesis Data Streams.

Weitere Informationen finden Sie unter Amazon Kinesis Stream as a source im Amazon EventBridge Release Guide.

Daten aus Kinesis Data Streams lesen mit AWS Glue

Mithilfe von AWS Glue Streaming ETL können Sie Streaming-Jobs zum Extrahieren, Transformieren und Laden (ETL) erstellen, die kontinuierlich ausgeführt werden und Daten aus Amazon Kinesis Data Streams verbrauchen. Die Jobs bereinigen und transformieren die Daten und laden die Ergebnisse anschließend in Amazon S3 S3-Datenseen oder JDBC Datenspeicher.

Weitere Informationen finden Sie unter ETLStreaming-Jobs AWS Glue im AWS Glue Versionshandbuch.

Daten aus Kinesis Data Streams mit Amazon Redshift lesen

Amazon Redshift unterstützt die Streaming-Erfassung von Amazon Kinesis Data Streams. Das Streaming-Erfassungs-Feature Amazon Redshift ermöglicht das Erfassen von Streaming-Daten mit geringer Latenz und hoher Geschwindigkeit aus Amazon Kinesis Data Streams in einer materialisierten Ansicht von Amazon Redshift. Durch die Amazon Redshift-Streaming-Aufnahme müssen Daten nicht mehr in Amazon S3 gespeichert werden, bevor sie in Amazon Redshift aufgenommen werden.

Weitere Informationen finden Sie unter Streaming-Erfassung im Handbuch zum Release von Amazon Redshift.

Lesen Sie mithilfe von Integrationen von Drittanbietern aus Kinesis Data Streams

Sie können Daten aus Amazon Kinesis Data Streams mit einer der folgenden Drittanbieteroptionen lesen, die in Kinesis Data Streams integriert sind. Wählen Sie die Option aus, über die Sie mehr erfahren möchten, und finden Sie Ressourcen und Links zu relevanter Dokumentation.

Themen

- Apache Flink
- Adobe Experience Platform

- Apache Druid
- Apache Spark
- Databricks
- Kafka Confluent Plattform
- Kinesumer
- Talend

Apache Flink

Apache Flink ist ein Framework und eine verteilte Verarbeitungs-Engine für statusbehaftete Berechnungen über unbegrenzte und begrenzte Datenströme. Weitere Informationen zur Nutzung von Kinesis Data Streams mit Apache Flink finden Sie unter Amazon Kinesis Data Streams Connector.

Adobe Experience Platform

Die Adobe Experience Platform ermöglicht es Unternehmen, Kundendaten aus jedem System zu zentralisieren und zu standardisieren. Anschließend werden Datenwissenschaft und Machine Learning angewendet, um das Design und die Bereitstellung umfassender, personalisierter Erlebnisse erheblich zu verbessern. Weitere Informationen zur Nutzung von Kinesis-Datenströmen mit der Adobe Experience Platform finden Sie unter Amazon Kinesis Connector.

Apache Druid

Druid ist eine hochleistungsfähige Echtzeit-Analysedatenbank, die Abfragen auf Streaming- und Batch-Daten in Sekundenschnelle und bei hoher Auslastung ermöglicht. Weitere Informationen zur Aufnahme von Kinesis-Datenströmen mit Apache Druid finden Sie unter Amazon Kinesis-Aufnahme.

Apache Spark

Apache Spark ist eine einheitliche Analytics-Engine für die großflächige Datenverarbeitung. Es bietet hochwertige APIs Funktionen in Java, Scala, Python und R sowie eine optimierte Engine, die allgemeine Ausführungsdiagramme unterstützt. Sie können Apache Spark verwenden, um Anwendungen zur Stream-Verarbeitung zu erstellen, die die Daten in Ihren Kinesis-Datenströmen nutzen.

Verwenden Sie den Amazon Kinesis Data Streams-Connector, um Kinesis-Datenstreams mit Apache Spark Structured Streaming zu nutzen. Dieser Connector unterstützt die Nutzung mit Enhanced

Fan-Out, wodurch Ihre Anwendung einen dedizierten Lesedurchsatz von bis zu 2 MB Daten pro Sekunde und Shard erhält. Weitere Informationen finden Sie unter Entwicklung benutzerdefinierter Verbraucher mit dediziertem Durchsatz (erweiterter Fan-Out).

Informationen zur Nutzung von Kinesis-Datenströmen mit Spark Streaming finden Sie unter Spark Streaming + Kinesis Integration.

Databricks

Databricks ist eine Cloud-basierte Plattform, die eine kollaborative Umgebung für Datentechnik, Datenwissenschaft und Machine Learning bietet. Weitere Informationen zur Nutzung von Kinesis-Datenströmen mithilfe von Databricks finden Sie unter Connect zu Amazon Kinesis herstellen.

Kafka Confluent Plattform

Confluent Platform basiert auf Kafka und bietet zusätzliche Features und Funktionen, mit denen Unternehmen Daten-Pipelines und Streaming-Anwendungen in Echtzeit aufbauen und verwalten können. Weitere Informationen zur Nutzung von Kinesis-Datenströmen mithilfe der Confluent Platform finden Sie unter Amazon Kinesis Source Connector for Confluent Platform.

Kinesumer

Kinesumer ist ein Go-Client, der einen clientseitigen Client für verteilte Verbrauchergruppen für Kinesis-Datenströme implementiert. Weitere Informationen finden Sie im Kinesumer-GitHub-Repository.

Talend

Talend ist eine Datenintegrations- und Verwaltungssoftware, die es Benutzern ermöglicht, Daten aus verschiedenen Quellen auf skalierbare und effiziente Weise zu sammeln, zu transformieren und miteinander zu verbinden. Weitere Informationen zur Nutzung von Kinesis-Datenströmen mit Talend finden Sie unter Talend mit einem Amazon Kinesis Kinesis-Stream Connect.

Problembehandlung bei Kinesis Data Streams Streams-Verbrauchern

Die folgenden Themen bieten Lösungen für häufig auftretende Probleme von Amazon Kinesis Data Streams Streams-Benutzern:

Einige Kinesis Data Streams Streams-Datensätze werden bei der Verwendung der Kinesis Client Library übersprungen

Der häufigste Grund für übersprungene Datensätzen ist ein Ausnahmefehler, der von `processRecords` ausgelöst wird. Die Kinesis Client Library (KCL) stützt sich auf Ihren `processRecords` Code, um alle Ausnahmen zu behandeln, die sich aus der Verarbeitung der Datensätze ergeben. Jede Ausnahme, die ausgelöst wird, `processRecords` wird von der KCL aufgenommen. Um unendliche Wiederholungsversuche bei einem wiederkehrenden Fehler zu vermeiden, KCL sendet der den Stapel von Datensätzen, die zum Zeitpunkt der Ausnahme verarbeitet wurden, nicht erneut. Das ruft KCL `processRecords` dann den nächsten Stapel von Datensätzen auf, ohne den Datensatzprozessor neu zu starten. Dies führt dazu, dass Konsumentenanwendungen übersprungene Datensätze erkennen. Zur Vermeidung übersprungener Datensätze müssen Sie alle Ausnahmen innerhalb von `processRecords` entsprechend behandeln.

Datensätze, die zu demselben Shard gehören, werden gleichzeitig von verschiedenen Datensatzprozessoren verarbeitet

Für jede laufende Kinesis Client Library (KCL) -Anwendung hat ein Shard nur einen Besitzer. Allerdings kann es vorkommen, dass mehrere Datensatzprozessoren vorübergehend denselben Shard verarbeiten. Im Fall einer Worker-Instance, die die Netzwerkkonnektivität verliert, KCL geht der davon aus, dass der nicht erreichbare Worker nach Ablauf der Failover-Zeit keine Datensätze mehr verarbeitet, und weist andere Worker-Instances an, die Kontrolle zu übernehmen. Für eine kurze Zeit werden dann möglicherweise Daten aus demselben Shard von neuen Datensatzprozessoren und Datensatzprozessoren des nicht erreichbaren Workers verarbeitet.

Legen Sie deshalb eine für Ihre Anwendung angemessene Failover-Zeit fest. Für Anwendungen mit geringer Latenz entspricht der Standardwert von 10 Sekunden möglicherweise dem Zeitraum, den Sie maximal bereit sind zu warten. In den Fällen jedoch, in denen Sie Verbindungsprobleme erwarten, beispielsweise bei Aufrufen über verschiedene geografische Bereiche hinweg, in denen häufiger mit Verbindungsproblemen zu rechnen ist, ist die Zeitspanne möglicherweise zu gering.

Ihre Anwendung sollte diesem Szenario abhelfen können, insbesondere, weil die Netzwerkanbindung in der Regel für den zuvor nicht erreichbaren Worker wiederhergestellt wird. Wenn die Shards eines Datensatzprozessors von einem anderen Datensatzprozessor übernommen werden, müssen folgende Fälle für ein ordnungsgemäßes Herunterfahren korrekt behandelt werden:

1. Nach Abschluss des aktuellen Aufrufs von KCL ruft der die Shutdown-Methode auf dem Datensatzprozessor mit dem Grund '' für das Herunterfahren auf. `processRecords` ZOMBIE Von Ihren Datensatzprozessoren wird erwartet, dass alle Ressourcen entsprechend bereinigt werden und die Datensatzprozessoren anschließend beendet werden.
2. Wenn Sie versuchen, von einem „Zombie“ -Arbeiter aus einen Checkpoint zu erreichen, werden die KCL Würfe ausgeführt. `ShutdownException` Nach Empfang der Ausnahme ist die aktuelle Methode durch Ihren Code sauber zu beenden.

Weitere Informationen finden Sie unter Behandeln Sie doppelte Datensätze.

Die Verbraucheranwendung liest langsamer als erwartet

Die häufigsten Gründe für einen langsameren Lesedurchsatz sind:

1. Bei mehreren Konsumentenanwendungen übersteigt die Anzahl der Lesevorgänge die pro Shard festgelegten Limits. Weitere Informationen finden Sie unter Kontingente und -Einschränkungen. Erhöhen Sie in diesem Fall die Anzahl der Shards im Kinesis-Datenstrom.
2. Das Limit für die maximale Anzahl von GetRecords pro Aufruf ist möglicherweise zu niedrig festgelegt. Wenn Sie den verwendenKCL, haben Sie den Worker möglicherweise mit einem niedrigen Wert für die `maxRecords` Eigenschaft konfiguriert. Grundsätzlich empfehlen wir die Verwendung der Systemvoreinstellungen für diese Eigenschaft.
3. Die Logik in Ihrem `processRecords` Aufruf kann aus verschiedenen Gründen länger als erwartet dauern. Möglicherweise ist die Logik CPU intensiv, blockiert I/O oder es kommt zu Engpässen bei der Synchronisation. Führen Sie leere Datensatzprozessoren aus und vergleichen Sie den Lesedurchsatz, um festzustellen, ob dies der Fall ist. Weitere Informationen dazu, wie Sie

eingehende Daten zeitgerecht verarbeiten können, finden Sie unter Verwenden Sie Resharding, Skalierung und Parallelverarbeitung, um die Anzahl der Shards zu ändern.

Wenn Sie nur eine Konsumentenanwendung haben, ist es immer möglich, mindestens zwei Mal schneller als die PUT-Rate zu lesen. Das liegt daran, dass Sie bis zu 1 000 Datensätze pro Sekunde für Schreibvorgänge bis zu einem maximalen Datenschreibvolumen von 1 MB pro Sekunde (einschließlich Partitionsschlüsseln) schreiben können. Jeder offene Shard kann bis zu 5 Lesetransaktionen pro Sekunde unterstützen, bis zu einer maximalen Gesamtdatenleserate von 2 MB pro Sekunde. Beachten Sie, dass jeder Lesevorgang (GetRecords-Aufruf) einen Stapel an Datensätzen erhält. Die Menge der Daten, die von GetRecords zurückgegeben werden, hängt von der Shard-Auslastung ab. Die maximale Größe der Daten, die GetRecords zurückgeben kann, ist 10 MB. Wenn ein Aufruf das Limit zurückgibt, lösen nachfolgende Aufrufe innerhalb der nächsten 5 Sekunden `ProvisionedThroughputExceededException` aus.

GetRecords gibt ein leeres Datensatz-Array zurück, auch wenn der Stream Daten enthält

Beim Verbrauchen oder Empfangen von Datensätzen kommt ein PULL-Modell zum Einsatz. Von Entwicklern wird erwartet, dass sie GetRecordsin einer Dauerschleife ohne Back-offs aufrufen. Jeder Aufruf von GetRecords gibt auch einen `ShardIterator`-Wert zurück, der in der nächsten Iteration der Schleife verwendet wird.

Die GetRecords-Operation blockiert nicht. Stattdessen erfolgt sofort eine Rückgabe, entweder mit relevanten Datensätzen oder mit einem leeren `Records`-Element. Ein leeres `Records`-Element wird unter zwei Bedingungen zurückgegeben:

1. Im Shard sind derzeit keine weiteren Daten vorhanden.
2. Es gibt keine Daten in der Nähe des Shard-Inhalts, auf den `ShardIterator` zeigt.

Die letzte Bedingung ist subtil, aber eine notwendige Design-Komponente, durch die unbegrenzte Suchzeiten (Latenzen) beim Abrufen von Datensätzen vermieden werden. Daher sollte die Anwendung, die Daten aus dem Stream verbraucht, eine Schleife durchlaufen, GetRecords aufrufen und leere Datensätze verarbeiten.

In einer Produktionsumgebung sollte die Endlosschleife nur verlassen werden, wenn der Wert von `NextShardIterator` `NULL` ist. Wenn `NextShardIterator` `NULL` ist, bedeutet dies, dass der aktuelle Shard geschlossen wurde und der Wert von `ShardIterator` ansonsten auf eine

Position hinter dem letzten Datensatz verweisen würde. Wenn die datenverbrauchende Anwendung SplitShard oder MergeShards nicht aufruft, bleibt der Shard offen und die Aufrufe von GetRecords geben niemals einen `NextShardIterator`-Wert von `NULL` zurück.

Wenn Sie die Kinesis Client Library (KCL) verwenden, wird das obige Verbrauchsmuster für Sie abstrahiert. Dies umfasst die automatische Handhabung einer Gruppe von Shards, die sich dynamisch ändert. Mit dem KCL stellt der Entwickler nur die Logik zur Verarbeitung eingehender Datensätze bereit. Dies ist möglich, da die Bibliothek GetRecords kontinuierlich für Sie aufruft.

Der Shard-Uterator läuft unerwartet ab

Bei jeder GetRecords-Anforderung (wie `NextShardIterator`) wird ein neuer Shard-Iterator zurückgegeben, den Sie bei der nächsten GetRecords-Anforderung (wie `ShardIterator`) verwenden. In der Regel läuft dieser Shard-Iterator nicht vor der Verwendung ab. Es kann jedoch sein, dass Shard-Iteratoren ihre Gültigkeit verlieren, weil Sie GetRecords länger als 5 Minuten nicht aufgerufen oder die Konsumentenanwendung neu gestartet haben.

Wenn der Shard-Iterator abläuft, ehe Sie ihn verwenden können, ist dies möglicherweise ein Hinweis darauf, dass die von Kinesis genutzte DynamoDB-Tabelle nicht genügend Kapazität hat, um die Lease-Daten zu speichern. Diese Situation tritt häufiger auf, wenn Sie viele Shards haben. Beheben Sie das Problem, indem Sie die Schreibkapazität der Shard-Tabelle erhöhen. Weitere Informationen finden Sie unter Verwenden Sie eine Leasetabelle, um nachzuverfolgen, welche Shards von der Consumer-Anwendung verarbeitet wurden KCL.

Die Verarbeitung von Verbraucherdaten hinkt hinterher

Bei den meisten Anwendungsfällen lesen die Konsumentenanwendungen die neuesten Daten aus dem Stream. Unter bestimmten Umständen hängt das Lesen des Konsumenten hinterher. Dieses Verhalten möglicherweise nicht erwünscht. Sehen Sie sich die häufigsten Gründe für ein zu langsames Lesen des Datenkonsumenten an, wenn Sie ermittelt haben, wie weit die Lesevorgänge des Konsumenten hinterherhängen.

Beginnen Sie mit der `GetRecords.IteratorAgeMilliseconds`-Metrik. Diese verfolgt die Leseposition aller Shards und Konsumenten im Stream. Beachten Sie: Wenn das Alter eines Iterators 50 % des Aufbewahrungszeitraums (standardmäßig 24 Stunden, anpassbar auf bis zu 365 Tage) überschreitet, besteht die Gefahr des Datenverlusts durch Datensatzablauf. Als schnelle Übergangslösung eignet sich eine Verlängerung des Aufbewahrungszeitraums. Auf diese Weise wird der Verlust von wichtigen Daten gestoppt, während Sie sich der weiteren Fehlerbehebung widmen. Weitere Informationen finden Sie unter Überwachen Sie den Amazon Kinesis Data Streams Streams-

Service mit Amazon CloudWatch. Identifizieren Sie als Nächstes anhand einer benutzerdefinierten CloudWatch Metrik, die von der Kinesis Client Library (KCL) ausgegeben wird, wie weit Ihre Verbraucheranwendung von jedem Shard liest. `MillisBehindLatest` Weitere Informationen finden Sie unter Überwachen Sie die Kinesis-Clientbibliothek mit Amazon CloudWatch.

Im Folgenden sind die häufigsten Gründe für ein Hinterherhängen von Konsumenten aufgeführt:

- Plötzliche starke Anstiege weisen auf ein vorübergehendes Problem hin `GetRecords.IteratorAgeMilliseconds` oder deuten `MillisBehindLatest` normalerweise auf ein vorübergehendes Problem hin, z. B. API Betriebsausfälle bei einer nachgeschalteten Anwendung. Prüfen Sie bei solch plötzlichen Anstiegen, ob eine der Metriken durchgängig dieses Verhalten aufweist.
- Eine allmählicher Anstieg dieser Metriken deutet darauf hin, dass ein Konsument Datensätze nicht schnell genug verarbeitet. Die häufigsten Ursachen für ein solches Verhalten sind unzureichende physische Ressourcen oder eine Datensatzverarbeitungslogik, die nicht für einen Anstieg des Stream-Durchsatzes skaliert wurde. Sie können dieses Verhalten überprüfen, indem Sie sich die anderen benutzerdefinierten CloudWatch Messwerte ansehen, die KCL im Zusammenhang mit dem `processTask` Vorgang ausgegeben werden, einschließlich `RecordProcessor.processRecords.TimeSuccess`, und. `RecordsProcessed`
  - Wenn Sie eine Erhöhung in der `processRecords.Time`-Metrik feststellen, die mit einem Anstieg des Durchsatzes korreliert, sollten Sie Ihre Datensatzverarbeitungslogik analysieren, um herauszufinden, warum sich diese nicht an den erhöhten Durchsatz anpasst.
  - Wenn Sie eine Erhöhung der `processRecords.Time`-Werte feststellen, die nicht in Zusammenhang mit einem erhöhten Durchsatz steht, prüfen Sie, ob im kritischen Pfad blockierende Aufrufe durchgeführt werden. Diese sind häufig der Grund für eine verlangsamte Datensatzverarbeitung. Alternativ können Sie auch die Parallelität durch eine Erhöhung der Anzahl der Shards erhöhen. Stellen Sie abschließend sicher, dass Sie bei Spitzenauslastung über eine ausreichende Menge an physischen Ressourcen (Arbeitsspeicher, CPU Auslastung usw.) auf den zugrunde liegenden Verarbeitungsknoten verfügen.

Unautorisierter KMS Masterkey-Zugriffsfehler

Dieser Fehler tritt auf, wenn eine Verbraucheranwendung ohne Berechtigungen für den KMS Masterschlüssel aus einem verschlüsselten Stream liest. Informationen zum Zuweisen von Berechtigungen für den Zugriff auf einen KMS Schlüssel einer Anwendung finden Sie unter Verwenden von Schlüsselrichtlinien in AWS KMS und Verwenden von IAM Richtlinien mit AWS KMS.

Beheben Sie andere häufig auftretende Probleme von Verbrauchern

- Warum kann der Kinesis-Data-Streams-Trigger meine Lambda-Funktion nicht aufrufen?
- Wie erkenne und behebe ich ReadProvisionedThroughputExceeded Ausnahmen in Kinesis Data Streams?
- Warum habe ich Probleme mit hoher Latenz bei Kinesis Data Streams?
- Warum gibt mein Kinesis-Datenstrom einen 500 Internal Server Error zurück?
- Wie behebe ich Fehler bei einer blockierten oder hängengebliebenen KCL Anwendung für Kinesis Data Streams?
- Kann ich verschiedene Anwendungen der Amazon Kinesis Client Library mit der gleichen Amazon-DynamoDB-Tabelle verwenden?

Optimieren Sie die Verbraucher von Amazon Kinesis Data Streams

Sie können Ihren Amazon Kinesis Data Streams Streams-Nutzer auf der Grundlage des spezifischen Verhaltens, das Sie beobachten, weiter optimieren.

Sehen Sie sich die folgenden Themen an, um Lösungen zu finden.

Themen

- Verbessern Sie die Verarbeitung mit niedriger Latenz
- Verarbeiten Sie serialisierte Daten AWS Lambda mithilfe der Kinesis Producer Library
- Verwenden Sie Resharding, Skalierung und Parallelverarbeitung, um die Anzahl der Shards zu ändern
- Behandeln Sie doppelte Datensätze
- Kümmere dich um das Starten, Herunterfahren und Drosseln

Verbessern Sie die Verarbeitung mit niedriger Latenz

Die Übertragungsverzögerung ist definiert als die end-to-end Latenz von dem Moment, in dem ein Datensatz in den Stream geschrieben wird, bis zu seinem Lesen durch eine Verbraucheranwendung. Diese Verzögerung variiert und hängt von vielen Faktoren ab, im Wesentlichen aber vom Abrufintervall der Konsumentenanwendungen.

