aws

Guía para desarrolladores

Amazon Kinesis Data Streams

Copyright © 2024 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon Kinesis Data Streams: Guía para desarrolladores

Copyright © 2024 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Las marcas comerciales y la imagen comercial de Amazon no se pueden utilizar en relación con ningún producto o servicio que no sea de Amazon, de ninguna manera que pueda causar confusión entre los clientes y que menosprecie o desacredite a Amazon. Todas las demás marcas comerciales que no son propiedad de Amazon son propiedad de sus respectivos propietarios, que pueden o no estar afiliados, relacionados o patrocinados por Amazon.

Table of Contents

¿Qué es Amazon Kinesis Data Streams?

Puede utilizar Amazon Kinesis Data Streams para recopilar y procesar grandes flujos de registros de datos en tiempo real. Puede crear aplicaciones de procesamiento de datos, conocidas como aplicaciones de Kinesis Data Streams. Una aplicación típica de Kinesis Data Streams lee datos de un flujo de datos como registros de datos. Estas aplicaciones pueden utilizar Kinesis Client Library y ejecutarse en instancias de Amazon EC2. Puede enviar los registros procesados a paneles de control, utilizarlos para generar alertas, cambiar dinámicamente las estrategias de precios y publicidad o enviar datos a una variedad de otros servicios de AWS . Para obtener información sobre las características y los precios de Kinesis Data Streams, consulte Amazon Kinesis Data Streams.

Kinesis Data Streams forma parte de la plataforma de transmisión de datos Kinesis, junto con Firehose, Kinesis Video Streams y Managed Service for Apache Flink.

Para obtener más información sobre las soluciones de AWS big data, consulte Big Data en. AWS
Para más información sobre las soluciones de datos de streaming de AWS , consulte ¿Qué son los datos de streaming?.

Temas

¿Qué puedo hacer con Kinesis Data Streams?

Puede utilizar Kinesis Data Streams para la ingesta y agregación rápidas y continuas de datos. El tipo de datos utilizado puede incluir datos de registros de infraestructura de TI, registros de aplicaciones, redes sociales, fuentes de datos de mercado y datos de secuencias de clics en sitios web. Dado que el tiempo de respuesta necesario para la admisión y el procesamiento de datos es en tiempo real, el procesamiento suele ser ligero.

Los siguientes son escenarios típicos de uso de Kinesis Data Streams:

Admisión y procesamiento acelerados de transmisiones de datos y registros

> Puede hacer que los generadores de datos los inserten directamente en una secuencia. Por ejemplo, se pueden enviar logs del sistema y de las aplicaciones, y estarán disponibles para su

procesamiento en cuestión de segundos. Eso evita que los datos registrados se pierdan si se produce un error en el front-end o en el servidor de la aplicación. Kinesis Data Streams permite la admisión acelerada de los datos, ya que no es necesario acumular lotes de datos en los servidores antes de enviarlos.

Métricas e informes en tiempo real

Puede utilizar los datos recopilados en Kinesis Data Streams para el análisis y la elaboración de informes sencillos en tiempo real. Por ejemplo, su aplicación de procesamiento de datos puede utilizar las métricas y los análisis de los registros de sistemas y aplicaciones a medida que entran los datos del streaming, en lugar de esperar a que lleguen lotes completos de datos.

Análisis de datos en tiempo real

Así se combina la eficacia del procesamiento paralelo con el valor de los datos en tiempo real. Por ejemplo, procese secuencias de clics de sitios web en tiempo real y, a continuación, analice el compromiso de usabilidad del sitio mediante varias aplicaciones diferentes de Kinesis Data Streams ejecutadas en paralelo.

Procesamiento de secuencias complejas

Puede crear grafos acíclicos dirigidos (DAG) de aplicaciones y flujos de datos de Kinesis Data Streams. Por lo general, esto implica colocar datos de varias aplicaciones de Kinesis Data Streams en otro flujo para su procesamiento posterior por una aplicación de Kinesis Data Streams diferente.

Ventajas del uso de Kinesis Data Streams

Aunque puede utilizar Kinesis Data Streams para resolver diversos problemas de flujo de datos, un uso común es la agregación de datos en tiempo real seguida de la carga de los datos agregados en un almacén de datos o clúster MapReduce.

Los datos se colocan en flujos de datos de Kinesis, lo que garantiza su durabilidad y elasticidad. El intervalo entre el momento en que se coloca un registro en la transmisión y el momento en que se puede recuperar (put-to-get retraso) suele ser inferior a 1 segundo. En otras palabras, una aplicación de flujos de datos de Kinesis puede empezar a consumir los datos del flujo casi inmediatamente después de agregarlos. El aspecto de servicio administrado de Kinesis Data Streams elimina la carga operativa de crear y ejecuta una canalización de admisión de datos. Puede crear aplicaciones de flujo de tipo MapReduce. La elasticidad de Kinesis Data Streams le permite escalar el flujo hacia arriba o hacia abajo, de modo que nunca pierda registros de datos antes de su vencimiento.

Varias aplicaciones de Kinesis Data Streams pueden consumir datos de un flujo, de forma que varias acciones, como el archivado y el procesamiento, puedan tener lugar de forma simultánea e independiente. Por ejemplo, dos aplicaciones pueden leer datos de la misma secuencia. La primera aplicación calcula agregados en ejecución y actualiza una tabla de Amazon DynamoDB, y la segunda aplicación comprime y archiva los datos en un almacén de datos como Amazon Simple Storage Service (Amazon S3). A continuación, un panel lee la tabla de DynamoDB con los agregados en ejecución para obtener informes. up-to-the-minute

Kinesis Client Library permite el consumo tolerante a fallos de datos de flujos y proporciona soporte de escalado para aplicaciones de Kinesis Data Streams.

Servicios relacionados

Para más información sobre el uso de clústeres de Amazon EMR para leer y procesar flujos de datos de Kinesis directamente, consulte Kinesis Connector.

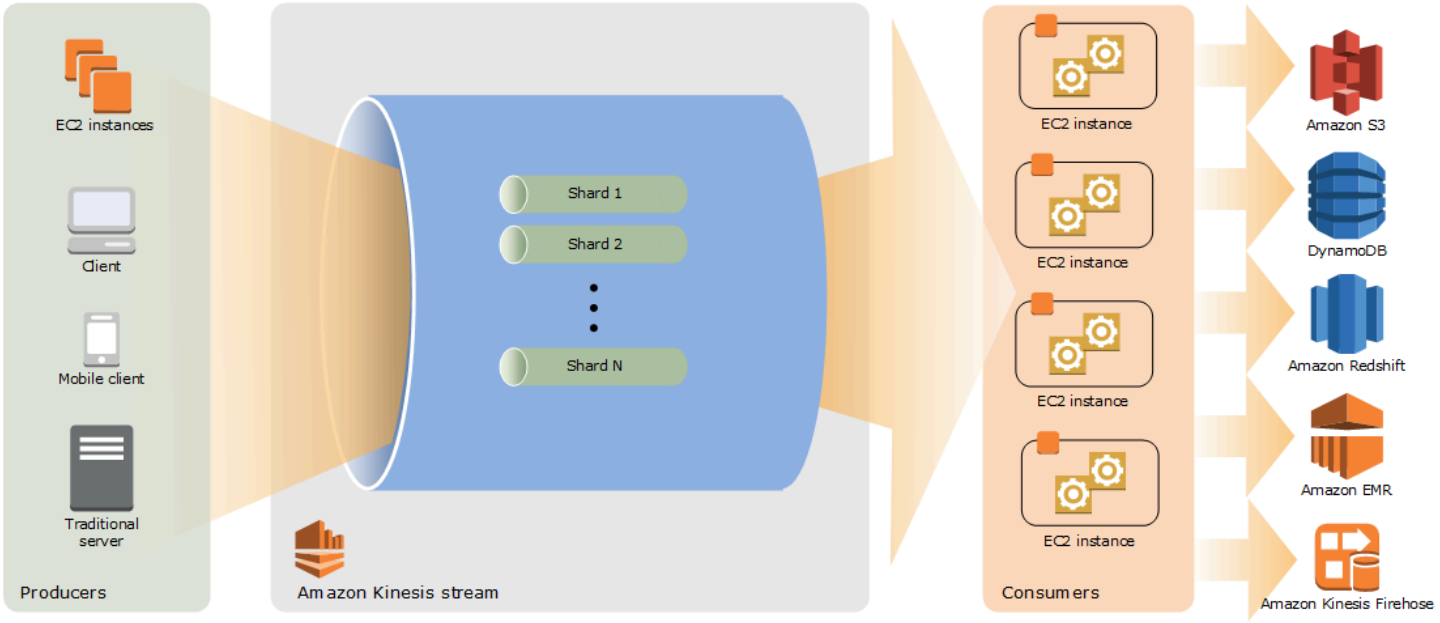
Terminología y conceptos de Amazon Kinesis Data Streams

A medida que utilice Amazon Kinesis Data Streams irá comprendiendo su arquitectura y terminología.

Temas

Arquitectura de alto nivel de Kinesis Data Streams

El siguiente diagrama ilustra la arquitectura de alto nivel de Kinesis Data Streams. Los productores envían datos continuamente a Kinesis Data Streams, mientras que los consumidores los procesan en tiempo real. Los consumidores (como una aplicación personalizada que se ejecute en Amazon EC2 o una transmisión de entrega de Amazon Data Firehose) pueden almacenar sus resultados mediante un AWS servicio como Amazon DynamoDB, Amazon Redshift o Amazon S3.



Terminología de Kinesis Data Streams

Kinesis Data Streams

Un flujo de datos de Kinesis es un conjunto de particiones. Cada fragmento tiene una secuencia de registros de datos. A su vez, cada registro de datos tiene un número de secuencia que asigna Kinesis Data Streams.

Registro de datos

Un registro de datos es la unidad de datos que se almacena en un flujo de datos de Kinesis. Los registros de datos se componen de un número de secuencia, una clave de partición y un blob de datos, que es una secuencia inmutable de bytes. Kinesis Data Streams no inspecciona, interpreta ni cambia los datos del blob de ninguna manera. Un blob de datos puede ser de hasta 1 MB.

Modo de capacidad

El modo de capacidad de un flujo de datos determina cómo se administra la capacidad y cómo se le cobra por el uso del flujo de datos. Actualmente, en Kinesis Data Streams, puede elegir entre un modo bajo demanda y un modo aprovisionado para sus transmisiones de datos. Para obtener más información, consulte Elegir el modo de capacidad del flujo de datos.

Con el modo bajo demanda, Kinesis Data Streams administra automáticamente las particiones para proporcionar el rendimiento necesario. Solo se le cobrará por el rendimiento real que utilice y Kinesis Data Streams se adapta automáticamente a las necesidades de rendimiento de las cargas de trabajo a medida que aumentan o disminuyen. Para obtener más información, consulte Modo bajo demanda.

Con el modo aprovisionado, debe especificar el número de particiones del flujo de datos. La capacidad total de un flujo de datos es la suma de las capacidades de las particiones. Puede aumentar o disminuir el número de particiones de un flujo de datos según sea necesario y se le cobrará una tarifa por hora según el número de particiones. Para obtener más información, consulte Modo aprovisionado.

Periodo de retención

El periodo de retención es el tiempo durante el cual se puede obtener acceso a los registros de datos después de que se agreguen a la secuencia. El periodo de retención de una secuencia se establece en un valor predeterminado de 24 horas tras su creación. Puede aumentar el período de retención

hasta 8760 horas (365 días) con la IncreaseStreamRetentionPeriodoperación y reducir el período de retención hasta un mínimo de 24 horas con la operación. DecreaseStreamRetentionPeriod Para las secuencias con periodos de retención superiores a 24 horas se aplican cargos adicionales. Para obtener más información, consulte los precios de Amazon Kinesis Data Streams.

Productor

Los productores colocan los registros en Amazon Kinesis Data Streams. Por ejemplo, un servidor web que envía datos de registro a una secuencia es un productor.

Consumidor

Los consumidores obtienen los registros de Amazon Kinesis Data Streams y los procesan. Estos consumidores se denominan Aplicación Amazon Kinesis Data Streams.

Aplicación Amazon Kinesis Data Streams

Una aplicación de Amazon Kinesis Data Streams es consumidor de un flujo que normalmente se ejecuta en una flota de instancias de EC2.

Es posible desarrollar dos tipos de consumidores: consumidores de distribución ramificada compartida y consumidores de distribución ramificada mejorada. Para conocer las diferencias entre ellos y saber cómo crear cada tipo de consumidor, consulte Lectura de datos de Amazon Kinesis Data Streams.

La salida de una aplicación de Kinesis Data Streams se puede utilizar como entrada para otro flujo, lo que permitirá crear topologías complejas que procesan datos en tiempo real. Una aplicación también puede enviar datos a una variedad de otros AWS servicios. Puede haber varias aplicaciones para una secuencia, y cada aplicación puede consumir datos procedentes de la secuencia de forma independiente y de forma simultánea.

Partición

Un fragmento es una sucesión de registros de datos identificados inequívocamente en una secuencia. Una secuencia consta de uno o varios fragmentos, cada uno de los cuales proporciona una unidad de capacidad fija. Cada partición puede admitir hasta 5 transacciones por segundo para las lecturas, hasta una velocidad máxima total de lectura de datos de 2 MB por segundo y hasta 1000 registros por segundo para las escrituras, hasta una velocidad máxima de escritura de datos total de 1 MB por segundo (incluidas las claves de partición). La capacidad de los datos

de la secuencia es una función del número de fragmentos que especifique para la secuencia. La capacidad total de la secuencia es la suma de las capacidades de sus fragmentos.

Si su velocidad de transferencia de datos aumenta, puede incrementar o reducir el número de fragmentos asignados a su secuencia. Para obtener más información, consulte Cambio de los fragmentos de una secuencia.

Clave de partición

Una clave de partición se utiliza para agrupar datos por partición en un flujo. Kinesis Data Streams divide los registros de datos que pertenecen a un flujo en varias particiones. Utiliza la clave de partición asociada a cada registro de datos para determinar a qué fragmento pertenece un registro de datos determinado. Las claves de partición son cadenas Unicode, con un límite de longitud máxima de 256 caracteres para cada clave. Se utiliza una función hash MD5 para asignar claves de partición a valores enteros de 128 bits y para asignar registros de datos asociados a fragmentos utilizando los rangos de claves hash de los fragmentos. Cuando una aplicación pone los datos en una secuencia, debe especificarse una clave de partición.

Sequence Number

Cada registro de datos tiene un número de secuencia único por clave de partición en su partición. Kinesis Data Streams asigna el número de secuencia después escribir en el flujo con `client.putRecords` o `client.putRecord`. Por lo general, los números secuenciales de una misma clave de partición aumentan con el paso del tiempo. Cuanto más largo sea el período de tiempo entre las solicitudes de escritura, mayores serán los números secuenciales.

> **Note**
>
> Los números secuenciales no se pueden utilizar como índices de conjuntos de datos dentro de la misma secuencia. Para separar lógicamente conjuntos de datos, utilice claves de partición o cree una secuencia independiente para cada conjunto de datos.

Kinesis Client Library

Kinesis Client Library se compila en la aplicación para permitir un consumo de datos tolerante a errores desde el flujo. Kinesis Client Library garantiza que para cada partición haya un procesador de registros que la ejecute y la procese. La biblioteca también simplifica la lectura de los datos desde la

secuencia. Kinesis Client Library utiliza una tabla de Amazon DynamoDB para almacenar los datos de control. Crea una tabla por cada aplicación que procesa datos.

Hay dos versiones principales de Kinesis Client Library. Deberá elegir una u otra en función del tipo de consumidor que desee crear. Para obtener más información, consulte Lectura de datos de Amazon Kinesis Data Streams.

Nombre de la aplicación

El nombre de una aplicación de Amazon Kinesis Data Streams la identifica. Cada una de sus aplicaciones debe tener un nombre único que corresponda a la AWS cuenta y la región utilizadas por la aplicación. Este nombre se utiliza como nombre para la tabla de control en Amazon DynamoDB y el espacio de nombres para las métricas de Amazon. CloudWatch

Cifrado del servidor

Amazon Kinesis Data Streams puede cifrar la información confidencial automáticamente cuando un productor la introduce en un flujo. Kinesis Data Streams utiliza claves maestras de AWS KMS para el cifrado. Para obtener más información, consulte Protección de datos en Amazon Kinesis Data Streams.

> **Note**
>
> Para leer o escribir en una secuencia cifrada, las aplicaciones productoras y consumidoras deben tener permiso para obtener acceso a la clave maestra. Para obtener más información sobre la concesión de permisos para aplicaciones productoras y consumidoras, consulte the section called “Permisos de para utilizar claves maestras de KMS generadas por el usuario”.

> **Note**
>
> El uso del cifrado del lado del servidor conlleva () costes. AWS Key Management Service AWS KMSPara obtener más información, consulte Precios deAWS Key Management Service.

Cuotas y límites

Amazon Kinesis Data Streams tiene las siguientes cuotas y límites de particiones y flujos.

Cuota	Modo bajo demanda	Modo aprovisionado
Número de flujos de datos	No existe una cuota máxima en el número de flujos de la cuenta de AWS. De forma predeterminada, puede crear hasta 50 flujos de datos con el modo de capacidad bajo demanda. Si necesita aumentar esta cuota, envíe un ticket de soporte.	No hay una cuota máxima en el número de flujos con el modo aprovisionado en una cuenta.
Número de fragmentos	No hay límite superior. La cantidad de particiones depende de la cantidad de datos ingeridos y del nivel de rendimiento que necesite. Kinesis Data Streams escala automáticamente el número de particiones en respuesta a los cambios en el volumen y el tráfico de datos.	No hay límite superior. El límite predeterminado es de 500 particiones por cuenta de AWS en las siguientes regiones de AWS: Este de EE. UU. (Norte de Virginia), Oeste de EE. UU. (Oregón) y Europa (Irlanda). Para las demás regiones, la cuota de particiones predeterminada es de 200 particiones por cuenta de AWS. Para solicitar un aumento de la cuota de particiones por flujo de datos, consulte Requesting a Quota Increase.
Rendimiento del flujo de datos	De forma predeterminada, los nuevos flujos de datos creados con el modo de capacidad bajo demanda	No hay límite superior. El rendimiento máximo depende de la cantidad de particiones aprovisionadas para el flujo.

Cuota	Modo bajo demanda	Modo aprovisionado
	tienen 4 MB/s de rendimiento de escritura y 8 MB/s de lectura. A medida que aumenta el tráfico, los flujos de datos con el modo de capacidad bajo demanda se escalan verticalmente hasta 200 MB/s de rendimiento de escritura y 400 MB/s de rendimiento de lectura. Si necesita aumentar la capacidad de escritura a 2 Gb/s y la capacidad de lectura a 4 Gb/s, envíe un ticket de soporte.	Cada partición puede admitir un rendimiento de escritura de hasta 1 MB/s o 1000 registros/s o un rendimiento de lectura de hasta 2 MB/s o 2000 registros/s. Si necesita más capacidad de adquisición, puede ampliar fácilmente el número de fragmentos de la secuencia mediante la AWS Management Console o la API UpdateShardCount.
Volumen de la carga de datos	El volumen máximo de carga de datos de un registro antes de `base64-encoding` es de 1 MB.	
Volumen de transacción de `GetRecords`	GetRecords puede recuperar hasta 10 MB de datos por llamada desde un único fragmento y hasta 10 000 registros por llamada. Cada llamada a `GetRecords` se contabiliza como una transacción de lectura. Cada fragmento puede admitir hasta cinco transacciones de lectura por segundo. Cada transacción de lectura puede proporcionar hasta 10 000 registros con una cuota superior de 10 MB por transacción.	
Velocidad de lectura de datos por partición	Cada fragmento puede admitir como máximo una velocidad de lectura de datos total de 2 MB por segundo a través de GetRecords. Si una llamada a `GetRecords` devuelve 10 MB, las llamadas posteriores que se realicen en los siguientes 5 segundos generarán una excepción.	
Número de consumidores registrados por flujo de datos	Puede crear hasta 20 consumidores registrados (límite de distribución mejorado) para cada flujo de datos.	

Cuota	Modo bajo demanda	Modo aprovisionado
Cambiar entre los modos aprovisionado y bajo demanda	Todos los flujos de datos de su cuenta de AWS permiten que se cambie entre el modo de capacidad bajo demanda y el modo de capacidad aprovisionado dos veces en un plazo de 24 horas.	

Límites de API

Al igual que la mayoría de las API de AWS, las operaciones de la API de Kinesis Data Streams tienen límites de velocidad. Los siguientes límites se aplican por cuenta y región de AWS. Para obtener más información sobre las API de Kinesis Data Streams, consulte la Referencia de la API de Amazon Kinesis.

Límites de las API del plano de control de KDS

En la siguiente sección se describen los límites de las API del plano de control de KDS. Las API del plano de control de KDS le permiten crear y administrar sus secuencias de datos. Estos límites se aplican por cuenta y región de AWS.