Für die meisten Anwendungen empfehlen wir, jeden Shard pro Anwendung einmal pro Sekunde abzurufen. So können mehrere Konsumentenanwendungen einen Stream gleichzeitig unabhängig voneinander verarbeiten, ohne an die Limits von 5 `GetRecords`-Aufrufen pro Sekunde von Amazon Kinesis Data Streams zu stoßen. Darüber hinaus ist die Verarbeitung größerer Datenstapel in Bezug auf Netzwerk- und andere nachgelagerte Latenzzeiten effizienter.

Die KCL Standardwerte entsprechen der bewährten Methode, alle 1 Sekunde eine Abfrage durchzuführen. Dieser Standard führt zu einer durchschnittlichen Verbreitungsverzögerung von weniger als 1 Sekunde.

Die Datensätze von Kinesis Data Streams können sofort nach dem Schreiben gelesen werden. In Fällen, in denen Daten sofort nach Bereitstellung verarbeitet werden müssen, ist dies eine wichtiger Aspekt. Sie können die Übertragungsverzögerung erheblich reduzieren, indem Sie die KCL Standardeinstellungen überschreiben und häufiger Abfragen durchführen, wie in den folgenden Beispielen gezeigt.

Java KCL-Konfigurationscode:

```
kinesisClientLibConfiguration = new
        KinesisClientLibConfiguration(applicationName,
        streamName,
        credentialsProvider,

 workerId).withInitialPositionInStream(initialPositionInStream).withIdleTimeBetweenReadsInMilli
```

Einstellung der Eigenschaftsdatei für Python und RubyKCL:

```
idleTimeBetweenReadsInMillis = 250
```

> **Note**
>
> Da für Kinesis Data Streams ein Limit von 5 `GetRecords`-Aufrufen pro Sekunde pro Shard gilt, kann das Festlegen der `idleTimeBetweenReadsInMillis`-Eigenschaft auf einen Wert von unter 200 ms dazu führen, dass Ihre Anwendung die `ProvisionedThroughputExceededException`-Ausnahme erkennt. Zu viele dieser Ausnahmen können zu exponentiellen Backoffs führen und bedeutsame unerwartete Latenzzeiten bei der Verarbeitung nach sich ziehen. Wenn Sie diese Eigenschaft auf 200 ms oder etwas mehr setzen und mehr als eine Verarbeitungsanwendung haben, kommt es zu einer ähnlichen Drosselung.

Verarbeiten Sie serialisierte Daten AWS Lambda mithilfe der Kinesis Producer Library

Die Kinesis Producer Library (KPL) fasst kleine benutzerformatierte Datensätze zu größeren Datensätzen von bis zu 1 MB zusammen, um den Durchsatz von Amazon Kinesis Data Streams besser zu nutzen. Die KCL für Java unterstützt zwar die Deaggregation dieser Datensätze, Sie müssen jedoch ein spezielles Modul verwenden, um Datensätze zu deaggregieren, wenn Sie sie als Verbraucher Ihrer Streams verwenden. AWS Lambda Den erforderlichen Projektcode und Anweisungen finden Sie GitHub unter Kinesis Producer Library Deaggregation Modules for Lambda. AWS Die Komponenten in diesem Projekt bieten Ihnen die Möglichkeit, KPL serialisierte Daten in Java AWS Lambda, Node.js und Python zu verarbeiten. Diese Komponenten können auch als Teil einer mehrsprachigen Anwendung verwendet werden. KCL

Verwenden Sie Resharding, Skalierung und Parallelverarbeitung, um die Anzahl der Shards zu ändern

Das Resharding ermöglicht Ihnen das Erhöhen oder Verringern der Anzahl der Shards in einem Stream zur Anpassung an die Datenrate. Das Resharding wird üblicherweise von einer administrativen Anwendung durchgeführt, die Datenverwaltungsmetriken zu Shards überwacht. Der KCL selbst initiiert zwar keine Resharding-Operationen, ist aber so konzipiert, dass er sich an Änderungen in der Anzahl der Shards anpasst, die sich aus dem Resharding ergeben.

Wie unter erwähntVerwenden Sie eine Leasetabelle, um nachzuverfolgen, welche Shards von der Consumer-Anwendung verarbeitet wurden KCL, KCL verfolgt der die Shards im Stream mithilfe einer Amazon DynamoDB-Tabelle. Wenn als Ergebnis von Resharding neue Shards erstellt werden, KCL erkennt der die neuen Shards und füllt neue Zeilen in der Tabelle auf. Die Auftragnehmer entdecken die neuen Shards automatisch und erstellen Prozessoren, um die Daten dieser neuen Shards zu verwalten. KCLAußerdem werden die Shards im Stream auf alle verfügbaren Worker und Datenprozessoren verteilt.

KCLDadurch wird sichergestellt, dass alle Daten, die vor dem Resharding in Shards vorhanden waren, zuerst verarbeitet werden. Nachdem diese Daten verarbeitet wurden, werden die Daten der neuen Shards an die Datensatzprozessoren gesendet. Auf diese Weise wird die Reihenfolge KCL beibehalten, in der Datensätze für einen bestimmten Partitionsschlüssel zum Stream hinzugefügt wurden.

Beispiel: Resharding, Skalierung und Parallelverarbeitung

Das folgende Beispiel zeigt, wie Sie mit Skalierung KCL und Resharding umgehen können:

- Zum Beispiel, wenn Ihre Anwendung auf einer EC2 Instance läuft und einen Kinesis-Datenstream mit vier Shards verarbeitet. Diese eine Instanz hat einen KCL Worker und vier Datensatzprozessoren (ein Datensatzprozessor für jeden Shard). Diese vier Datensatzprozessoren werden parallel innerhalb desselben Prozesses ausgeführt.
- Wenn Sie dann die Anwendung für die Verwendung einer weiteren Instance skalieren, verarbeiten zwei Instances einen Stream mit vier Shards. Wenn der KCL Worker auf der zweiten Instanz startet, erfolgt ein Lastenausgleich mit der ersten Instanz, sodass jede Instanz nun zwei Shards verarbeitet.
- Dann beschließen Sie, Ihre vier Shards auf fünf Shards aufzuteilen. Die KCL wiederum koordiniert die Verarbeitung zwischen den Instanzen: Eine Instanz verarbeitet drei Shards, und die andere verarbeitet zwei Shards. Eine ähnliche Koordinierung erfolgt, wenn Sie Shards zusammenführen.

Wenn Sie den verwenden, sollten Sie in der Regel sicherstellenKCL, dass die Anzahl der Instanzen die Anzahl der Shards nicht überschreitet (außer für Ausfallbereitschaftszwecke). Jeder Shard wird von genau einem KCL Worker verarbeitet und verfügt über genau einen entsprechenden Datensatzprozessor, sodass Sie nie mehrere Instanzen benötigen, um einen Shard zu verarbeiten. Ein Auftragnehmer kann allerdings eine beliebige Anzahl von Shards verarbeiten, deshalb ist es kein Problem, wenn Sie mehr Shards als Instances haben.

Testen Sie eine Kombination der folgenden Ansätze, wenn Sie die Verarbeitung in Ihrer Anwendung optimieren möchten:

- Erhöhung der Instance-Größe (da alle Datensatzprozessoren innerhalb eines Prozesses parallel ausgeführt werden)
- Erhöhung der Anzahl der Instances bis zur maximalen Anzahl offener Shards (da Shards unabhängig verarbeitet werden können)
- Erhöhung der Anzahl der Shards (fördert die Parallelität)

Beachten Sie, dass Sie Auto Scaling für die automatische Skalierung Ihrer Instances basierend auf entsprechenden Metriken nutzen können. Weitere Informationen finden Sie im Amazon EC2 Auto Scaling Scaling-Benutzerhandbuch.

Wenn Resharding die Anzahl der Shards im Stream erhöht, erhöht die entsprechende Erhöhung der Anzahl der Datensatzprozessoren die Auslastung der EC2 Instances, die sie hosten. Wenn die Instances Teil einer Auto Scaling-Gruppe sind und die Last entsprechend steigt, fügt die Auto Scaling-Gruppe weitere Instances hinzu, damit die höhere Last ausgeglichen wird. Konfigurieren Sie Ihre Instances so, dass Ihre Anwendung Amazon Kinesis Data Streams beim Startup gestartet wird. So werden zusätzliche Auftragnehmer und Datensatzprozessoren in der neuen Instance sofort aktiviert.

Weitere Informationen zum Resharding finden Sie unter Einen Stream erneut teilen.

Behandeln Sie doppelte Datensätze

Es gibt zwei primäre Gründe, warum Datensätze mehr als einmal an Ihre Anwendung von Amazon Kinesis Data Streams übermittelt werden: Wiederholungsversuche des Produzenten und Wiederholungsversuche des Konsumenten. Ihre Anwendung muss in der Lage sein, einzelne Datensätze mehrere Male angemessen zu verarbeiten.

Der Produzent versucht es erneut

Stellen Sie sich einen Produzenten vor, der nach dem Aufruf von `PutRecord` aber noch vor dem Erhalt einer Bestätigung von Amazon Kinesis Data Streams einen netzwerkbedingten Timeout erfährt. In diesem Fall ist es fraglich, ob der Datensatz auch an Kinesis Data Streams übermittelt wurde. Davon ausgehend, dass jeder Datensatz für die Anwendung wichtig ist, ist der Produzent so aufgebaut, dass er den Aufruf mit denselben Daten wiederholt. Wenn beide `PutRecord`-Aufrufe mit denselben Daten erfolgreich an Kinesis Data Streams übergeben werden, sind anschließend zwei Datensätze von Kinesis Data Streams vorhanden. Obwohl die beiden Datensätze identische Daten enthalten, haben Sie trotzdem eindeutige Sequenznummern. Anwendungen, die Garantien benötigen, sollten einen Primärschlüssel in den Datensatz einbetten, sodass doppelte Datensätze später bei der Verarbeitung entfernt werden. Beachten Sie, dass die Anzahl der Duplikate aufgrund von Wiederholungen durch den Produzenten verglichen mit der Anzahl der Duplikate aufgrund von Wiederholungen durch den Konsumenten in der Regel nicht sehr hoch ist.

> **Note**
>
> Wenn Sie den verwenden AWS SDK`PutRecord`, erfahren Sie mehr über das SDK Wiederholungsverhalten im Benutzerhandbuch AWS SDKs und Tools.

Der Verbraucher versucht es erneut

Wiederholungen durch den Konsumenten (Datenverarbeitungsanwendungen) kommen vor, wenn Datenprozessoren neu gestartet werden. Datensatzprozessoren für denselben Shard werden in den folgenden Fällen neu gestartet:

1. Ein Auftragnehmer wird unerwartet beendet
2. Es werden Auftragnehmer-Instances hinzugefügt oder entfernt
3. Es werden Shards zusammengeführt oder getrennt
4. Die Anwendung wird bereitgestellt

In all diesen Fällen wird die shards-to-worker-to -record-processor-Zuordnung kontinuierlich aktualisiert, um die Lastenausgleichsverarbeitung zu berücksichtigen. Shard-Prozessoren, die in andere Instances migriert wurden, starten das Verarbeiten von Datensätzen am letzten Prüfpunkt neu. Dies führt zu einer doppelten Datensatzverarbeitung, wie in dem folgenden Beispiel dargestellt. Weitere Informationen zur Lastverteilung finden Sie unter Verwenden Sie Resharding, Skalierung und Parallelverarbeitung, um die Anzahl der Shards zu ändern.

Beispiel: Wiederholte Versuche eines Verbrauchers führen zu erneut zugestellten Datensätzen

In diesem Beispiel haben Sie eine Anwendung, die kontinuierlich Datensätze aus einem Stream liest, Datensätze in eine lokale Datei aggregiert und die Datei in Amazon S3 hochlädt. Der Einfachheit halber gehen wir davon aus, dass nur 1 Shard und 1 Auftragnehmer den Shard verarbeiten. Beachten Sie die folgende Beispielabfolge von Ereignissen und gehen Sie davon aus, dass der letzte Prüfpunkt bei Datensatznummer 10.000 gesetzt wurde.

1. Ein Auftragnehmer liest den nächsten Datensatzstapel aus dem Shard (Datensätze 10.001 bis 20.000).
2. Anschließend übermittelt der Auftragnehmer diesen an den zugehörigen Datensatzprozessor.
3. Der Datensatzprozessor aggregiert die Daten, erstellt eine Amazon-S3-Datei und lädt die Datei erfolgreich in Amazon S3 hoch.
4. Der Auftragnehmer wird unerwartet beendet, noch ehe er einen neuen Prüfpunkt setzen kann.
5. Anwendung, Auftragnehmer und Datensatzprozessor starten neu.
6. Der Auftragnehmer beginnt beim letzten erfolgreich gesetzten Prüfpunkt mit dem Lesen, hier Datensatznummer 10.001.

Somit werden die Datensätze 10.001 bis 20.000 mehr als einmal verwendet.

Widerstandsfähigkeit gegenüber Wiederholungsversuchen durch Verbraucher

Auch wenn Datensätze mehrmals verarbeitet werden, kann die Anwendung vorgeben, dass Datensätze nur einmal verarbeitet wurden (idempotente Verarbeitung). Die Lösungen für dieses Problem variieren in Komplexität und Genauigkeit. Wenn das Ziel der resultierenden Daten doppelte Datensätze ordnungsgemäß handhaben kann, sollten Sie sich auf diese Anwendung verlassen, um eine idempotente Verarbeitung zu erreichen. Mit Opensearch können Sie beispielsweise eine Kombination aus Versionierung und Unique verwenden, um doppelte Verarbeitung IDs zu verhindern.

Die Beispielanwendung aus dem vorherigen Abschnitt liest kontinuierlich Datensätze aus dem Stream, aggregiert diese in eine lokale Datei und lädt diese Datei in Amazon S3 hoch. Wie gezeigt, werden die Datensätze 10 001 bis 20 000 mehr als einmal verarbeitet, sodass es mehrere Amazon-S3-Dateien mit identischen Daten gibt. Eine Möglichkeit, Duplikate in diesem Beispiel zu vermeiden, besteht darin, sicherzustellen, dass bei Schritt 3 das folgende Schema verwendet wird:

1. Der Datensatzprozessor verwendet eine feste Anzahl von Datensätzen pro Amazon-S3-Datei, beispielsweise 5 000.
2. Der Dateiname nutzt folgendes Schema: Amazon-S3-Präfix, Shard-ID und `First-Sequence-Num`. In diesem Fall könnte dies `sample-shard000001-10001` sein.
3. Nach dem Hochladen der Amazon-S3-Datei setzen Sie durch Angabe von `Last-Sequence-Num` einen Prüfpunkt. In diesem Fall würden Sie bei Datensatznummer 15.000 einen Prüfpunkt setzen.

Bei diesem Schema hat die resultierende Amazon-S3-Datei auch bei einer mehrfachen Verarbeitung der Datensätze denselben Namen und Dateninhalt. Die Wiederholungen führen nur dazu, dass dieselben Daten mehrmals in dieselbe Datei geschrieben werden.

Bei einer Reshard-Operation ist die Anzahl der im Shard verbliebenen Datensätze möglicherweise kleiner als benötigte Anzahl Ihrer gewünschten, festen Anzahl. In diesem Fall muss die `shutdown()`-Methode die Datei an Amazon S3 übertragen und bei der letzten Sequenznummer einen Prüfpunkt setzen. Die obige Schema ist auch mit Reshard-Operationen kompatibel.

Kümmere dich um das Starten, Herunterfahren und Drosseln

Nachfolgend einige Überlegungen, die Sie in das Design Ihrer Amazon Kinesis Data Streams einfließen lassen sollten.

Themen

- Starten Sie Datenproduzenten und Datenkonsumenten
- Fahren Sie eine Amazon Kinesis Data Streams Streams-Anwendung herunter
- Drosselung lesen

Starten Sie Datenproduzenten und Datenkonsumenten

Standardmäßig KCL beginnt der mit dem Lesen von Datensätzen am Anfang des Streams, dem zuletzt hinzugefügten Datensatz. Wenn bei dieser Konfiguration eine datenproduzierende Anwendung Datensätze zum Stream hinzufügt, ehe empfangsbereite Datensatzprozessoren ausgeführt werden, werden diese Datensätze nach dem Start der Datensatzprozessoren nicht gelesen.

Damit Datenprozessoren immer am Anfang des Streams mit dem Lesen beginnen, legen Sie den folgenden Wert in der Eigenschaftendatei Ihrer Amazon Kinesis Data Streams fest:

```
initialPositionInStream = TRIM_HORIZON
```

Standardmäßig speichert Amazon Kinesis Data Streams alle Daten für 24 Stunden. Er unterstützt auch eine erweiterte Aufbewahrung von bis zu 7 Tagen und die langfristige Aufbewahrung von bis zu 365 Tagen. Dieser Zeitrahmen wird als Aufbewahrungszeitraum bezeichnet. Wenn Sie die Startposition auf `TRIM_HORIZON` festsetzen, beginnt der Datensatzprozessor entsprechend des definierten Aufbewahrungszeitraums mit den ältesten Daten. Wenn ein Datensatzprozessor nach Ablauf des Aufbewahrungszeitraums gestartet wird, sind trotz `TRIM_HORIZON`-Einstellung einige der Datensätze aus dem Stream nicht mehr verfügbar. Aus diesem Grund sollten Verbraucheranwendungen immer aus dem Stream lesen und anhand der CloudWatch Metrik überwachen, `GetRecords.IteratorAgeMilliseconds` ob die Anwendungen mit den eingehenden Daten Schritt halten.

In einigen Fällen mag es unerheblich sein, wenn Datensatzprozessoren die ersten Datensätze im Stream auslassen. Sie könnten beispielsweise einige erste Datensätze durch den Stream laufen lassen, um zu testen, ob der Stream end-to-end erwartungsgemäß funktioniert. Nach dieser anfänglichen Verifizierung starten Sie dann die Auftragnehmer und beginnen mit der Einleitung von Produktionsdaten in den Stream.

Weitere Informationen zur `TRIM_HORIZON`-Einstellung finden Sie unter Verwenden Sie Shard-Iteratoren.

Fahren Sie eine Amazon Kinesis Data Streams Streams-Anwendung herunter

Wenn Ihre Amazon Kinesis Data Streams Streams-Anwendung ihre vorgesehene Aufgabe abgeschlossen hat, sollten Sie sie herunterfahren, indem Sie die EC2 Instances beenden, auf denen sie ausgeführt wird. Sie können die Instances über die AWS Management Console oder die AWS CLI beenden.

Nachdem Sie Ihre Amazon Kinesis Data Streams Streams-Anwendung heruntergefahren haben, sollten Sie die Amazon DynamoDB-Tabelle löschen, mit der der Status der KCL Anwendung verfolgt wurde.

Drosselung lesen

Der Durchsatz eines Streams wird auf Shard-Ebene bereitgestellt. Jeder Shard hat einen Lesedurchsatz von bis zu 5 Transaktionen pro Sekunde für Lesevorgänge, bis zu einer maximalen Gesamtdatenleserate von 2 MB pro Sekunde. Wenn eine Anwendung (oder eine Gruppe von Anwendungen im selben Stream) versucht, Daten schneller aus einem Shard abzurufen, drosselt Kinesis Data Streams die entsprechenden GET-Operationen.

Wenn in einer Amazon Kinesis Data Streams ein Datensatzprozessor Daten schneller als das Limit verarbeitet, beispielsweise bei einem Ausfall, kommt es zu einem Failover. Da die Interaktionen zwischen der Anwendung und Kinesis Data Streams KCL verwaltet werden, treten Drosselungsausnahmen eher im KCL Code als im Anwendungscode auf. Da diese Ausnahmen jedoch KCL protokolliert werden, werden sie in den Protokollen angezeigt.

Wenn Sie feststellen, dass Ihre Anwendung permanent gedrosselt wird, sollten Sie eine Erhöhung der Shards für diesen Stream in Betracht ziehen.

Kinesis Data Streams überwachen

Sie können Datenströme in Amazon Kinesis Data Streams mit den folgenden Features überwachen:

- CloudWatch Metriken — Kinesis Data Streams sendet CloudWatch benutzerdefinierte Amazon-Metriken mit detaillierter Überwachung für jeden Stream.
- Kinesis Agent — Der Kinesis Agent veröffentlicht benutzerdefinierte CloudWatch Metriken, anhand derer beurteilt werden kann, ob der Agent erwartungsgemäß funktioniert.
- APIProtokollierung — Kinesis Data Streams verwendet AWS CloudTrail , um API Aufrufe zu protokollieren und die Daten in einem Amazon S3 S3-Bucket zu speichern.
- Kinesis Client Library — Die Kinesis Client Library (KCL) bietet Metriken pro Shard, Worker und Anwendung. KCL
- Kinesis Producer Library — Die Kinesis Producer Library (KPL) bietet Metriken pro Shard, Worker und Anwendung. KPL

Weitere Informationen zu allgemeinen Überwachungsproblemen, Fragen und zur Fehlerbehebung finden Sie im Folgenden:

- Welche Metriken sollte ich verwenden, um Probleme mit Kinesis Data Streams zu überwachen und zu beheben?
- Warum steigt der IteratorAgeMilliseconds Wert von Kinesis Data Streams ständig?

Überwachen Sie den Amazon Kinesis Data Streams Streams-Service mit Amazon CloudWatch

Amazon Kinesis Data Streams und Amazon CloudWatch sind integriert, sodass Sie CloudWatch Metriken für Ihre Kinesis-Datenstreams sammeln, anzeigen und analysieren können. Um beispielsweise die Shard-Nutzung zu verfolgen, können Sie die Kennzahlen `IncomingBytes` und `OutgoingBytes` überwachen und mit der Anzahl der Shards im Stream vergleichen.

Stream-Metriken und Metriken auf Shard-Ebene, die Sie konfigurieren, werden automatisch erfasst und in jede Minute übertragen. CloudWatch Die Metriken werden für zwei Wochen archiviert. Nach Ablauf dieses Zeitraums werden die Daten verworfen.

Die folgende Tabelle beschreibt die Überwachung von Kinesis-Datenströmen auf grundlegender Stream- und erweiterter Shard-Ebene.