Límites de las API del plano de control

API	Límite de llamadas a la API	Por cuenta/flujo	Descripción
AddTagsToStream	5 transacciones por segundo (TPS)	Por flujo	50 etiquetas por flujo de datos
CreateStream	5 TPS	Por cuenta	No existe una cuota máxima en el número de secuencias que se pueden tener en una cuenta. Obtendrá una `LimitExceededException` al realizar una solicitud `CreateStream` cuando intente

API	Límite de llamadas a la API	Por cuenta/flujo	Descripción
			realizar una de las siguientes acciones: • Tener más de cinco secuencias en el estado `CREATING` en un momento dado. • Crear más particion es de los autorizad os para su cuenta.
DecreaseStreamRetentionPeriod	5 TPS	Por flujo	El valor mínimo del período de retención de una secuencia de datos es de 24 horas.
DeleteResourcePolicy	5 TPS	Por cuenta	Si necesita aumentar este límite, envíe un ticket de soporte.
DeleteStream	5 TPS	Por cuenta	
DeregisterStreamConsumer	5 TPS	Por flujo	
DescribeLimits	1 TPS	Por cuenta	
DescribeStream	10 TPS	Por cuenta	
DescribeStreamConsumer	20 TPS	Por flujo	
DescribeStreamSummary	20 TPS	Por cuenta	

API	Límite de llamadas a la API	Por cuenta/flujo	Descripción
DisableEnhancedMonitoring	5 TPS	Por flujo	
EnableEnhancedMonitoring	5 TPS	Por flujo	
GetResourcePolicy	5 TPS	Por cuenta	Si necesita aumentar este límite, envíe un ticket de soporte.
IncreaseStreamRetentionPeriod	5 TPS	Por flujo	El valor máximo del periodo de retención de un flujo es de 8760 horas (365 días).
ListShards	1000 TPS	Por flujo	
ListStreamConsumers	5 TPS	Por flujo	
ListStreams	5 TPS	Por cuenta	
ListTagsForStream	5 TPS	Por flujo	
MergeShards	5 TPS	Por flujo	Solo se aplica a los aprovisionados.
PutResourcePolicy	5 TPS	Por cuenta	Si necesita aumentar este límite, envíe un ticket de soporte.

API	Límite de llamadas a la API	Por cuenta/flujo	Descripción
RegisterStreamConsumer	5 TPS	Por flujo	Puede registrar hasta 20 consumidores por secuencia de datos. Un consumidor determinado solo se puede registrar en una secuencia de datos a la vez. Solo se pueden crear cinco consumidores simultáneamente. En otras palabras, no puede tener más de 5 consumidores en un estado `CREATING` al mismo tiempo. Registro de un sexto consumidor mientras haya 5 en un estado `CREATING`
RemoveTagsFromStream	5 TPS	Por flujo	
SplitShard	5 TPS	Por flujo	Solo aplicable a los aprovisionados
StartStreamEncryption		Por flujo	Puede aplicar una nueva clave de AWS KMS para el cifrado del servidor 25 veces en un periodo de 24 horas.

API	Límite de llamadas a la API	Por cuenta/flujo	Descripción
StopStreamEncryption		Por flujo	Puede deshabilitar correctamente el cifrado del lado del servidor 25 veces en un período de 24 horas sucesivas.
UpdateShardCount		Por flujo	Solo se aplica a los aprovisionados. El límite predeterminado de número de particiones es de 10 000. Hay límites adicionales en esta API. Para obtener más información, consulte UpdateShardCount.
UpdateStreamMode		Por flujo	Todos los flujos de datos de su cuenta de AWS permiten que se cambie entre el modo de capacidad bajo demanda y el modo de capacidad aprovisionado dos veces en un plazo de 24 horas.

Límites de las API del plano de datos de KDS

En la siguiente sección se describen los límites de las API del plano de datos de KDS. Las API del plano de datos de KDS le permiten utilizar sus secuencias de datos para recopilar y procesar registros de datos en tiempo real. Estos límites se aplican por partición en sus secuencias de datos.

Límites de las API del plano de datos

API	Límite de llamadas a la API	Límite de carga	Detalles adicionales
GetRecords	5 TPS	El número máximo de registros que se pueden devolver por llamada es de 10 000. El volumen máximo de datos que `GetRecords` puede devolver es de 10 MB.	Si una llamada devuelve esta cantidad de datos, las llamadas posteriores realizadas en los siguientes cinco segundos generan `ProvisionedThroughputExceededException`. Si no hay suficiente rendimiento aprovisionado en el flujo, las llamadas posteriores realizadas en el siguiente segundo generan `ProvisionedThroughputExceededException`.
GetShardIterator	5 TPS		Un iterador de particiones caduca cinco minutos después de devolverse al solicitante. Si una

API	Límite de llamadas a la API	Límite de carga	Detalles adicionales
			solicitud GetShardIterator se realiza con demasiada frecuencia, recibirá ProvisionedThroughputExceededException.
PutRecord	1000 TPS	Cada partición puede admitir escrituras de hasta 1000 registros por segundo, hasta un total máximo de escritura de datos de 1 MB por segundo.	
PutRecords		Cada solicitud PutRecords puede admitir hasta 500 registros. Cada registro puede ser tan grande como 1 MB, hasta un límite de 5 MB para toda la solicitud, incluidas las claves de partición. Cada partición puede admitir escrituras de hasta 1000 registros por segundo, hasta un total máximo de escritura de datos de 1 MB por segundo.	

API	Límite de llamadas a la API	Límite de carga	Detalles adicionales
SubscribeToShard	Puede hacer una llamada a Subscribe ToShard por segundo por consumido r registrado por partición.		Si vuelve a llamar a SubscribeToShard con el mismo ConsumerARN y ShardId en los cinco segundos posterior es a una llamada correcta, obtendrá ResourceInUseExcep tion.

Aumento de las cuotas

Puede utilizar Service Quotas para solicitar un aumento de una cuota, si esta es ajustable. Algunas solicitudes se resuelven automáticamente, mientras que otras se envían a AWS Support. Puede hacer un seguimiento del estado de una solicitud de aumento de cuota que se haya enviado a AWS Support. Las solicitudes para aumentar las cuotas de servicio no reciben soporte prioritario. Si tiene una solicitud urgente, póngase en contacto con AWS Support. Para obtener más información, consulte ¿Qué es Service Quotas?

Para solicitar un aumento de la cuota de servicio, siga el procedimiento descrito en Solicitar un aumento de cuota.

Configuración de Amazon Kinesis Data Streams

Antes de usar Amazon Kinesis Data Streams por primera vez, realice las siguientes tareas.

Tareas

- Inscripción en AWS
- Descargar bibliotecas y herramientas
- Configurar el entorno de desarrollo

Inscripción en AWS

Al inscribirse en Amazon Web Services (AWS), la cuenta de AWS se inscribe automáticamente en todos los servicios de AWS, incluido Kinesis Data Streams. Solo se le cobrará por los servicios que utilice.

Si ya dispone de una cuenta de AWS, pase a la siguiente tarea. Si no dispone de una cuenta de AWS, utilice el siguiente procedimiento para crear una.

Para inscribirse en una cuenta de AWS

1. Abra https://portal.aws.amazon.com/billing/signup.
2. Siga las instrucciones en línea.

   Parte del procedimiento de inscripción consiste en recibir una llamada telefónica e indicar un código de verificación en el teclado del teléfono.

   Al registrase en una Cuenta de AWS, se crea un Usuario raíz de la cuenta de AWS. El usuario raíz tiene acceso a todos los recursos y Servicios de AWS de esa cuenta. Como práctica recomendada de seguridad, asigne acceso administrativo a un usuario administrativo y utilice únicamente el usuario raíz para realizar la ejecución tareas que requieren acceso de usuario raíz.

Descargar bibliotecas y herramientas

Las siguientes bibliotecas y herramientas le permitirán trabajar con Kinesis Data Streams:

- La Referencia de la API de Amazon Kinesis es el conjunto básico de operaciones que admite Kinesis Data Streams. Para obtener más información sobre la realización de operaciones básicas utilizando el código de Java, consulte lo siguiente:
  - Desarrollo de productores mediante la API de Amazon Kinesis Data Streams con AWS SDK for Java
  - Desarrollo de consumidores personalizados con rendimiento compartido con AWS SDK for Java
  - Creación y administración de secuencias
- Los SDK de AWS para Go, Java, JavaScript, .NET, Node.js, PHP, Python y Ruby incluyen muestras de Kinesis Data Streams, así como también compatibilidad. Si su versión de AWS SDK for Java no incluye muestras de Kinesis Data Streams, puede descargarlas de GitHub.
- Kinesis Client Library (KCL) proporciona una forma sencilla de utilizar el modelo de programación para el procesamiento de datos. KCL puede ser de ayuda para comenzar rápidamente a utilizar Kinesis Data Streams en Java, Node.js, .NET, Python y Ruby. Para obtener más información, consulte Lectura de datos de secuencias.
- AWS Command Line Interface es compatible con Kinesis Data Streams. La AWS CLI permite controlar varios servicios de AWS desde la línea de comandos y automatizarlos mediante scripts.

Configurar el entorno de desarrollo

Para utilizar KCL, asegúrese de que su entorno de desarrollo de Java cumpla los siguientes requisitos:

- Java 1.7 (Java SE 7 JDK) o posterior. Puede descargar el software de Java más reciente desde Descargas de Java SE en el sitio web de Oracle.
- Paquete de Apache Commons (código, cliente HTTP e inicio de sesión)
- Procesador Jackson JSON

Tenga en cuenta que AWS SDK for Java incluye Apache Commons y Jackson en la carpeta de terceros. Sin embargo, el SDK para Java funciona con Java 1.6, mientras que Kinesis Client Library requiere Java 1.7.

Introducción a Amazon Kinesis Data Streams

La información de esta sección lo ayuda a comenzar a utilizar Amazon Kinesis Data Streams. Si es la primera vez que utiliza Kinesis Data Streams, familiarícese antes con los conceptos y los términos que encontrará en Terminología y conceptos de Amazon Kinesis Data Streams.

En esta sección le indicamos cómo realizar operaciones básicas en Amazon Kinesis Data Streams mediante la AWS Command Line Interface. Se exponen principios fundamentales del flujo de datos en Kinesis Data Streams y los pasos necesarios para insertar y obtener datos de un flujo de datos de Kinesis.

Temas

Para acceder a la CLI, necesita un ID de clave de acceso y una clave de acceso secreta. Cuando sea posible, utilice credenciales temporales en lugar de claves de acceso. Las credenciales temporales incluyen un ID de clave de acceso y una clave de acceso secreta, pero, además, incluyen un token de seguridad que indica cuándo caducan las credenciales. Para obtener más información, consulte Uso de credenciales temporales con AWS en la Guía del usuario de IAM.

Puede encontrar instrucciones detalladas paso a paso sobre IAM y la configuración de la clave de seguridad en Creación de un usuario de IAM.

Los comandos específicos que utilizaremos en esta sección se expresarán literalmente, excepto cuando algunos valores específicos deben ser diferentes en cada ejecución. Además, los ejemplos utilizan la región Oeste de EE. UU. (Oregón), pero los pasos de esta sección funcionan en cualquiera de las regiones que admiten Kinesis Data Streams.

Instalar y configurar la AWS CLI

Instalar AWS CLI

Para ver los pasos detallados sobre cómo instalar la AWS CLI para los sistemas operativos Windows, Linux, OS X y Unix, consulte Instalación de la AWS CLI.

Utilice el siguiente comando para ver una lista de las opciones y servicios disponibles:

```
aws help
```

Va a utilizar el servicio Kinesis Data Streams, por lo que puede consultar los subcomandos de la AWS CLI relacionados con Kinesis Data Streams mediante el siguiente comando:

```
aws kinesis help
```

Este comando da como resultado los comandos de Kinesis Data Streams disponibles:

```
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
```

Esta lista de comandos se corresponde con la API de Kinesis Data Streams documentada en la Referencia de la API de Amazon Kinesis Service. Por ejemplo, el comando `create-stream` se corresponde con la acción de la API `CreateStream`.

La AWS CLI ya está instalada correctamente, pero no se ha configurado aún. Este proceso se describe en la siguiente sección.

Configurar AWS CLI

Para el uso general, el comando `aws configure` es la forma más rápida de configurar la instalación de la AWS CLI. Para obtener más información, consulte Configuración de AWS CLI.

Realizar operaciones básicas en Kinesis Data Stream con la AWS CLI

En esta sección se describe el uso básico de un flujo de datos de Kinesis desde la línea de comandos mediante la AWS CLI. Asegúrese de estar familiarizado con los conceptos que se abordan en Terminología y conceptos de Amazon Kinesis Data Streams.

> **Note**
>
> Una vez creado un flujo, su cuenta generará gastos nominales por el uso de Kinesis Data Streams, ya que Kinesis Data Streams no está disponible en el nivel gratuito de AWS. Cuando haya terminado con este tutorial, elimine sus recursos de AWS para dejar de incurrir en cargos. Para obtener más información, consulte Paso 4: Eliminación.

Temas

Paso 1: Crear una secuencia

El primer paso es crear una secuencia y verificar que se haya creado correctamente. Utilice el siguiente comando para crear una secuencia llamada "Foo":

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

A continuación, escriba el siguiente comando para comprobar el progreso de creación de la secuencia:

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

Debería obtener un resultado similar al siguiente ejemplo:

```
{
    "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
    }
}
```

En este ejemplo, la secuencia tiene el estado CREATING, lo que significa que aún no está lista para su uso. Compruébelo de nuevo en unos minutos, y debería ver un resultado parecido al siguiente ejemplo:

```
{
    "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
    }
}
```

Este resultado contiene también información de la que no debería preocuparse para el objetivo de este tutorial. El elemento principal, por el momento, es `"StreamStatus": "ACTIVE"`, que le indica que la secuencia está lista para ser utilizada, y la información en el fragmento único que ha solicitado. También puede verificar la existencia de su nuevo secuencia mediante el comando `list-streams`, tal y como se muestra aquí:

```
aws kinesis list-streams
```

Salida:

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

Paso 2: Insertar un registro

Ahora que ya tiene una secuencia activa, está listo para insertar algunos datos. En este tutorial, utilizará el comando más sencillo posible, `put-record`, que inserta un único registro de datos que contiene el texto "testdata" en la secuencia:

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

Este comando, si se ejecuta correctamente, dará como resultado algo similar a lo del siguiente ejemplo:

```
{
```

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

¡Enhorabuena, acaba de agregar datos a una secuencia! A continuación verá cómo obtener datos a partir de la secuencia.

Paso 3: Obtener el registro

GetShardIterator

Antes de poder obtener datos de la secuencia, necesita obtener el iterador de fragmentos para el fragmento que le interese. Un iterador de fragmentos representa la posición de la secuencia y el fragmento a partir de la cual realizará la lectura el consumidor (en este caso, el comando `get-record`). Utilizará el comando `get-shard-iterator`, tal y como se indica a continuación:

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

Recuerde que los comandos de `aws kinesis` utilizan la API de Kinesis Data Streams, por lo que si le interesa alguno de los parámetros que se muestran, puede obtener más información sobre ellos en el tema de referencia de la API `GetShardIterator`. Una ejecución correcta se traducirá en un resultado parecido al siguiente ejemplo (desplace la ventana hacia los lados para leer el resultado completo):

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

Esta larga cadena de caracteres aparentemente aleatorios es el iterador de fragmentos (el suyo será diferente). Tendrá que copiar y pegar el iterador de fragmentos en el comando "get" que se muestra a continuación. Los iteradores de fragmentos tienen una vida útil de 300 segundos, un tiempo que debería ser suficiente para que pueda copiar y pegar el iterador de fragmentos en el siguiente comando. Tenga en cuenta que tendrá que eliminar las líneas nuevas de su iterador de fragmentos antes de pegarlo en el siguiente comando. Si recibe un mensaje de error que informa de que el iterador de fragmentos ya no es válido, solo tiene que ejecutar de nuevo el comando `get-shard-iterator`.

GetRecords

El comando `get-records` obtiene los datos del flujo y llama a `GetRecords` en la API de Kinesis Data Streams. El iterador de fragmentos especifica la posición del fragmento desde la que quiera empezar a leer los registros de datos de forma secuencial. Si no hay registros disponibles en la parte del fragmento a la que señala el iterador, `GetRecords` devolverá una lista vacía. Tenga en cuenta que puede necesitar varias llamadas para dar con una parte del fragmento que contenga registros.

En el siguiente ejemplo del comando `get-records` (desplace la ventana hacia los lados para ver el comando completo):

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

Si ejecuta este tutorial a partir de un procesador de comandos de tipo Unix, como bash, puede automatizar la adquisición del iterador de fragmentos utilizando un comando anidado, como este (desplace la ventana hacia los lados para ver el comando completo):

```
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
```

Si ejecuta este tutorial a partir de un sistema compatible con PowerShell, puede automatizar la adquisición del iterador de fragmentos utilizando un comando como este (desplace la ventana hacia los lados para ver el comando completo):

```
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])
```

El resultado satisfactorio del comando `get-records` solicitará registros de su secuencia para el fragmento especificado al obtener el iterador de fragmentos, como en el siguiente ejemplo (desplace la ventana hacia los lados para leer el resultado completo):

```
{
  "Records":[ {
```

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

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

Tenga en cuenta que `get-records` se describe anteriormente como una solicitud, lo que significa que puede recibir cero o más registros incluso cuando haya registros en la secuencia, y los registros recibidos pueden no ser todos los que se encuentren actualmente en la secuencia. Esto es perfectamente normal, y el código de producción sondeará la secuencia en busca de registros a intervalos apropiados (esta velocidad de sondeo variará en función de los requisitos de diseño específicos de su aplicación).

Lo primero de lo que se dará cuenta acerca de su registro en esta parte del tutorial es que los datos parecen ser inútiles: no se tratará de un texto claro como el `testdata` que le enviamos. Esto se debe a la forma en la que `put-record` utiliza la codificación Base64 para permitirle enviar datos binarios. Sin embargo, la compatibilidad con Kinesis Data Streams de la AWS CLI no incluye la descodificación Base64, porque la descodificación Base64 aplicada a contenido binario sin procesar que se envía a stdout puede producir comportamientos no deseados y problemas de seguridad potenciales en determinadas plataformas y terminales. Si utiliza un descodificador Base64 (por ejemplo, https://www.base64decode.org/) para descodificar manualmente `dGVzdGRhdGE=` verá que, en realidad, es `testdata`. Esto es suficiente para los objetivos de este tutorial porque, en la práctica, la AWS CLI no suele utilizarse para consumir datos, sino más bien para monitorear el estado de la secuencia y obtener información, como se ha mostrado anteriormente (`describe-stream` y `list-streams`). En futuros tutoriales le mostraremos cómo crear aplicaciones consumidoras de calidad mediante Kinesis Client Library (KCL), en las que los procesos relacionados con Base64 se realizan en su nombre. Para más información acerca de KCL, consulte Desarrollo de consumidores personalizados con rendimiento compartido mediante KCL.

El comando `get-records` no siempre devolverá todos los registros en la secuencia/fragmento especificado. Si ocurre esto, use el `NextShardIterator` del último resultado para obtener el siguiente conjunto de registros. Por lo tanto, si se estaban insertando más datos en la secuencia (la situación normal en aplicaciones de producción), podría mantener el sondeo de datos cada

vez con `get-records`. Sin embargo, si no llama a `get-records` utilizando el siguiente iterador de fragmentos dentro del plazo de vida útil del iterador (300 segundos), obtendrá un mensaje de error y tendrá que utilizar el comando `get-shard-iterator` para obtener un nuevo iterador de fragmentos.

Además, en este resultado también se incluye `MillisBehindLatest`, que es el número de milisegundos a los que se encuentra la respuesta de la operación GetRecords del extremo del flujo, lo que indica el retraso de la aplicación consumidora con respecto al momento actual. Un valor de cero indica que el procesamiento de registros está actualizado y que no hay nuevos registros para procesar en este momento. En el caso de este tutorial, es posible que vea un número bastante grande si se ha ido tomando el tiempo de ir leyendo sobre la marcha. Eso no supone ningún problema. De manera predeterminada, los registros de datos permanecen en un flujo durante 24 horas, y podrá recuperarlos. Este periodo de tiempo se denomina periodo de retención y se puede configurar para durar hasta 365 días.

Tenga en cuenta que un resultado satisfactorio de `get-records` siempre tendrá un `NextShardIterator`, aunque no haya más registros actualmente en la secuencia. Este es un modelo de sondeo que asume que un productor puede insertar más registros en la secuencia en cualquier momento determinado. Aunque puede escribir sus propias rutinas de sondeo, si utiliza la KCL mencionada anteriormente para el desarrollo de aplicaciones consumidoras, este sondeo se realizará automáticamente.