Typ	Beschreibung
Grundlegend (Stream-Ebene)	Daten auf Stream-Ebene werden automatisch und kostenlos jede Minute gesendet.
Erweitert (Shard-Ebene)	Daten auf Shard-Ebene werden für zusätzlic he Kosten jede Minute gesendet. Um diese Datenebene zu erhalten, müssen Sie sie speziell für den Stream aktivieren, der den Vorgang verwendet. EnableEnhancedMoni toring Informationen zu den Preisen finden Sie auf der CloudWatch Amazon-Produktseite.

Dimensionen und Metriken von Amazon Kinesis Data Streams

Kinesis Data Streams sendet Metriken CloudWatch auf zwei Ebenen: auf Stream-Ebene und optional auf Shard-Ebene. Metriken auf Stream-Ebene eignen sich unter normalen Bedingungen für die häufigsten Überwachungsanwendungsfälle. Metriken auf Shard-Ebene beziehen sich auf bestimmte Überwachungsaufgaben, die sich normalerweise auf die Fehlerbehebung beziehen, und werden mithilfe des Vorgangs aktiviert. EnableEnhancedMonitoring

Eine Erläuterung der CloudWatch anhand von Metriken gesammelten Statistiken finden Sie unter CloudWatch Statistiken im CloudWatch Amazon-Benutzerhandbuch.

Themen

- Grundlegende Metriken auf Stream-Ebene
- Verbesserte Metriken auf Shard-Ebene
- Dimensionen für Amazon Kinesis Data Streams Streams-Metriken
- Empfohlene Amazon Kinesis Data Streams Streams-Metriken

Grundlegende Metriken auf Stream-Ebene

Der AWS/Kinesis-Namespace enthält die folgenden Metriken auf Stream-Ebene.

Kinesis Data Streams sendet diese Metriken auf Stream-Ebene an CloudWatch jede Minute. Diese Metriken sind jederzeit verfügbar:

Metrik	Beschreibung
GetRecords.Bytes	Anzahl der Byte, die im angegebenen Zeitraum vom Kinesis-Stream abgerufen wurden. Die Statistiken Minimum, Maximum und Durchschnitt geben die Byte in einem einzelnen GetRecords -Vorgang für den Stream im angegebenen Zeitraum an. Metrikname auf Shard-Ebene: OutgoingBytes Abmessungen: StreamName Gültige Statistiken: Minimum, Maximum, Durchschnitt, Summe, Stichproben Einheiten: Byte
GetRecords.IteratorAge	Diese Metrik ist veraltet. Verwenden Sie GetRecords.IteratorAgeMilliseconds .
GetRecords.IteratorAgeMilliseconds	Alter des letzten Datensatzes in allen GetRecords -Aufrufen an einen Kinesis-Stream, gemessen im angegebenen Zeitraum. Das Alter ist die Differenz zwischen der aktuellen Zeit und der Zeit, zu der der letzte Datensatz des GetRecords -Aufrufs in den Stream geschrieben wurde. Die Statistiken Minimum und Maximum können zum Nachverfolgen des Fortschritts von Kinesis-Verbraucheranwendungen verwendet werden. Wenn der Wert null beträgt, sind die gelesenen Datensätze vollständig mit dem Stream aktualisiert. Metrikname auf Shard-Ebene: IteratorAgeMilliseconds

Metrik	Beschreibung
	Abmessungen: StreamName Gültige Statistiken: Minimum, Maximum, Durchschnitt, Stichproben Einheiten: Millisekunden
`GetRecords.Latency`	Die Dauer pro `GetRecords` -Vorgang, gemessen im angegebenen Zeitraum. Abmessungen: StreamName Statistiken: Minimum, Maximum, Durchschnitt Einheiten: Millisekunden
`GetRecords.Records`	Anzahl der Datensätze, die im angegebenen Zeitraum vom Shard abgerufen wurden. Die Statistiken Minimum, Maximum und Durchschnitt geben die Datensätze in einem einzelnen `GetRecords` -Vorgang für den Stream im angegebenen Zeitraum an. Metrikname auf Shard-Ebene: `OutgoingRecords` Abmessungen: StreamName Gültige Statistiken: Minimum, Maximum, Durchschnitt, Summe, Stichproben Einheiten: Anzahl
`GetRecords.Success`	Anzahl der erfolgreichen `GetRecords` -Vorgänge pro Stream im angegebenen Zeitraum. Abmessungen: StreamName Gültige Statistiken: Durchschnitt, Summe, Stichproben Einheiten: Anzahl

Metrik	Beschreibung
`IncomingBytes`	Anzahl der Byte, die im angegebenen Zeitraum erfolgreich an den Kinesis-Stream übertragen wurden. Diese Metrik enthält die Byte von `PutRecord` - und `PutRecords` -Vorgängen. Die Statistiken Minimum, Maximum und Durchschnitt geben die Byte in einem einzelnen PUT-Vorgang für den Stream im angegebenen Zeitraum an. Metrikname auf Shard-Ebene: `IncomingBytes` Abmessungen: StreamName Gültige Statistiken: Minimum, Maximum, Durchschnitt, Summe, Stichproben Einheiten: Byte
`IncomingRecords`	Anzahl der Datensätze, die im angegebenen Zeitraum erfolgreich an den Kinesis-Stream übertragen wurden. Diese Metrik enthält die Anzahlen der Datensätze von `PutRecord` - und `PutRecords` -Vorgängen. Die Statistiken Minimum, Maximum und Durchschnitt geben die Datensätze in einem einzelnen PUT-Vorgang für den Stream im angegebenen Zeitraum an. Metrikname auf Shard-Ebene: `IncomingRecords` Abmessungen: StreamName Gültige Statistiken: Minimum, Maximum, Durchschnitt, Summe, Stichproben Einheiten: Anzahl

Metrik	Beschreibung
PutRecord.Bytes	Anzahl der Bytes, die im angegebenen Zeitraum mithilfe des Vorgangs PutRecord an den Kinesis-Stream übertragen wurden. Abmessungen: StreamName Gültige Statistiken: Minimum, Maximum, Durchschnitt, Summe, Stichproben Einheiten: Byte
PutRecord.Latency	Die Dauer pro PutRecord -Vorgang, gemessen im angegebenen Zeitraum. Abmessungen: StreamName Statistiken: Minimum, Maximum, Durchschnitt Einheiten: Millisekunden
PutRecord.Success	Anzahl der erfolgreichen PutRecord -Vorgänge pro Kinesis-Stream im angegebenen Zeitraum. Der Durchschnitt gibt den Prozentsatz der erfolgreichen Schreibvorgänge in einem Stream an. Abmessungen: StreamName Gültige Statistiken: Durchschnitt, Summe, Stichproben Einheiten: Anzahl

Metrik	Beschreibung
PutRecords.Bytes	Anzahl der Bytes, die im angegebenen Zeitraum mithilfe des Vorgangs PutRecords an den Kinesis-Stream übertragen wurden. Abmessungen: StreamName Gültige Statistiken: Minimum, Maximum, Durchschnitt, Summe, Stichproben Einheiten: Byte
PutRecords.Latency	Die Dauer pro PutRecords -Vorgang, gemessen im angegebenen Zeitraum. Abmessungen: StreamName Statistiken: Minimum, Maximum, Durchschnitt Einheiten: Millisekunden
PutRecords.Records	Diese Metrik ist veraltet. Verwenden Sie PutRecords.SuccessfulRecords . Abmessungen: StreamName Gültige Statistiken: Minimum, Maximum, Durchschnitt, Summe, Stichproben Einheiten: Anzahl
PutRecords.Success	Anzahl der PutRecords -Vorgänge pro Kinesis-Stream mit mindestens einem erfolgreichen Datensatz, gemessenen im angegebenen Zeitraum. Abmessungen: StreamName Gültige Statistiken: Durchschnitt, Summe, Stichproben Einheiten: Anzahl

Metrik	Beschreibung
PutRecords.TotalRecords	Die gesamte Anzahl der Datensätze in einem PutRecords -Vorgang pro Kinesis-Datenstrom, gemessenen im angegebenen Zeitraum. Abmessungen: StreamName Gültige Statistiken: Minimum, Maximum, Durchschnitt, Summe, Stichproben Einheiten: Anzahl
PutRecords.SuccessfulRecords	Anzahl der erfolgreichen Datensätze in einem PutRecords -Vorgang pro Kinesis-Datenstrom, gemessenen im angegebenen Zeitraum. Abmessungen: StreamName Gültige Statistiken: Minimum, Maximum, Durchschnitt, Summe, Stichproben Einheiten: Anzahl
PutRecords.FailedRecords	Anzahl der Datensätze, die aufgrund von internen Fehlern in einem PutRecords -Vorgang pro Kinesis-Datenstrom zurückgewiesen wurden, gemessen über den angegebenen Zeitraum. Gelegentliche interne Fehler sind zu erwarten und sollten erneut versucht werden. Abmessungen: StreamName Gültige Statistiken: Minimum, Maximum, Durchschnitt, Summe, Stichproben Einheiten: Anzahl

Metrik	Beschreibung
PutRecords.ThrottledRecords	Anzahl der Datensätze, die aufgrund von Drosselung in einem PutRecords -Vorgang pro Kinesis-Datenstrom zurückgewiesen wurden, gemessen über den angegebenen Zeitraum. Abmessungen: StreamName Gültige Statistiken: Minimum, Maximum, Durchschnitt, Summe, Stichproben Einheiten: Anzahl
ReadProvisionedThroughputExceeded	Anzahl von GetRecords -Aufrufen, die im angegebenen Zeitraum für den Stream gedrosselt wurden. Die am häufigsten verwendete Statistik für diese Metrik ist Durchschnitt. Wenn die Statistik Minimum den Wert 1 hat, wurden im angegebenen Zeitraum sämtliche Datensätze für den Stream gedrosselt. Wenn die Statistik Maximum den Wert 0 (null) hat, wurden im angegebenen Zeitraum keine Datensätze für den Stream gedrosselt. Metrikname auf Shard-Ebene: ReadProvisionedThroughputExceeded Abmessungen: StreamName Gültige Statistiken: Minimum, Maximum, Durchschnitt, Summe, Stichproben Einheiten: Anzahl

Metrik	Beschreibung
`SubscribeToShard.RateExceeded`	Diese Metrik wird ausgelöst, wenn ein neuer Abonnementversuch fehlschlägt, da bereits ein aktives Abonnement von demselben Verbraucher existiert, oder wenn Sie die Anzahl der Aufrufe pro Sekunde überschreiten, die für diese Operation zulässig ist. Abmessungen: StreamName, ConsumerName
`SubscribeToShard.Success`	Diese Metrik zeichnet auf, ob das SubscribeToShard Abonnement erfolgreich eingerichtet wurde. Das Abonnement besteht nur für maximal 5 Minuten. Aus diesem Grund wird diese Metrik mindestens einmal alle 5 Minuten ausgegeben. Abmessungen: StreamName, ConsumerName
`SubscribeToShardEvent.Bytes`	Anzahl der Bytes, die im angegebenen Zeitraum vom Shard empfangen wurden. Die Statistiken Minimum, Maximum und Durchschnitt geben die Bytes an, die in einem einzelnen Ereignis für den angegebenen Zeitraum veröffentlicht wurden. Metrikname auf Shard-Ebene: `OutgoingBytes` Abmessungen: StreamName, ConsumerName Gültige Statistiken: Minimum, Maximum, Durchschnitt, Summe, Stichproben Einheiten: Byte

Metrik	Beschreibung
`SubscribeToShardEvent.MillisBehindLatest`	Die Anzahl der Millisekunden, die die gelesenen Datensätze von der Spitze des Streams entfernt sind, gibt an, wie weit der Verbraucher hinter der aktuellen Uhrzeit zurückliegt. Abmessungen:, StreamName ConsumerName Gültige Statistiken: Minimum, Maximum, Durchschnitt, Stichproben Einheiten: Millisekunden
`SubscribeToShardEvent.Records`	Anzahl der Datensätze, die im angegebenen Zeitraum vom Shard empfangen wurden. Die Statistiken Minimum, Maximum und Durchschnitt geben die Datensätze in einem einzelnen Ereignis für den angegebenen Zeitraum an. Metrikname auf Shard-Ebene: `OutgoingRecords` Abmessungen: StreamName, ConsumerName Gültige Statistiken: Minimum, Maximum, Durchschnitt, Summe, Stichproben Einheiten: Anzahl
`SubscribeToShardEvent.Success`	Diese Metrik wird jedes Mal ausgegeben, wenn ein Ereignis erfolgreich veröffentlicht wird. Es wird nur ausgegeben, wenn ein aktives Abonnement besteht. Abmessungen: StreamName, ConsumerName Gültige Statistiken: Minimum, Maximum, Durchschnitt, Summe, Stichproben Einheiten: Anzahl

Metrik	Beschreibung
`WriteProvisionedThroughputExceeded`	Anzahl der Datensätze, die im angegebenen Zeitraum infolge der Drosselung für den Stream abgelehnt wurden. Diese Metrik enthält die Drosselung infolge von `PutRecord` - und `PutRecords` -Vorgängen. Die am häufigsten verwendete Statistik für diese Metrik ist Durchschnitt. Wenn die Statistik Minimum einen Wert ungleich null hat, wurden im angegebenen Zeitraum Datensätze für den Stream gedrosselt. Wenn die Statistik Maximum den Wert 0 (null) hat, wurden im angegebenen Zeitraum keine Datensätze für den Stream gedrosselt. Metrikname auf Shard-Ebene: `WriteProvisionedThroughputExceeded` Abmessungen: StreamName Gültige Statistiken: Minimum, Maximum, Durchschnitt, Summe, Stichproben Einheiten: Anzahl

Verbesserte Metriken auf Shard-Ebene

Der `AWS/Kinesis`-Namespace enthält die folgenden Metriken auf Shard-Ebene.

Kinesis sendet die folgenden Shard-Level-Metriken an CloudWatch jede Minute. Jede Metrikdimension erstellt eine CloudWatch Metrik und führt ungefähr `PutMetricData` API 43.200 Aufrufe pro Monat durch. Diese Metriken sind nicht standardmäßig aktiviert. Von Kinesis übermittelte erweiterte Metriken sind kostenpflichtig. Weitere Informationen finden Sie unter Amazon CloudWatch Pricing unter der Überschrift Amazon CloudWatch Custom Metrics. Die Kosten fallen pro Shard pro Metrik pro Monat an.

Metrik	Beschreibung
IncomingBytes	Anzahl der Byte, die im angegebenen Zeitraum erfolgreich an den Shard übertragen wurden. Diese Metrik enthält die Byte von PutRecord - und PutRecords -Vorgängen. Die Statistiken Minimum, Maximum und Durchschnitt geben die Byte in einem einzelnen PUT-Vorgang für den Shard im angegebenen Zeitraum an. Metrikname auf Stream-Ebene: IncomingBytes Abmessungen: StreamName, ShardId Gültige Statistiken: Minimum, Maximum, Durchschnitt, Summe, Stichproben Einheiten: Byte
IncomingRecords	Anzahl der Datensätze, die im angegebenen Zeitraum erfolgreich an den Shard übertragen wurden. Diese Metrik enthält die Anzahlen der Datensätze von PutRecord - und PutRecords -Vorgängen. Die Statistiken Minimum, Maximum und Durchschnitt geben die Datensätze in einem einzelnen PUT-Vorgang für den Shard im angegebenen Zeitraum an. Metrikname auf Stream-Ebene: IncomingRecords Abmessungen: StreamName, ShardId Gültige Statistiken: Minimum, Maximum, Durchschnitt, Summe, Stichproben Einheiten: Anzahl
IteratorAgeMilliseconds	Alter des letzten Datensatzes in allen GetRecords -Aufrufen an einen Shard, gemessen im angegebenen Zeitraum. Das Alter ist die Differenz zwischen der aktuellen Zeit und der Zeit, zu der der letzte Datensatz

Metrik	Beschreibung
	des `GetRecords` -Aufrufs in den Stream geschrieben wurde. Die Statistiken Minimum und Maximum können zum Nachverfolgen des Fortschritts von Kinesis-V erbraucheranwendungen verwendet werden. Wenn der Wert 0 (null) beträgt, sind die gelesenen Datensätze vollständig mit dem Stream aktualisiert. Metrikname auf Stream-Ebene: `GetRecords.IteratorAgeMilliseconds` Abmessungen: StreamName, ShardId Gültige Statistiken: Minimum, Maximum, Durchschnitt, Stichproben Einheiten: Millisekunden
`OutgoingBytes`	Anzahl der Byte, die im angegebenen Zeitraum vom Shard abgerufen wurden. Die Statistiken Minimum, Maximum und Durchschnitt geben die Bytes an, die für den Shard im angegebenen Zeitraum in einem einzelnen `GetRecords` -Vorgang zurückgegeben oder bei einem einzelnen `SubscribeToShard` -Ereignis veröffentlicht wurden. Metrikname auf Stream-Ebene: `GetRecords.Bytes` Abmessungen: StreamName, ShardId Gültige Statistiken: Minimum, Maximum, Durchschnitt, Summe, Stichproben Einheiten: Byte

Metrik	Beschreibung
`OutgoingRecords`	Anzahl der Datensätze, die im angegebenen Zeitraum vom Shard abgerufen wurden. Die Statistiken Minimum, Maximum und Durchschnitt geben die Datensätze an, die für den Shard im angegebenen Zeitraum in einem einzelnen `GetRecords` -Vorgang zurückgegeben oder bei einem einzelnen `SubscribeToShard` -Ereignis veröffentlicht wurden. Metrikname auf Stream-Ebene: `GetRecords.Records` Abmessungen: StreamName, ShardId Gültige Statistiken: Minimum, Maximum, Durchschnitt, Summe, Stichproben Einheiten: Anzahl

Metrik	Beschreibung
`ReadProvisionedThroughputExceeded`	Anzahl von `GetRecords` -Aufrufen, die im angegebenen Zeitraum für den Shard gedrosselt wurden. Diese Anzahl von Ausnahmen deckt alle Dimensionen mit folgenden Einschränkungen ab: 5 Lesevorgänge pro Shard pro Sekunde oder 2 MB pro Sekunde pro Shard. Die am häufigsten verwendete Statistik für diese Metrik ist Durchschnitt. Wenn die Statistik Minimum den Wert 1 hat, wurden im angegebenen Zeitraum sämtliche Datensätze für den Shard gedrosselt. Wenn die Statistik Maximum den Wert 0 (null) hat, wurden im angegebenen Zeitraum keine Datensätze für den Shard gedrosselt. Metrikname auf Stream-Ebene: `ReadProvisionedThroughputExceeded` Abmessungen: StreamName, ShardId Gültige Statistiken: Minimum, Maximum, Durchschnitt, Summe, Stichproben Einheiten: Anzahl

Metrik	Beschreibung
`WriteProvisionedThroughputExceeded`	Anzahl der Datensätze, die im angegebenen Zeitraum infolge der Drosselung für den Shard abgelehnt wurden. Diese Metrik enthält die Drosselung infolge von `PutRecord` - und `PutRecords` -Vorgängen und deckt alle Dimensionen mit folgenden Einschränkungen ab: 1 000 Lesevorgänge pro Shard pro Sekunde oder 1 MB pro Sekunde pro Shard. Die am häufigsten verwendete Statistik für diese Metrik ist Durchschnitt. Wenn die Statistik Minimum einen Wert ungleich null hat, wurden im angegebenen Zeitraum Datensätze für den Shard gedrosselt. Wenn die Statistik Maximum den Wert 0 (null) hat, wurden im angegebenen Zeitraum keine Datensätze für den Shard gedrosselt. Metrikname auf Stream-Ebene: `WriteProvisionedThroughputExceeded` Abmessungen: StreamName, ShardId Gültige Statistiken: Minimum, Maximum, Durchschnitt, Summe, Stichproben Einheiten: Anzahl

Dimensionen für Amazon Kinesis Data Streams Streams-Metriken

Dimension	Beschreibung
`StreamName`	Der Name des Kinesis-Streams. Alle verfügbaren Statistiken werden von `StreamName` gefiltert.

Empfohlene Amazon Kinesis Data Streams Streams-Metriken

Verschiedene Metriken für Amazon Kinesis Data Streams könnten für Kunden von Kinesis Data Streams von besonderem Interesse sein. Die folgende Liste enthält empfohlene Metriken und ihre Verwendung.

Metrik	Nutzungshinweise
`GetRecords.IteratorAgeMilliseconds`	Verfolgt die Leseposition über alle Shards und Kunden in dem Stream. Wenn das Alter eines Iterators 50 % des Aufbewahrungszeitraums (standardmäßig 24 Stunden, anpassbar auf bis zu 7 Tage) überschreitet, besteht die Gefahr des Datenverlusts durch Ablauf des Datensatzes. Wir empfehlen Ihnen, CloudWatch Alarme für die Maximum-Statistik zu verwenden, um Sie zu warnen, bevor dieser Verlust ein Risiko darstellt. Ein Beispielszenario mit dieser Metrik finden Sie unter Die Verarbeitung von Verbraucherdaten hinkt hinterher.
`ReadProvisionedThroughputExceeded`	Wenn Ihre konsumentenseitige Datensatzverarbeitung zurückfällt, ist es oft schwierig zu erkennen, wo der Engpass ist. Verwenden Sie diese Metrik, um zu bestimmen, ob Ihre Lesevorgänge aufgrund einer Überschreitung Ihrer Durchsatzgrenzwerte gedrosselt werden. Die am häufigsten verwendete Statistik für diese Metrik ist Durchschnitt.
`WriteProvisionedThroughputExceeded`	Diese dient dem gleichen Zweck wie die Metrik `ReadProvisionedThroughputExceeded` , jedoch für die Producer- (Put) Seite des Streams. Die am häufigsten verwendete Statistik für diese Metrik ist Durchschnitt.
`PutRecord.Success` , `PutRecords.Success`	Wir empfehlen die Verwendung von CloudWatch Alarmen in der Statistik „Durchschnitt“, um anzuzeigen, wenn Datensätze nicht in den Stream aufgenommen werden. Wählen Sie einen oder beide Put-Typen, je nachdem, was Ihr Producer verwendet. Wenn Sie die Kinesis Producer Library (KPL) verwenden, verwenden Sie`PutRecords.Success` .
`GetRecords.Success`	Wir empfehlen die Verwendung von CloudWatch Alarmen in der Statistik „Durchschnitt“, um anzuzeigen, wenn Datensätze aus dem Stream fehlerhaft sind.

Greifen Sie auf CloudWatch Amazon-Metriken für Kinesis Data Streams zu

Sie können Metriken für Kinesis Data Streams mithilfe der CloudWatch Konsole, der Befehlszeile oder der CloudWatch API überwachen. Die folgenden Verfahren zeigen, wie Sie mithilfe dieser verschiedenen Verfahren auf die Metriken zugreifen können.

So greifen Sie über die Konsole auf Metriken zu CloudWatch

1. Öffnen Sie die CloudWatch Konsole unter https://console.aws.amazon.com/cloudwatch/.
2. Wählen Sie in der Navigationsleiste eine Region aus.
3. Wählen Sie im Navigationsbereich Metriken aus.
4. Wählen Sie im Bereich CloudWatch Metriken nach Kategorie die Option Kinesis Metrics aus.
5. Klicken Sie auf die entsprechende Zeile, um die Statistiken für das angegebene MetricNameund StreamNameanzuzeigen.

   Hinweis: Die meisten Namen der Konsolenstatistiken stimmen mit den entsprechenden oben aufgeführten CloudWatch Metriknamen überein, mit Ausnahme von Lesedurchsatz und Schreibdurchsatz. Diese Statistiken werden in Intervallen von 5 Minuten berechnet: Der Schreibdurchsatz überwacht die `IncomingBytes` CloudWatch Metrik und der Lesedurchsatz überwacht `GetRecords.Bytes` die Metrik.
6. (Optional) Wählen Sie im Diagrammbereich eine Statistik und einen Zeitraum aus, und erstellen Sie dann mit diesen Einstellungen einen CloudWatch Alarm.

Um auf Metriken zuzugreifen, verwenden Sie AWS CLI

Verwenden Sie die Listen-Metriken und get-metric-statisticsBefehle.

Um auf Metriken zuzugreifen, verwenden Sie CloudWatch CLI

Verwenden Sie die mon-get-statsBefehle mon-list-metricsund.

Um mit dem auf Metriken zuzugreifen CloudWatch API

Verwenden Sie die GetMetricStatisticsOperationen ListMetricsund.

Überwachen Sie den Zustand des Kinesis Data Streams Streams-Agenten mit Amazon CloudWatch

Der Agent veröffentlicht benutzerdefinierte CloudWatch Metriken mit dem Namespace. AWS KinesisAgent Anhand dieser Metriken können Sie beurteilen, ob der Agent Daten wie angegeben an Kinesis Data Streams sendet und ob sie fehlerfrei sind und die entsprechende Menge CPU an Speicherressourcen auf dem Datenproduzenten verbrauchen. Metriken wie die Anzahl der gesendeten Datensätze und Bytes sind nützlich, um zu verstehen, wie der Agent Daten an den Stream; übergibt. Wenn diese Metriken um einige Prozent unter die erwarteten Schwellenwerte oder auf Null sinken, kann dies auf Probleme mit der Konfiguration, Netzwerkfehler oder Probleme mit dem Zustand des Agenten hinweisen. Metriken wie Serververbrauch CPU und Speicherverbrauch sowie Agenten-Fehlerzähler geben Aufschluss über die Ressourcennutzung des Datenproduzenten und geben Aufschluss über mögliche Konfigurations- oder Hostfehler. Schließlich protokolliert der Agent auf Serviceausnahmen, um die Untersuchung von Agentenproblemen zu unterstützen. Diese Kennzahlen werden in der Region gemeldet, die in der Agentenkonfigurationseinstellung `cloudwatch.endpoint` angegeben ist. Cloudwatch-Metriken, die von mehreren Kinesis-Agenten veröffentlicht wurden, werden aggregiert oder kombiniert. Weitere Informationen zur Agentenkonfiguration finden Sie unter Geben Sie die Einstellungen für die Agentenkonfiguration an

Überwachen Sie mit CloudWatch

Der Kinesis Data Streams Streams-Agent sendet die folgenden Metriken an CloudWatch.

Metrik	Beschreibung
`BytesSent`	Die Anzahl von Byte, die über den angegebenen Zeitraum an Kinesis Data Streams gesendet wurden. Einheiten: Byte
`RecordSendAttempts`	Die Anzahl der versuchten Datensätze (zum ersten Mal oder wiederholt) in einem Aufruf an `PutRecords` in dem angegebenen Zeitraum. Einheiten: Anzahl
`RecordSendErrors`	Die Anzahl der Datensätze mit Fehlerstatus in einem Aufruf an `PutRecords` , einschließlich wiederholter Versuche, in dem angegebenen Zeitraum.

Metrik	Beschreibung
	Einheiten: Anzahl
`ServiceErrors`	Die Anzahl der Aufrufe an `PutRecords` , die zu einem Servicefehler führten (außer Ablehnungsfehlern) in dem angegebenen Zeitraum. Einheiten: Anzahl

Amazon Kinesis Data Streams API Streams-Aufrufe protokollieren mit AWS CloudTrail

Amazon Kinesis Data Streams ist in einen Service integriert AWS CloudTrail, der eine Aufzeichnung der Aktionen bereitstellt, die von einem Benutzer, einer Rolle oder einem AWS Service in Kinesis Data Streams ausgeführt wurden. CloudTrail erfasst alle API Aufrufe von Kinesis Data Streams als Ereignisse. Zu den erfassten Aufrufen gehören Aufrufe von der Kinesis Data Streams Streams-Konsole und Code-Aufrufe der Kinesis Data Streams API Streams-Operationen. Wenn Sie einen Trail erstellen, können Sie die kontinuierliche Bereitstellung von CloudTrail Ereignissen an einen Amazon S3 S3-Bucket aktivieren, einschließlich Ereignissen für Kinesis Data Streams. Wenn Sie keinen Trail konfigurieren, können Sie die neuesten Ereignisse trotzdem in der CloudTrail Konsole im Ereignisverlauf anzeigen. Anhand der von gesammelten Informationen können Sie die Anfrage CloudTrail, die an Kinesis Data Streams gestellt wurde, die IP-Adresse, von der aus die Anfrage gestellt wurde, wer die Anfrage gestellt hat, wann sie gestellt wurde, und weitere Details ermitteln.

Weitere Informationen darüber CloudTrail, einschließlich der Konfiguration und Aktivierung, finden Sie im AWS CloudTrail Benutzerhandbuch.

Kinesis Data Streams Streams-Informationen in CloudTrail

CloudTrail ist für Ihr AWS Konto aktiviert, wenn Sie das Konto erstellen. Wenn unterstützte Ereignisaktivitäten in Kinesis Data Streams auftreten, wird diese Aktivität zusammen mit anderen AWS Serviceereignissen im CloudTrail Ereignisverlauf in einem Ereignis aufgezeichnet. Sie können aktuelle Ereignisse in Ihrem AWS Konto anzeigen, suchen und herunterladen. Weitere Informationen finden Sie unter Ereignisse mit CloudTrail Ereignisverlauf anzeigen.

Für eine fortlaufende Aufzeichnung von Ereignissen in Ihrem AWS Konto, einschließlich Ereignissen für Kinesis Data Streams, erstellen Sie einen Trail. Ein Trail ermöglicht CloudTrail die Übermittlung von Protokolldateien an einen Amazon S3 S3-Bucket. Wenn Sie einen Trail in der Konsole erstellen,

gilt der Trail standardmäßig für alle AWS Regionen. Der Trail protokolliert Ereignisse aus allen Regionen der AWS Partition und übermittelt die Protokolldateien an den von Ihnen angegebenen Amazon S3 S3-Bucket. Darüber hinaus können Sie andere AWS Dienste konfigurieren, um die in den CloudTrail Protokollen gesammelten Ereignisdaten weiter zu analysieren und darauf zu reagieren. Weitere Informationen finden Sie hier:

- Übersicht zum Erstellen eines Trails
- CloudTrail Unterstützte Dienste und Integrationen
- Konfiguration von SNS Amazon-Benachrichtigungen für CloudTrail
- Empfangen von CloudTrail Protokolldateien aus mehreren Regionen und Empfangen von CloudTrail Protokolldateien von mehreren Konten

Kinesis Data Streams unterstützt die Protokollierung der folgenden Aktionen als Ereignisse in CloudTrail Protokolldateien:

- 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

Jeder Ereignis- oder Protokolleintrag enthält Informationen zu dem Benutzer, der die Anforderung generiert hat. Die Identitätsinformationen unterstützen Sie bei der Ermittlung der folgenden Punkte:

- Ob die Anfrage mit Root- oder AWS Identity and Access Management (IAM) Benutzeranmeldedaten gestellt wurde.
- Gibt an, ob die Anforderung mit temporären Sicherheitsanmeldeinformationen für eine Rolle oder einen Verbundbenutzer gesendet wurde.
- Ob die Anfrage von einem anderen AWS Dienst gestellt wurde.

Weitere Informationen finden Sie im CloudTrail userIdentity Element.

Beispiel: Einträge in der Kinesis Data Streams Streams-Protokolldatei

Ein Trail ist eine Konfiguration, die die Übertragung von Ereignissen als Protokolldateien an einen von Ihnen angegebenen Amazon S3 S3-Bucket ermöglicht. CloudTrail Protokolldateien enthalten einen oder mehrere Protokolleinträge. Ein Ereignis stellt eine einzelne Anforderung aus einer beliebigen Quelle dar und enthält unter anderem Informationen über die angeforderte Aktion, das Datum und die Uhrzeit der Aktion sowie über die Anforderungsparameter. CloudTrail Protokolldateien sind kein geordneter Stack-Trace der öffentlichen API Aufrufe, sie erscheinen also nicht in einer bestimmten Reihenfolge.

Das folgende Beispiel zeigt einen CloudTrail Protokolleintrag, der die `MergeShards` Aktionen`CreateStream`,`DescribeStream`,`ListStreams`, `DeleteStreamSplitShard`, und demonstriert.

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


Überwachen Sie die Kinesis-Clientbibliothek mit Amazon CloudWatch

Die Kinesis Client Library (KCL) für Amazon Kinesis Data Streams veröffentlicht in Ihrem Namen benutzerdefinierte CloudWatch Amazon-Metriken, wobei der Name Ihrer KCL Anwendung als Namespace verwendet wird. Sie können diese Metriken anzeigen, indem Sie zur CloudWatch Konsole navigieren und Benutzerdefinierte Metriken auswählen. Weitere Informationen zu benutzerdefinierten Metriken finden Sie unter Veröffentlichen benutzerdefinierter Metriken im CloudWatch Amazon-Benutzerhandbuch.

Für die CloudWatch von der hochgeladenen Metriken fällt eine geringe Gebühr an. Insbesondere KCL fallen Gebühren für Amazon CloudWatch Custom Metrics und Amazon CloudWatch API Requests an. Weitere Informationen finden Sie unter CloudWatch Amazon-Preise.

Themen

- Metriken und Namespace
- Metrische Ebenen und Dimensionen
- Metrische Konfiguration
- Liste der Metriken

Metriken und Namespace

Der Namespace, der zum Hochladen von Metriken verwendet wird, ist der Anwendungsname, den Sie beim Starten von angeben. KCL

Metrische Ebenen und Dimensionen

Es gibt zwei Optionen, um zu steuern, in welche Metriken hochgeladen werden CloudWatch:

Metrikstufen

Jeder Metrik ist eine individuelle Stufe zugewiesen. Wenn Sie eine Berichtsebene für Kennzahlen festlegen, werden Metriken, deren individuelle Ebene unter der Berichtsebene liegt, nicht an diese gesendet CloudWatch. Die Stufen sind: `NONE`, `SUMMARY` und `DETAILED`. Die Standardeinstellung ist`DETAILED`; das heißt, alle Metriken werden an gesendet CloudWatch. Die Berichtsstufe `NONE` bedeutet, dass keine Kennzahlen gesendet werden. Informationen dazu, welchen Metriken welche Stufen zugewiesen werden, finden Sie unter Liste der Metriken.

aktivierte Dimensionen

Jeder KCL Metrik sind Dimensionen zugeordnet, an die auch gesendet werden CloudWatch. Wenn in KCL 2.x für die Verarbeitung eines einzelnen Datenstroms konfiguriert KCL ist, sind alle Metrikdimensionen (`OperationShardId`, und`WorkerIdentifier`) standardmäßig aktiviert. Außerdem kann die `Operation` Dimension in KCL 2.x nicht deaktiviert werden, wenn sie für die Verarbeitung eines einzelnen Datenstroms konfiguriert KCL ist. In KCL 2.x sind alle Metrikdimensionen (,, und`WorkerIdentifier`) standardmäßig aktiviert `Operation` `ShardIdStreamId`, wenn KCL sie für die Verarbeitung mehrerer Datenströme konfiguriert ist. Außerdem können die Dimensionen und die `StreamId` Dimensionen in KCL 2.x nicht deaktiviert werden, wenn sie für die `Operation` Verarbeitung mehrerer Datenströme konfiguriert KCL ist. `StreamId`Die Dimension ist nur für die Metriken pro Shard verfügbar.

In KCL 1.x sind standardmäßig nur die `ShardId` Dimensionen `Operation` und die Dimension aktiviert, und die `WorkerIdentifier` Dimension ist deaktiviert. In KCL 1.x kann die `Operation` Dimension nicht deaktiviert werden.

Weitere Informationen zu CloudWatch metrischen Dimensionen finden Sie im Abschnitt Dimensionen des Themas Amazon CloudWatch Concepts im CloudWatch Amazon-Benutzerhandbuch.

Wenn die `WorkerIdentifier` Dimension aktiviert ist und bei jedem Neustart eines bestimmten Workers ein anderer Wert für die KCL Worker-ID-Eigenschaft verwendet wird, werden neue Messwerte mit neuen `WorkerIdentifier` Dimensionswerten an CloudWatch gesendet. Wenn Sie möchten, dass der `WorkerIdentifier` Dimensionswert bei bestimmten KCL Worker-Neustarts identisch ist, müssen Sie bei der Initialisierung für jeden Worker explizit denselben

Worker-ID-Wert angeben. Beachten Sie, dass der Worker-ID-Wert für jeden aktiven KCL Worker für alle KCL Worker eindeutig sein muss.

Metrische Konfiguration

Metrikebenen und aktivierte Dimensionen können mithilfe der KinesisClientLibConfiguration Instanz konfiguriert werden, die beim Starten der KCL Anwendung an Worker übergeben wird. In MultiLangDaemon diesem Fall können die `metricsEnabledDimensions` Eigenschaften `metricsLevel` und in der Datei .properties angegeben werden, die zum Starten der MultiLangDaemon KCL Anwendung verwendet wird.

Metrischen Ebenen kann einer von drei Werten zugewiesen werden: NONESUMMARY, oderDETAILED. Bei aktivierten Dimensionswerten muss es sich um kommagetrennte Zeichenketten mit der Liste der Dimensionen handeln, die für die CloudWatch Metriken zulässig sind. Die von der KCL Anwendung verwendeten Dimensionen sind `OperationShardId`, und. `WorkerIdentifier`

Liste der Metriken

In den folgenden Tabellen sind die KCL Metriken nach Umfang und Funktionsweise gruppiert aufgeführt.

Themen

- Metriken KCL pro Anwendung
- Kennzahlen pro Mitarbeiter
- Metriken pro Shard

Metriken KCL pro Anwendung

Diese Metriken werden für alle KCL Mitarbeiter innerhalb des Anwendungsbereichs aggregiert, wie im CloudWatch Amazon-Namespace definiert.

Themen

- InitializeTask
- ShutdownTask
- ShardSyncTask
- BlockOnParentTask

- PeriodicShardSyncManager
- MultistreamTracker

InitializeTask

Der `InitializeTask` Vorgang ist für die Initialisierung des Datensatzprozessors für die Anwendung verantwortlich. KCL Zur Logik dieser Operation gehört der Abruf eines Shard-Iterators von Kinesis Data Streams und die Initialisierung des Datensatzprozessors.

Metrik	Beschreibung
KinesisDataFetcher. getIterator. Erfolg	Anzahl erfolgreicher `GetShardIterator` Operationen pro KCL Anwendung. Metrikstufe: Detailed Einheiten: Anzahl
KinesisDataFetcher. getIterator. Zeit	Zeit, die pro `GetShardIterator` Vorgang für die jeweilige KCL Anwendung benötigt wird. Metrikstufe: Detailed Einheiten: Millisekunden
RecordProcessor.in itialize.TIME	Zeitbedarf für die Initialisierungsmethode des Datensatzprozessors. Metrikstufe: Summary Einheiten: Millisekunden
Herzlichen Glückwuns ch	Anzahl der erfolgreichen Initialisierungen des Datensatzprozessors. Metrikstufe: Summary Einheiten: Anzahl
Zeit	Zeit, die der KCL Worker für die Initialisierung des Datensatzprozessors benötigt hat. Metrikstufe: Summary

Metrik	Beschreibung
	Einheiten: Millisekunden

ShutdownTask

Die - `ShutdownTask` Operation initiiert die Abschaltsequenz für die Shard-Verarbeitung. Dies kann auftreten, da ein Shard geteilt oder zusammengeführt ist, oder wenn das Shard-Lease vom Worker verloren ging. In beiden Fällen wird die Datensatzprozessorfunktion `shutdown()` aufgerufen. Neue Shards werden auch erkannt, wenn ein Shard geteilt oder zusammengeführt wurde, was zur Erstellung von einem oder zwei neuen Shards führt.

Metrik	Beschreibung
CreateLease. Erfolg	Häufigkeit, mit der neue untergeordnete Shards nach dem Herunterf ahren der übergeordneten Shard erfolgreich zur DynamoDB-Tabelle der KCL Anwendung hinzugefügt wurden. Metrikstufe: Detailed Einheiten: Anzahl
CreateLease.Zeit	Zeitaufwand für das Hinzufügen neuer untergeordneter Shard-Inf ormationen zur KCL DynamoDB-Tabelle der Anwendung. Metrikstufe: Detailed Einheiten: Millisekunden
UpdateLease. Erfolg	Anzahl der erfolgreichen endgültigen Checkpoints während des Schließens des Datensatzprozessors. Metrikstufe: Detailed Einheiten: Anzahl
UpdateLease.Zeit	Zeitbedarf für die Checkpoint-Operation während des Schließens des Datensatzprozessors. Metrikstufe: Detailed

Metrik	Beschreibung
	Einheiten: Millisekunden
RecordProcessor.Herunterfahren.Zeit	Zeitbedarf für die Schließungsmethode des Datensatzprozessors. Metrikstufe: Summary Einheiten: Millisekunden
Herzlichen Glückwunsch	Anzahl der erfolgreichen Schließungsvorgänge. Metrikstufe: Summary Einheiten: Anzahl
Zeit	Zeit, die der KCL Worker für die Shutdown-Aufgabe benötigt hat. Metrikstufe: Summary Einheiten: Millisekunden

ShardSyncTask

Bei `ShardSyncTask` diesem Vorgang werden Änderungen an den Shard-Informationen für den Kinesis-Datenstrom erkannt, sodass neue Shards von der Anwendung verarbeitet werden können. KCL

Metrik	Beschreibung
CreateLease. Erfolg	Anzahl der erfolgreichen Versuche, der DynamoDB-Tabelle der KCL Anwendung neue Shard-Informationen hinzuzufügen. Metrikstufe: Detailed Einheiten: Anzahl
CreateLease. Zeit	Zeit, die für das Hinzufügen neuer Shard-Informationen zur DynamoDB-Tabelle der KCL Anwendung benötigt wurde. Metrikstufe: Detailed

Metrik	Beschreibung
	Einheiten: Millisekunden
Herzlichen Glückwunsch	Anzahl der erfolgreichen Shard Synchronisierungsoperationen. Metrikstufe: Summary Einheiten: Anzahl
Zeit	Zeitbedarf für die Shard-Synchronisierungsoperation. Metrikstufe: Summary Einheiten: Millisekunden

BlockOnParentTask

Wenn der Shard geteilt oder mit anderen Shards zusammengeführt wird, werden neue untergeordnete Shards erstellt. Durch diesen `BlockOnParentTask` Vorgang wird sichergestellt, dass die Datensatzverarbeitung für die neuen Shards erst beginnt, wenn die übergeordneten Shards vollständig von der verarbeitet wurden. KCL

Metrik	Beschreibung
Herzlichen Glückwunsch	Anzahl der erfolgreichen Prüfungen für den Abschluss der übergeordneten Shards. Metrikstufe: Summary Einheiten: Anzahl
Zeit	Zeitbedarf für den Abschluss der übergeordneten Shards. Metrikstufe: Summary Einheit: Millisekunden

PeriodicShardSyncManager

Der `PeriodicShardSyncManager` ist dafür verantwortlich, die Datenströme zu untersuchen, die von der KCL Verbraucheranwendung verarbeitet werden, Datenströme zu identifizieren, die teilweise ausgeliehen wurden, und sie zur Synchronisation weiterzuleiten.

Die folgenden Metriken sind verfügbar, wenn sie für die Verarbeitung eines einzelnen Datenstroms konfiguriert KCL NumStreamsWithPartialLeases ist (dann wird der Wert von NumStreamsToSync und auf 1 gesetzt) und auch, wenn sie für die Verarbeitung mehrerer Datenströme konfiguriert KCL ist.