Si llama a `get-records` hasta que no haya más registros en la secuencia y el fragmento a los que está recurriendo, verá resultados con registros vacíos similares a los del siguiente ejemplo (desplace la ventana hacia los lados para ver el resultado completo):

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

Paso 4: Eliminación

Por último, tendrá que eliminar su secuencia para liberar recursos y evitar cargos no deseados en su cuenta, como hemos advertido anteriormente. Haga esto cada vez que haya creado una secuencia y no lo vaya a usar, ya que los cargos se acumulan por cada secuencia, independientemente de si inserta o extrae datos de él o no. El comando de limpieza es sencillo:

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

Si se completa correctamente, no se obtienen resultados. Por eso, puede que prefiera usar `describe-stream` para comprobar el progreso de la eliminación:

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

Si ejecuta este comando inmediatamente después del comando eliminado, probablemente verá un resultado que en parte es parecido al siguiente ejemplo:

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

Tras eliminar por completo la secuencia, `describe-stream` devolverá un error del tipo "no encontrado":

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

Tutoriales de ejemplo de Amazon Kinesis Data Streams

Los tutoriales de ejemplo de esta sección están diseñados para comprender mejor los conceptos y la funcionalidad de Amazon Kinesis Data Streams.

Temas

Tutorial: Procesamiento de datos de operaciones bursátiles en tiempo real con KPL y KCL 2.x

El escenario que planteamos en este tutorial comprende la adquisición de operaciones bursátiles en un flujo de datos y la escritura de una aplicación de Amazon Kinesis Data Streams sencilla que realiza cálculos con dicho flujo. Aprenderá a enviar un flujo de registros a Kinesis Data Streams e implementar una aplicación que consume y procesa dichos registros casi en tiempo real.

> **Important**
>
> Una vez creado un flujo, su cuenta generará gastos nominales por el uso de Kinesis Data Streams, ya que Kinesis Data Streams no está disponible en el nivel gratuito de AWS. Una vez que se inicia la aplicación consumidora, se aplicarán cargos nominales por el uso de Amazon DynamoDB. La aplicación consumidora utiliza DynamoDB para realizar un seguimiento del estado de procesamiento. Cuando termine con esta aplicación, elimine sus recursos de AWS para dejar de incurrir en gastos. Para obtener más información, consulte Paso 7: Conclusión.

El código no obtiene acceso a datos bursátiles reales, sino que, en su lugar, simula la secuencia de operaciones bursátiles. Lo hace a través de un generador de operaciones bursátiles aleatorias que parte de datos reales del mercado para los 25 principales valores por capitalización en febrero

de 2015. Si tiene acceso a una secuencia de operaciones bursátiles en tiempo real, puede que le interese derivar estadísticas útiles y puntuales a partir de dicha secuencia. Por ejemplo, es posible que desee realizar un análisis de ventana deslizante en el que se determina el valor más popular adquirido durante los últimos 5 minutos. O también cabe la posibilidad de que quiera recibir una notificación cada vez que haya una orden de venta que sea demasiado grande (es decir, con demasiadas acciones). Puede ampliar el código de esta serie para proporcionar esta funcionalidad.

Puede seguir los pasos de este tutorial en su ordenador de escritorio o portátil y ejecutar tanto el código de productor como el de consumidor en la misma máquina o en cualquier plataforma que sea compatibles con los requisitos definidos.

Los ejemplos mostrados utilizan la región Oeste de EE. UU. (Oregón), pero funcionan en cualquiera de las regiones de AWS compatibles con Kinesis Data Streams.

Tareas

Requisitos previos

Debe cumplir los siguientes requisitos para completar este tutorial:

Cuenta de Amazon Web Services

Antes de comenzar, asegúrese de estar familiarizado con los conceptos que se abordan en Terminología y conceptos de Amazon Kinesis Data Streams, especialmente los relacionados con secuencias, fragmentos, productores y consumidores. También es útil haber completado los pasos de la siguiente guía: Instalar y configurar la AWS CLI.

Necesita una cuenta de AWS y un navegador web para obtener acceso a la AWS Management Console.

Para acceder a la consola, utilice su nombre de usuario y contraseña de IAM para iniciar sesión en AWS Management Console desde la página de inicio de sesión de IAM. Para más información sobre las credenciales de seguridad de AWS, incluido el acceso programático y las alternativas a las credenciales a largo plazo, consulte Credenciales de seguridad de AWS en la Guía del usuario de IAM. Para información sobre cómo iniciar sesión en su cuenta de Cuenta de AWS, consulte Cómo iniciar sesión en AWS en la Guía del usuario de AWS Sign-In.

Para obtener más información sobre IAM; y las instrucciones de configuración de la clave de seguridad, consulte Creación de un usuario de IAM.

Requisitos de software del sistema

El sistema que está utilizando para ejecutar la aplicación debe tener instalada la versión de Java 7 o superior. Para descargar e instalar la versión más reciente del kit de desarrollo de Java (JDK), visite el sitio web de instalación de Java SE de Oracle.

Si tiene una aplicación IDE de Java, como Eclipse, puede abrir el código fuente, editarlo, compilarlo y ejecutarlo.

Necesitará la última versión de AWS SDK for Java. Si utiliza Eclipse como IDE, puede instalar el Kit de herramientas de AWS para Eclipse en su lugar.

La aplicación de consumo requiere la versión 2.2.9 o superior de Kinesis Client Library (KCL), que se puede obtener en GitHub en https://github.com/awslabs/amazon-kinesis-client/tree/master.

Pasos siguientes

Paso 1: Crear una secuencia de datos

Paso 1: Crear una secuencia de datos

En primer lugar, debe crear la secuencia de datos que utilizará en los pasos siguientes de este tutorial.

Para crear un flujo

1. Inicie sesión en la AWS Management Console y abra la consola de Kinesis en https://console.aws.amazon.com/kinesis.
2. Elija Data Streams (Secuencias de datos) en el panel de navegación.
3. En la barra de navegación, expanda el selector de regiones y elija una región.

4. Elija Create Kinesis stream (Crear secuencia de Kinesis).
5. Escriba un nombre para la secuencia (por ejemplo **`StockTradeStream`**).
6. Introduzca **1** como número de fragmentos, pero deje Calcular el número de particiones que va a necesitar contraído.
7. Elija Create Kinesis stream (Crear secuencia de Kinesis).

En la página de lista Flujos de Kinesis, el estado del flujo aparece como `CREATING` mientras se crean los flujos. Cuando la secuencia está lista para usarse, el estado cambia a `ACTIVE`.

Si elige el nombre del flujo, en la página que aparece, la ficha Detalles muestra un resumen de la configuración del flujo de datos. La sección Monitoreo muestra información de supervisión del flujo.

Pasos siguientes

Paso 2: Crear una política y un usuario de IAM

Paso 2: Crear una política y un usuario de IAM

Las prácticas recomendadas de seguridad de AWS imponen el uso de permisos específicos para controlar el acceso a diferentes recursos. AWS Identity and Access Management (IAM) le permite administrar usuarios y permisos de usuario en AWS. Una política de IAM enumera de forma explícita las acciones que se pueden realizar y los recursos a los que se pueden aplicar dichas acciones.

A continuación se indican los permisos mínimos que suelen requerirse para los productores y consumidores de Kinesis Data Streams.

Productor

Acciones	Recurso	Finalidad
`DescribeStream` , `DescribeStreamSummary` , `DescribeStreamConsumer`	Flujo de datos de Kinesis	Antes de intentar leer registros, el consumidor comprueba s existe y si está activa, y si los fragmentos se encuentran en
`SubscribeToShard` , `RegisterStreamConsumer`	Flujo de datos de Kinesis	Suscribe y registra a los consumidores en un fragmento.

Acciones	Recurso	Finalidad
`PutRecord` , `PutRecords`	Flujo de datos de Kinesis	Escribe registros en Kinesis Data Streams.

Consumidor

Acciones	Recurso	Finalidad
`DescribeStream`	Flujo de datos de Kinesis	Antes de intentar leer registros, el consumidor comprueba s existe y si está activa, y si los fragmentos se encuentran en
`GetRecords` , `GetShardIterator`	Flujo de datos de Kinesis	Lee registros de un fragmento.
`CreateTable` , `DescribeTable` , `GetItem`, `PutItem`, `Scan`, `UpdateItem`	La tabla de Amazon DynamoDB.	Si el consumidor se desarrolla con Kinesis Client Library (K necesita permisos en una tabla de DynamoDB para realiza estado de procesamiento de la aplicación.
`DeleteItem`	La tabla de Amazon DynamoDB.	Para cuando el consumidor realiza operaciones de división de Kinesis Data Streams.
`PutMetricData`	Registros de Amazon CloudWatch	KCL también carga métricas a CloudWatch, que son útiles aplicación.

Para este tutorial, creará una única política de IAM que otorgue todos los permisos anteriores. En producción, es posible que desee crear dos políticas, una para los productores y otra para los consumidores.

Para crear una política de IAM

1. Busque el nombre de recurso de Amazon (ARN) de la nueva secuencia de datos que creó en el paso anterior. Puede encontrar este ARN como Stream ARN (ARN de la secuencia) en la parte superior de la pestaña Details (Detalles). El formato del ARN es el siguiente:

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

region

El código de la región de AWS por ejemplo, `us-west-2`. Para obtener más información, consulte Conceptos de región y zona de disponibilidad.

cuenta

El ID de la cuenta de AWS, como se muestra en Configuración de cuenta.

nombre

El nombre de la secuencia de datos que creó en el paso anterior, que es `StockTradeStream`.

2. Determine el ARN de la tabla de DynamoDB que utilizará el consumidor (y que creará la primera instancia de consumidor). Debe tener el siguiente formato:

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

La región y el ID de cuenta son idénticos a los valores del ARN del flujo de datos que se utiliza para este tutorial, pero el nombre es el nombre de la tabla de DynamoDB creada y utilizada por la aplicación del consumidor. KCL utiliza el nombre de la aplicación como nombre de tabla. En este paso, utilice `StockTradesProcessor` para el nombre de la tabla DynamoDB, ya que ese es el nombre de la aplicación utilizado en los pasos posteriores de este tutorial.

3. En la consola de IAM, en Políticas (https://console.aws.amazon.com/iam/home#policies), elija Crear política. Si es la primera vez que trabaja con políticas de IAM, elija Introducción, Crear política.
4. Elija Seleccionar junto a Generador de políticas.
5. Elija Amazon Kinesis como servicio AWS.
6. Seleccione `DescribeStream`, `GetShardIterator`, `GetRecords`, `PutRecord` y `PutRecords` como acciones permitidas.
7. Introduzca el ARN del flujo de datos que está utilizando en este tutorial.
8. Utilice Add Statement (Añadir instrucción) para cada uno de los siguientes:

Servicio de AWS	Acciones	ARN
Amazon DynamoDB	`CreateTable` , `DeleteItem` , `DescribeTable` , `GetItem`, `PutItem`, `Scan`, `UpdateItem`	El ARN de la tabla de DynamoDB creada en el paso 2 de este procedimiento.
Amazon CloudWatch	`PutMetricData`	*

El asterisco (*) que se utiliza cuando no es necesario especificar un ARN. En este caso, se debe a que no existe ningún recurso específico en CloudWatch para el que se invoque la acción `PutMetricData`.

9. Elija Next Step (Paso siguiente).
10. Cambie Policy Name (Nombre de política) a `StockTradeStreamPolicy`, revise el código y elija Create Policy (Crear política).

El documento de política resultante debería tener el siguiente aspecto:

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

Para crear un usuario de IAM

1. Abra la consola de IAM en https://console.aws.amazon.com/iam/.
2. En la página Users (Usuarios), elija Add user (Añadir usuario).
3. En User name, escriba `StockTradeStreamUser`.

4. En Access type (Tipo de acceso), elija Programmatic access (Acceso por programa) y, a continuación, elija Next: Permissions (Siguiente: Permisos).
5. Elija Attach existing policies directly (Adjuntar directamente políticas existentes).
6. Busque la política que creó en el procedimiento anterior por su nombre (`StockTradeStreamPolicy`. Seleccione la casilla situada a la izquierda del nombre de la política y, a continuación, elija Next: Review (Siguiente: Revisión).
7. Revise los detalles y el resumen y, a continuación, elija Create user (Crear usuario).
8. Copie el valor de Access key ID (ID de clave de acceso) y guárdelo de modo confidencial. En Secret access key (Clave de acceso secreta), elija Show (Mostrar) y guarde también esa clave de forma confidencial.
9. Pegue las claves de acceso y secreta en un archivo local en una ubicación segura a la que solo pueda obtener acceso usted. Para esta aplicación, cree un archivo denominado `~/.aws/credentials` (con permisos estrictos). El archivo debe tener el siguiente formato:

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

Asociar una política de IAM a un usuario

1. En la consola de IAM, abra Políticas y elija Acciones de política.
2. Elija `StockTradeStreamPolicy` y Asociar.
3. Elija `StockTradeStreamUser` y Asociar política.

Pasos siguientes

Paso 3: Descargar y crear el código

Paso 3: Descargar y crear el código

Este tema proporciona código de implementación de ejemplo para la ingesta de operaciones bursátiles de muestra en la secuencia de datos (productor) y el procesamiento de estos datos (consumidor).

Para descargar y compilar el código

1. Descargue el código fuente del repositorio https://github.com/aws-samples/amazon-kinesis-learning GitHub en su ordenador.
2. Cree un proyecto en su IDE con el código fuente, adhiriéndose a la estructura de directorios proporcionada.
3. Añada las siguientes bibliotecas al proyecto:
   - Amazon Kinesis Client Library (KCL)
   - SDK de AWS
   - Apache HttpCore
   - Apache HttpClient
   - Apache Commons Lang
   - Apache Commons Logging
   - Guava (Google Core Libraries For Java)
   - Jackson Annotations
   - Jackson Core
   - Jackson Databind
   - Jackson Dataformat: CBOR
   - Joda Time
4. Según el IDE, el proyecto puede compilarse automáticamente. Si no, compile el proyecto con los pasos apropiados para su IDE.

Si completa estos pasos correctamente, estará preparado para pasar a la siguiente sección, the section called “Paso 4: Implementar el productor”.

Pasos siguientes

Paso 4: Implementar el productor

Este tutorial utiliza el escenario del mundo real de la monitorización del mercado bursátil. Los siguientes principios explican brevemente cómo este escenario se asigna al productor y su estructura de código compatible.

Consulte el código fuente y revise la siguiente información.

Clase StockTrade

Un comercio de acciones individual está representado por una instancia de la clase StockTrade. Esta instancia contiene atributos como el símbolo de cotización, el precio, el número de acciones, el tipo de transacción (compra o venta) y un ID que identifica de forma exclusiva la transacción. Esta clase se implementa por usted.

Registro de la secuencia

Una secuencia es una serie de registros. Un registro es la sucesión en serie de una instancia de `StockTrade` en formato JSON. Por ejemplo:

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

Clase StockTradeGenerator

StockTradeGenerator tiene un método denominado `getRandomTrade()` que devuelve una nueva operación de comercio de existencias generada aleatoriamente cada vez que se invoca. Esta clase se implementa por usted.

Clase StockTradesWriter

El método `main` del productor, StockTradesWriter recupera continuamente una operación aleatoria y luego la envía a Kinesis Data Streams mediante la ejecución de las siguientes tareas:

1. Lee los nombres de la secuencia y el nombre de la región como entrada.
2. Utiliza el `KinesisAsyncClientBuilder` para establecer la región, las credenciales y la configuración de cliente.
3. Comprueba que la secuencia existe y está activa (si no, sale con un error).
4. En un bucle continuo, llama al método `StockTradeGenerator.getRandomTrade()` y después al método `sendStockTrade` para enviar la transacción a la secuencia cada 100 milisegundos.

El método `sendStockTrade` de la clase `StockTradesWriter` tiene el siguiente código:

```
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);
        }
    }
```

Consulte el siguiente desglose del código:

- La API de `PutRecord` espera una matriz de bytes y debe convertir la transacción al formato JSON. Esta única línea de código realiza esa operación:

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

- Antes de poder enviar la transacción, debe crear una nueva instancia de `PutRecordRequest` (denominada solicitud en este caso). Cada llamada a `request` requiere el nombre de la secuencia, la clave de partición y el blob de datos.

```
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();
```

  El ejemplo utiliza un ticker como clave de partición, que asigna el registro a una partición específica. En la práctica, debería tener cientos o miles de claves de partición por fragmento, de forma que los registros se dispersen de forma uniforme en toda la secuencia. Para obtener más información acerca de cómo agregar datos a una secuencia, consulte Escritura de datos en Amazon Kinesis Data Streams.

  Ahora `request` estará listo para el envío al cliente (la operación PUT):

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

- Siempre es útil agregar funciones de comprobación y registro de errores. Este código registra las condiciones de error:

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

  Agregue el bloque try/catch a la operación `put`:

```
try {
  kinesisClient.putRecord(request).get();
} catch (InterruptedException e) {
```

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

Esto se debe a que una operación put de Kinesis Data Streams puede fallar debido a un error de red o debido a que la secuencia de datos alcanza sus límites de rendimiento y se estrangula. Le recomendamos comprobar detalladamente su política de reintentos para las operaciones `put` de modo que evite la pérdida de datos, por ejemplo utilizando un reintento simple.

- El registro de estado resulta útil, pero es opcional:

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

El productor que se muestra aquí utiliza la funcionalidad de registro único de la API de Kinesis Data Streams, `PutRecord`. En la práctica, si un solo productor genera una gran cantidad de registros, suele ser más eficaz utilizar la funcionalidad de varios registros de `PutRecords` y enviar lotes de registros de una vez. Para obtener más información, consulte Escritura de datos en Amazon Kinesis Data Streams.

Para ejecutar el productor

1. Compruebe que el par de clave de acceso y el par de clave secreta recuperadas en Paso 2: Crear una política y un usuario de IAM se guardan en el archivo `~/.aws/credentials`.
2. Ejecute la clase `StockTradeWriter` con los siguientes argumentos:

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

Si ha creado su secuencia en una región diferente a `us-west-2` tendrá que especificar esa región aquí.

Debería ver una salida similar a esta:

```
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
```

Ahora, Kinesis Data Streams está ingiriendo sus operaciones bursátiles.

Pasos siguientes

Paso 5: Implementar el consumidor

Paso 5: Implementar el consumidor

La aplicación del consumidor en este tutorial procesa continuamente las operaciones bursátiles de su flujo de datos. A continuación, genera como resultado las acciones que más se compran y venden cada minuto. La aplicación se basa en la biblioteca Kinesis Client Library (KCL), que se ocupa de gran parte de las tareas que normalmente deben afrontar las aplicaciones consumidoras. Para obtener más información, consulte Uso de Kinesis Client Library.

Consulte el código fuente y revise la siguiente información.

Clase StockTradesProcessor

Clase principal del consumidor, que proporcionamos por usted y realiza las siguientes tareas:

- Lee los nombres de aplicación, secuencia y región que se pasan como argumentos.
- Crea una instancia `KinesisAsyncClient` con el nombre de región.
- Crea una instancia de `StockTradeRecordProcessorFactory` que sirve instancias de `ShardRecordProcessor`, implementadas por una instancia de `StockTradeRecordProcessor`.

- Crea una instancia `ConfigsBuilder` con las instancias `KinesisAsyncClient`, `StreamName`, `ApplicationName` y `StockTradeRecordProcessorFactory`. Esto es útil para crear todas las configuraciones con valores predeterminados.
- Crea un programador de KCL (anteriormente, en las versiones 1.x de KCL se conocía como trabajador de KCL) con la instancia `ConfigsBuilder`.
- El programador crea un nuevo hilo para cada fragmento (asignado a esta instancia de consumidor), que realiza un bucle continuo para leer registros de la secuencia de datos. A continuación, invoca a la instancia de `StockTradeRecordProcessor` para procesar cada lote de registros recibido.

Clase StockTradeRecordProcessor

Implementación de la instancia `StockTradeRecordProcessor`, que a su vez implementa tres métodos necesarios: `initialize`, `processRecords`, `leaseLost`, `shardEnded` y `shutdownRequested`.

El KCL utiliza los métodos `initialize` y `shutdownRequested` para que el procesador de registros sepa cuándo debe estar listo para comenzar a recibir registros y cuándo debe esperar dejar de recibir registros, respectivamente, de modo que pueda realizar cualquier tarea de configuración y terminación específica de la aplicación. `leaseLost` y `shardEnded` se utilizan para implementar cualquier lógica de qué hacer cuando se pierde un arrendamiento o un procesamiento ha llegado al final de un fragmento. En este ejemplo, simplemente registramos mensajes que indican estos eventos.

Le proporcionamos el código para estos métodos. El procesamiento principal sucede en el método `processRecords`, que a su vez utiliza `processRecord` para cada registro. Este último método se proporciona como el código esqueleto mayormente vacío para que lo implemente en el siguiente paso, donde se explica con mayor detalle.

También hay que resaltar la implementación de métodos de compatibilidad para `processRecord`: `reportStats` y `resetStats`, que están vacíos en el código fuente original.