Metrik	Beschreibung
NumStreamsToSync	Die Anzahl der Datenströme (pro AWS Konto), die von der Verbrauch eranwendung verarbeitet werden und teilweise Leases enthalten und zur Synchronisation weitergeleitet werden müssen. Metrikstufe: Summary Einheiten: Anzahl
NumStreamsWithPart ialLeases	Die Anzahl der Datenströme (pro AWS Konto), die die Verbrauch eranwendung verarbeitet und die Teilleasingverträge enthalten. Metrikstufe: Summary Einheiten: Anzahl
Herzlichen Glückwuns ch	Die Anzahl der Male, in denen `PeriodicShardSyncManager` erfolgreich Teil-Leases in den Datenströmen identifizieren konnte, die die Verbraucheranwendung verarbeitet. Metrikstufe: Summary Einheiten: Anzahl
Zeit	Die Zeit (in Millisekunden), die `PeriodicShardSyncManager` benötigt wird, um die Datenströme zu untersuchen, die von der Verbraucheranwendung verarbeitet werden, um festzustellen, welche Datenströme eine Shard-Synchronisierung erfordern. Metrikstufe: Summary

Metrik	Beschreibung
	Einheiten: Millisekunden

MultistreamTracker

Die `MultistreamTracker` Schnittstelle ermöglicht es Ihnen, KCL Verbraucheranwendungen zu erstellen, die mehrere Datenströme gleichzeitig verarbeiten können.

Metrik	Beschreibung
DeletedStreams. Zählen	Die Anzahl der in diesem Zeitraum gelöschten Datenströme. Metrikstufe: Summary Einheiten: Anzahl
ActiveStreams. Zählen	Die Anzahl der aktiven Datenströme, die verarbeitet werden. Metrikstufe: Summary Einheiten: Anzahl
StreamsPendingDeletion. Zählen	Die Anzahl der Datenströme, die aufgrund von `FormerStreamsLeasesDeletionStrategy` zur Löschung anstehen. Metrikstufe: Summary Einheiten: Anzahl

Kennzahlen pro Mitarbeiter

Diese Metriken werden für alle Datensatzprozessoren aggregiert, die Daten aus einem Kinesis-Datenstream, z. B. einer EC2 Amazon-Instance, verbrauchen.

Themen

- RenewAllLeases
- TakeLeases

RenewAllLeases

Die Operation `RenewAllLeases` erneuert periodisch alle Shard-Leases, die zu einer bestimmten Worker-Instance gehören.

Metrik	Beschreibung
RenewLease. Erfolg	Anzahl der erfolgreichen Lease-Erneuerungen durch den Worker. Metrikstufe: Detailed Einheiten: Anzahl
RenewLease.Zeit	Zeitbedarf für die Operation der Lease-Erneuerung. Metrikstufe: Detailed Einheiten: Millisekunden
CurrentLeases	Anzahl der Shard-Leases, die einem Worker nach der Erneuerung aller Leases gehören. Metrikstufe: Summary Einheiten: Anzahl
LostLeases	Anzahl der Shard-Leases, die nach einem Versuch zur Erneuerung aller Leases eines Workers verloren gingen. Metrikstufe: Summary Einheiten: Anzahl
Herzlichen Glückwunsch	Häufigkeit erfolgreicher Lease-Erneuerungsoperationen für den Worker. Metrikstufe: Summary Einheiten: Anzahl
Zeit	Zeitbedarf für die Erneuerung aller Leases für den Worker. Metrikstufe: Summary

Metrik	Beschreibung
	Einheiten: Millisekunden

TakeLeases

Bei diesem `TakeLeases` Vorgang wird die Verarbeitung von Datensätzen zwischen allen KCL Mitarbeitern ausgeglichen. Wenn der aktuelle KCL Mitarbeiter weniger Shard-Leasingverträge als erforderlich hat, nimmt er Shard-Leasingverträge von einem anderen Mitarbeiter an, der überlastet ist.

Metrik	Beschreibung
ListLeases. Erfolg	Häufigkeit, mit der alle Shard-Leases erfolgreich aus der DynamoDB-Tabelle der KCL Anwendung abgerufen wurden. Metrikstufe: Detailed Einheiten: Anzahl
ListLeases. Zeit	Zeit, die benötigt wurde, um alle Shard-Leases aus der DynamoDB-Tabelle der KCL Anwendung abzurufen. Metrikstufe: Detailed Einheiten: Millisekunden
TakeLease. Erfolg	Häufigkeit, mit der der Arbeiter erfolgreich Shard-Leasingverträge von anderen Mitarbeitern angenommen hat. KCL Metrikstufe: Detailed Einheiten: Anzahl
TakeLease. Zeit	Zeitbedarf für die Aktualisierung der Lease-.Tabelle mit Leases des Workers. Metrikstufe: Detailed Einheiten: Millisekunden
NumWorkers	Gesamtzahl der Worker, wie von einem spezifischen Worker definiert.

Metrik	Beschreibung
	Metrikstufe: Summary Einheiten: Anzahl
NeededLeases	Anzahl der Shard-Leases, die der aktuelle Worker für eine ausgeglichene Shard-Verarbeitungslast benötigt. Metrikstufe: Detailed Einheiten: Anzahl
LeasesToTake	Anzahl der Leases, die der Worker zu übernehmen versuchen wird. Metrikstufe: Detailed Einheiten: Anzahl
TakenLeases	Anzahl der von dem Worker erfolgreich übernommenen Leases. Metrikstufe: Summary Einheiten: Anzahl
TotalLeases	Gesamtzahl der Shards, die die KCL Anwendung verarbeitet. Metrikstufe: Detailed Einheiten: Anzahl
ExpiredLeases	Gesamtzahl der Shards, die nicht von einem Worker verarbeitet werden, wie von einem spezifischen Worker identifiziert. Metrikstufe: Summary Einheiten: Anzahl
Herzlichen Glückwunsch	Häufigkeit des erfolgreichen Abschlusses der `TakeLeases` -Operation. Metrikstufe: Summary Einheiten: Anzahl

Metrik	Beschreibung
Zeit	Zeitbedarf für die `TakeLeases` -Operation für einen Worker. Metrikstufe: Summary Einheiten: Millisekunden

Metriken pro Shard

Diese Metriken werden über einen einzigen Datensatzprozessor aggregiert.

ProcessTask

Die `ProcessTask` Operation ruft GetRecordsmit der aktuellen Iteratorposition auf, um Datensätze aus dem Stream abzurufen, und ruft die Datensatzprozessorfunktion auf. `processRecords`

Metrik	Beschreibung
KinesisDataFetcher. getRecords. Erfolg	Anzahl der erfolgreichen `GetRecords` -Operationen pro Kinesis-D atenstrom-Shard. Metrikstufe: Detailed Einheiten: Anzahl
KinesisDataFetcher. getRecords. Zeit	Zeitbedarf für die `GetRecords` -Operation für den Kinesis-Datenstrom-Shard. Metrikstufe: Detailed Einheiten: Millisekunden
UpdateLease.Erfolg	Anzahl der erfolgreichen Checkpoints durch den Datensatzprozessor für einen bestimmten Shard. Metrikstufe: Detailed Einheiten: Anzahl
UpdateLease.Zeit	Zeitbedarf für jede Checkpoint-Operation eines bestimmten Shards.

Metrik	Beschreibung
	Metrikstufe: Detailed Einheiten: Millisekunden
DataBytesProcessed	Gesamtgröße der verarbeiteten Datensätze in Byte bei jedem `ProcessTask` -Aufruf. Metrikstufe: Summary Einheiten: Byte
RecordsProcessed	Anzahl der bei jedem `ProcessTask` -Aufruf verarbeiteten Datensätze. Metrikstufe: Summary Einheiten: Anzahl
ExpiredIterator	Nummer der beim Anruf ExpiredIteratorException `GetRecords` empfangenen Personen. Metrikstufe: Summary Einheiten: Anzahl
MillisBehindLatest	Zeitraum, um den der aktuelle Iterator seit dem letzten Datensatz (Tip) in dem Shard verspätet ist. Dieser Wert ist kleiner oder gleich der Zeitdiffe renz zwischen dem letzten Datensatz in einer Antwort und dem aktuellen Zeitpunkt. Dies ist eine genauere Angabe dafür, wie weit ein Shard vom Tip entfernt ist, als durch den Vergleich der Zeitstempel im letzten Antwortdatensatz möglich ist. Dieser Wert gilt für den letzten Datensatz stapel und ist kein Durchschnittswert aller Zeitstempel in den Datensätz en. Metrikstufe: Summary Einheiten: Millisekunden

Metrik	Beschreibung
RecordProcessor. processRecords. Zeit	Zeitbedarf für die `processRecords` -Methode des Datensatzprozessors. Metrikstufe: Summary Einheiten: Millisekunden
Herzlichen Glückwunsch	Anzahl der erfolgreichen Verarbeitungsaufgabenoperationen. Metrikstufe: Summary Einheiten: Anzahl
Zeit	Zeitbedarf für die Verarbeitungsaufgabenoperation. Metrikstufe: Summary Einheiten: Millisekunden

Überwachen Sie die Kinesis Producer Library mit Amazon CloudWatch

Die Kinesis Producer Library (KPL) für Amazon Kinesis Data Streams veröffentlicht in Ihrem Namen benutzerdefinierte CloudWatch Amazon-Metriken. Sie können sich diese Metriken ansehen, indem Sie zur CloudWatch Konsole navigieren und Benutzerdefinierte Metriken auswählen. Weitere Informationen zu benutzerdefinierten Metriken finden Sie unter Veröffentlichen benutzerdefinierter Metriken im CloudWatch Amazon-Benutzerhandbuch.

Für die CloudWatch von der hochgeladenen Metriken fällt eine geringe Gebühr an. Insbesondere KPL fallen Gebühren für Amazon CloudWatch Custom Metrics und Amazon CloudWatch API Requests an. Weitere Informationen finden Sie unter CloudWatch Amazon-Preise. Für die Erfassung lokaler Kennzahlen fallen keine CloudWatch Gebühren an.

Themen

- Metriken, Dimensionen und Namespaces
- Metrische Ebene und Granularität

- Lokaler Zugriff und CloudWatch Amazon-Upload
- Liste der Metriken

Metriken, Dimensionen und Namespaces

Sie können beim Starten von einen Anwendungsnamen angebenKPL, der dann beim Hochladen von Metriken als Teil des Namespaces verwendet wird. Dies ist optional; es KPL stellt einen Standardwert bereit, wenn kein Anwendungsname festgelegt ist.

Sie können das auch so konfigurierenKPL, dass den Metriken beliebige zusätzliche Dimensionen hinzugefügt werden. Dies ist nützlich, wenn Sie detailliertere Daten in Ihren Metriken haben möchten. CloudWatch Sie können beispielsweise den Host-Namen als Dimension hinzufügen, damit Sie ungleichmäßige Lastverteilungen in der Flotte erkennen. Alle KPL Konfigurationseinstellungen sind unveränderlich, sodass Sie diese zusätzlichen Dimensionen nach der Initialisierung der KPL Instanz nicht ändern können.

Metrische Ebene und Granularität

Es gibt zwei Optionen, um die Anzahl der hochgeladenen Metriken zu CloudWatch steuern:

Metrikstufe

Dies ist ein grobes Maß für die Wichtigkeit der Metrik. Jede Metrik wird eine Stufe zugewiesen. Wenn Sie eine Stufe festlegen, werden Metriken, deren Stufen darunter liegen, nicht an diese gesendet CloudWatch. Die Stufen sind `NONE`, `SUMMARY` und `DETAILED`. Die Standardeinstellung ist `DETAILED`; das heißt, alle Metriken. `NONE` bedeutet keine Metriken, dieser Stufe werden daher keine Metriken zugewiesen.

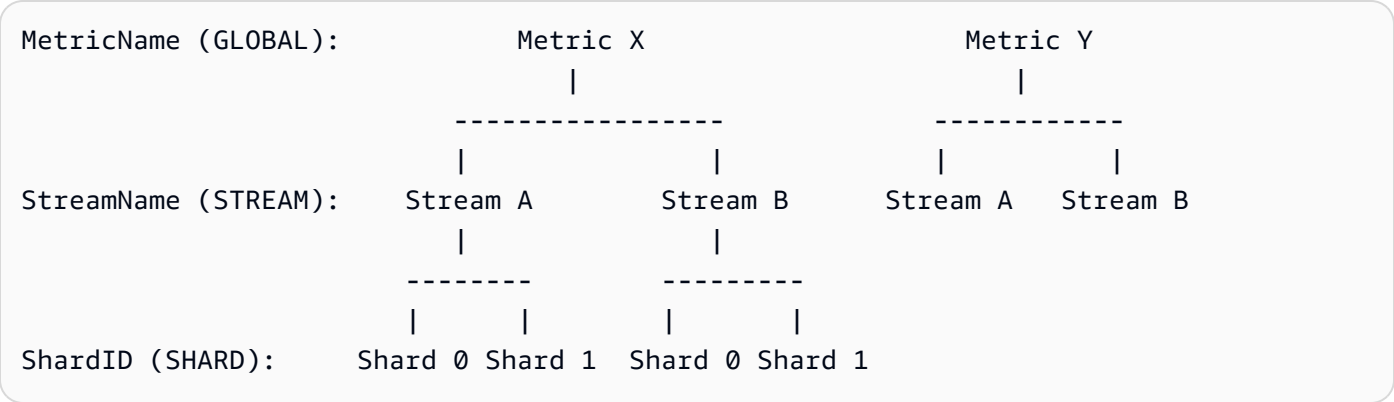
Granularität

Dies steuert, ob deine Metrik mit weiteren Granularitätsstufen ausgegeben werden soll. Die Stufen sind `GLOBAL`, `STREAM` und `SHARD`. Die Standardeinstellung ist `SHARD`mit den detailliertesten Metriken.

Bei Auswahl von `SHARD` werden Metriken mit dem Stream-Namen und der Shard-ID als Dimensionen ausgegeben. Dazu wird die gleiche Metrik auch nur mit der Stream-Name-Dimension und ohne die Stream-Name-Dimension ausgegeben. Das bedeutet, dass für eine bestimmte Metrik zwei Streams mit jeweils zwei Shards sieben CloudWatch Metriken erzeugen: eine für jeden Shard, eine für jeden Stream und eine insgesamt. Alle beschreiben dieselben

Statistiken, aber auf unterschiedlichen Granularitätsebenen. Eine Illustration finden Sie im folgenden Diagramm.

Die unterschiedlichen Granularitätsebenen bilden eine Hierarchie, und alle Metriken in dem System bilden einen Baum, dessen Wurzel die Metriknamen bilden:



Nicht alle Metriken sind auf Shard-Ebene verfügbar; einige sind ihrer Natur nach global oder auf die Stream-Ebene beschränkt. Sie werden nicht auf Shard-Ebene produziert, auch wenn Sie Kennzahlen auf Shard-Ebene aktiviert haben (`Metric Y` im Diagramm oben).

Wenn Sie eine zusätzliche Dimension angeben, müssen Sie Werte für `tuple:<DimensionName, DimensionValue, Granularity>` angeben. Die Granularität legt fest, wo die benutzerdefinierte Dimension in die Hierarchie eingefügt wird: `GLOBAL` bedeutet, dass die zusätzliche Dimension hinter dem Kennzahlnamen eingefügt wird, `STREAM` bedeutet, dass sie hinter dem Stream-Namen und `SHARD`, dass sie hinter der Shard-ID eingefügt wird. Wenn mehrere Dimensionen pro Granularitätsstufe angegeben sind, werden sie in der angegebenen Reihenfolge eingefügt.

Lokaler Zugriff und CloudWatch Amazon-Upload

Die Metriken für die aktuelle KPL Instanz sind lokal in Echtzeit verfügbar. Sie können sie KPL jederzeit abfragen, um sie abzurufen. Der berechnet KPL lokal die Summe, den Durchschnitt, das Minimum, das Maximum und die Anzahl aller Metriken, wie in CloudWatch.

Sie erhalten statistische Werte, die seit dem Start des Programms bis zum aktuellen Zeitpunkt kumuliert wurden, oder Sie können ein laufendes Zeitfenster über die letzten N Sekunden verwenden, wobei N eine ganze Zahl zwischen 1 und 60 ist.

Alle Metriken sind für den Upload verfügbar. CloudWatch Dies ist besonders nützlich für die Aggregierung von Daten über mehrere Hosts sowie Überwachungs- und Alarmvorgänge. Diese Funktionalität ist nicht lokal verfügbar.

Wie zuvor beschrieben, können Sie wählen, welche Metriken mit der Metrik-Ebene und Granularität hochgeladen werden sollen. Metriken, die nicht hochgeladen wurde, sind lokal verfügbar.

Das Hochladen einzelner Datenpunkte ist nicht sinnvoll, da dies zu Millionen von Uploads pro Sekunde führen könnte, wenn der Datenverkehr hoch ist. Aus diesem Grund werden Metriken KPL lokal in 1-Minuten-Buckets zusammengefasst und ein Statistikobjekt pro Minute pro CloudWatch aktivierter Metrik hochgeladen.

Liste der Metriken

Metrik	Beschreibung
`UserRecordsReceived`	Zählt, wie viele logische Benutzerdatensätze vom KPL Core für PUT-Operationen empfangen wurden. Auf Shard-Ebene nicht verfügbar. Metrikstufe: Detailed Einheit: Anzahl
`UserRecordsPending`	Regelmäßige Stichprobe, wie viele Benutzer Datensätze derzeit ausstehen. Ein Datensatz ist ausstehend, wenn er entweder derzeit gepuffert ist und darauf wartet, gesendet zu werden, oder wenn er direkt zu dem Backend-Service gesendet wird. Auf Shard-Ebene nicht verfügbar. Das KPL bietet eine spezielle Methode zum Abrufen dieser Metrik auf globaler Ebene, sodass Kunden ihre Put-Rate verwalten können. Metrikstufe: Detailed Einheit: Anzahl
`UserRecordsPut`	Anzahl der logischen Benutzerdatensätze mit erfolgreichen Put-Operationen.

Metrik	Beschreibung
	Fehlgeschlagene Datensätze für diese Metrik werden KPL nicht mitgezählt. Dadurch kann der Durchschnittswert die Erfolgsrate, die Anzahl die Gesamtzahl der Versuche und die Differenz zwischen beiden die Anzahl der Fehlschläge angeben. Metrikstufe: Summary Einheit: Anzahl
`UserRecordsDataPut`	Bytes in den logischen Benutzer-Datensätzen mit erfolgreichen Put-Operationen. Metrikstufe: Detailed Einheit: Byte
`KinesisRecordsPut`	Anzahl der Datensätze der Kinesis Data Streams mit erfolgreichen Put-Operationen (jeder Datensatz der Kinesis Data Streams kann mehrere Benutzerdatensätze enthalten). Das KPL gibt für fehlgeschlagene Datensätze eine Null aus. Dadurch kann der Durchschnittswert die Erfolgsrate, die Anzahl die Gesamtzahl der Versuche und die Differenz zwischen beiden die Anzahl der Fehlschläge angeben. Metrikstufe: Summary Einheit: Anzahl
`KinesisRecordsDataPut`	Bytes in den Datensätzen der Kinesis Data Streams. Metrikstufe: Detailed Einheit: Byte

Metrik	Beschreibung
`ErrorsByCode`	Anzahl der einzelnen Arten von Fehlercodes. Dies führt eine weitere Dimension von `ErrorCode` zusätzlich zu den normalen Dimension en ein, wie z. B. `StreamName` und `ShardId`. Nicht jeder Fehler kann zu einem Shard verfolgt werden. Die Fehler, die nicht verfolgt werden können, werden nur auf Stream- oder globaler Ebene ausgegebe n. Diese Metrik erfasst Informationen zu Dingen wie Drosselung, Änderungen der Shard-Zuordnung, internen Fehlern, nicht verfügbaren Services, Zeitüberschreitungen u. dgl. Kinesis Data Streams API Streams-Fehler werden einmal pro Kinesis Data Streams Streams-Datensatz gezählt. Mehrere Benutzerdatensätze innerhalb eines Datensatzes von Kinesis Data Streams erzeugen keine Mehrfachzählungen. Metrikstufe: Summary Einheit: Anzahl
`AllErrors`	Dies wird von den gleichen Fehlern ausgelöst wie Fehler nach Code, jedoch ohne Unterscheidung zwischen den Typen. Dies ist nützlich zur allgemeinen Überwachung der Fehlerrate, ohne dass eine manuelle Summe der Anzahlen der verschiedenen Arten von Fehlern erforderlich ist. Metrikstufe: Summary Einheit: Anzahl

Metrik	Beschreibung
`RetriesPerRecord`	Anzahl der pro Benutzerdatensatz durchgeführten erneuten Versuche. Für Datensätze, die bei einem Versuch erfolgreich waren, wird Null ausgegeben. Die Daten werden in dem Moment ausgegeben, in dem ein Benutzerdatensatz abgeschlossen wird (wenn er entweder erfolgreich war, oder wenn kein Wiederholungsversuch mehr möglich ist). Wenn der Datensatz einen großen Wert hat, kann time-to-live es bei dieser Metrik zu einer erheblichen Verzögerung kommen. Metrikstufe: Detailed Einheit: Anzahl
`BufferingTime`	Die Zeit zwischen dem Eintreffen eines Benutzerdatensatzes im Backend KPL und dem Abflug zum Backend. Diese Informationen werden an den Benutzer pro Datensatz zurückgegeben, sind jedoch auch als aggregierte Statistik verfügbar. Metrikstufe: Summary Einheit: Millisekunden
`Request Time`	Die Zeit zum Ausführen von `PutRecordsRequests` . Metrikstufe: Detailed Einheit: Millisekunden
`User Records per Kinesis Record`	Die Anzahl der logischen Benutzerdatensätze, die in einem einzigen Datensatz der Kinesis Data Streams zusammengefasst sind. Metrikstufe: Detailed Einheit: Anzahl

Metrik	Beschreibung
`Amazon Kinesis Records per PutRecords Request`	Die Anzahl der Datensätze von Kinesis Data Streams, die zu einem einzigen `PutRecordsRequest` aggregiert wurden. Auf Shard-Ebene nicht verfügbar. Metrikstufe: Detailed Einheit: Anzahl
`User Records per PutRecords Request`	Die Gesamtanzahl der Benutzerdatensätze in einer `PutRecordsRequest`. Diese Zahl entspricht ungefähr dem Produkt der beiden vorherigen Metriken. Auf Shard-Ebene nicht verfügbar. Metrikstufe: Detailed Einheit: Anzahl

Sicherheit in Amazon Kinesis Data Streams

Cloud-Sicherheit AWS hat höchste Priorität. Als AWS Kunde profitieren Sie von einer Rechenzentrums- und Netzwerkarchitektur, die auf die Anforderungen der sicherheitssensibelsten Unternehmen zugeschnitten sind.