El método `processRecords` se implementa por usted, y realiza los pasos siguientes:

- En cada registro que se pase, llama a su `processRecord`.
- Si ha pasado al menos 1 minuto desde el último informe, llama a `reportStats()`, que imprime las últimas estadísticas y, a continuación, a `resetStats()`, que elimina las estadísticas para que el próximo intervalo incluya solo registros nuevos.
- Establece el momento del siguiente informe.

- Si ha transcurrido al menos un minuto desde el último punto de comprobación de la base de datos, llama a `checkpoint()`.
- Establece el momento del siguiente punto de comprobación.

Este método utiliza intervalos de 60 segundos para la velocidad de elaboración de informes y puntos de comprobación. Para más información sobre el punto de control, consulte Uso de Kinesis Client Library.

Clase StockStats

Esta clase proporciona retención de datos y seguimiento de estadísticas para las acciones más populares a lo largo del tiempo. Proporcionamos este código por usted y contiene los siguientes métodos:

- `addStockTrade(StockTrade)`: inserta el `StockTrade` dado en las estadísticas de ejecución.
- `toString()`: devuelve las estadísticas en una cadena con formato.

Esta clase realiza un seguimiento de los valores más populares, ya que realiza un recuento continuo del número total de transacciones para cada valor y del recuento máximo. Asimismo, actualiza estos recuentos cada vez que llega una operación nueva.

Agregar código para los métodos de la clase `StockTradeRecordProcessor`, tal y como se muestra en los pasos siguientes.

Para implementar el consumidor

1. Implemente el método `processRecord` creando una instancia de un objeto `StockTrade` con el tamaño correcto y añadiéndole los datos de registro, registrando una advertencia si hay algún problema.

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

2. Implemente un método `reportStats` sencillo. Si lo desea, puede modificar el formato de salida según sus preferencias.

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

3. Por último, implemente el método `resetStats`, lo que creará una nueva instancia de `stockStats`.

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

4. Implemente los siguientes métodos requeridos por la interfaz `ShardRecordProcessor`

```
@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);
    }
}
```

Para ejecutar el consumidor

1. Ejecute el productor que escribió en para insertar registros de operaciones bursátiles simuladas en la secuencia.
2. Compruebe que la clave de acceso y el par de claves secretas recuperadas anteriormente (al crear el usuario de IAM) se guardaron en el archivo `~/.aws/credentials`.
3. Ejecute la clase `StockTradesProcessor` con los siguientes argumentos:

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

   Tenga en cuenta que si ha creado su secuencia en una región diferente a `us-west-2` tiene que especificar esa región aquí.

Después de un minuto, debería ver un resultado similar a este, actualizado cada minuto:

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

```
Most popular stock being sold: PTR, 14 sells.
**********************************************************
```

Pasos siguientes

[Paso 6: Ampliar el consumidor (opcional)]

Paso 6: Ampliar el consumidor (opcional)

En esta sección opcional, mostramos cómo puede ampliar el código del consumidor para un escenario ligeramente más complicado.

Si desea conocer cuáles son las mayores operaciones de venta de cada minuto, puede modificar la clase `StockStats` en tres lugares para dar cabida a esta nueva prioridad.

Para ampliar el consumidor

1. Agregue nuevas variables de instancia:

```
// 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. Agregue el siguiente código a `addStockTrade`:

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

3. Modifique el método `toString` para imprimir la información adicional:

```
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);
}
```

Si ejecuta el consumidor ahora (recuerde ejecutar también el productor), debería ver un resultado similar a este:

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

Pasos siguientes

[Paso 7: Conclusión]

Paso 7: Conclusión

Como paga por utilizar el flujo de datos de Kinesis, asegúrese de eliminarla, así como la tabla de Amazon DynamoDB correspondiente, cuando haya terminado con ella. Se aplican cargos nominales sobre una secuencia activa incluso aunque no esté enviando ni recibiendo registros. Esto se debe a que una secuencia activa está utilizando los recursos "escuchando" de forma continua los registros entrantes y las solicitudes para obtener registros.

Para eliminar la secuencia y la tabla

1. Cierre los productores y los consumidores que puedan estar ejecutándose aún.
2. Abra la consola de Kinesis en https://console.aws.amazon.com/kinesis.

3. Seleccione la secuencia que haya creado para esta aplicación (`StockTradeStream`).
4. Elija Delete Stream (Eliminar secuencia).
5. Abra la consola de DynamoDB en https://console.aws.amazon.com/dynamodb/.
6. Elimine la tabla `StockTradesProcessor`.

Resumen

El procesamiento de una gran cantidad de datos casi en tiempo real no requiere escribir ningún código mágico ni desarrollar una infraestructura enorme. Es tan sencillo como escribir lógica para procesar una pequeña cantidad de datos (como escribir `processRecord(Record)`), pero con Kinesis Data Streams para escalarla de forma que funcione con una gran cantidad de datos transmitidos. No tiene que preocuparse de cómo escalar su procesamiento, ya que Kinesis Data Streams lo administra por usted. Lo único que tiene que hacer es enviar sus registros de streaming a Kinesis Data Streams y escribir la lógica para procesar cada nuevo registro recibido.

A continuación se muestran algunas posibles mejoras para esta aplicación.

Agregación en todos los fragmentos

> En la actualidad, obtiene estadísticas derivadas de agrupar los registros de datos que se reciben de un único proceso de trabajo desde un único fragmento. (Un fragmento no puede ser procesado por más de un proceso de trabajo en una sola aplicación al mismo tiempo). Lógicamente, si escala y tiene más de un fragmento, es posible que quiera realizar la agregación en todos los fragmentos. Podrá hacerlo con una arquitectura de canalización en la que el resultado de cada proceso de trabajo se envíe a otra secuencia con un único fragmento, que se procesará por parte de un proceso de trabajo que agregue los resultados de la primera etapa. Dado que los datos de la primera etapa están limitados (a una muestra por minuto y fragmento), pueden ser administrados fácilmente por un fragmento.

Procesamiento de la escala

> Cuando la secuencia se escala para tener muchos fragmentos (porque hay muchos productores enviando datos), el método para aumentar el procesamiento es agregar más procesos de trabajo. Puede ejecutar los procesos de trabajo en instancias de Amazon EC2 y utilizar grupos de escalado automático.

Use conectores para Amazon S3/DynamoDB/Amazon Redshift/Storm

A medida que un flujo se procesa continuamente, su salida puede enviarse a otros destinos. AWS proporciona conectores para integrar Kinesis Data Streams con otros servicios de AWS y herramientas de terceros.

Tutorial: Procesamiento de operaciones bursátiles en tiempo real con KPL y KCL 1.x

El escenario que planteamos en este tutorial comprende la adquisición de operaciones bursátiles en un flujo de datos y la escritura de una aplicación de Amazon Kinesis Data Streams sencilla que realiza cálculos con dicho flujo. Aprenderá a enviar un flujo de registros a Kinesis Data Streams e implementar una aplicación que consume y procesa dichos registros casi en tiempo real.

> **Important**
>
> Una vez creado un flujo, su cuenta generará gastos nominales por el uso de Kinesis Data Streams, ya que Kinesis Data Streams no está disponible en el nivel gratuito de AWS. Una vez que se inicia la aplicación consumidora, se aplicarán cargos nominales por el uso de Amazon DynamoDB. La aplicación consumidora utiliza DynamoDB para realizar un seguimiento del estado de procesamiento. Cuando termine con esta aplicación, elimine sus recursos de AWS para dejar de incurrir en gastos. Para obtener más información, consulte Paso 7: Conclusión.

El código no obtiene acceso a datos bursátiles reales, sino que, en su lugar, simula la secuencia de operaciones bursátiles. Lo hace a través de un generador de operaciones bursátiles aleatorias que parte de datos reales del mercado para los 25 principales valores por capitalización en febrero de 2015. Si tiene acceso a una secuencia de operaciones bursátiles en tiempo real, puede que le interese derivar estadísticas útiles y puntuales a partir de dicha secuencia. Por ejemplo, es posible que desee realizar un análisis de ventana deslizante en el que se determina el valor más popular adquirido durante los últimos 5 minutos. O también cabe la posibilidad de que quiera recibir una notificación cada vez que haya una orden de venta que sea demasiado grande (es decir, con demasiadas acciones). Puede ampliar el código de esta serie para proporcionar esta funcionalidad.

Puede seguir los pasos de este tutorial en su equipo de escritorio o portátil y ejecutar el código tanto del consumidor como del productor en la misma máquina o en cualquier plataforma que cumpla los requisitos definidos, como Amazon Elastic Compute Cloud (Amazon EC2).

Los ejemplos mostrados utilizan la región Oeste de EE. UU. (Oregón), pero funcionan en cualquiera de las regiones de AWS compatibles con Kinesis Data Streams.

Tareas

- Requisitos previos
- Paso 1: Crear una secuencia de datos
- Paso 2: Crear una política y un usuario de IAM
- Paso 3: Descargar y compilar el código de implementación
- Paso 4: Implementar el productor
- Paso 5: Implementar el consumidor
- Paso 6: Ampliar el consumidor (opcional)
- Paso 7: Conclusión

Requisitos previos

A continuación, se indican los requisitos para completar el Tutorial: Procesamiento de operaciones bursátiles en tiempo real con KPL y KCL 1.x.

Cuenta de Amazon Web Services

Antes de comenzar, asegúrese de estar familiarizado con los conceptos que se abordan en Terminología y conceptos de Amazon Kinesis Data Streams, especialmente los relacionados con secuencias, fragmentos, productores y consumidores. También le será de ayuda haber completado Instalar y configurar la AWS CLI.

Necesita una cuenta de AWS y un navegador web para obtener acceso a la AWS Management Console.

Para acceder a la consola, utilice su nombre de usuario y contraseña de IAM para iniciar sesión en AWS Management Console desde la página de inicio de sesión de IAM. Para más información sobre las credenciales de seguridad de AWS, incluido el acceso programático y las alternativas a las credenciales a largo plazo, consulte Credenciales de seguridad de AWS en la Guía del usuario de

IAM. Para información sobre cómo iniciar sesión en su cuenta de Cuenta de AWS, consulte Cómo iniciar sesión en AWS en la Guía del usuario de AWS Sign-In.

Para obtener más información sobre IAM; y las instrucciones de configuración de la clave de seguridad, consulte Creación de un usuario de IAM.

Requisitos de software del sistema

El sistema utilizado para ejecutar la aplicación debe tener instalado Java 7 o superior. Para descargar e instalar la versión más reciente del kit de desarrollo de Java (JDK), visite el sitio web de instalación de Java SE de Oracle.

Si tiene una aplicación IDE de Java, como Eclipse, puede abrir el código fuente, editarlo, compilarlo y ejecutarlo.

Necesitará la última versión de AWS SDK for Java. Si utiliza Eclipse como IDE, puede instalar el Kit de herramientas de AWS para Eclipse en su lugar.

La aplicación de consumidor requiere la versión 1.2.1 o superior de Kinesis Client Library (KCL), que puede obtener en GitHub en Kinesis Client Library (Java).

Pasos siguientes

Paso 1: Crear una secuencia de datos

Paso 1: Crear una secuencia de datos

En el primer paso del Tutorial: Procesamiento de operaciones bursátiles en tiempo real con KPL y KCL 1.x, creará la secuencia que utilizará en los pasos siguientes.

Para crear un flujo

1. Inicie sesión en la AWS Management Console y abra la consola de Kinesis en https://console.aws.amazon.com/kinesis.
2. Elija Data Streams (Secuencias de datos) en el panel de navegación.
3. En la barra de navegación, expanda el selector de regiones y seleccione una región.
4. Elija Create Kinesis stream (Crear secuencia de Kinesis).
5. Escriba un nombre para la secuencia (por ejemplo **`StockTradeStream`**).
6. Introduzca **1** como número de fragmentos, pero deje Calcular el número de particiones que va a necesitar contraído.

7. Elija Create Kinesis stream (Crear secuencia de Kinesis).

En la página de lista Flujos de Kinesis, el estado del flujo es `CREATING` mientras se está creando. Cuando la secuencia está lista para usarse, el estado cambia a `ACTIVE`. Elija el nombre del flujo. En la página que aparece, la pestaña Detalles muestra un resumen de la configuración del flujo. La sección Monitoreo muestra información de supervisión del flujo.

Información adicional sobre los fragmentos

Cuando comience a utilizar Kinesis Data Streams fuera de este tutorial, es posible que tenga que planificar el proceso de creación de flujos más detenidamente. Debe planificarlo en función de la demanda máxima esperada cuando aprovisione los fragmentos. Con este escenario como ejemplo, el tráfico del mercado bursátil de los Estados Unidos alcanza su máximo durante el día (hora del este), por lo que las estimaciones de la demanda deben realizarse a para ese momento del día. A continuación, podrá elegir el aprovisionamiento adecuado para la demanda máxima prevista o escalar su secuencia en respuesta a las fluctuaciones de la demanda.

Un fragmento es una unidad de capacidad de rendimiento. En la página Crear flujo de Kinesis, expanda Estimar el número de particiones que necesitará. Introduzca el tamaño medio de los registros, el número máximo de registros escritos por segundo, y el número de aplicaciones consumidoras utilizando las directrices siguientes:

Tamaño medio de registro

Estimación del tamaño medio calculado de los registros. Si no conoce este valor, utilice la estimación del tamaño máximo del registro para este valor.

Número máximo de registros escritos

Tenga en cuenta el número de entidades que proporcionan los datos y el número aproximado de registros por segundo que producen cada una de ellas. Por ejemplo, si está obteniendo operaciones bursátiles de 20 servidores y cada uno genera 250 operaciones por segundo, el número total de operaciones (registros) por segundo es de 5000.

Número de aplicaciones consumidoras

Número de aplicaciones que leen independientemente desde la secuencia para procesarla de una forma diferente y producir resultados distintos. Cada aplicación puede tener varias instancias en ejecución en diferentes máquinas (es decir, que se ejecutan en un clúster) para poder procesar una secuencia de gran volumen.

Si el número estimado de fragmentos que se muestra supera su límite de fragmentos actual, puede que tenga que enviar una solicitud para aumentar ese límite antes de crear una secuencia con ese número de fragmentos. Para solicitar un aumento del límite de particiones, utilice el formulario Límites de Kinesis Data Streams. Para más información sobre los flujos y particiones, consulte Creación y administración de secuencias.

Pasos siguientes

Paso 2: Crear una política y un usuario de IAM

Paso 2: Crear una política y un usuario de IAM

Las prácticas recomendadas de seguridad de AWS imponen el uso de permisos específicos para controlar el acceso a diferentes recursos. AWS Identity and Access Management (IAM) le permite administrar usuarios y permisos de usuario en AWS. Una política de IAM enumera de forma explícita las acciones que se pueden realizar y los recursos a los que se pueden aplicar dichas acciones.

Los siguientes permisos son los permisos mínimos normalmente necesarios para un productor y un consumidor de Kinesis Data Streams.

Productor

Acciones	Recurso	Finalidad
`DescribeStream` , `DescribeStreamSummary` , `DescribeStreamConsumer`	Flujo de datos de Kinesis	Antes de intentar escribir registros, el productor comprueba está activa, si los fragmentos se encuentran en la secuenci un consumidor.
`SubscribeToShard` , `RegisterStreamConsumer`	Flujo de datos de Kinesis	Suscribe y registra los consumidores en un fragmento de K
`PutRecord` , `PutRecords`	Flujo de datos de Kinesis	Escriba registros en Kinesis Data Streams.

Consumidor

Acciones	Recurso	Finalidad
`DescribeStream`	Flujo de datos de Kinesis	Antes de intentar leer registros, el consumidor comprueba s si está activa, y si los fragmentos se encuentran en la secu
`GetRecords` , `GetShardIterator`	Flujo de datos de Kinesis	Lee registros de una partición de Kinesis Data Streams.
`CreateTable` , `DescribeTable` , `GetItem`, `PutItem`, `Scan`, `UpdateItem`	La tabla de Amazon DynamoDB.	Si el consumidor se desarrolla utilizando Kinesis Client Libr permisos para que una tabla de DynamoDB realice un segu procesamiento de la aplicación. El primer consumidor que s
`DeleteItem`	La tabla de Amazon DynamoDB.	Para cuando el consumidor realiza operaciones de división de Kinesis Data Streams.
`PutMetricData`	Registros de Amazon CloudWatch	KCL también carga métricas a CloudWatch, que son útiles aplicación.

Para esta aplicación, debe crear una única política de IAM; que conceda todos los permisos anteriores. En la práctica, es posible que quiera crear dos políticas, uno para los productores y otra para los consumidores.

Para crear una política de IAM

1. Localice el nombre de recurso de Amazon (ARN) de la nueva secuencia. Puede encontrar este ARN como Stream ARN (ARN de la secuencia) en la parte superior de la pestaña Details (Detalles). El formato del ARN es el siguiente:

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

   region

   El código de la región; por ejemplo, `us-west-2`. Para obtener más información, consulte Conceptos de región y zona de disponibilidad.

cuenta

El ID de la cuenta de AWS, como se muestra en Configuración de cuenta.

nombre

El nombre de la secuencia de Paso 1: Crear una secuencia de datos, que es `StockTradeStream`.

2. Determinar el ARN de la tabla de DynamoDB para que lo utilice el consumidor (creado por la primera instancia del consumidor). Debe tener el siguiente formato:

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

La región y la cuenta y provienen del mismo lugar que en el paso anterior, pero esta vez name es el nombre de la tabla creada y utilizada por la aplicación consumidora. La KCL utilizada por el consumidor utiliza el nombre de la aplicación como nombre de la tabla. Utilice `StockTradesProcessor`, que será el nombre de aplicación que se utilizará más tarde.

3. En la consola de IAM, en Políticas (https://console.aws.amazon.com/iam/home#policies), elija Crear política. Si es la primera vez que trabaja con políticas de IAM, elija Introducción, Crear política.
4. Elija Seleccionar junto a Generador de políticas.
5. Elija Amazon Kinesis como servicio AWS.
6. Seleccione `DescribeStream`, `GetShardIterator`, `GetRecords`, `PutRecord` y `PutRecords` como acciones permitidas.
7. Introduzca el ARN que ha creado en el paso 1.
8. Utilice Add Statement (Añadir instrucción) para cada uno de los siguientes:

Servicio de AWS	Acciones	ARN
Amazon DynamoDB	`CreateTable` , `DeleteItem` , `DescribeTable` , `GetItem`, `PutItem`, `Scan`, `UpdateItem`	El ARN que ha creado en el paso 2.
Amazon CloudWatch	`PutMetricData`	*

El asterisco (*) que se utiliza cuando no es necesario especificar un ARN. En este caso, se debe a que no existe ningún recurso específico en CloudWatch para el que se invoque la acción `PutMetricData`.

9. Elija Next Step (Paso siguiente).
10. Cambie Policy Name (Nombre de política) a `StockTradeStreamPolicy`, revise el código y elija Create Policy (Crear política).

El documento de política resultante debería tener un aspecto similar a este:

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

Para crear un usuario de IAM

1. Abra la consola de IAM en https://console.aws.amazon.com/iam/.
2. En la página Users (Usuarios), elija Add user (Añadir usuario).
3. En User name, escriba `StockTradeStreamUser`.
4. En Access type (Tipo de acceso), elija Programmatic access (Acceso por programa) y, a continuación, elija Next: Permissions (Siguiente: Permisos).
5. Elija Attach existing policies directly (Adjuntar directamente políticas existentes).
6. Busque por nombre la política que ha creado. Seleccione la casilla situada a la izquierda del nombre de la política y, a continuación, elija Next: Review (Siguiente: Revisión).
7. Revise los detalles y el resumen y, a continuación, elija Create user (Crear usuario).
8. Copie el valor de Access key ID (ID de clave de acceso) y guárdelo de modo confidencial. En Secret access key (Clave de acceso secreta), elija Show (Mostrar) y guarde también esa clave de forma confidencial.

9. Pegue las claves de acceso y secreta en un archivo local en una ubicación segura a la que solo pueda obtener acceso usted. Para esta aplicación, cree un archivo denominado `~/.aws/credentials` (con permisos estrictos). El archivo debe tener el siguiente formato:

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

Asociar una política de IAM a un usuario

1. En la consola de IAM, abra Políticas y elija Acciones de política.
2. Elija `StockTradeStreamPolicy` y Asociar.
3. Elija `StockTradeStreamUser` y Asociar política.

Pasos siguientes

Paso 3: Descargar y compilar el código de implementación

Paso 3: Descargar y compilar el código de implementación

Se facilita la estructura del código para el the section called “Tutorial: Procesamiento de operaciones bursátiles en tiempo real con KPL y KCL 1.x”. Contiene una implementación de stub tanto para la adquisición de secuencias de operaciones bursátiles (productor) como para el procesamiento de los datos (consumidor). El siguiente procedimiento muestra cómo completar las implementaciones.

Para descargar y compilar el código de implementación

1. Descargue el código fuente en el equipo.
2. Cree un proyecto en su IDE favorito con el código fuente, utilizando la estructura de directorios proporcionada.
3. Añada las siguientes bibliotecas al proyecto:
   - Amazon Kinesis Client Library (KCL)
   - SDK de AWS
   - Apache HttpCore
   - Apache HttpClient

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

4. Según el IDE, el proyecto puede compilarse automáticamente. Si no, compile el proyecto con los pasos apropiados para su IDE.

Si completa estos pasos correctamente, estará preparado para pasar a la siguiente sección, the section called “Paso 4: Implementar el productor”. Si la compilación genera errores en cualquier etapa, deberá investigarlos y corregirlos antes de continuar.

Pasos siguientes

Paso 4: Implementar el productor

La aplicación del Tutorial: Procesamiento de operaciones bursátiles en tiempo real con KPL y KCL 1.x utiliza un escenario real de monitorización del mercado bursátil. Los siguientes principios explicar brevemente la forma en que este escenario se asocia con la estructura del productor y el código de apoyo.

Consulte el código fuente y revise la siguiente información.

Clase StockTrade

Una operación bursátil está representada por una instancia de la clase `StockTrade`. Esta instancia contiene atributos como el símbolo de cotización, el precio, el número de acciones, el tipo de transacción (compra o venta) y un ID que identifica de forma exclusiva la transacción. Esta clase se implementa por usted.

Registro de la secuencia

Una secuencia es una serie de registros. Un registro es la sucesión en serie de una instancia de `StockTrade` en formato JSON. Por ejemplo:

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

Clase StockTradeGenerator

`StockTradeGenerator` tiene un método denominado `getRandomTrade()` que proporciona una nueva operación bursátil generada aleatoriamente cada vez que se invoca. Esta clase se implementa por usted.

Clase StockTradesWriter

El método `main` del productor, `StockTradesWriter` recupera continuamente una operación aleatoria y luego la envía a Kinesis Data Streams mediante la ejecución de las siguientes tareas:

1. Lee los nombres de la secuencia y la región como entrada.
2. Crea un `AmazonKinesisClientBuilder`.
3. Utiliza el compilador de clientes para establecer la región, las credenciales y la configuración de cliente.
4. Crea un cliente de `AmazonKinesis` mediante el compilador de clientes.
5. Comprueba que la secuencia existe y está activa (si no, sale con un error).
6. En un bucle continuo, llama al método `StockTradeGenerator.getRandomTrade()` y después al método `sendStockTrade` para enviar la transacción a la secuencia cada 100 milisegundos.

El método `sendStockTrade` de la clase `StockTradesWriter` tiene el siguiente código:

```
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);
    }
}
```

Consulte el siguiente desglose del código:

- La API de `PutRecord` espera una matriz de bytes, y debe convertir la `trade` a formato JSON. Esta única línea de código realiza esa operación:

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

- Antes de poder enviar la transacción, debe crear una nueva instancia de `PutRecordRequest` (denominada `putRecord` en este caso):

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

  Cada llamada a `PutRecord` requiere el nombre de la secuencia, la clave de partición y el blob de datos. El siguiente código rellenará estos campos en el objeto `putRecord` utilizando sus métodos `setXxxx()`:

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

  El ejemplo utiliza un ticker como clave de partición, que asigna el registro a una partición específica. En la práctica, debería tener cientos o miles de claves de partición por fragmento, de forma que los registros se dispersen de forma uniforme en toda la secuencia. Para obtener más información acerca de cómo agregar datos a una secuencia, consulte Agregar datos a una secuencia.

Ahora `putRecord` estará listo para el envío al cliente (la operación `put`):

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

- Siempre es útil agregar funciones de comprobación y registro de errores. Este código registra las condiciones de error:

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

Agregue el bloque try/catch a la operación `put`:

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

Esto se debe a que una operación `put` de Kinesis Data Streams puede fallar debido a un error de red o debido a que el flujo alcanza sus límites de rendimiento y se ve limitado. Le recomendamos comprobar detalladamente su política de reintentos para las operaciones `put` de modo que evite la pérdida de datos, por ejemplo utilizando un reintento simple.

- El registro de estado resulta útil, pero es opcional:

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

El productor que se muestra aquí utiliza la funcionalidad de registro único de la API de Kinesis Data Streams, `PutRecord`. En la práctica, si un solo productor genera una gran cantidad de registros, suele ser más eficaz utilizar la funcionalidad de varios registros de `PutRecords` y enviar lotes de registros de una vez. Para obtener más información, consulte Agregar datos a una secuencia.

Para ejecutar el productor

1. Compruebe que la clave de acceso y el par de claves secretas recuperadas anteriormente (al crear el usuario de IAM) se guardaron en el archivo `~/.aws/credentials`.
2. Ejecute la clase `StockTradeWriter` con los siguientes argumentos:

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

   Si ha creado su secuencia en una región diferente a `us-west-2` tendrá que especificar esa región aquí.

Debería ver una salida similar a esta:

```
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
```

El flujo de operaciones bursátiles está siendo adquirido por Kinesis Data Streams.

Pasos siguientes

Paso 5: Implementar el consumidor

Paso 5: Implementar el consumidor

La aplicación consumidora del Tutorial: Procesamiento de operaciones bursátiles en tiempo real con KPL y KCL 1.x procesa de forma continua la secuencia de operaciones bursátiles que se ha creado en . A continuación, genera como resultado las acciones que más se compran y venden cada minuto. La aplicación se basa en la biblioteca Kinesis Client Library (KCL), que se ocupa de gran parte de las tareas que normalmente deben afrontar las aplicaciones consumidoras. Para obtener más información, consulte Desarrollo de consumidores de KCL 1.x.

Consulte el código fuente y revise la siguiente información.

Clase StockTradesProcessor

Clase principal del consumidor, que proporcionamos por usted y realiza las siguientes tareas:

- Lee los nombres de aplicación, secuencia y región que se pasan como argumentos.
- Lee las credenciales de `~/.aws/credentials`.
- Crea una instancia de `RecordProcessorFactory` que sirve instancias de `RecordProcessor`, implementadas por una instancia de `StockTradeRecordProcessor`.
- Crea un proceso de trabajo de KCL con la instancia `RecordProcessorFactory` y una configuración estándar que incluye el nombre del flujo, las credenciales y el nombre de la aplicación.
- El proceso de trabajo crea un subproceso nuevo para cada partición (asignado a esta instancia del consumidor), que se ejecuta en bucle continuamente para leer registros de Kinesis Data Streams. A continuación, invoca a la instancia de `RecordProcessor` para procesar cada lote de registros recibido.

Clase StockTradeRecordProcessor

Implementación de la instancia `RecordProcessor`, que a su vez implementa tres métodos necesarios: `initialize`, `processRecords` y `shutdown`.

Como indican sus nombres, `initialize` y `shutdown` se utilizan en Kinesis Client Library para que el procesador de registros sepa cuándo debe estar listo para empezar a recibir registros y cuándo debe esperar dejar de recibirlos, respectivamente, de modo que pueda realizar cualquier tarea de configuración y finalización específica de la aplicación. Le proporcionamos el código de los mismos. El procesamiento principal sucede en el método `processRecords`, que a su vez utiliza `processRecord` para cada registro. Este último método se proporciona principalmente como código estructural prácticamente vacío, para que pueda implementarlo en el siguiente paso, donde se explica más a fondo.

También hay que resaltar la implementación de métodos de apoyo para `processRecord`: `reportStats`y `resetStats`, que están vacíos en el código fuente original.

El método `processRecords` se implementa por usted, y realiza los pasos siguientes:

- Para cada registro que se pase, llama a `processRecord`.

- Si ha pasado al menos 1 minuto desde el último informe, llama a `reportStats()`, que imprime las últimas estadísticas y, a continuación, a `resetStats()`, que elimina las estadísticas para que el próximo intervalo incluya solo registros nuevos.
- Establece el momento del siguiente informe.
- Si ha transcurrido al menos un minuto desde el último punto de comprobación de la base de datos, llama a `checkpoint()`.
- Establece el momento del siguiente punto de comprobación.

Este método utiliza intervalos de 60 segundos para la velocidad de elaboración de informes y puntos de comprobación. Para obtener más información sobre los puntos de comprobación, consulte Información adicional sobre el consumidor.

Clase StockStats

Esta clase proporciona retención de datos y seguimiento de estadísticas para las acciones más populares a lo largo del tiempo. Proporcionamos este código por usted y contiene los siguientes métodos:

- `addStockTrade(StockTrade)`: inserta el `StockTrade` dado en las estadísticas de ejecución.
- `toString()`: devuelve las estadísticas en una cadena con formato.

Esta clase realiza un seguimiento de los valores más populares, ya que realiza un recuento continuo del número total de transacciones para cada valor y del recuento máximo. Asimismo, actualiza estos recuentos cada vez que llega una operación nueva.

Agregar código para los métodos de la clase `StockTradeRecordProcessor`, tal y como se muestra en los pasos siguientes.

Para implementar el consumidor

1. Implemente el método `processRecord` creando una instancia de un objeto `StockTrade` con el tamaño correcto y añadiéndole los datos de registro, registrando una advertencia si hay algún problema.

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

```
    return;
}
stockStats.addStockTrade(trade);
```

2. Implemente un método `reportStats` sencillo. Si lo desea, puede modificar el formato de salida según sus preferencias.

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

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

3. Por último, implemente el método `resetStats`, lo que creará una nueva instancia de `stockStats`.

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

Para ejecutar el consumidor

1. Ejecute el productor que escribió en para insertar registros de operaciones bursátiles simuladas en la secuencia.
2. Compruebe que la clave de acceso y el par de claves secretas recuperadas anteriormente (al crear el usuario de IAM) se guardaron en el archivo `~/.aws/credentials`.
3. Ejecute la clase `StockTradesProcessor` con los siguientes argumentos:

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

   Tenga en cuenta que si ha creado su secuencia en una región diferente a `us-west-2` tiene que especificar esa región aquí.

Después de un minuto, debería ver un resultado similar a este, actualizado cada minuto:

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

Información adicional sobre el consumidor

Si está familiarizado con las ventajas de Kinesis Client Library, que se tratan en Desarrollo de consumidores de KCL 1.x y en otros artículos, es posible que se pregunte por qué debería utilizarla aquí. Aunque solo se utiliza un único flujo de particiones y una única instancia de consumidor para procesarlo, aún así es más fácil implementar el consumidor mediante KCL. Compare los pasos de implementación del código en la sección del productor con la del consumidor, y podrá ver en comparación la facilidad de implementar un consumidor. Esto se debe, en gran medida, a los servicios que proporciona la KCL.

En esta aplicación, se centrará en la implementación de una clase de procesador de registros que puede procesar registros individuales. No tiene que preocuparse de la forma en que se obtienen los registros de Kinesis Data Streams. KCL recibe los registros e invoca el procesador de registros cuando hay nuevos registros disponibles. Tampoco tiene que preocuparse del número de fragmentos o instancias del consumidor que hay. Si la secuencia se amplía, no será necesario volver a escribir su aplicación para administrar más de un fragmento o una instancia del consumidor.

El término creación de puntos de comprobación implica registrar el punto de la secuencia con los registros de datos que se han consumido y procesado hasta el momento, de forma que si la aplicación se bloquea, la secuencia se lee a partir de ese punto y no desde su inicio. El tema de los puntos de comprobación, los diversos patrones de diseño y las prácticas recomendadas al respecto quedan fuera del ámbito de este capítulo. Sin embargo, es algo que puede encontrar en entornos de producción.

Como aprendió en , las operaciones `put` de la API de Kinesis Data Streams toman una clave de partición como entrada. Kinesis Data Streams utiliza una clave de partición como mecanismo para dividir los registros en varias particiones (cuando hay más de una partición en el flujo). La misma clave de partición siempre se dirige al mismo fragmento. Esto permite que el consumidor que procesa un determinado fragmento se diseñe con el supuesto de que los registros con la misma clave de partición solo se envían a ese consumidor, y que ningún registro con la misma clave de partición acaba en ningún otro consumidor. Por lo tanto, un proceso de trabajo del consumidor puede agregar todos los registros con la misma clave de partición sin preocuparse de que podrían faltar datos necesarios.

En esta aplicación, el procesamiento de registros por parte del consumidor no es intensivo, por lo que puede utilizar una partición y realizar el procesamiento en el mismo subproceso que el subproceso de KCL. En la práctica, sin embargo, piense primero en aumentar el número de fragmentos. En algunos casos, es posible que desee cambiar el procesamiento a otro subproceso o utilizar un grupo de subprocesos si espera que el procesamiento de registros sea intenso. De esta

forma, KCL puede obtener nuevos registros con mayor rapidez, mientras que los demás subprocesos pueden procesar los registros en paralelo. El diseño con múltiples subprocesos no es trivial y debe abordarse con técnicas avanzadas, por lo que aumentar el número de fragmentos suele ser la forma más eficaz y sencilla de escalar.

Pasos siguientes

Paso 6: Ampliar el consumidor (opcional)

Paso 6: Ampliar el consumidor (opcional)

Es posible que la aplicación del Tutorial: Procesamiento de operaciones bursátiles en tiempo real con KPL y KCL 1.x ya sea suficiente para sus objetivos. En esta sección opcional, mostramos cómo puede ampliar el código del consumidor para un escenario ligeramente más complicado.

Si desea conocer cuáles son las mayores operaciones de venta de cada minuto, puede modificar la clase `StockStats` en tres lugares para dar cabida a esta nueva prioridad.

Para ampliar el consumidor

1. Agregue nuevas variables de instancia:

```
// 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. Agregue el siguiente código a `addStockTrade`:

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

3. Modifique el método `toString` para imprimir la información adicional:

```
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);
}
```

Si ejecuta el consumidor ahora (recuerde ejecutar también el productor), debería ver un resultado similar a este:

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

Pasos siguientes

Paso 7: Conclusión

Paso 7: Conclusión

Como paga por utilizar el flujo de datos de Kinesis, asegúrese de eliminarla, así como la tabla de Amazon DynamoDB correspondiente, cuando haya terminado con ella. Se aplican cargos nominales sobre una secuencia activa incluso aunque no esté enviando ni recibiendo registros. Esto se debe a que una secuencia activa está utilizando los recursos "escuchando" de forma continua los registros entrantes y las solicitudes para obtener registros.

Para eliminar la secuencia y la tabla

1. Cierre los productores y los consumidores que puedan estar ejecutándose aún.
2. Abra la consola de Kinesis en https://console.aws.amazon.com/kinesis.
3. Seleccione la secuencia que haya creado para esta aplicación (`StockTradeStream`).
4. Elija Delete Stream (Eliminar secuencia).
5. Abra la consola de DynamoDB en https://console.aws.amazon.com/dynamodb/.
6. Elimine la tabla `StockTradesProcessor`.

Resumen

El procesamiento de una gran cantidad de datos casi en tiempo real no requiere escribir ningún código mágico ni desarrollar una infraestructura enorme. Es tan sencillo como escribir lógica para procesar una pequeña cantidad de datos (como escribir `processRecord(Record)`), pero con Kinesis Data Streams para escalarla de forma que funcione con una gran cantidad de datos transmitidos. No tiene que preocuparse de cómo escalar su procesamiento, ya que Kinesis Data Streams lo administra por usted. Lo único que tiene que hacer es enviar sus registros de streaming a Kinesis Data Streams y escribir la lógica para procesar cada nuevo registro recibido.

A continuación se muestran algunas posibles mejoras para esta aplicación.

Agregación en todos los fragmentos

> En la actualidad, obtiene estadísticas derivadas de agrupar los registros de datos que se reciben de un único proceso de trabajo desde un único fragmento. (Un fragmento no puede ser procesado por más de un proceso de trabajo en una sola aplicación al mismo tiempo). Lógicamente, si escala y tiene más de un fragmento, es posible que quiera realizar la agregación en todos los fragmentos. Podrá hacerlo con una arquitectura de canalización en la que el resultado de cada proceso de trabajo se envíe a otra secuencia con un único fragmento, que se procesará por parte de un proceso de trabajo que agregue los resultados de la primera etapa. Dado que los datos de la primera etapa están limitados (a una muestra por minuto y fragmento), pueden ser administrados fácilmente por un fragmento.

Procesamiento de la escala

> Cuando la secuencia se escala para tener muchos fragmentos (porque hay muchos productores enviando datos), el método para aumentar el procesamiento es agregar más procesos de trabajo. Puede ejecutar los procesos de trabajo en instancias de Amazon EC2 y utilizar grupos de escalado automático.

Use conectores para Amazon S3/DynamoDB/Amazon Redshift/Storm

> A medida que un flujo se procesa continuamente, su salida puede enviarse a otros destinos. AWS proporciona conectores para integrar Kinesis Data Streams con otros servicios de AWS y herramientas de terceros.

Pasos siguientes

- Para más información acerca del uso de la API de Kinesis Data Streams, consulte Desarrollo de productores mediante la API de Amazon Kinesis Data Streams con AWS SDK for Java, Desarrollo de consumidores personalizados con rendimiento compartido con AWS SDK for Java y Creación y administración de secuencias.
- Para más información sobre Kinesis Client Library, consulte Desarrollo de consumidores de KCL 1.x.
- Para obtener más información sobre cómo optimizar su aplicación, consulte Temas avanzados.

Tutorial: Analizar datos bursátiles en tiempo real con aplicaciones de Managed Service para Apache Flink para Flink

El escenario que planteamos en este tutorial comprende la adquisición de operaciones bursátiles en un flujo de datos y la escritura de una aplicación de Amazon Managed Service para Apache Flink sencilla que realiza cálculos con dicho flujo. Aprenderá a enviar un flujo de registros a Kinesis Data Streams e implementar una aplicación que consume y procesa dichos registros casi en tiempo real.

Con las aplicaciones de Managed Service para Apache Flink para aplicaciones Flink, puede utilizar Java o Scala para procesar y analizar datos de streaming. El servicio le permite crear y ejecutar código Java o Scala en orígenes de flujo para realizar análisis de series temporales, alimentar paneles en tiempo real y crear métricas en tiempo real.

Puede crear aplicaciones Flink en Managed Service para Apache Flink mediante bibliotecas de código abierto basadas en Apache Flink. Apache Flink es un marco y un motor usados habitualmente para procesar flujos de datos.

> **Important**
>
> Después de crear dos flujos de datos y una aplicación, su cuenta generará gastos nominales por el uso de Kinesis Data Streams y Managed Service para Apache Flink, ya que no pueden utilizarse en el nivel gratuito de AWS. Cuando termine con esta aplicación, elimine sus recursos de AWS para dejar de incurrir en gastos.

El código no obtiene acceso a datos bursátiles reales, sino que, en su lugar, simula la secuencia de operaciones bursátiles. Lo hace a través de un generador de operaciones bursátiles aleatorias. Si

tiene acceso a una secuencia de operaciones bursátiles en tiempo real, puede que le interese derivar estadísticas útiles y puntuales a partir de dicha secuencia. Por ejemplo, es posible que desee realizar un análisis de ventana deslizante en el que se determina el valor más popular adquirido durante los últimos 5 minutos. O también cabe la posibilidad de que quiera recibir una notificación cada vez que haya una orden de venta que sea demasiado grande (es decir, con demasiadas acciones). Puede ampliar el código de esta serie para proporcionar esta funcionalidad.

Los ejemplos mostrados utilizan la región Oeste de EE. UU. (Oregón), pero funcionan en cualquiera de las regiones de AWS compatibles con Managed Service para Apache Flink.

Tareas

- Requisitos previos para completar los ejercicios
- Paso 1: Configurar una cuenta de AWS y crear un usuario administrador
- Paso 2: Configuración de la AWS Command Line Interface (AWS CLI)
- Paso 3: Crear y ejecutar una aplicación de Managed Service para Apache Flink para Flink

Requisitos previos para completar los ejercicios

Para completar los pasos de esta guía, debe disponer de lo siguiente:

- Java Development Kit (JDK), versión 8. Establezca la variable de entorno `JAVA_HOME` para señalar la ubicación de la instalación del JDK.
- Le recomendamos utilizar un entorno de desarrollo (como Eclipse Java Neon o IntelliJ Idea) para desarrollar y compilar su aplicación.
- Cliente Git. Si aún no lo ha hecho, instale el cliente Git.
- Apache Maven Compiler Plugin. Maven debe estar en su ruta de trabajo. Para probar la instalación de Apache Maven, introduzca lo siguiente:

```
$ mvn -version
```

Para empezar, vaya a Paso 1: Configurar una cuenta de AWS y crear un usuario administrador.

Paso 1: Configurar una cuenta de AWS y crear un usuario administrador

Antes de utilizar aplicaciones de Amazon Managed Service para Apache Flink para Flink por primera vez, complete las siguientes tareas:

1. Registrarse en AWS
2. Creación de un usuario de IAM

Registrarse en AWS

Cuando se registre en Amazon Web Services (AWS), su cuenta de AWS se registrará automáticamente en todos los servicios de AWS, incluido Amazon Managed Service para Apache Flink. Solo se le cobrará por los servicios que utilice.