Sicherheit ist eine gemeinsame Verantwortung von Ihnen AWS und Ihnen. Das Modell der geteilten Verantwortung beschreibt dies als Sicherheit der Cloud und Sicherheit in der Cloud:

- Sicherheit der Cloud — AWS ist verantwortlich für den Schutz der Infrastruktur, die AWS Dienste in der AWS Cloud ausführt. AWS bietet Ihnen auch Dienste, die Sie sicher nutzen können. Die Wirksamkeit unserer Sicherheitsfunktionen wird regelmäßig von externen Prüfern im Rahmen des AWS -Compliance-Programms getestet und überprüft. Weitere Informationen zu den Compliance-Programmen für Kinesis Data Streams finden Sie unter Durch das Compliance-Programm abgedeckte AWS -Services.
- Sicherheit in der Cloud — Ihre Verantwortung richtet sich nach dem AWS Dienst, den Sie nutzen. In Ihre Verantwortung fallen außerdem weitere Faktoren, wie z. B. die Vertraulichkeit der Daten, die Anforderungen Ihrer Organisation sowie geltende Gesetze und Vorschriften.

Diese Dokumentation hilft Ihnen zu verstehen, wie Sie das Modell der geteilten Verantwortung bei der Verwendung von Kinesis Data Streams einsetzen können. Die folgenden Themen veranschaulichen, wie Sie Kinesis Data Streams zur Erfüllung Ihrer Sicherheits- und Compliance-Ziele konfigurieren können. Sie erfahren auch, wie Sie andere AWS Dienste nutzen können, mit denen Sie Ihre Kinesis Data Streams Streams-Ressourcen überwachen und sichern können.

Themen

Datenschutz in Amazon Kinesis Data Streams

Die serverseitige Verschlüsselung mit AWS Key Management Service (AWS KMS) -Schlüsseln macht es Ihnen leicht, strenge Datenverwaltungsanforderungen zu erfüllen, indem Sie Ihre ruhenden Daten in Amazon Kinesis Data Streams verschlüsseln.

> **Note**
>
> Wenn Sie für den Zugriff AWS über eine Befehlszeilenschnittstelle oder eine andere FIPS 140-2 validierte kryptografische Module benötigen, verwenden Sie einen Endpunkt. API FIPS Weitere Informationen zu den verfügbaren FIPS Endpunkten finden Sie unter Federal Information Processing Standard () 140-2. FIPS

Themen

Was ist serverseitige Verschlüsselung für Kinesis Data Streams?

Serverseitige Verschlüsselung ist eine Funktion in Amazon Kinesis Data Streams, die Daten automatisch verschlüsselt, bevor sie gespeichert werden, indem ein von Ihnen festgelegter AWS KMS Kundenhauptschlüssel (CMK) verwendet wird. Die Daten werden verschlüsselt, bevor sie in die Speicherschicht des Kinesis-Streams geschrieben werden. Nach Abruf aus dem Speicher werden Sie entschlüsselt. Dadurch sind Ihre ruhenden Daten innerhalb des Services Kinesis Data Streams verschlüsselt. Sie erfüllen so die strengen regulatorischen Anforderungen und erhöhen die Sicherheit Ihrer Daten.

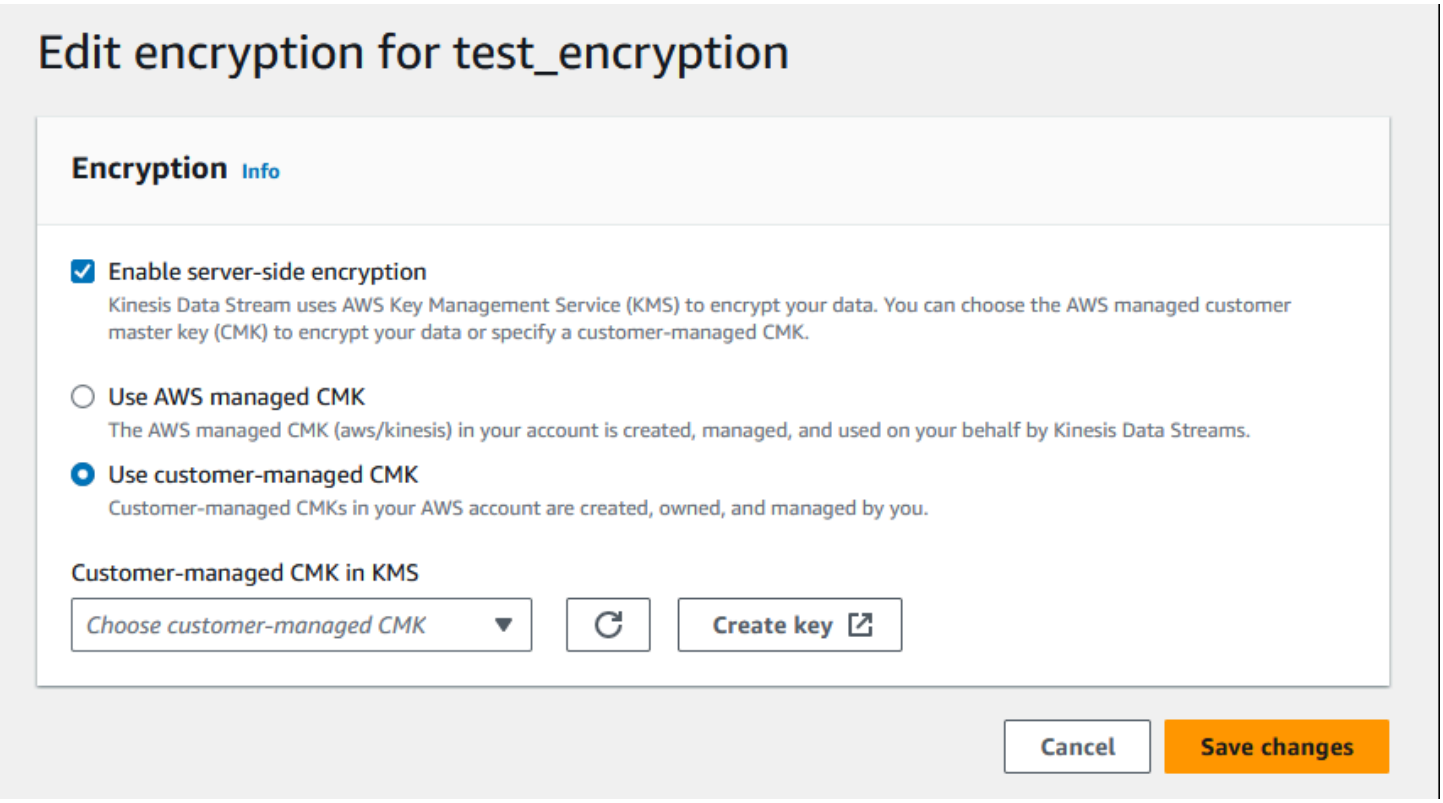
Durch die serverseitige Verschlüsselung müssen Ihre Kinesis-Stream-Produzenten und -Verbraucher keine Masterschlüssel oder kryptographische Operationen verwalten. Ihre Daten werden automatisch

verschlüsselt, wenn sie den Kinesis Data Streams Streams-Dienst betreten und verlassen, sodass Ihre Daten im Ruhezustand verschlüsselt sind. AWS KMS stellt alle Masterschlüssel bereit, die von der serverseitigen Verschlüsselungsfunktion verwendet werden. AWS KMS macht es einfach, einen CMK für Kinesis zu verwenden, der von AWS einem benutzerdefinierten AWS KMS CMK oder einem in den AWS KMS Service importierten Masterschlüssel verwaltet wird.

> **Note**
>
> Bei der serverseitigen Verschlüsselung werden eingehende Daten nur dann verschlüsselt, wenn die Funktion aktiviert ist. Bereits in einem nicht verschlüsselten Stream vorhandene Daten werden nach Aktivierung der serverseitigen Verschlüsselung nicht verschlüsselt.

Wenn Sie Ihre Datenströme verschlüsseln und den Zugriff mit anderen Principals teilen, müssen Sie die entsprechenden Berechtigungen sowohl in der Schlüsselrichtlinie für den Schlüssel als auch in den AWS KMS IAM Richtlinien für das externe Konto erteilen. Weitere Informationen finden Sie unter Zulassen, dass Benutzer mit anderen Konten einen KMS Schlüssel verwenden.

Wenn Sie die serverseitige Verschlüsselung für einen Datenstrom mit AWS verwaltetem KMS Schlüssel aktiviert haben und den Zugriff über eine Ressourcenrichtlinie gemeinsam nutzen möchten, müssen Sie zur Verwendung des vom Kunden verwalteten Schlüssels (CMK) wechseln, wie im Folgenden dargestellt:



Darüber hinaus müssen Sie den für die gemeinsame Nutzung verantwortlichen Entitäten den Zugriff auf Ihre Daten gewährenCMK, indem Sie Funktionen für die KMS kontenübergreifende gemeinsame Nutzung nutzen. Stellen Sie sicher, dass Sie die Änderungen auch in den IAM Richtlinien für die gemeinsam genutzten Haupteinheiten vornehmen. Weitere Informationen finden Sie unter Benutzern mit anderen Konten die Verwendung eines KMS Schlüssels gestatten.

Überlegungen zu Kosten, Regionen und Leistung

Wenn Sie serverseitige Verschlüsselung anwenden, fallen Kosten für AWS KMS API Nutzung und Schlüssel an. Im Gegensatz zu benutzerdefinierten KMS Hauptschlüsseln wird der `(Default) aws/kinesis` Kundenhauptschlüssel (CMK) kostenlos angeboten. Sie müssen jedoch weiterhin die API Nutzungskosten bezahlen, die Amazon Kinesis Data Streams in Ihrem Namen entstehen.

APINutzungskosten fallen für alle anCMK, auch für benutzerdefinierte. Kinesis Data Streams ruft AWS KMS ungefähr alle fünf Minuten ab, wenn der Datenschlüssel gewechselt wird. In einem Monat mit 30 Tagen sollten die Gesamtkosten für AWS KMS API Anrufe, die über einen Kinesis-Stream initiiert werden, weniger als ein paar Dollar betragen. Diese Kosten hängen von der Anzahl der Benutzeranmeldedaten ab, die Sie bei Ihren Datenproduzenten und -verbrauchern verwenden, da für jede Benutzeranmeldeberechtigung ein eindeutiger API Anruf erforderlich ist. AWS KMS Wenn Sie eine IAM Rolle für die Authentifizierung verwenden, führt jeder Assumer-Rollenaufruf zu eindeutigen

Benutzeranmeldedaten. Um KMS Kosten zu sparen, sollten Sie Benutzeranmeldeinformationen zwischenspeichern, die beim Assume-Role-Aufruf zurückgegeben werden.

Im Folgenden werden die Kosten pro Ressource aufgeführt:

Schlüssel

- Das CMK für Kinesis, das von AWS (alias =`aws/kinesis`) verwaltet wird, ist kostenlos.
- Für benutzergenerierte KMS Schlüssel fallen KMS Schlüsselkosten an. Weitere Informationen finden Sie unter AWS Key Management Service – Preise.

APINutzungskosten fallen für alle anCMK, auch für benutzerdefinierte. Kinesis Data Streams ruft KMS ungefähr alle 5 Minuten auf, wenn der Datenschlüssel rotiert wird. In einem Monat mit 30 Tagen sollten die Gesamtkosten für KMS API Anrufe, die über einen Kinesis-Datenstrom initiiert werden, weniger als ein paar Dollar betragen. Bitte beachten Sie, dass diese Kosten mit der Anzahl der Benutzeranmeldedaten, die Sie bei Ihren Datenproduzenten und -verbrauchern verwenden, steigen, da für jeden Benutzeranmeldenachweis ein eindeutiger API Anruf erforderlich ist. AWS KMS Wenn Sie IAM Rollen für die Authentifizierung verwenden, assume-role-call führt jede Rolle zu eindeutigen Benutzeranmeldedaten. Um KMS Kosten zu sparen, sollten Sie die von der assume-role-call zurückgegebenen Benutzeranmeldedaten zwischenspeichern.

KMSAPIVerwendung

Für jeden verschlüsselten Stream ruft der Kinesis-Dienst den AWS KMS Dienst etwa 12 Mal alle 5 Minuten auf, wenn von TIP einem einzigen IAM Konto/Benutzerzugriffsschlüssel für mehrere Leser und Autoren gelesen und verwendet wird. Wenn kein Text gelesen wird, TIP kann dies zu einer höheren Anzahl von Anfragen an den Service führen. AWS KMS APIAnfragen zur Generierung neuer Datenverschlüsselungsschlüssel sind mit AWS KMS Nutzungskosten verbunden. Weitere Informationen finden Sie unter Preise für AWS Key Management Service – Verwendung.

Verfügbarkeit serverseitiger Verschlüsselung nach Regionen

Derzeit ist die serverseitige Verschlüsselung von Kinesis-Streams in allen Regionen verfügbar, die für Kinesis Data Streams unterstützt werden, einschließlich AWS GovCloud (USA West) und den Regionen China. Weitere Informationen zu unterstützten Regionen für Kinesis Data Streams finden Sie unter https://docs.aws.amazon.com/general/latest/gr/ak.html.

Performanceaspekte

Aufgrund des Serviceaufwands bei der Verschlüsselung erhöht die serverseitige Verschlüsselung die typische Latenzzeit von `PutRecord`, `PutRecords` und `GetRecords` um 100µs.

Wie fange ich mit serverseitiger Verschlüsselung an?

Der einfachste Weg, mit serverseitiger Verschlüsselung zu beginnen, ist die Verwendung des AWS Management Console und des Amazon Kinesis KMS Service Keys,. `aws/kinesis`

Im Folgenden wird die Aktivierung der serverseitigen Verschlüsselung für einen Kinesis-Stream beschrieben.

So aktivieren Sie die serverseitige Verschlüsselung für einen Kinesis-Stream

1. Melden Sie sich bei der Amazon Kinesis Data Streams Streams-Konsole an AWS Management Console und öffnen Sie sie.
2. Wählen oder erstellen Sie einen Kinesis-Stream in der AWS Management Console.
3. Wählen Sie die Registerkarte Details aus.
4. Wählen Sie unter Server-side encryption (Serverseitige Verschlüsselung) die Option edit (Bearbeiten) aus.
5. Sofern Sie keinen benutzergenerierten KMS Masterschlüssel verwenden möchten, stellen Sie sicher, dass der KMSAWS/Kinesis-Masterschlüssel (Standard) ausgewählt ist. Dies ist der vom Kinesis-Dienst generierte KMS Hauptschlüssel. Wählen Sie Enabled (Aktiviert) und anschließend Save (Speichern) aus.

   > **Note**
   >
   > Der standardmäßige Kinesis-Servicemasterkey ist kostenlos. Für die API Anrufe von Kinesis an den AWS KMS Service fallen jedoch KMS Nutzungskosten an.

6. Der Stream ist vorübergehend im Zustand ausstehend. Nachdem der Stream in einen aktiven Zustand mit aktivierter Verschlüsselung zurückgekehrt ist, werden alle in den Stream geschriebenen eingehenden Daten mit dem ausgewählten KMS Masterschlüssel verschlüsselt.
7. Um die serverseitige Verschlüsselung zu deaktivieren, wählen Sie Deaktiviert unter Serverseitige Verschlüsselung in und wählen Sie dann Speichern. AWS Management Console

Benutzergenerierte Schlüssel erstellen und verwenden KMS

In diesem Abschnitt wird beschrieben, wie Sie Ihre eigenen KMS Schlüssel erstellen und verwenden können, anstatt den von Amazon Kinesis verwalteten Masterschlüssel zu verwenden.

Benutzergenerierte Schlüssel KMS erstellen

Anweisungen zum Erstellen eigener Schlüssel finden Sie unter Creating Keys im AWS Key Management Service Developer Guide. Nachdem Sie Schlüssel für Ihr Konto erstellt haben, gibt der Kinesis Data Streams Streams-Dienst diese Schlüssel in der KMSHauptschlüsselliste zurück.

Verwenden von benutzergenerierten Schlüsseln KMS

Nachdem Ihren Verbrauchern, Produzenten und Administratoren die richtigen Berechtigungen zugewiesen wurden, können Sie benutzerdefinierte KMS Schlüssel in Ihrem eigenen AWS Konto oder einem anderen AWS Konto verwenden. Alle KMS Hauptschlüssel in Ihrem Konto werden in der KMSMaster-Key-Liste im angezeigt AWS Management Console.

Um benutzerdefinierte KMS Hauptschlüssel verwenden zu können, die sich in einem anderen Konto befinden, benötigen Sie Berechtigungen zur Verwendung dieser Schlüssel. Sie müssen auch den Namen ARN des KMS Hauptschlüssels im ARN Eingabefeld in der angeben AWS Management Console.

Berechtigungen zur Verwendung von benutzergenerierten Schlüsseln KMS

Bevor Sie serverseitige Verschlüsselung mit einem benutzergenerierten KMS Schlüssel verwenden können, müssen Sie AWS KMS Schlüsselrichtlinien konfigurieren, um die Verschlüsselung von Streams sowie die Verschlüsselung und Entschlüsselung von Stream-Datensätzen zu ermöglichen. Beispiele und weitere Informationen zu AWS KMS Berechtigungen finden Sie unter AWS KMSAPIPermissions: Actions and Resources Reference.

> **Note**
>
> Die Verwendung des standardmäßigen Dienstschlüssels für die Verschlüsselung erfordert keine Anwendung benutzerdefinierter IAM Berechtigungen.

Bevor Sie benutzergenerierte KMS Masterschlüssel verwenden, stellen Sie sicher, dass Ihre Kinesis-Stream-Produzenten und -Verbraucher (IAMPrincipals) Benutzer in der KMS Master-Key-Richtlinie sind. Andernfalls schlägt das Schreiben in und das Lesen aus dem Stream fehl. Dies kann zu einem

Datenverlust, einer verspäteten Verarbeitung oder abgestürzten Anwendungen führen. Sie können die Berechtigungen für KMS Schlüssel mithilfe von Richtlinien verwalten. IAM Weitere Informationen finden Sie unter IAMRichtlinien verwenden mit AWS KMS.

Beispiel für Herstellerberechtigungen

Die Kinesis-Stream-Produzenten benötigen die Berechtigung `kms:GenerateDataKey`.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
        "Effect": "Allow",
        "Action": [
            "kms:GenerateDataKey"
        ],
        "Resource": "arn:aws:kms:us-
west-2:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab"
    },
    {
        "Effect": "Allow",
        "Action": [
            "kinesis:PutRecord",
            "kinesis:PutRecords"
        ],
        "Resource": "arn:aws:kinesis:*:123456789012:MyStream"
    }
  ]
}
```

Beispiel für Verbraucherberechtigungen

Ihre Kinesis-Stream-Konsumenten benötigen die Berechtigung `kms:Decrypt`.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
        "Effect": "Allow",
        "Action": [
            "kms:Decrypt"
        ],
```

```
        "Resource": "arn:aws:kms:us-
west-2:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab"
    },
    {
        "Effect": "Allow",
        "Action": [
            "kinesis:GetRecords",
            "kinesis:DescribeStream"
        ],
        "Resource": "arn:aws:kinesis:*:123456789012:MyStream"
    }
  ]
}
```

Amazon Managed Service für Apache Flink und AWS Lambda verwenden Sie Rollen, um Kinesis-Streams zu nutzen. Stellen Sie sicher, dass Sie die `kms:Decrypt`-Berechtigung für die Rollen erteilen, die diese Konsumenten innehaben.

Stream-Administratorberechtigungen

Kinesis-Stream-Administratoren benötigen eine Autorisierung für den Aufruf von `kms:List*` und `kms:DescribeKey*`.

Überprüfen Sie die KMS wichtigsten Berechtigungen und beheben Sie Fehler

Nachdem Sie die Verschlüsselung für einen Kinesis-Stream aktiviert haben, empfehlen wir Ihnen, den Erfolg Ihrer `putRecordputRecords`, und `getRecords` -Anrufe anhand der folgenden CloudWatch Amazon-Metriken zu überwachen:

- `PutRecord.Success`
- `PutRecords.Success`
- `GetRecords.Success`

Weitere Informationen finden Sie unter Kinesis Data Streams überwachen

Verwenden Sie Amazon Kinesis Data Streams mit Schnittstellenendpunkten VPC

Sie können einen VPC Schnittstellenendpunkt verwenden, um zu verhindern, dass der Datenverkehr zwischen Ihren Amazon VPC - und Kinesis Data Streams das Amazon-Netzwerk verlässt. VPCSchnittstellenendpunkte benötigen kein Internet-Gateway, NAT Gerät, VPN Verbindung oder AWS Direct Connect Verbindung. VPCSchnittstellenendpunkte werden von einer AWS Technologie unterstützt AWS PrivateLink, die private Kommunikation zwischen AWS Diensten über eine elastic network interface mit Private IPs in Ihrem Amazon VPC ermöglicht. Weitere Informationen finden Sie unter Amazon Virtual Private Cloud und Interface VPC Endpoints (AWS PrivateLink).

Themen

Verwenden Sie VPC Schnittstellenendpunkte für Kinesis Data Streams

Für den Einstieg müssen Sie die Einstellungen für Ihre Streams, Produzenten oder Verbraucher nicht ändern. Erstellen Sie einfach einen VPC Schnittstellenendpunkt, damit Ihr Kinesis Data Streams Streams-Verkehr von und zu Ihren VPC Amazon-Ressourcen über den VPC Schnittstellenendpunkt fließen kann. Weitere Informationen finden Sie unter Erstellen eines Schnittstellenendpunkts.