Con Managed Service para Apache Flink, solo paga por los recursos que utiliza. Si es un nuevo cliente de AWS, puede comenzar a utilizar Managed Service para Apache Flink de forma gratuita. Para obtener más información, consulte Capa gratuita de AWS.

Si ya dispone de una cuenta de AWS, pase a la siguiente tarea. Si no dispone de una cuenta de AWS, siga estos pasos para crear una.

Para crear una cuenta de AWS

1. Abra https://portal.aws.amazon.com/billing/signup.
2. Siga las instrucciones que se le indiquen.

   Parte del procedimiento de registro consiste en recibir una llamada telefónica e indicar un código de verificación en el teclado del teléfono.

   Al registrarse en una Cuenta de AWS, se crea un Usuario raíz de la cuenta de AWS. El usuario raíz tendrá acceso a todos los Servicios de AWS y recursos de esa cuenta. Como práctica recomendada de seguridad, asigne acceso administrativo a un usuario administrativo y utilice únicamente el usuario raíz para realizar tareas que requieran acceso de usuario raíz.

Anote su ID de cuenta de AWS porque lo necesitará en la siguiente tarea.

Creación de un usuario de IAM

Para poder acceder a los servicios de AWS, como Amazon Managed Service para Apache Flink, debe proporcionar credenciales. Esto es así para que el servicio puede determinar si usted tiene permisos para obtener acceso a los recursos que son propiedad de tal servicio. La AWS Management Console exige que introduzca su contraseña.

Puede crear claves de acceso para su cuenta de AWS para tener acceso a la AWS CLI (AWS Command Line Interface) o a la API. Sin embargo, no es recomendable acceder a AWS con las credenciales de su cuenta de AWS. En su lugar, le recomendamos que utilice AWS Identity and Access Management (IAM). Cree un usuario de IAM, añada el usuario a un grupo de IAM con permisos administrativos y, a continuación, conceda permisos administrativos al usuario de IAM que ha creado. Luego, podrá acceder a AWS mediante una URL especial y esas credenciales de usuario de IAM.

Si se registró en AWS pero no ha creado un usuario de IAM, puede crear uno con la consola de IAM.

En los ejercicios de introducción de esta guía se supone que tiene un usuario con permisos de administrador (`adminuser`). Siga el procedimiento para crear `adminuser` en su cuenta.

Para crear un grupo de administradores

1. Inicie sesión en la AWS Management Console y abra la consola de IAM en https://console.aws.amazon.com/iam/.
2. En el panel de navegación, elija Grupos y, a continuación, elija Crear nuevo grupo.
3. En Nombre de grupo, escriba el nombre del grupo, como **`Administrators`**, y seleccione Paso siguiente.
4. En la lista de políticas, seleccione la casilla de verificación situada junto a la AdministratorAccesspolítica. Puede utilizar el menú de filtros y el cuadro de búsqueda para filtrar la lista de políticas.
5. Elija Siguiente paso y luego seleccione Crear grupo.

El grupo nuevo figura en la lista Nombre del grupo.

Para crear un usuario de IAM para usted, agréguelo al grupo Administradores y cree una contraseña

1. En el panel de navegación, elija Users y luego elija la opción Add user.
2. En el cuadro Nombre de usuario, escriba un nombre de usuario.
3. Seleccione Acceso mediante programación y Acceso con la consola de administración de AWS.
4. Elija Siguiente: permisos.
5. Seleccione la casilla de verificación situada junto al grupo de Administradores. A continuación, seleccione Next: Review.

6. Seleccione la opción Crear un usuario.

Para iniciar sesión como el nuevo usuario de IAM

1. Cierre la sesión de la AWS Management Console.
2. Utilice el siguiente formato de URL para iniciar sesión en la consola:

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

   aws_account_number es su ID de cuenta de AWS sin guiones. Por ejemplo, si su ID de cuenta de AWS es 1234-5678-9012, reemplace *aws_account_number* por **123456789012**. Para más información sobre cómo encontrar su número de cuenta, consulte ID de cuenta de AWS y alias en la Guía del usuario de IAM.
3. Escriba el nombre y la contraseña del usuario de IAM que acaba de crear. Cuando haya iniciado sesión, en la barra de navegación se mostrará: *su_nombre_de_usuario* @ *su_id_de_cuenta_de_aws*.

> **Note**
>
> Si no desea que la dirección URL de la página de inicio de sesión contenga el ID de su cuenta de AWS, puede crear un alias de cuenta.

Para crear o eliminar un alias de cuenta

1. Abra la consola de IAM en https://console.aws.amazon.com/iam/.
2. En el panel de navegación, elija Panel.
3. Busque el enlace de inicio de sesión de los usuarios de IAM.
4. Para crear el alias, elija Personalizar. Escriba el nombre que desea utilizar para el alias y, a continuación, elija Sí, crear.
5. Para eliminar el alias, elija Personalizar y, a continuación, elija Sí, eliminar. La dirección URL de inicio de sesión volverá a utilizar el ID de cuenta de AWS.

Para iniciar sesión después de crear un alias de cuenta, use la siguiente dirección URL:

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

Para verificar el enlace de inicio de sesión de los usuarios de IAM de su cuenta, abra la consola de IAM y compruebe el valor de IAM users sign-in link (Enlace de inicio de sesión de los usuarios de IAM) en el panel.

Para obtener más información sobre IAM, consulte lo siguiente:

- AWS Identity and Access Management (IAM)
- Introducción
- Guía del usuario de IAM

Paso siguiente

Paso 2: Configuración de la AWS Command Line Interface (AWS CLI)

Paso 2: Configuración de la AWS Command Line Interface (AWS CLI)

En este paso, descargue y configure la AWS CLI para utilizarla con las aplicaciones de Amazon Managed Service para Apache Flink para Flink.

> **Note**
>
> En los ejercicios introductorios de esta guía se presupone que está utilizando las credenciales de administrador (`adminuser`) en su cuenta para realizar las operaciones.

> **Note**
>
> Si ya tiene la AWS CLI instalada, es posible que tenga que actualizarla para obtener las últimas funcionalidades. Para obtener más información, consulte Instalación de la interfaz de la línea de comandos de AWS en la Guía del usuario de AWS Command Line Interface. Ejecute el siguiente comando para comprobar la versión de la AWS CLI:
>
> ```
> aws --version
> ```
>
> Los ejercicios de este tutorial requieren la siguiente versión de la AWS CLI o posterior:
>
> ```
> aws-cli/1.16.63
> ```

Para configurar la AWS CLI

1. Descargue y configure la AWS CLI. Para obtener instrucciones, consulte los siguientes temas en la Guía del usuario de la AWS Command Line Interface:
   - Instalación de la AWS Command Line Interface
   - Configuración de la AWS CLI
2. Añada un perfil con nombre para el usuario administrador en el archivo de configuración de la AWS CLI. Puede utilizar este perfil cuando ejecute los comandos de la AWS CLI. Para obtener más información sobre los perfiles con nombre, consulte Perfiles con nombre en la Guía del usuario de AWS Command Line Interface.

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

   Para ver una lista de las regiones de AWS disponibles, consulte este artículo sobre las regiones y puntos de enlace de AWS en la Referencia general de Amazon Web Services.
3. Verifique la configuración introduciendo el siguiente comando de ayuda en el símbolo del sistema:

   ```
   aws help
   ```

Tras configurar una AWS cuenta y laAWS CLI, puede realizar el siguiente ejercicio, en el que configurará una aplicación de ejemplo y probará la end-to-end configuración.

Paso siguiente

Paso 3: Crear y ejecutar una aplicación de Managed Service para Apache Flink para Flink

Paso 3: Crear y ejecutar una aplicación de Managed Service para Apache Flink para Flink

En este ejercicio, se creará una aplicación de Managed Service para Apache Flink para Flink con flujos de datos como origen y destino.

Esta sección contiene los siguientes pasos:

- Crear dos Amazon Kinesis Data Streams
- Escribir registros de muestra en el flujo de entrada
- Descargar y examinar el código de Java de streaming de Apache Flink
- Compilar el código de la aplicación
- Cargar el código de Java de streaming de Apache Flink
- Crear y ejecutar la aplicación de Managed Service para Apache Flink

Crear dos Amazon Kinesis Data Streams

Antes de crear una aplicación de Managed Service para Apache Flink para Flink en este ejercicio, cree dos flujos de datos de Kinesis (`ExampleInputStream` y `ExampleOutputStream`). Su aplicación utiliza estos flujos para los flujos de origen y destino de la aplicación.

Puede crear estos flujos mediante la consola de Amazon Kinesis o el siguiente comando de la AWS CLI. Para obtener instrucciones de la consola, consulte Creación y actualización de secuencias de datos.

Para crear flujos de datos (AWS CLI)

1. Para crear el primer flujo (`ExampleInputStream`), utilice el siguiente comando de la AWS CLI `create-stream` de Amazon Kinesis.

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

2. Para crear el segundo flujo que la aplicación utilizará para escribir la salida, ejecute el mismo comando, cambiando el nombre a `ExampleOutputStream`.

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

Escribir registros de muestra en el flujo de entrada

En esta sección, se utiliza un script de Python para escribir registros de muestra en el flujo para que la aplicación los procese.

> **Note**
>
> Esta sección requiere AWS SDK for Python (Boto).

1. Cree un archivo denominado `stock.py` con el siguiente contenido:

```
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. Más adelante en el tutorial, se ejecuta el script `stock.py` para enviar datos a la aplicación.

```
$ python stock.py
```

Descargar y examinar el código de Java de streaming de Apache Flink

El código de la aplicación Java para estos ejemplos está disponible en GitHub. Para descargar el código de la aplicación, haga lo siguiente:

1. Clone el repositorio remoto con el siguiente comando:

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

2. Vaya al directorio `GettingStarted`.

El código de la aplicación se encuentra en los archivos `CustomSinkStreamingJob.java` y `CloudWatchLogSink.java`. Tenga en cuenta lo siguiente en relación con el código de la aplicación:

- La aplicación utiliza un origen de Kinesis para leer del flujo de origen. El siguiente fragmento crea el receptor de Kinesis:

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

Compilar el código de la aplicación

En esta sección, se utiliza el compilador Apache Maven para crear el código de Java para la aplicación. Para obtener más información sobre la instalación de Apache Maven y el Java Development Kit (JDK), consulte Requisitos previos para completar los ejercicios.

La aplicación de Java requiere los siguientes componentes:

- Un archivo Project Object Model (pom.xml). Este archivo contiene información sobre la configuración y las dependencias de la aplicación, incluidas las bibliotecas de aplicaciones de Managed Service para Apache Flink para Flink.
- Un método `main` que contiene la lógica de la aplicación.

> **Note**
>
> Para utilizar el conector de Kinesis para la siguiente aplicación, es necesario descargar el código fuente del conector y compilarlo tal y como se describe en la documentación de Apache Flink.

Para crear y compilar el código de la aplicación

1. Cree una aplicación de Java/Maven en su entorno de desarrollo. Para obtener más información acerca de cómo crear una aplicación, consulte la documentación de su entorno de desarrollo:
   - Creating your first Java project (Eclipse Java Neon)
   - Creating, Running and Packaging Your First Java Application (IntelliJ Idea)
2. Utilice el siguiente código para un archivo llamado `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");
    }
}
```

Tenga en cuenta lo siguiente en relación con el ejemplo de código anterior:

- Este archivo contiene el método `main` que define la funcionalidad de la aplicación.
- La aplicación crea conectores de origen y recepción para obtener acceso a recursos externos usando un objeto `StreamExecutionEnvironment`.
- La aplicación crea conectores de origen y recepción mediante propiedades estáticas. Para utilizar propiedades dinámicas de la aplicación, utilice los métodos `createSourceFromApplicationProperties` y `createSinkFromApplicationProperties` para crear los conectores. Estos métodos leen las propiedades de la aplicación para configurar los conectores.

3. Para utilizar el código de la aplicación, compile y empaquete el código en un archivo JAR. Puede compilar y empaquetar el código de una de las dos formas siguientes:

   - Utilice la herramienta de línea de comandos de Maven. Cree su archivo JAR ejecutando el siguiente comando en el directorio que contiene el archivo `pom.xml`:

```
mvn package
```

- Use el entorno de desarrollo. Consulte la documentación de su entorno de desarrollo para obtener más información.

Puede cargar el paquete como un archivo JAR o puede comprimir el paquete y cargarlo como un archivo ZIP. Si crea su aplicación mediante la AWS CLI, debe especificar el tipo de contenido del código (JAR o ZIP).

4. Si hay errores al compilar, verifique que la variable de entorno `JAVA_HOME` se ha configurado correctamente.

Si la aplicación se compila correctamente, se crea el siguiente archivo:

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

Cargar el código de Java de streaming de Apache Flink

En esta sección, creará un bucket de Amazon Simple Storage Service (Amazon S3) y cargará el código de la aplicación.

Para cargar el código de la aplicación

1. Abra la consola de Amazon S3 en https://console.aws.amazon.com/s3/.
2. Elija Crear bucket.
3. Escriba **ka-app-code-*<username>*** en el campo Nombre del bucket. Añada un sufijo al nombre del bucket, como su nombre de usuario, para que sea único a nivel global. Elija Siguiente.
4. En el paso Configurar opciones, deje los ajustes tal y como están y elija Siguiente.
5. En el paso Establecer permisos, deje los ajustes tal y como están y elija Siguiente.
6. Elija Crear bucket.
7. En la consola de Amazon S3, elija el <username>bucket ka-app-code- y elija Upload.
8. En el paso Seleccionar archivos, elija Añadir archivos. Vaya al archivo `java-getting-started-1.0.jar` que creó en el paso anterior. Elija Siguiente.
9. En el paso Establecer permisos, deje los ajustes tal y como están. Elija Siguiente.

10. En el paso Establecer propiedades, deje los ajustes tal y como están. Seleccione Cargar.

El código de la aplicación ya está almacenado en un bucket de Amazon S3 al que la aplicación puede acceder.

Crear y ejecutar la aplicación de Managed Service para Apache Flink

Puede crear y ejecutar una aplicación de Managed Service para Apache Flink para Flink mediante la consola o la AWS CLI.

> **Note**
>
> Cuando crea la aplicación mediante la consola, sus recursos AWS Identity and Access Management (de IAM) y de Amazon CloudWatch Logs se crean automáticamente. Si crea la aplicación mediante la AWS CLI, debe crear estos recursos por separado.

Temas

- Crear y ejecutar la aplicación (consola)
- Crear y ejecutar la aplicación (AWS CLI)

Crear y ejecutar la aplicación (consola)

Siga estos pasos para crear, configurar, actualizar y ejecutar la aplicación mediante la consola.

Crear la aplicación

1. Abra la consola de Kinesis en https://console.aws.amazon.com/kinesis.
2. En el panel de Amazon Kinesis, elija Crear aplicación de análisis.
3. En la página Kinesis Analytics - Crear aplicación, proporcione la siguiente información:
   - En Nombre de la aplicación, escriba **`MyApplication`**.
   - En Descripción, escriba **`My java test app`**.
   - En Tiempo de ejecución, escriba Apache Flink 1.6.
4. Para Permisos de acceso, seleccione Crear o actualizar rol de IAM **`kinesis-analytics-MyApplication-us-west-2`**.
5. Elija Crear aplicación.

> **Note**
>
> Cuando crea una aplicación de Managed Service para Apache Flink para Flink mediante la consola, tiene la opción de tener un rol de IAM y una política creada para su aplicación. La aplicación utiliza este rol y la política para acceder a los recursos dependientes. Estos recursos de IAM reciben un nombre usando el nombre de la aplicación y la región tal y como se indica a continuación:
>
> - Política: `kinesis-analytics-service-MyApplication-us-west-2`
> - Rol: `kinesis-analytics-MyApplication-us-west-2`

Editar la política de IAM

Edite la política de IAM para agregar permisos de acceso a los flujos de datos de Kinesis.

1. Abra la consola de IAM en https://console.aws.amazon.com/iam/.
2. Elija Políticas. Elija la política **kinesis-analytics-service-MyApplication-us-west-2** que la consola creó en su nombre en la sección anterior.
3. En la página Resumen, elija Editar política. Seleccione la pestaña JSON.
4. Añada la sección subrayada de la siguiente política de ejemplo a la política. Reemplace el ID de la cuenta de muestra (*012345678901*) por el ID de su cuenta.

```
{
    "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"
        }
    ]
```

```
}
```

Configurar la aplicación

1. En la MyApplicationpágina, selecciona Configurar.
2. En la página Configurar aplicación, proporcione la Ubicación del código:
   - Para el bucket de Amazon S3, introduzca **`ka-app-code-<username>`**.
   - En Ruta al objeto de Amazon S3, introduzca **`java-getting-started-1.0.jar`**.
3. En Acceso a los recursos de la aplicación, en Permisos de acceso, seleccione Crear o actualizar rol de IAM **`kinesis-analytics-MyApplication-us-west-2`**.
4. En Propiedades, en ID de grupo, escriba **`ProducerConfigProperties`**.
5. Escriba las siguientes propiedades y valores de la aplicación:

Clave	Valor
`flink.inputstream.initpos`	**`LATEST`**
`aws:region`	**`us-west-2`**
`AggregationEnabled`	**`false`**

6. En Monitorización, asegúrese de que el Nivel de métricas de monitorización se ha establecido en Aplicación.
7. Para el CloudWatch registro, active la casilla Activar.
8. Seleccione Actualizar.

> **Note**
>
> Si decide habilitar el CloudWatch registro, Managed Service for Apache Flink crea un grupo de registros y un flujo de registros automáticamente. Los nombres de estos recursos son los siguientes:
>
> - Grupo de registro: `/aws/kinesis-analytics/MyApplication`
> - Flujo de registro: `kinesis-analytics-log-stream`

Ejecutar la aplicación

1. En la MyApplicationpágina, seleccione Ejecutar. Confirme la acción.
2. Cuando la aplicación se está ejecutando, actualice la página. La consola muestra el Gráfico de la aplicación.

Detener la aplicación

En la MyApplicationpágina, selecciona Detener. Confirme la acción.

Actualizar la aplicación

Mediante la consola, puede actualizar la configuración de la aplicación, tal como sus propiedades, ajustes de monitorización y la ubicación o el nombre de archivo JAR de la aplicación. También puede volver a cargar el JAR de la aplicación del bucket de Amazon S3 si necesita actualizar el código de la aplicación.

En la MyApplicationpágina, elija Configurar. Actualice la configuración de la aplicación y elija Actualizar.

Crear y ejecutar la aplicación (AWS CLI)

En esta sección se utiliza la AWS CLI para crear y ejecutar la aplicación de Managed Service para Apache Flink. Las aplicaciones de Managed Service para Apache Flink para Flink utilizan el comando `kinesisanalyticsv2` de la AWS CLI para crear e interactuar con las aplicaciones de Managed Service para Apache Flink.

Crear una política de permisos

En primer lugar, debe crear una política de permisos con dos instrucciones: una que concede permisos para la acción `read` en el flujo de origen y otra que concede permisos para las acciones `write` en el flujo de recepción. A continuación, asocie la política a un rol de IAM (que se crea en la siguiente sección). Por lo tanto, cuando Managed Service para Apache Flink asume el rol, el servicio tiene los permisos necesarios para leer desde el flujo de origen y escribir en el flujo de recepción.

Utilice el siguiente código para crear la política de permisos `KAReadSourceStreamWriteSinkStream`. Reemplace *`username`* por el nombre de usuario que se utilizó para crear el bucket de Amazon S3 para almacenar el código de la aplicación. Sustituya el ID de la cuenta en los nombres de recurso de Amazon (ARN) (*`012345678901`*) por el ID de su cuenta.

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

Para step-by-step obtener instrucciones sobre cómo crear una política de permisos, consulte el tutorial sobre cómo crear y adjuntar su primera política gestionada por el cliente en la Guía del usuario de IAM.

> **Note**
>
> Para acceder a otros servicios de AWS, puede usar AWS SDK for Java. Managed Service para Apache Flink configura automáticamente las credenciales requeridas por el SDK con las del rol de IAM de ejecución del servicio que está asociado a su aplicación. No hace falta realizar ningún otro paso.

Creación de un rol de IAM

En esta sección se crea un rol de IAM que la aplicación de Managed Service para Apache Flink para Flink puede adoptar para leer un flujo de origen y escribir en el flujo de destino.

Managed Service para Apache Flink no puede acceder a su flujo sin permisos. Estos permisos se conceden a través del rol de IAM. Cada rol de IAM tiene dos políticas asociadas. La política de confianza concede a Managed Service para Apache Flink permiso para asumir el rol, y la política de permisos determina lo que Managed Service para Apache Flink puede hacer después de asumir el rol.

Usted deberá asociar la política de permisos que ha creado en la sección anterior a este rol.