Die Kinesis Producer Library (KPL) und die Kinesis Consumer Library (KCL) rufen AWS Dienste wie Amazon CloudWatch und Amazon DynamoDB entweder über öffentliche Endpunkte oder private VPC Schnittstellenendpunkte auf, je nachdem, welche verwendet werden. Wenn Ihre KCL Anwendung beispielsweise in einer DynamoDB-Schnittstelle VPC mit aktivierten VPC Endpunkten ausgeführt wird, werden Aufrufe zwischen DynamoDB und Ihrer KCL Anwendung über den Schnittstellenendpunkt geleitet. VPC

Steuern Sie den Zugriff auf VPC Endpunkte für Kinesis Data Streams

VPCMit Endpunktrichtlinien können Sie den Zugriff kontrollieren, indem Sie entweder eine Richtlinie an einen VPC Endpunkt anhängen oder indem Sie zusätzliche Felder in einer Richtlinie verwenden, die einem IAM Benutzer, einer Gruppe oder einer Rolle zugeordnet ist, um den Zugriff darauf zu beschränken, dass der Zugriff nur über den angegebenen Endpunkt erfolgt. VPC Diese Richtlinien können verwendet werden, um den Zugriff auf bestimmte Streams auf einen bestimmten VPC

Endpunkt zu beschränken, wenn sie zusammen mit den IAM Richtlinien verwendet werden, um nur Zugriff auf Kinesis-Datenstream-Aktionen über den angegebenen VPC Endpunkt zu gewähren.

Es folgen Beispiele für Endpunktrichtlinien für den Zugriff auf Kinesis-Daten-Streams.

- VPCRichtlinienbeispiel: Nur-Lese-Zugriff — Diese Beispielrichtlinie kann an einen Endpunkt angehängt werden. VPC (Weitere Informationen finden Sie unter Steuern des Zugriffs auf VPC Amazon-Ressourcen). Es beschränkt die Aktionen darauf, einen Kinesis-Datenstrom nur über den VPC Endpunkt aufzulisten und zu beschreiben, an den er angehängt ist.

```
{
  "Statement": [
    {
      "Sid": "ReadOnly",
      "Principal": "*",
      "Action": [
        "kinesis:List*",
        "kinesis:Describe*"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

- VPCRichtlinienbeispiel: Beschränken Sie den Zugriff auf einen bestimmten Kinesis-Datenstrom — diese Beispielrichtlinie kann an einen VPC Endpunkt angehängt werden. Sie schränkt den Zugriff auf einen bestimmten Datenstrom über den VPC Endpunkt ein, an den er angehängt ist.

```
{
  "Statement": [
    {
      "Sid": "AccessToSpecificDataStream",
      "Principal": "*",
      "Action": "kinesis:*",
      "Effect": "Allow",
      "Resource": "arn:aws:kinesis:us-east-1:123456789012:stream/MyStream"
    }
  ]
}
```

- IAMBeispiel für eine Richtlinie: Beschränken Sie den Zugriff auf einen bestimmten Stream nur von einem bestimmten VPC Endpunkt aus. Diese Beispielrichtlinie kann einem IAM Benutzer, einer Rolle oder einer Gruppe zugewiesen werden. Es schränkt den Zugriff auf einen bestimmten Kinesis-Datenstrom so ein, dass er nur von einem bestimmten VPC Endpunkt aus erfolgt.

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

Verfügbarkeit von VPC Endpunktrichtlinien für Kinesis Data Streams

Kinesis Data Streams VPC Streams-Schnittstellenendpunkte mit Richtlinien werden in den folgenden Regionen unterstützt:

- Europa (Paris)
- Europa (Irland)
- USA Ost (Nord-Virginia)
- Europa (Stockholm)
- USA Ost (Ohio)
- Europa (Frankfurt)
- Südamerika (São Paulo)
- Europa (London)
- Asien-Pazifik (Tokio)
- USA West (Nordkalifornien)

- Asien-Pazifik (Singapur)
- Asien-Pazifik (Sydney)
- China (Peking)
- China (Ningxia)
- Asia Pacific (Hong Kong)
- Naher Osten (Bahrain)
- Naher Osten () UAE
- Europa (Milan)
- Afrika (Kapstadt)
- Asien-Pazifik (Mumbai)
- Asien-Pazifik (Seoul)
- Kanada (Zentral)
- USA West (Oregon) außer usw2-az4
- AWS GovCloud (US-Ost)
- AWS GovCloud (US-West)
- Asien-Pazifik (Osaka)
- Europa (Zürich)
- Asien-Pazifik (Hyderabad)

Steuern des Zugriffs auf Amazon Kinesis Data Streams Streams-Ressourcen mithilfe IAM

AWS Identity and Access Management (IAM) ermöglicht Ihnen Folgendes:

- Erstellen Sie Benutzer und Gruppen unter Ihrem AWS Konto
- Weisen Sie jedem Benutzer unter Ihrem AWS Konto eindeutige Sicherheitsanmeldedaten zu
- Kontrollieren Sie die Berechtigungen der einzelnen Benutzer zur Ausführung von Aufgaben mithilfe von AWS Ressourcen
- Erlauben Sie den Benutzern eines anderen AWS Kontos, Ihre AWS Ressourcen gemeinsam zu nutzen

- Erstellen Sie Rollen für Ihr AWS Konto und definieren Sie die Benutzer oder Dienste, die diese Rollen übernehmen können
- Verwenden Sie bestehende Identitäten für Ihr Unternehmen, um Berechtigungen zur Ausführung von Aufgaben unter Verwendung von AWS Ressourcen zu erteilen

Durch die Verwendung IAM mit Kinesis Data Streams können Sie steuern, ob Benutzer in Ihrer Organisation eine Aufgabe mithilfe bestimmter Kinesis Data Streams API Streams-Aktionen ausführen können und ob sie bestimmte AWS Ressourcen verwenden können.

Wenn Sie eine Anwendung mithilfe der Kinesis Client Library (KCL) entwickeln, muss Ihre Richtlinie Berechtigungen für Amazon DynamoDB und Amazon beinhalten. DynamoDB KCL verwendet DynamoDB CloudWatch, um Statusinformationen für die Anwendung zu verfolgen und in Ihrem Namen KCL Metriken an CloudWatch diese zu senden. CloudWatch Weitere Informationen zu finden Sie unter. KCL Entwickeln Sie KCL 1.x-Verbraucher

Weitere Informationen zu IAM finden Sie unter:

- AWS Identity and Access Management (IAM)
- Erste Schritte
- IAM-Benutzerhandbuch

Weitere Informationen zu IAM Amazon DynamoDB finden Sie unter Using IAM to Control Access to Amazon DynamoDB Resources im Amazon DynamoDB DynamoDB-Entwicklerhandbuch.

Weitere Informationen zu IAM und Amazon CloudWatch finden Sie unter Steuern des Benutzerzugriffs auf Ihr AWS Konto im CloudWatch Amazon-Benutzerhandbuch.

Inhalt

Richtliniensyntax

Eine IAM Richtlinie ist ein JSON Dokument, das aus einer oder mehreren Aussagen besteht. Jede Anweisung ist folgendermaßen strukturiert:

```
{
  "Statement":[{
    "Effect":"effect",
    "Action":"action",
    "Resource":"arn",
    "Condition":{
      "condition":{
        "key":"value"
        }
      }
    }
  ]
}
```

Eine Anweisung kann aus verschiedenen Elementen bestehen:

- Effect: Der effect-Wert kann `Allow` oder `Deny` lauten. Standardmäßig sind IAM Benutzer nicht berechtigt, Ressourcen und API Aktionen zu verwenden, sodass alle Anfragen abgelehnt werden. Dieser Standardwert kann durch eine explizite Zugriffserlaubnis überschrieben werden. Eine explizite Zugriffsverweigerung überschreibt jedwede Zugriffserlaubnis.
- Aktion: Die Aktion ist die spezifische API Aktion, für die Sie die Erlaubnis erteilen oder verweigern.
- Resource: Die von einer Aktion betroffene Ressource. Um eine Ressource in der Anweisung anzugeben, müssen Sie ihren Amazon-Ressourcennamen (ARN) verwenden.
- Condition: Bedingungen sind optional. Mit ihrer Hilfe können Sie bestimmen, wann Ihre Richtlinie wirksam wird.

Wenn Sie IAM Richtlinien erstellen und verwalten, möchten Sie möglicherweise den IAMPolicy Generator und den IAMPolicy Simulator verwenden.

Aktionen für Kinesis Data Streams

In einer IAM Richtlinienerklärung können Sie jede API Aktion von jedem Dienst angeben, der dies unterstütztIAM. Verwenden Sie für Kinesis Data Streams das folgende Präfix mit dem Namen

der API Aktion:`kinesis:`. Beispiel: `kinesis:CreateStream`, `kinesis:ListStreams` und `kinesis:DescribeStreamSummary`.

Um mehrere Aktionen in einer einzigen Anweisung anzugeben, trennen Sie sie wie folgt durch Kommata:

```
"Action": ["kinesis:action1", "kinesis:action2"]
```

Sie können auch mehrere Aktionen mittels Platzhaltern angeben. Beispielsweise können Sie alle Aktionen festlegen, deren Name mit dem Wort "Get" beginnt:

```
"Action": "kinesis:Get*"
```

Um alle Operationen von Kinesis Data Streams anzugeben, verwenden Sie den Platzhalter * folgendermaßen:

```
"Action": "kinesis:*"
```

Die vollständige Liste der Kinesis Data Streams API Streams-Aktionen finden Sie in der Amazon Kinesis Kinesis-Referenz API.

Amazon-Ressourcennamen (ARNs) für Kinesis Data Streams

Jede IAM Richtlinienerklärung gilt für die Ressourcen, die Sie mithilfe ihrer ARNs angeben.

Verwenden Sie das folgende ARN Ressourcenformat für Kinesis-Datenstreams:

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

Beispielsweise:

```
"Resource": arn:aws:kinesis:*:111122223333:stream/my-stream
```

Beispielrichtlinien für Kinesis Data Streams

Die folgenden Beispielrichtlinien zeigen, wie Sie den Benutzerzugriff auf Kinesis-Datenströme steuern könnten.

Example 1: Allow users to get data from a stream

Example

Diese Richtlinie erlaubt einem Benutzer oder einer Gruppe, die Operationen `DescribeStreamSummary`, `GetShardIterator` und `GetRecords` auf dem angegebenen Stream und `ListStreams` auf einem beliebigen Stream auszuführen. Diese Richtlinie könnte auf Benutzer angewendet werden, die Daten aus einem spezifischen Stream abrufen können sollten.

```
{
    "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

Diese Richtlinie erlaubt einem Benutzer oder einer Gruppe, die Operation `PutRecord` mit einem beliebigen Stream des Kontos zu verwenden. Diese Richtlinie könnte auf Benutzer angewendet werden, die Daten zu allen Streams in einem Konto hinzufügen können sollten.

```
{
```

```
    "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

Diese Richtlinie erlaubt einem Benutzer oder einer Gruppe, eine beliebige Operation von Kinesis Data Streams auf dem angegebenen Stream zu verwenden. Diese Richtlinie könnte auf Benutzer angewendet werden, die administrative Kontrolle über einen bestimmten Stream haben sollten.

```
{
    "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

Diese Richtlinie erlaubt einem Benutzer oder einer Gruppe, eine beliebige Operation von Kinesis Data Streams auf einem beliebigen Stream in einem Konto zu verwenden. Da diese Richtlinie vollen Zugriff auf alle Ihre Streams gewährt, sollten Sie sie auf Administratoren beschränken.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": [
                "arn:aws:kinesis:*:111122223333:stream/*"
            ]
        }
    ]
}
```

Teilen Sie Ihren Datenstream mit einem anderen Konto

> **Note**
>
> Die Kinesis Producer Library unterstützt derzeit nicht die Angabe eines Streams ARN beim Schreiben in einen Datenstream. Verwenden Sie den AWS SDK, wenn Sie in einen kontoübergreifenden Datenstream schreiben möchten.

Fügen Sie Ihrem Datenstream eine ressourcenbasierte Richtlinie hinzu, um einem anderen Konto, IAM Benutzer oder einer anderen Rolle Zugriff zu gewähren. IAM Ressourcenbasierte Richtlinien sind JSON Richtliniendokumente, die Sie an eine Ressource wie einen Datenstrom anhängen. Diese Richtlinien erteilen dem angegebenen Prinzipal die Berechtigung zum Ausführen bestimmter Aktionen für diese Ressource und definieren, unter welchen Bedingungen diese gilt. Eine Richtlinie kann mehrere Anweisungen enthalten. Sie müssen in einer ressourcenbasierten Richtlinie einen Prinzipal angeben. Zu den Prinzipalen können Konten, Benutzer, Rollen, Verbundbenutzer oder Dienste gehören. AWS Sie können Richtlinien in der Kinesis Data Streams Streams-Konsole konfigurieren, API oderSDK.

Beachten Sie, dass für die gemeinsame Nutzung des Zugriffs mit registrierten Verbrauchern wie Enhanced Fan Out eine Richtlinie sowohl für den Datenstream ARN als auch für den Verbraucher ARN erforderlich ist.

Aktivieren Sie den kontoübergreifenden Zugriff

Um den kontoübergreifenden Zugriff zu ermöglichen, können Sie in einer ressourcenbasierten Richtlinie ein ganzes Konto oder IAM Entitäten in einem anderen Konto als Prinzipal angeben. Durch das Hinzufügen eines kontoübergreifenden Auftraggebers zu einer ressourcenbasierten Richtlinie ist nur die halbe Vertrauensbeziehung eingerichtet. Wenn sich der Prinzipal und die Ressource in getrennten AWS Konten befinden, müssen Sie außerdem eine identitätsbasierte Richtlinie verwenden, um dem Prinzipal Zugriff auf die Ressource zu gewähren. Wenn jedoch eine ressourcenbasierte Richtlinie Zugriff auf einen Prinzipal in demselben Konto gewährt, ist keine zusätzliche identitätsbasierte Richtlinie erforderlich.

Weitere Informationen zur Verwendung ressourcenbasierter Richtlinien für den kontenübergreifenden Zugriff finden Sie unter Kontoübergreifender Ressourcenzugriff in. IAM

Datenstream-Administratoren können mithilfe von AWS Identity and Access Management Richtlinien angeben, wer auf was Zugriff hat. Das bedeutet, welcher Prinzipal kann Aktionen für welche Ressourcen und unter welchen Bedingungen ausführen. Das `Action` Element einer JSON Richtlinie beschreibt die Aktionen, mit denen Sie den Zugriff in einer Richtlinie zulassen oder verweigern können. Richtlinienaktionen haben normalerweise denselben Namen wie der zugehörige AWS API Vorgang.

Aktionen von Kinesis Data Streams, die freigegeben werden können:

Aktion	Zugriffsebene
DescribeStreamConsumer	Konsument
DescribeStreamSummary	Datenstrom
GetRecords	Datenstrom
GetShardIterator	Datenstrom
ListShards	Datenstrom
PutRecord	Datenstrom
PutRecords	Datenstrom

Aktion	Zugriffsebene
SubscribeToShard	Konsument

Im Folgenden finden Sie Beispiele für die Verwendung einer ressourcenbasierten Richtlinie, um kontoübergreifenden Zugriff auf Ihren Datenstrom oder registrierten Verbraucher zu gewähren.

Um eine kontoübergreifende Aktion durchzuführen, müssen Sie den Stream ARN für den Zugriff auf den Datenstream und den Consumer ARN für den registrierten Consumer-Zugriff angeben.

Beispiel für ressourcenbasierte Richtlinien für Kinesis-Datenstreams

Die Freigabe eines registrierten Verbrauchers erfordert aufgrund der erforderlichen Maßnahmen sowohl eine Datenstromrichtlinie als auch eine Verbraucherrichtlinie.

> **Note**
>
> Nachfolgend finden Sie Beispiele für gültige Werte für `Principal`:
>
> - `{"AWS": "123456789012"}`
> - IAMBenutzer — `{"AWS": "arn:aws:iam::123456789012:user/user-name"}`
> - IAMRolle — `{"AWS":["arn:aws:iam::123456789012:role/role-name"]}`
> - Mehrere Prinzipale (kann eine Kombination aus Konto, Benutzer, Rolle sein) – `{"AWS": ["123456789012", "123456789013", "arn:aws:iam::123456789012:user/user-name"]}`

Example 1: Write access to the data stream

Example

```
{
    "Version": "2012-10-17",
    "Id": "__default_write_policy_ID",
    "Statement": [
        {
            "Sid": "writestatement",
            "Effect": "Allow",
```

```
            "Principal": {
                "AWS": "Account12345"
            },
            "Action": [
                "kinesis:DescribeStreamSummary",
                "kinesis:ListShards",
                "kinesis:PutRecord",
                "kinesis:PutRecords"
            ],
            "Resource": "arn:aws:kinesis:us-east-2:123456789012:stream/
datastreamABC"
        }
    ]
}
```

Example 2: Read access to the data stream

Example

```
{
    "Version": "2012-10-17",
    "Id": "__default_sharedthroughput_read_policy_ID",
    "Statement": [
        {
            "Sid": "sharedthroughputreadstatement",
            "Effect": "Allow",
            "Principal": {
                "AWS": "Account12345"
            },
            "Action": [
                "kinesis:DescribeStreamSummary",
                "kinesis:ListShards",
                "kinesis:GetRecords",
                "kinesis:GetShardIterator"
            ],
            "Resource": "arn:aws:kinesis:us-east-2:123456789012:stream/
datastreamABC"
        }
    ]
}
```

Example 3: Share enhanced fan-out read access to a registered consumer

Example

Erklärung zur Datenstromrichtlinie:

```
{
    "Version": "2012-10-17",
    "Id": "__default_sharedthroughput_read_policy_ID",
    "Statement": [
        {
            "Sid": "consumerreadstatement",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::Account12345:role/role-name"
            },
            "Action": [
                "kinesis:DescribeStreamSummary",
                "kinesis:ListShards"
            ],
            "Resource": "arn:aws:kinesis:us-east-2:123456789012:stream/
datastreamABC"
        }
    ]
}
```

Erklärung zur Verbraucherrichtlinie:

```
{
    "Version": "2012-10-17",
    "Id": "__default_efo_read_policy_ID",
    "Statement": [
        {
            "Sid": "eforeadstatement",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::Account12345:role/role-name"
            },
            "Action": [
                "kinesis:DescribeStreamConsumer",
                "kinesis:SubscribeToShard"
            ],
```

```
            "Resource": "arn:aws:kinesis:us-east-2:123456789012:stream/
datastreamABC/consumer/consumerDEF:1674696300"
        }
    ]
}
```

Platzhalter (*) werden für Aktionen oder Prinzipal-Felder nicht unterstützt, um das Prinzip der geringsten Berechtigung aufrecht zu erhalten.

Verwalten Sie die Richtlinie für Ihren Datenstrom programmatisch

Abgesehen von bietet Kinesis Data Streams drei Optionen APIS für die Verwaltung Ihrer Datenstream-Richtlinie: AWS Management Console

- PutResourcePolicy
- GetResourcePolicy
- DeleteResourcePolicy

Verwenden Sie `PutResourePolicy`, um eine Richtlinie für einen Datenstrom oder Verbraucher anzufügen oder zu überschreiben. Verwenden Sie `GetResourcePolicy`, um eine Richtlinie für den angegebenen Datenstrom oder Verbraucher zu überprüfen und anzuzeigen. Verwenden Sie `DeleteResourcePolicy`, um eine Richtlinie für den angegebenen Datenstrom oder Verbraucher zu löschen.

Richtlinienbeschränkungen

Für die Ressourcenrichtlinien von Kinesis Data Streams gelten folgende Einschränkungen:

- Platzhalter (*) werden nicht unterstützt, um zu verhindern, dass umfassender Zugriff über die Ressourcenrichtlinien gewährt wird, die direkt mit einem Datenstrom oder einem registrierten Verbraucher verknüpft sind. Prüfen Sie außerdem sorgfältig die folgenden Richtlinien, um sicherzustellen, dass sie keinen breiten Zugriff gewähren:
  - Identitätsbasierte Richtlinien, die mit zugehörigen AWS Prinzipalen verknüpft sind (z. B. Rollen) IAM
  - Ressourcenbasierte Richtlinien, die mit zugehörigen AWS Ressourcen verknüpft sind (z. B. Schlüssel) AWS Key Management Service KMS

- AWS Service Principals werden für Principals nicht unterstützt, um zu verhindern, dass Stellvertreter verwirrt werden.
- Verbund-Prinzipale werden nicht unterstützt.
- Canonical−Benutzer werden nicht unterstützt. IDs
- Die Größe der Richtlinie darf 20 KB nicht überschreiten.

Teilen Sie den Zugriff auf verschlüsselte Daten

Wenn Sie die serverseitige Verschlüsselung für einen Datenstrom mit AWS verwaltetem KMS Schlüssel aktiviert haben und den Zugriff über eine Ressourcenrichtlinie gemeinsam nutzen möchten, müssen Sie zur Verwendung des vom Kunden verwalteten Schlüssels () CMK wechseln. Weitere Informationen finden Sie unter Was ist serverseitige Verschlüsselung für Kinesis Data Streams?. Darüber hinaus müssen Sie den für die gemeinsame Nutzung verantwortlichen Entitäten den Zugriff auf Ihre Daten gewähren und dabei Funktionen CMK zur KMS kontenübergreifenden Nutzung nutzen. Stellen Sie sicher, dass Sie die Änderungen auch in den IAM Richtlinien für die gemeinsam genutzten Haupteinheiten vornehmen. Weitere Informationen finden Sie unter Benutzern mit anderen Konten die Verwendung eines KMS Schlüssels gestatten.

Konfigurieren Sie eine AWS Lambda Funktion zum Lesen aus Kinesis Data Streams in einem anderen Konto

Ein Beispiel dafür, wie Sie eine Lambda-Funktion zum Lesen von Kinesis Data Streams in einem anderen Konto konfigurieren können, finden Sie unter Teilen Sie den Zugriff mit kontoübergreifenden Funktionen AWS Lambda.

Teilen Sie den Zugriff mithilfe ressourcenbasierter Richtlinien

> **Note**
>
> Das Aktualisieren einer vorhandenen ressourcenbasierten Richtlinie bedeutet, dass die bestehende Richtlinie ersetzt wird. Stellen Sie daher sicher, dass Sie in Ihrer neuen Richtlinie alle erforderlichen Informationen angeben.

Teilen Sie den Zugriff mit kontoübergreifenden Funktionen AWS Lambda

Lambda-Operator

1. Gehen Sie zur IAMKonsole, um eine IAM Rolle zu erstellen, die als Lambda-Ausführungsrolle für Ihre AWS Lambda Funktion verwendet wird. Fügen Sie die verwaltete IAM Richtlinie hinzu`AWSLambdaKinesisExecutionRole`, die über die erforderlichen Kinesis Data Streams- und Lambda-Aufrufberechtigungen verfügt. Diese Richtlinie gewährt auch Zugriff auf alle potenziellen Ressourcen von Kinesis Data Streams, auf die Sie möglicherweise Zugriff haben.
2. Erstellen Sie in der AWS Lambda Konsole eine AWS Lambda Funktion zur Verarbeitung von Datensätzen in einem Kinesis Data Streams Streams-Datenstream und wählen Sie während der Einrichtung für die Ausführungsrolle die Rolle aus, die Sie im vorherigen Schritt erstellt haben.
3. Stellen Sie dem Ressourcenbesitzer in Kinesis Data Streams die Ausführungsrolle zur Konfiguration der Ressourcenrichtlinie bereit.
4. Schließen Sie die Einrichtung der Lambda-Funktion ab.

Besitzer der Ressource in Kinesis Data Streams

1. Rufen Sie die kontoübergreifende Lambda-Ausführungsrolle ab, welche die Lambda-Funktion aufruft.
2. Wählen Sie in der Konsole von Amazon Kinesis Data Streams den Datenstrom aus. Wählen Sie die Registerkarte Datenstrom-Freigabe und anschließend die Schaltfläche Freigaberichtlinie erstellen, um den visuellen Richtlinieneditor zu starten. Um einen registrierten Verbraucher innerhalb eines Datenstroms freizugeben, wählen Sie den Verbraucher aus und wählen Sie dann Freigaberichtlinie erstellen aus. Sie können die JSON Richtlinie auch direkt schreiben.
3. Geben Sie die kontoübergreifende Lambda-Ausführungsrolle als Prinzipal und die genauen Aktionen von Kinesis Data Streams an, auf die Sie Zugriff gewähren. Stellen Sie sicher, dass Sie die Aktion `kinesis:DescribeStream` einbeziehen. Weitere Informationen über beispielhafte Ressourcenrichtlinien für Kinesis Data Streams finden Sie unter Beispiel für ressourcenbasierte Richtlinien für Kinesis-Datenstreams.
4. Wählen Sie Richtlinie erstellen oder verwenden Sie die PutResourcePolicy, um die Richtlinie an Ihre Ressource anzuhängen.

Teilen Sie den Zugriff mit kontoübergreifenden Verbrauchern KCL

- Wenn Sie KCL 1.x verwenden, stellen Sie sicher, dass Sie KCL 1.15.0 oder höher verwenden.

- Wenn Sie KCL 2.x verwenden, stellen Sie sicher, dass Sie 2.5.3 oder höher verwendenKCL.

KCLBetreiber

1. Stellen Sie dem Eigentümer der Ressource Ihren IAM Benutzer oder Ihre IAM Rolle zur Ausführung der KCL Anwendung zur Verfügung.
2. Fragen Sie den Besitzer der Ressource nach dem Datenstream oder dem DatenverbraucherARN.
3. Stellen Sie sicher, dass Sie den bereitgestellten Stream ARN als Teil Ihrer KCL Konfiguration angeben.

   - Für KCL 1.x: Verwenden Sie den KinesisClientLibConfigurationKonstruktor und stellen Sie den Stream bereit. ARN
   - Für KCL 2.x: Sie können nur den Stream ARN oder StreamTrackerdie Kinesis Client Library bereitstellen. ConfigsBuilder Geben Sie für StreamTracker den Stream ARN und die Erstellungsepoche aus der DynamoDB-Leasetabelle an, die von der Bibliothek generiert wird. Wenn Sie von einem gemeinsamen registrierten Benutzer lesen möchten, z. B. Enhanced Fan-Out, verwenden Sie den Benutzer StreamTracker und geben Sie ihn auch an. ARN

Besitzer der Ressource in Kinesis Data Streams

1. Ermitteln Sie den kontoübergreifenden IAM Benutzer oder die IAM Rolle, die die Anwendung ausführen soll. KCL
2. Wählen Sie in der Konsole von Amazon Kinesis Data Streams den Datenstrom aus. Wählen Sie die Registerkarte Datenstrom-Freigabe und anschließend die Schaltfläche Freigaberichtlinie erstellen, um den visuellen Richtlinieneditor zu starten. Um einen registrierten Verbraucher innerhalb eines Datenstroms freizugeben, wählen Sie den Verbraucher aus und wählen Sie dann Freigaberichtlinie erstellen aus. Sie können die JSON Richtlinie auch direkt schreiben.
3. Geben Sie den IAM Benutzer oder die IAM Rolle der kontoübergreifenden KCL Anwendung als Principal und die genauen Kinesis Data Streams Streams-Aktionen an, auf die Sie den Zugriff teilen. Weitere Informationen über beispielhafte Ressourcenrichtlinien für Kinesis Data Streams finden Sie unter Beispiel für ressourcenbasierte Richtlinien für Kinesis-Datenstreams.
4. Wählen Sie Richtlinie erstellen oder verwenden Sie die PutResourcePolicy, um die Richtlinie an Ihre Ressource anzuhängen.

Teilen Sie den Zugriff auf verschlüsselte Daten

Wenn Sie die serverseitige Verschlüsselung für einen Datenstrom mit AWS verwaltetem KMS Schlüssel aktiviert haben und den Zugriff über eine Ressourcenrichtlinie gemeinsam nutzen möchten, müssen Sie zur Verwendung des vom Kunden verwalteten Schlüssels () CMK wechseln. Weitere Informationen finden Sie unter Was ist serverseitige Verschlüsselung für Kinesis Data Streams?. Darüber hinaus müssen Sie den für die gemeinsame Nutzung verantwortlichen Entitäten den Zugriff auf Ihre Daten gewähren und dabei Funktionen CMK zur KMS kontenübergreifenden Nutzung nutzen. Stellen Sie sicher, dass Sie die Änderungen auch in den IAM Richtlinien für die gemeinsam genutzten Haupteinheiten vornehmen. Weitere Informationen finden Sie unter Benutzern mit anderen Konten die Verwendung eines KMS Schlüssels gestatten.

Konformitätsprüfung für Amazon Kinesis Data Streams

Externe Prüfer bewerten die Sicherheit und Konformität von Amazon Kinesis Data Streams im Rahmen mehrerer AWS Compliance-Programme. Dazu gehören SOC PCIRAMP, Fed HIPAA und andere.

Eine Liste der AWS Dienstleistungen im Rahmen bestimmter Compliance-Programme finden Sie unter AWS Services im Umfang der einzelnen Compliance-Programme. Allgemeine Informationen finden Sie unter AWS -Compliance-Programme.

Sie können Prüfberichte von Drittanbietern unter herunterladen AWS Artifact. Weitere Informationen finden Sie unter Berichte in AWS Artifact herunterladen.

Ihre Compliance-Verantwortung bei Verwendung von Kinesis Data Streams hängt von der Vertraulichkeit der Daten, den Compliance-Zielen des Unternehmens und den geltenden Gesetzen und Vorschriften ab. Falls Ihre Nutzung von Kinesis Data Streams der Einhaltung von Standards wieHIPAA, oder Fed unterliegtPCI, AWS bietet Fed RessourcenRAMP, die Ihnen helfen:

- Schnellstartanleitungen zu Sicherheit und Compliance — In diesen Bereitstellungsleitfäden werden architektonische Überlegungen erörtert und Schritte für die Implementierung von Umgebungen beschrieben, auf denen Sicherheit und Compliance im Vordergrund stehen. AWS
- Whitepaper Architecting for HIPAA Security and Compliance — In diesem Whitepaper wird beschrieben, wie Unternehmen damit -konforme Anwendungen erstellen können AWS . HIPAA
- AWS Compliance-Ressourcen — Diese Sammlung von Arbeitsmappen und Leitfäden, die möglicherweise auf Ihre Branche und Ihren Standort zutreffen

- AWS Config— Dieser AWS Service bewertet, wie gut Ihre Ressourcenkonfigurationen internen Praktiken, Branchenrichtlinien und Vorschriften entsprechen.
- AWS Security Hub— Dieser AWS Service bietet einen umfassenden Überblick über Ihren Sicherheitsstatus und hilft Ihnen AWS , die Einhaltung der Sicherheitsstandards und bewährten Verfahren der Sicherheitsbranche zu überprüfen.

Resilienz in Amazon Kinesis Data Streams

Die AWS globale Infrastruktur basiert auf AWS Regionen und Availability Zones. AWS Regionen bieten mehrere physisch getrennte und isolierte Availability Zones, die über Netzwerke mit niedriger Latenz, hohem Durchsatz und hoher Redundanz miteinander verbunden sind. Mithilfe von Availability Zones können Sie Anwendungen und Datenbanken erstellen und ausführen, die automatisch Failover zwischen Availability Zones ausführen, ohne dass es zu Unterbrechungen kommt. Availability Zones sind besser hoch verfügbar, fehlertoleranter und skalierbarer als herkömmliche Infrastrukturen mit einem oder mehreren Rechenzentren.

Weitere Informationen zu AWS Regionen und Availability Zones finden Sie unter AWS Globale Infrastruktur.

Zusätzlich zur AWS globalen Infrastruktur bietet Kinesis Data Streams mehrere Funktionen, die Sie bei der Unterstützung Ihrer Datenausfallsicherheit und Ihrer Backup-Anforderungen unterstützen.

Notfallwiederherstellung in Amazon Kinesis Data Streams

Wenn Sie Daten aus einem Stream mit einer Anwendung von Amazon Kinesis Data Streams verarbeiten, sind Ausfälle auf folgenden Ebenen möglich:

- Ausfall eines Datensatzprozessors
- Ausfall eines Auftragnehmers oder einer Instance der Anwendung, die den Auftragnehmer instanziiert hat
- Eine EC2 Instanz, die eine oder mehrere Instanzen der Anwendung hostet, könnte ausfallen

Prozessorfehler aufzeichnen

Der Worker ruft mithilfe von ExecutorServiceJava-Aufgaben Methoden der Datensatzverarbeitung auf. Schlägt die Ausführung einer Aufgabe fehlt, behält der Auftragnehmer die Kontrolle über den Shard, den der Datensatzprozessor verarbeitet hat. Der Auftragnehmer startet eine neue

Datensatzprozessoraufgabe für die Verarbeitung des Shards. Weitere Informationen finden Sie unter Drosselung lesen.

Fehler beim Worker oder bei der Anwendung

Wenn ein Worker oder eine Instance der Anwendung Amazon Kinesis Data Streams ausfällt, sollten Sie die Situation erkennen können und angemessen reagieren. Löst beispielsweise die `Worker.run`-Methode eine Ausnahme aus, müssen Sie diese abfangen und verwalten.

Fällt die Anwendung selbst aus, sollten Sie dies erkennen, damit Sie einen Neustart durchführen können. Wenn die Anwendung gestartet wird, instanziiert sie einen neuen Auftragnehmer, der wiederum neue Datensatzprozessoren instanziiert, denen automatisch Shards zur Verarbeitung zugewiesen werden. Dies können dieselben Shards sein, die der Datensatzprozessor vor dem Ausfall verarbeitet hat, oder neue Shards.

In einer Situation, in der der Worker oder die Anwendung ausfällt, der Fehler nicht erkannt wird und andere Instanzen der Anwendung auf anderen EC2 Instanzen ausgeführt werden, behandeln die Worker auf diesen anderen Instanzen den Fehler. Sie erstellen weitere Datensatzprozessoren für die Verarbeitung der Shards des ausgefallenen Auftragnehmers. Die Belastung dieser anderen EC2 Instanzen nimmt entsprechend zu.

Das hier beschriebene Szenario geht davon aus, dass der Worker oder die Anwendung zwar ausgefallen ist, die EC2 Hosting-Instance jedoch weiterhin läuft und daher nicht von einer Auto Scaling Scaling-Gruppe neu gestartet wird.

Ausfall der EC2 Amazon-Instanz

Wir empfehlen, dass Sie die EC2 Instances für Ihre Anwendung in einer Auto Scaling Scaling-Gruppe ausführen. Auf diese Weise startet die Auto Scaling Scaling-Gruppe automatisch eine neue Instance, um sie zu ersetzen, wenn eine der EC2 Instances ausfällt. Sie sollten die Instances so konfigurieren, dass Ihre Anwendung Amazon Kinesis Data Streams beim Start gestartet wird.

Infrastruktursicherheit in Kinesis Data Streams

Als verwalteter Service ist Amazon Kinesis Data Streams durch die AWS globalen Netzwerksicherheitsverfahren geschützt, die im Whitepaper Amazon Web Services: Sicherheitsprozesse im Überblick beschrieben werden.

Sie verwenden AWS veröffentlichte API Aufrufe, um über das Netzwerk auf Kinesis Data Streams zuzugreifen. Clients müssen Transport Layer Security (TLS) 1.2 oder höher unterstützen. Clients

müssen außerdem Cipher Suites mit Perfect Forward Secrecy (PFS) wie Ephemeral Diffie-Hellman () oder Elliptic Curve Ephemeral Diffie-Hellman (DHE) unterstützen. ECDHE Die meisten modernen Systeme wie Java 7 und höher unterstützen diese Modi.

Darüber hinaus müssen Anfragen mithilfe einer Zugriffsschlüssel-ID und eines geheimen Zugriffsschlüssels, der einem Prinzipal zugeordnet ist, signiert werden. IAM Alternativ können Sie mit AWS Security Token Service (AWS STS) temporäre Sicherheitsanmeldeinformationen erstellen, um die Anforderungen zu signieren.

Bewährte Sicherheitsmethoden für Kinesis Data Streams

Amazon Kinesis Data Streams enthält eine Reihe von Sicherheitsfunktionen, die Sie bei der Entwicklung und Implementierung Ihrer eigenen Sicherheitsrichtlinien berücksichtigen sollten. Die folgenden bewährten Methoden sind allgemeine Richtlinien und keine vollständige Sicherheitslösung. Da diese bewährten Methoden für Ihre Umgebung möglicherweise nicht angemessen oder ausreichend sind, sollten Sie sie als hilfreiche Überlegungen und nicht als bindend ansehen.

Implementieren des Zugriffs mit geringsten Berechtigungen

Beim Erteilen von Berechtigungen entscheiden Sie, wer welche Berechtigungen für welche Ressourcen von Kinesis Data Streams erhält. Sie aktivieren die spezifischen Aktionen, die daraufhin für die betreffenden Ressourcen erlaubt sein sollen. Aus diesem Grund sollten Sie nur Berechtigungen gewähren, die zum Ausführen einer Aufgabe erforderlich sind. Die Implementierung der geringstmöglichen Zugriffsrechte ist eine grundlegende Voraussetzung zum Reduzieren des Sicherheitsrisikos und der Auswirkungen, die aufgrund von Fehlern oder böswilligen Absichten entstehen könnten.

Verwenden Sie Rollen IAM

Produzenten- und Client-Anwendungen müssen über gültige Anmeldeinformationen für den Zugriff auf Kinesis-Datenströme verfügen. Sie sollten AWS Anmeldeinformationen nicht direkt in einer Client-Anwendung oder in einem Amazon S3 S3-Bucket speichern. Dabei handelt es sich um langfristige Anmeldeinformationen, die nicht automatisch rotiert werden und bedeutende geschäftliche Auswirkungen haben könnten, wenn sie kompromittiert werden.

Stattdessen sollten Sie eine IAM Rolle verwenden, um temporäre Anmeldeinformationen für Ihre Produzenten- und Client-Anwendungen für den Zugriff auf Kinesis-Datenstreams zu verwalten. Wenn Sie eine Rolle verwenden, müssen Sie keine langfristigen Anmeldeinformationen (z. B.

Benutzername und Passwort oder Zugriffsschlüssel) für den Zugriff auf andere Ressourcen verwenden.

Weitere Informationen finden Sie in den folgenden Themen im IAMBenutzerhandbuch:

- IAMRollen
- Gängige Szenarien für Rollen: Benutzer, Anwendungen und Services

Implementieren Sie serverseitige Verschlüsselung in abhängigen Ressourcen

Daten im Ruhezustand und Daten während der Übertragung können in Kinesis Data Streams verschlüsselt werden. Weitere Informationen finden Sie unter Datenschutz in Amazon Kinesis Data Streams.

Wird zur Überwachung CloudTrail von Anrufen API verwendet

Kinesis Data Streams ist in einen Dienst integriert AWS CloudTrail, der eine Aufzeichnung der Aktionen bereitstellt, die von einem Benutzer, einer Rolle oder einem AWS Dienst in Kinesis Data Streams ausgeführt wurden.

Anhand der von gesammelten Informationen können Sie die Anfrage CloudTrail, die an Kinesis Data Streams gestellt wurde, die IP-Adresse, von der aus die Anfrage gestellt wurde, wer die Anfrage gestellt hat, wann sie gestellt wurde, und weitere Details ermitteln.

Weitere Informationen finden Sie unter the section called “Amazon Kinesis Data Streams API Streams-Aufrufe protokollieren mit AWS CloudTrail”.

Dokumentverlauf

Die folgende Tabelle enthält wichtige Änderungen an der Dokumentation zu Amazon Kinesis Data Streams.

Änderung	Beschreibung	Änderungsdatum
Unterstützung für die gemeinsame Nutzung von Datenströmen zwischen Konten hinzugefügt.	Teilen Sie Ihren Datenstream mit einem anderen Konto hinzugefügt.	22. November 2023
Unterstützung für die Kapazitätsmodi „On-Demand“ und „Bereitgestellter Datenstrom“ wurde hinzugefügt.	Wählen Sie den Datenstream-Kapazitätsmodus hinzugefügt.	29. November 2021
Neue Inhalte für die serverseitige Verschlüsselung.	Datenschutz in Amazon Kinesis Data Streams hinzugefügt.	7. Juli 2017
Neue Inhalte für verbesserte CloudWatch Metriken.	Aktualisiert Kinesis Data Streams überwachen.	19. April 2016
Neue Inhalte zur Erweiterung von Kinesis-Agenten.	Aktualisiert Schreiben Sie mit Kinesis Agent in Amazon Kinesis Data Streams.	11. April 2016
Neue Inhalte für die Nutzung von Kinesis-Agenten.	Schreiben Sie mit Kinesis Agent in Amazon Kinesis Data Streams hinzugefügt.	2. Oktober 2015

Änderung	Beschreibung	Änderungsdatum
KPLInhalt für Version 0.10.0 aktualisieren.	Entwickeln Sie Produzenten mithilfe der Amazon Kinesis Producer Library () KPL hinzugefügt.	15. Juli 2015
Aktualisiere das Thema KCL Metriken für konfigurierbare Metriken.	Überwachen Sie die Kinesis-Clientbibliothek mit Amazon CloudWatch hinzugefügt.	9. Juli 2015
Inhalte neu organisiert.	Wichtige Neuorganisation von Inhalten für einen bessere Strukturansicht und eine logischere Gruppierung.	01. Juli 2015
Neues Thema im KPL Entwickle rhandbuch.	Entwickeln Sie Produzenten mithilfe der Amazon Kinesis Producer Library () KPL hinzugefügt.	02. Juni 2015
Neues Thema zu KCL Metriken.	Überwachen Sie die Kinesis-Clientbibliothek mit Amazon CloudWatch hinzugefügt.	19. Mai 2015
Support fürKCL. NET	Entwickeln Sie einen Kinesis Client Library-Benutzer in. NET hinzugefügt.	1. Mai 2015
Support für KCL Node.js	Entwickeln Sie einen Kinesis Client Library-Consumer in Node.js hinzugefügt.	26. März 2015
Support für KCL Ruby	Links zur KCL Ruby-Bibliothek hinzugefügt.	12. Januar 2015
Neu API PutRecord s	Es wurden Informationen zu New PutRecords API to hinzugefügtthe section called “Fügen Sie mehrere Datensätze hinzu mit PutRecords”.	15. Dezember 2014
Support für Markierungen	Kennzeichnen Sie Ihre Streams in Amazon Kinesis Data Streams hinzugefügt.	11. September 2014

Änderung	Beschreibung	Änderungsdatum
Neue CloudWatch Metrik	`GetRecords.IteratorAgeMilliseconds` - Metrik zu Dimensionen und Metriken von Amazon Kinesis Data Streams hinzugefügt.	3. September 2014
Neues Kapitel zur Überwachung	Kinesis Data Streams überwachen und Überwachen Sie den Amazon Kinesis Data Streams Streams-Service mit Amazon CloudWatch hinzugefügt.	30. Juli 2014
Standard-Shard-Limit	Kontingente und -Einschränkungen aktualisiert: Das Standard-Shard-Limit wurde von 5 auf 10 erhöht.	25. Februar 2014
Standard-Shard-Limit	Kontingente und -Einschränkungen aktualisiert: Das Standard-Shard-Limit wurde von 2 auf 5 erhöht.	28. Januar 2014
APIVersionsupdates	Updates für Version 2013-12-02 der Kinesis Data Streams. API	12. Dezember 2013
Erstversion	Erstveröffentlichung des Entwicklerhandbuchs für Amazon Kinesis.	14. November 2013

Die vorliegende Übersetzung wurde maschinell erstellt. Im Falle eines Konflikts oder eines Widerspruchs zwischen dieser übersetzten Fassung und der englischen Fassung (einschließlich infolge von Verzögerungen bei der Übersetzung) ist die englische Fassung maßgeblich.