Para crear un rol de IAM

1. Abra la consola de IAM en https://console.aws.amazon.com/iam/.
2. En el panel de navegación, elija Roles, Crear rol.
3. En Seleccionar tipo de entidad de confianza, elija Servicio de AWS. En Elegir el servicio que usará este rol, elija Kinesis. En Seleccionar su caso de uso, elija Kinesis Analytics.

   Elija Siguiente: permisos.
4. En la página Asociar políticas de permisos, elija Siguiente: Revisión. Asociará políticas de permisos después de crear el rol.
5. En la página Crear rol, escriba **`KA-stream-rw-role`** como Nombre de rol. Elija Crear rol.

   Ahora ha creado un nuevo rol de IAM llamado `KA-stream-rw-role`. A continuación, actualice las políticas de confianza y permisos del rol.
6. Asocie la política de permisos al rol.

   > **Note**
   >
   > Para este ejercicio, Managed Service para Apache Flink asume este rol tanto para leer datos de un flujo de datos de Kinesis (origen) como para escribir la salida en otro flujo de datos de Kinesis. Asocie la política que ha creado en el paso anterior, the section called "Crear una política de permisos".

   a. En la página Resumen, elija la pestaña Permisos.

b. Seleccione Asociar políticas.

c. En el campo de búsqueda, escriba **`KAReadSourceStreamWriteSinkStream`** (la política que ha creado en la sección anterior).

d. Elija la ReadInputStreamWriteOutputStream política de KA y, a continuación, seleccione Adjuntar política.

Ahora ha creado el rol de ejecución de servicio que utiliza la aplicación para obtener acceso a los recursos. Anote el ARN del nuevo rol.

Para step-by-step obtener instrucciones sobre cómo crear un rol, consulte Creación de un rol de IAM (consola) en la Guía del usuario de IAM.

Crear la aplicación de Managed Service para Apache Flink

1. Copie el siguiente código JSON en un archivo denominado `create_request.json`. Cambie el ARN del rol de ejemplo por el ARN del rol que ha creado antes. Reemplace el sufijo del ARN del bucket (*`username`*) por el sufijo que eligió en la sección anterior. Reemplace el ID de la cuenta de ejemplo (*`012345678901`*) del rol de ejecución del servicio por el ID de su cuenta.

```
{
    "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. Ejecute la acción `CreateApplication` con la solicitud anterior para crear la aplicación:

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

Se ha creado la aplicación. Puede iniciar la aplicación en el siguiente paso.

Iniciar la aplicación

En esta sección, se utiliza la acción `StartApplication` para iniciar la aplicación.

Para iniciar la aplicación

1. Guarde el siguiente código JSON en un archivo denominado `start_request.json`.

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

2. Ejecute la acción `StartApplication` con la solicitud anterior para iniciar la aplicación:

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

Ya se debe estar ejecutando la aplicación. Puedes comprobar las métricas de Managed Service for Apache Flink en la CloudWatch consola de Amazon para comprobar que la aplicación funciona.

Detener la aplicación

En esta sección, se utiliza la acción `StopApplication` para detener la aplicación.

Para detener la aplicación

1. Guarde el siguiente código JSON en un archivo denominado `stop_request.json`.

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

2. Ejecute la acción `StopApplication` con la siguiente solicitud para detener la aplicación:

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

La aplicación se habrá detenido.

Tutorial: Uso de AWS Lambda con secuencias de datos de Amazon Kinesis Data Streams

En este tutorial, creará una función de Lambda para consumir eventos de un flujo de datos de Kinesis. En este escenario de ejemplo, una aplicación personalizada escribe registros en un flujo de datos de Kinesis. AWS A continuación, Lambda sondea este flujo de datos y, cuando detecta nuevos registros de datos, invoca su función de Lambda. AWS A continuación, Lambda ejecuta la función de Lambda asumiendo el rol de ejecución que especificó al crear la función de Lambda.

Para obtener instrucciones detalladas paso a paso, consulte Tutorial: Uso de AWS Lambda con Amazon Kinesis.

> **Note**
>
> En este tutorial, se presupone que tiene algunos conocimientos sobre las operaciones básicas de Lambda y la consola de Lambda. Si aún no lo ha hecho, siga las instrucciones de Introducción a AWS Lambda para crear su primera función de Lambda.

Solución de datos de flujos de AWS para Amazon Kinesis

La solución de datos de flujos de AWS para Amazon Kinesis configura automáticamente los servicios de AWS necesarios para capturar, almacenar, procesar y entregar fácilmente los datos de flujos. La solución ofrece varias opciones para resolver casos de uso de datos de flujos que utilizan varios servicios de AWS, incluidos Kinesis Data Streams, AWS Lambda, Amazon API Gateway y Amazon Managed Service para Apache Flink.

Cada solución incluye los siguientes componentes:

- Un paquete de AWS CloudFormation para implementar el ejemplo completo.
- Un panel de CloudWatch para mostrar las métricas de las aplicaciones.
- CloudWatch emite alarmas sobre las métricas de aplicaciones más pertinentes.
- Todas las políticas y roles de IAM necesarios.

La solución se encuentra aquí: Solución de datos de flujos para Amazon Kinesis

Creación y administración de secuencias

Amazon Kinesis Data Streams adquiere una gran cantidad de datos en tiempo real, almacena de forma duradera los datos y hace que estos estén disponibles para su consumo. La unidad de datos almacenada por Kinesis Data Streams es un registro de datos. Una secuencia de datos representa un grupo de registros de datos. Los registros de datos de una secuencia de datos se distribuyen en fragmentos.

Un fragmento contiene una sucesión de registros de datos en una secuencia. Sirve como unidad de rendimiento base de un flujo de datos de Kinesis. Una partición admite 1 MB/s y 1000 registros por segundo para escrituras y 2 MB/s para lecturas en los modos de capacidad bajo demanda y aprovisionada. Los límites de la partición garantizan un rendimiento predecible, lo que facilita el diseño y funcionamiento de un flujo de datos de alta fiabilidad.

Temas

Elegir el modo de capacidad del flujo de datos

Temas

¿Qué es el modo de capacidad de un flujo de datos?

Un modo de capacidad determina cómo se administra la capacidad de un flujo de datos y cómo se le cobra por el uso de su flujo de datos. En Amazon Kinesis Data Streams, puede elegir entre un modo bajo demanda y un modo aprovisionado para sus flujos de datos.

- Bajo demanda: los flujos de datos con un modo bajo demanda no requieren planificación de la capacidad y se escalan automáticamente para gestionar gigabytes de escritura y rendimiento de lectura por minuto. Con el modo bajo demanda, Kinesis Data Streams administra automáticamente las particiones para proporcionar el rendimiento necesario.
- Aprovisionado: para el flujo de datos con un modo aprovisionado, debe especificar el número de particiones para el flujo de datos. La capacidad total de un flujo de datos es la suma de las capacidades de las particiones. Puede aumentar o reducir el número de particiones de un flujo de datos según sea necesario.

Puede utilizar las API `PutRecord` y `PutRecords` de Kinesis Data Streams para escribir datos en sus flujos de datos tanto en el modo de capacidad bajo demanda como en el de capacidad aprovisionada. Para recuperar datos, ambos modos de capacidad admiten consumidores predeterminados que utilizan la API `GetRecords` y consumidores de Distribución ramificada mejorada (EFO) que utilizan la API `SubscribeToShard`.

Todas las capacidades de Kinesis Data Streams, incluidos el modo de retención, el cifrado y las métricas de supervisión, entre otros, son compatibles tanto para el modo bajo demanda como para el modo aprovisionado. Kinesis Data Streams proporciona una alta durabilidad y disponibilidad tanto en el modo bajo demanda como en el modo de capacidad aprovisionada.

Modo bajo demanda

Los flujos de datos en el modo bajo demanda no requieren planificación de capacidad y se escalan automáticamente para manejar gigabytes de rendimiento de escritura y lectura por minuto. El modo bajo demanda simplifica la ingesta y el almacenamiento de grandes volúmenes de datos con baja latencia, ya que elimina el aprovisionamiento y la gestión de servidores, almacenamiento o rendimiento. Puede ingerir miles de millones de registros al día sin ninguna sobrecarga operativa.

El modo bajo demanda es ideal para responder a las necesidades de un tráfico de aplicaciones muy variable e impredecible. Ya no tendrá que aprovisionar estas cargas de trabajo para los picos de capacidad, lo que puede dar lugar a costos más elevados debido al bajo uso. El modo bajo demanda es adecuado para cargas de trabajo con patrones de tráfico impredecibles y muy variables.

Con el modo de capacidad bajo demanda, paga por GB de datos escritos y leídos de sus flujos de datos. No necesita especificar el rendimiento de lectura y escritura que espera de su aplicación. Kinesis Data Streams se adapta instantáneamente a sus cargas de trabajo a medida que aumentan o disminuyen. Para obtener más información, consulte los Precios de Amazon Kinesis Data Streams.

Puede crear un nuevo flujo de datos con el modo bajo demanda mediante la consola, las API o los comandos de la CLI de Kinesis Data Streams.

Un flujo de datos en el modo bajo demanda admite hasta el doble del rendimiento de escritura máximo observado en los 30 días anteriores. A medida que el rendimiento de escritura de su flujo de datos alcanza un nuevo pico, Kinesis Data Streams escala automáticamente la capacidad del flujo de datos. Por ejemplo, si el flujo de datos tiene un rendimiento de escritura que varía entre 10 MB/s y 40 MB/s, Kinesis Data Streams garantiza que pueda alcanzar fácilmente el doble de su rendimiento máximo anterior, es decir, 80 MB/s. Si el mismo flujo de datos mantiene un nuevo rendimiento máximo de 50 MB/s, Kinesis Data Streams garantiza que hay capacidad suficiente para ingerir 100 MB/s de rendimiento de escritura. Sin embargo, puede producirse una limitación de escritura si el tráfico aumenta a más del doble del pico anterior en un periodo de 15 minutos. En este caso, es necesario reintentar las peticiones limitadas.

La capacidad de lectura agregada de un flujo de datos con el modo bajo demanda aumenta proporcionalmente al rendimiento de escritura. Esto ayuda a garantizar que las aplicaciones de consumo siempre tengan un rendimiento de lectura adecuado para procesar los datos entrantes en tiempo real. Con la API `GetRecords` se obtiene al menos el doble de rendimiento de escritura que de lectura de datos. Le recomendamos que utilice una aplicación de consumo con la API `GetRecord`, para que tenga espacio suficiente de recuperación cuando la aplicación necesite reponerse de un tiempo de inactividad. Se recomienda que utilice la capacidad de distribución ramificada mejorada de Kinesis Data Streams para escenarios que requieran agregar más de una aplicación consumidora. La distribución ramificada mejorada admite la adición de hasta 20 aplicaciones consumidoras a un flujo de datos mediante la API `SubscribeToShard`, y cada aplicación consumidora tiene un rendimiento dedicado.

Manejo de excepciones de rendimiento de lectura y escritura

Con el modo de capacidad bajo demanda (igual que con el modo de capacidad aprovisionada), debe especificar una clave de partición con cada registro para escribir datos en su flujo de datos. Kinesis Data Streams utiliza sus claves de partición para distribuir los datos entre las particiones. Kinesis Data Streams supervisa el tráfico de cada partición. Cuando el tráfico entrante supera los 500 KB/s por partición divide la partición en 15 minutos. Los valores de clave hash de la partición principal se redistribuyen uniformemente entre las particiones secundarias.

Si el tráfico entrante supera el doble de su pico anterior, puede experimentar excepciones de lectura o escritura durante unos 15 minutos, incluso cuando sus datos se distribuyan uniformemente entre las particiones. Le recomendamos que reintente todas estas solicitudes para que todos los registros se almacenen correctamente en Kinesis Data Streams.

Puede experimentar excepciones de lectura y escritura si utiliza una clave de partición que provoque una distribución desigual de los datos y los registros asignados a una partición concreta superan sus límites. Con el modo bajo demanda, el flujo de datos se adapta automáticamente para manejar patrones de distribución de datos desiguales a menos que una sola clave de partición supere los límites de rendimiento de 1 MB/s y 1000 registros por segundo de una partición.

En el modo bajo demanda, Kinesis Data Streams divide las particiones de manera uniforme cuando detecta un aumento del tráfico. Sin embargo, no detecta ni aísla las claves hash que dirigen una mayor parte del tráfico entrante a una partición concreta. Si utiliza claves de partición muy desiguales, es posible que siga recibiendo excepciones de escritura. Para estos casos de uso, le recomendamos que utilice el modo de capacidad aprovisionada que admite divisiones de particiones granulares.

Modo aprovisionado

Con el modo aprovisionado, después de crear el flujo de datos, puede aumentar o reducir dinámicamente la capacidad de su partición mediante la API AWS Management Console o la UpdateShardCountAPI. Puede realizar actualizaciones mientras haya una aplicación productora o consumidora de Kinesis Data Streams que escriba o lea datos del flujo.

El modo aprovisionado es adecuado para tráfico predecible con requisitos de capacidad fáciles de prever. Puede utilizar el modo aprovisionado si desea un control detallado sobre cómo se distribuyen los datos entre las particiones.

Con el modo aprovisionado, debe especificar el número de particiones del flujo de datos. Para determinar el tamaño de un flujo de datos con el modo aprovisionado, necesita los siguientes valores de entrada:

- El tamaño promedio del registro de datos escrito en el flujo en kilobytes (KB), redondeado al 1 KB más cercano (`average_data_size_in_KB`).
- El número de registros de datos escritos en la secuencia de y leídos desde esta por segundo (`records_per_second`).
- El número de consumidores, que son aplicaciones de Kinesis Data Streams que consumen datos de forma simultánea e independiente del flujo (`number_of_consumers`).

- El ancho de banda de escritura entrante (en KB`incoming_write_bandwidth_in_KB`), que es igual a `average_data_size_in_KB` multiplicado por `records_per_second`.
- El ancho de banda de lectura saliente (en KB`outgoing_read_bandwidth_in_KB`), que es igual a `incoming_write_bandwidth_in_KB` multiplicado por `number_of_consumers`.

Puede calcular el número de particiones (`number_of_shards`) que su flujo necesita mediante los valores de entrada de la siguiente fórmula.

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

En el modo aprovisionado, es posible que se produzcan excepciones en el rendimiento de lectura y escritura si no se configura el flujo de datos para manejar los picos de rendimiento. En este caso, debe escalar manualmente su flujo de datos para acomodar el tráfico de datos.

También puede experimentar excepciones de lectura y escritura si utiliza una clave de partición que provoque una distribución desigual de los datos y los registros signados a una partición superan sus límites. Para resolver este problema en el modo aprovisionado, identifique estas particiones y divídalas manualmente para acomodar mejor su tráfico. Para obtener más información, consulte Redistribución de un flujo.

Cambio entre los modos de capacidad

Puede cambiar el modo de capacidad de su flujo de datos de bajo demanda a aprovisionado, o de aprovisionado a bajo demanda. Todos los flujos de datos de su cuenta de AWS permiten que se cambie entre el modo de capacidad bajo demanda y el modo de capacidad aprovisionado dos veces en un plazo de 24 horas.

El cambio entre los modos de capacidad de un flujo de datos no provoca ninguna interrupción en las aplicaciones que utilizan este flujo de datos. Puede seguir escribiendo y leyendo de este flujo de datos. Al cambiar entre los modos de capacidad, ya sea de bajo demanda a provisionado o de provisionado a bajo demanda, el estado del flujo pasa a Actualizando. Debe esperar a que el estado del flujo de datos pase a Activo para poder modificar de nuevo sus propiedades.

Al cambiar del modo de capacidad aprovisionada al modo bajo demanda, el flujo de datos conserva inicialmente el recuento de particiones que tenía antes de la transición y, a partir de ese momento, Kinesis Data Streams supervisa el tráfico de datos y escala el recuento de particiones de este flujo de datos bajo demanda en función del rendimiento de escritura.

Cuando cambia del modo bajo demanda al modo aprovisionado, el flujo de datos también conserva inicialmente el recuento de particiones que tenía antes de la transición, pero a partir de este momento, el usuario es responsable de supervisar y ajustar el recuento de particiones de este flujo de datos para adaptarse adecuadamente al rendimiento de escritura.

Creación de un flujo a través de la consola de administración de AWS

Puede crear un flujo mediante la consola de Kinesis Data Streams, la API de Kinesis Data Streams o la AWS Command Line Interface (AWS CLI).

Para crear una secuencia de datos con la consola

1. Inicie sesión en la AWS Management Console y abra la consola de Kinesis en https://console.aws.amazon.com/kinesis.
2. En la barra de navegación, expanda el selector de regiones y seleccione una región.
3. Elija Crear flujo de datos.
4. En la página Crear flujo de Kinesis, escriba un nombre para su flujo de datos y, a continuación, elija el modo de capacidad Bajo demanda o Aprovisionado. El modo Bajo demanda está seleccionado de forma predeterminada. Para obtener más información, consulte Elegir el modo de capacidad del flujo de datos.

   Con el modo Bajo demanda, puede seleccionar Crear flujo de Kinesis para crear su flujo de datos. Con el modo aprovisionado, debe especificar el número de particiones que necesita y, a continuación, elegir Crear flujo de Kinesis.

   En la página Flujos de Kinesis, el valor Estado del flujo es Creándose mientras se crea. Cuando el flujo está listo para usarse, el valor Estado cambia a Activo.
5. Elija el nombre del flujo. La página Detalles del flujo muestra un resumen de la configuración del flujo, junto con información de monitoreo.

Creación de un flujo mediante la API de Kinesis Data Streams

- Para más información sobre la creación de flujos mediante la API de Kinesis Data Streams, consulte Creación de un flujo a través de las API.

Para crear una secuencia con la AWS CLI

- Para obtener más información acerca de la creación de una secuencia con la AWS CLI, consulte el comando create-stream.

Creación de un flujo a través de las API

Siga estos pasos para crear su flujo de datos de Kinesis.

Creación del cliente de Kinesis Data Streams

Antes de poder trabajar con flujos de datos de Kinesis, debe crear un objeto cliente. El siguiente código Java crea una instancia de un compilador de clientes y la usa para definir la región, las credenciales y la configuración del cliente. A continuación, crea un objeto cliente.

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

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

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

Para obtener más información, consulte Regiones y puntos de conexión de Kinesis Data Streams en Referencia general de AWS.

Creación de la secuencia

Ahora que ya creó su cliente de Kinesis Data Streams, puede crear un flujo con el que trabajar, lo que puede hacer con la consola de Kinesis Data Streams o mediante programación. Para crear un flujo de forma programada, cree una instancia de un objeto `CreateStreamRequest` y especifique un nombre para el flujo y el número de particiones que ha de usar esto.

- Bajo demanda:

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

- Aprovisionado:

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

El nombre de la secuencia identifica la secuencia. El nombre se adapta a la cuenta de AWS que utiliza la aplicación. También se limita a la región. Es decir, dos flujos en dos cuentas diferentes de AWS pueden tener el mismo nombre y dos flujo en la misma cuenta de AWS pero en dos regiones diferentes pueden tener el mismo nombre, pero no dos flujos en la misma cuenta y en la misma región.

El rendimiento de la secuencia es una función del número de fragmentos; se requieren más fragmentos para un rendimiento con mayor aprovisionamiento. El aumento en el número de particiones también hace que se incremente el costo que carga AWS por el flujo. Para obtener más información sobre cómo calcular un número adecuado de fragmentos para una aplicación, consulte Elegir el modo de capacidad del flujo de datos.

Tras configurar el objeto `createStreamRequest`, cree una secuencia llamando al método `createStream` en el cliente. Después de llamar a `createStream`, espere a que la secuencia alcance el estado `ACTIVE` antes de realizar cualquier operación en la secuencia. Para comprobar el estado de la secuencia, llame al método `describeStream`. Sin embargo, `describeStream` arroja una excepción si la secuencia no existe. Por lo tanto, incluya la llamada a `describeStream` en un bloque `try/catch`.

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

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

  try {
    DescribeStreamResult describeStreamResponse =
 client.describeStream( describeStreamRequest );
```

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

Actualización de una secuencia

Puede actualizar los detalles de un flujo mediante la consola de Kinesis Data Streams, la API de Kinesis Data Streams o la AWS CLI.

> **Note**
>
> Puede habilitar el cifrado en el servidor para secuencias existentes o para las secuencias que acaba de crear.

Para actualizar una secuencia de datos con la consola

1. Abra la consola de Amazon Kinesis en https://console.aws.amazon.com/kinesis.
2. En la barra de navegación, expanda el selector de regiones y seleccione una región.
3. Elija el nombre de la secuencia en la lista. La página Detalles del flujo muestra un resumen de la configuración del flujo e información de monitoreo.
4. Para cambiar entre los modos de capacidad bajo demanda y aprovisionada para un flujo de datos, seleccione Editar modo de capacidad en la pestaña Configuración. Para obtener más información, consulte Elegir el modo de capacidad del flujo de datos.

> **Important**
>
> Para cada flujo de datos de la cuenta de AWS, puede cambiar entre los modos bajo demanda y aprovisionado dos veces en 24 horas.

5. En el caso de un flujo de datos con el modo aprovisionado, para editar el número de particiones, seleccione Editar particiones aprovisionadas en la pestaña Configuración y, a continuación, ingrese un nuevo recuento de particiones.
6. Para habilitar el cifrado del servidor para registros de datos, elija Editar en la sección Cifrado en el servidor. Seleccione una clave de KMS para utilizarla como clave maestra de cifrado, o utilice la clave maestra predeterminada, aws/kinesis, administrada por Kinesis. Si habilita el cifrado para una secuencia y utiliza su propia clave maestra de AWS KMS, asegúrese de que las aplicaciones productoras y consumidoras tengan acceso a la clave maestra de AWS KMS que haya utilizado. Para asignar permisos a una aplicación de modo que tenga acceso a una clave de AWS KMS generada por el usuario, consulte the section called “Permisos de para utilizar claves maestras de KMS generadas por el usuario”.
7. Para editar el periodo de retención de datos, elija Editar en la sección Periodo de retención de datos y, a continuación, escriba un nuevo periodo de retención de datos.
8. Si ha habilitado las métricas personalizadas en su cuenta, elija Editar en la sección Métricas en el nivel de partición y, a continuación, especifique las métricas para su flujo. Para obtener más información, consulte the section called “Supervisión del servicio con CloudWatch”.

Actualización de una secuencia con la API

Para actualizar los detalles de una secuencia a través de la API, consulte los siguientes métodos:

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

- UpdateShardCount

Actualización de una secuencia con la AWS CLI

Para obtener información acerca de cómo actualizar una secuencia con la AWS CLI, consulte la Referencia de la CLI de Kinesis.

Visualización de secuencias

Como se describe en la sección anterior, las transmisiones se asignan a la cuenta de AWS asociada con las credenciales de AWS que se utilizan para instanciar el cliente de Kinesis Data Streams y también a la región especificada para el cliente. Una cuenta de AWS podría tener muchos flujos activos en un momento dado. Puede listar sus flujos en la consola de Kinesis Data Streams o mediante programación. El código de esta sección muestra cómo listar todos los flujos de su cuenta de AWS.

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

Este código de muestra crea primero una nueva instancia de `ListStreamsRequest` y llama a su método `setLimit` para especificar que se debería devolver un máximo de 20 secuencias por cada llamada a `listStreams`. Si no especifica un valor para `setLimit`, Kinesis Data Streams devuelve un número de flujos menor o igual que el número de la cuenta. A continuación, el código pasa `listStreamsRequest` al método `listStreams` del cliente. El valor `listStreams` devuelto se almacena en un objeto `ListStreamsResult`. El código llama al método `getStreamNames` en este objeto y almacena los nombres de secuencia devueltos en la lista `streamNames`. Tenga en cuenta que Kinesis Data Streams podría devolver menos flujos de los especificados en el límite establecido, incluso si hay más flujos que en la cuenta y región. Para garantizar que pueda recuperar todas las secuencias, utilice el método `getHasMoreStreams` tal y como se describe en el siguiente código de muestra.

```
while (listStreamsResult.getHasMoreStreams())
{
    if (streamNames.size() > 0) {
      listStreamsRequest.setExclusiveStartStreamName(streamNames.get(streamNames.size()
 - 1));
```

```
    }
    listStreamsResult = client.listStreams(listStreamsRequest);
    streamNames.addAll(listStreamsResult.getStreamNames());
}
```

Este código llama al método `getHasMoreStreams` en `listStreamsRequest` para comprobar si hay secuencias adicionales disponibles más allá de las que se devuelven en la llamada inicial a `listStreams`. En caso afirmativo, el código llama al método `setExclusiveStartStreamName` con el nombre de la última secuencia que se devolviera en la llamada anterior a `listStreams`. El método `setExclusiveStartStreamName` hace que la siguiente llamada a `listStreams` se inicie después de dicha secuencia. El grupo de nombres de secuencia que devuelve esa secuencia se añade a la lista `streamNames`. Este proceso continúa hasta que todos los nombres de secuencia se han recopilado en la lista.

Las secuencias que devuelve `listStreams` pueden estar en uno de los siguientes estados:

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

Puede comprobar el estado de una secuencia mediante el método `describeStream`, tal y como se muestra en la sección anterior, Creación de un flujo a través de las API.

Listado de fragmentos

Un flujo de datos puede tener una o varias particiones. Hay dos métodos para listar (o recuperar) las particiones de un flujo de datos.

Temas

- ListShards API: recomendada
- DescribeStream API: en desuso

ListShards API: recomendada

El método recomendado para enumerar o recuperar los fragmentos de un flujo de datos es utilizar la ListShardsAPI. El siguiente ejemplo muestra cómo obtener una lista de las particiones de un flujo de

datos. Para obtener una descripción completa de la operación principal utilizada en este ejemplo y de todos los parámetros que puede configurar para la operación, consulte. 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());
        }
    }
}
```

Para ejecutar el ejemplo de código anterior puede utilizar un archivo POM como el siguiente.

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

    <groupId>kinesis.data.streams.samples</groupId>
    <artifactId>shards</artifactId>
    <version>1.0-SNAPSHOT</version>
    <build>
```

```
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <configuration>
                    <source>8</source>
                    <target>8</target>
                </configuration>
            </plugin>
        </plugins>
    </build>
    <dependencies>
        <dependency>
            <groupId>software.amazon.awssdk</groupId>
            <artifactId>kinesis</artifactId>
            <version>2.0.0</version>
        </dependency>
    </dependencies>
</project>
```

Con la `ListShards` API, puede usar el ShardFilterparámetro para filtrar la respuesta de la API. Solo puede especificar un filtro a la vez.

Si utilizas el `ShardFilter` parámetro al invocar la ListShards API, `Type` es la propiedad obligatoria y debes especificarla. Si especifica los tipos `AT_TRIM_HORIZON`, `FROM_TRIM_HORIZON` o `AT_LATEST`, no necesita especificar las propiedades opcionales `ShardId` o `Timestamp`.

Si especifica el tipo `AFTER_SHARD_ID`, también deberá proporcionar el valor de la propiedad opcional `ShardId`. La `ShardId` propiedad tiene una funcionalidad idéntica a la del `ExclusiveStartShardId` parámetro de la ListShards API. Cuando se especifica la propiedad `ShardId`, la respuesta incluye las particiones que comienzan con la partición cuyo ID sigue inmediatamente al `ShardId` proporcionado.

Si especifica el tipo `AT_TIMESTAMP` o `FROM_TIMESTAMP_ID`, también deberá proporcionar el valor de la propiedad opcional `Timestamp`. Si especifica el tipo `AT_TIMESTAMP`, se devolverán todas las particiones que estaban abiertas en la marca de tiempo proporcionada. Si se especifica el tipo `FROM_TIMESTAMP`, se devolverán todas las particiones a partir de la fecha y hora indicadas hasta TIP.

Important

Las API `DescribeStreamSummary` y `ListShard` proporcionan una forma más escalable de recuperar información sobre sus flujos de datos. Más específicamente, las cuotas de la DescribeStream API pueden provocar limitaciones. Para obtener más información, consulte Cuotas y límites. Tenga en cuenta también que las cuotas de `DescribeStream` se comparten entre todas las aplicaciones que interactúan con todos los flujos de datos de su cuenta de AWS. Las cuotas de la ListShards API, por otro lado, son específicas para un único flujo de datos. Por lo tanto, no solo se obtiene un TPS más alto con la ListShards API, sino que la acción se amplía mejor a medida que se crean más flujos de datos.
Le recomendamos que migre a todos los productores y consumidores que utilizan la DescribeStream API para que, en su lugar, invoquen la API DescribeStreamSummary y las ListShard API. Para identificar a estos productores y consumidores, recomendamos utilizar Athena para analizar los CloudTrail registros, ya que los agentes de usuario de KPL y KCL se capturan en las llamadas a la API.

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

También recomendamos que las integraciones de AWS Lambda y Amazon Firehose con Kinesis Data Streams que invocan la API `DescribeStream` se reconfiguren para que las integraciones invoquen en su lugar `DescribeStreamSummary` y `ListShards`. En específico, para AWS Lambda, debe actualizar su asignación de orígenes de eventos. En el caso de Amazon Firehose, deben actualizarse los permisos de IAM correspondientes para que incluyan el permiso de IAM de `ListShards`.

DescribeStream API: en desuso

> **Important**
>
> La siguiente información describe una forma actualmente obsoleta de recuperar fragmentos de un flujo de datos a través de la API. DescribeStream En la actualidad, se recomienda utilizar la API `ListShards` para recuperar las particiones que componen el flujo de datos.

El objeto de respuesta que devuelve el método `describeStream` le permite recuperar información sobre los fragmentos que componen la secuencia. Para recuperar los fragmentos, llame al método `getShards` de este objeto. Este método podría no devolver todos los fragmentos de la secuencia en una única llamada. En el siguiente código, comprobaremos el método `getHasMoreShards` en `getStreamDescription` para ver si hay fragmentos adicionales que no se hayan devuelto. Si es así, es decir, si este método devuelve `true`, seguiremos insistiendo en llamar a `getShards` en bucle, añadiendo cada nuevo lote de fragmentos devueltos a nuestra lista de fragmentos. El bucle se cierra cuando `getHasMoreShards` devuelve `false`; es decir, cuando todos los fragmentos se han devuelto. Tenga en cuenta que `getShards` no devolverá fragmentos que se encuentren en el estado `EXPIRED`. Para obtener más información sobre los estados de los fragmentos, incluido el estado `EXPIRED`, consulte Direccionamiento de datos, persistencia de datos y estado de fragmentos tras los cambios en los fragmentos.

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

Eliminar una secuencia

Puede eliminar un flujo con la consola de Kinesis Data Streams o mediante programación. Para eliminar una secuencia de forma programada, utilice `DeleteStreamRequest` tal y como se muestra en el siguiente código.

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

Cierre todas las aplicaciones que utilicen la secuencia antes de eliminarla. Si una aplicación intenta utilizar secuencias eliminadas, recibirá excepciones `ResourceNotFound`. Además, si posteriormente crea una secuencia nueva con el mismo nombre que la secuencia anterior y las aplicaciones que utilizaban esta siguen en ejecución, dichas aplicaciones podrían intentar interactuar con la secuencia nueva como si fuera la secuencia anterior, con resultados impredecibles.

Cambio de los fragmentos de una secuencia

> **Important**
>
> Puedes volver a fragmentar tu transmisión mediante la API. UpdateShardCount También puede seguir realizando las divisiones y las fusiones como explicamos aquí.

Amazon Kinesis Data Streams permite efectuar cambios en las particiones, lo que le implica ajustar el número de particiones de la secuencia para adaptarse a los cambios en la velocidad del flujo de datos en la secuencia. Los cambios en las particiones se consideran operaciones avanzadas. Si es la primera vez que utiliza Kinesis Data Streams, vuelva a este tema después de familiarizarse con todos los demás aspectos de Kinesis Data Streams.

Existen dos tipos de operaciones de cambio de los fragmentos: división de fragmento y fusión de fragmentos. En una división, se divide un único fragmento en dos fragmentos. En una fusión, se combinan dos fragmentos en un único fragmento. Los cambios en los fragmentos siempre se realizan en pares, es decir, no se puede dividir en más de dos fragmentos en una sola operación y no se pueden fusionar más de dos fragmentos en una sola operación. El fragmento o par de fragmentos sobre los que actúa la operación de cambio se denominan fragmentos principales. El fragmento o par de fragmentos que surge como resultado de la operación de cambio se denominan fragmentos secundarios.

La división aumenta el número de fragmentos en la secuencia, y por tanto incrementa la capacidad de datos de la misma. Dado que se le cobrará por fragmento, la división aumenta el costo de su secuencia. Del mismo modo, la fusión reduce el número de fragmentos en la secuencia, y por tanto disminuye la capacidad de datos y el costo de la misma.

Los cambios en los fragmentos suele realizarlos una aplicación administrativa distinta de las aplicaciones productoras (put) y consumidoras (get). Una aplicación administrativa de este tipo supervisa el rendimiento general de la transmisión en función de las métricas proporcionadas por Amazon CloudWatch o en función de las métricas recopiladas de los productores y los consumidores. La aplicación administrativa también necesita un conjunto más amplio de permisos de IAM que las consumidoras o productoras, ya que estas normalmente no necesitan acceso a las API utilizadas para realizar los cambios en las particiones. Para más información sobre los permisos de IAM para Kinesis Data Streams, consulte Control del acceso a los recursos de Amazon Kinesis Data Streams mediante IAM.

Para más información sobre cambios en las particiones, consulte ¿Cómo puedo cambiar el número de particiones abiertas en Kinesis Data Streams?

Temas

Estrategias para cambios en las particiones

El objetivo de realizar cambios en las particiones en Amazon Kinesis Data Streams es que la secuencia pueda adaptarse a los cambios en la velocidad del flujo de datos. Puede dividir fragmentos para aumentar la capacidad (y los costos) de la secuencia. Puede fusionar fragmentos para reducir el costo (y la capacidad) de la secuencia.

Una forma de hacerlo sería dividir todas las particiones del flujo, lo que duplicaría su capacidad. Sin embargo, esto podría facilitar una capacidad adicional mayor que la que necesita realmente y, por tanto, crear costos innecesarios.

También puede utilizar métricas para determinar cuáles son sus particiones calientes o frías, es decir, particiones que reciben muchos más datos o muchos menos datos de lo esperado.

A continuación, puede dividir los fragmentos "calientes" de manera selectiva para aumentar la capacidad de las claves hash que se dirigen a esos fragmentos. Del mismo modo, podría fusionar fragmentos "fríos "para aprovechar mejor su capacidad.

Puede obtener algunos datos de rendimiento para su transmisión a partir de las CloudWatch métricas de Amazon que publica Kinesis Data Streams. Sin embargo, también puede recopilar sus propias métricas para las secuencias. Podría registrar los valores de las claves hash generadas por las claves de partición de sus registros de datos. Recuerde especificar la clave de partición en el momento en el que añada el registro a la secuencia.

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

Kinesis Data Streams utiliza MD5 para procesar la clave hash de la clave de partición. Dado que especifica la clave de partición para el registro, podría utilizar MD5 para calcular el valor de la clave hash de dicho registro y registrarla.

También puede registrar la ID de los fragmentos a los que están asignados los registros de datos. Puede consultar la ID de fragmento utilizando el método `getShardId` del objeto `putRecordResults` que devuelve el método `putRecords` y el objeto `putRecordResult` que devuelve el método `putRecord`.

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

Con los valores de ID de fragmento y clave hash, puede determinar qué fragmentos y claves hash reciben la mayor o menor cantidad de tráfico. A continuación, puede utilizar los cambios en los fragmentos para proporcionar más o menos capacidad, según proceda para estas claves.

División de un fragmento

Para dividir una partición en Amazon Kinesis Data Streams, debe especificar cómo deben redistribuirse los valores de clave hash de la partición principal a las particiones secundarias. Cuando añade un registro de datos a una secuencia, se asigna a un fragmento según un valor de clave hash. El valor de clave hash es el hash MD5 de la clave de partición que especifica para el registro de datos en el momento de añadirlo a la secuencia. Los registros de datos que tengan la misma clave de partición también tienen el mismo valor de clave hash.

Los valores de clave hash posibles para un fragmento dado constituyen un conjunto de valores enteros positivos, ordenados y contiguos. Esta serie de posibles valores de clave hash viene dada por:

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

Al dividir el fragmento especifica un valor de este rango. Ese valor de clave hash y todos los valores de clave hash superiores se distribuyen a uno de los fragmentos secundarios. Todos los valores de clave hash inferiores se distribuyen a los demás fragmentos secundarios.

El código siguiente ilustra una operación de división de fragmentos que redistribuye las claves hash de manera uniforme entre cada uno de los fragmentos secundarios, básicamente dividiendo a la mitad el fragmento principal. Esta es solo una de las posibles formas de dividir el fragmento principal. Podría, por ejemplo, dividir el fragmento de forma que el tercio más bajo de las claves del fragmento principal vaya a uno de los fragmentos secundarios y los dos tercios más altos de las claves vayan al otro fragmento secundario. Sin embargo, para muchas aplicaciones, la división de fragmentos a la mitad resulta un enfoque eficaz.

El código asume que `myStreamName` contiene el nombre de la secuencia y la variable de objeto `shard` contiene el fragmento que se ha de dividir. Empiece por crear una instancia de un nuevo objeto `splitShardRequest` y por introducir un nombre para la secuencia y una ID para el fragmento.

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

Determine el valor de clave hash que está a medio camino entre los valores mínimo y máximo del fragmento. Este será el valor de clave hash inicial para el fragmento secundario que contendrá la mitad superior de las claves hash del fragmento principal. Especifique este valor en el método `setNewStartingHashKey`. Solo necesita especificar este valor. Kinesis Data Streams distribuye automáticamente las claves hash por debajo de este valor a la otra partición primaria creada por la división. El último paso consiste en llamar al método `splitShard` en el cliente de Kinesis Data Streams.

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

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

El primer paso tras este procedimiento se muestra en Esperar a que una secuencia se active de nuevo.

Fusión de dos fragmentos

Una operación de fusión toma dos fragmentos especificados y los combina en un único fragmento. Tras la fusión, el fragmento secundario recibe los datos de todos los valores de claves hash que cubrían los dos fragmentos principales.

Adyacencia de fragmentos

Para fusionar dos fragmentos, estos deben ser adyacentes. Se considera que dos fragmentos son adyacentes si la unión de los rangos de las claves hash de ambos forma un conjunto continuo sin huecos. Por ejemplo, supongamos que tiene dos fragmentos, uno con un rango de clave hash de 276...381 y el otro con un rango de clave hash de 382...454. Podría fusionar estos dos fragmentos en uno, que tendría un rango de clave hash de 276...454.

Por tomar otro ejemplo, supongamos que tiene dos fragmentos, uno con un rango de clave hash de 276...381 y el otro con un rango de clave hash de 455...560. No es posible combinar estos dos fragmentos, ya que habría uno o varios fragmentos entre ellos abarcando el rango 382...454.

El conjunto de las particiones `OPEN` de un flujo, como grupo, siempre abarca el rango completo de valores de clave hash de MD5. Para obtener más información sobre los estados de los fragmentos, como `CLOSED`, consulte Direccionamiento de datos, persistencia de datos y estado de fragmentos tras los cambios en los fragmentos.

Para identificar fragmentos candidatos a la fusión, debe filtrar todos los que están en un estado `CLOSED`. Las particiones que están `OPEN` (es decir, no `CLOSED`) tienen un número secuencial final de `null`. Puede comprobar el número secuencial final de un fragmento con:

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

Después de filtrar los fragmentos cerrados, ordene los fragmentos restantes por el valor de clave hash más alto que permita cada fragmento. Puede recuperar este valor mediante:

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

Si hay dos fragmentos adyacentes en la lista filtrada y ordenada, se pueden fusionar.

Código para la operación de fusión

Con el siguiente código se fusionan dos fragmentos. El código asume que `myStreamName` contiene el nombre de la secuencia y las variables de objeto `shard1` y `shard2` contienen los dos fragmentos adyacentes que se han de fusionar.

Para la operación de fusión, comience por crear una instancia de un nuevo objeto `mergeShardsRequest`. Especifique el nombre de la secuencia con el método `setStreamName`. A continuación, especifique qué dos fragmentos quiere fusionar con los métodos `setShardToMerge` y `setAdjacentShardToMerge`. Por último, llame al método `mergeShards` en el cliente de Kinesis Data Streams para realizar la operación.

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

El primer paso tras este procedimiento se muestra en Esperar a que una secuencia se active de nuevo.

Después de realizar cambios en los fragmentos

Después de realizar cualquier tipo de procedimiento que implique cambios en las particiones en Amazon Kinesis Data Streams y antes de reanudar el procesamiento de registros normal, son necesarios otros procedimientos y consideraciones. Estos pasos se describen en las siguientes secciones.

Temas

- Esperar a que una secuencia se active de nuevo
- Direccionamiento de datos, persistencia de datos y estado de fragmentos tras los cambios en los fragmentos

Esperar a que una secuencia se active de nuevo

Tras realizar una operación de cambio de fragmentos, ya sea `splitShard` o `mergeShards`, tendrá que esperar a que la secuencia vuelva a estar activa. El código que ha de usar es el mismo que al esperar a que una secuencia se active tras la creación de una secuencia. Ese código es el siguiente:

```
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" );
}
```

Direccionamiento de datos, persistencia de datos y estado de fragmentos tras los cambios en los fragmentos

Kinesis Data Streams es un servicio de streaming de datos en tiempo real, es decir, que las aplicaciones deberían presuponer que los datos fluyen de forma continua a través de las particiones del flujo. Cuando realiza cambios en los fragmentos, los registros de datos dirigidos a los fragmentos principales se redireccionan a los fragmentos secundarios en función de los valores de clave hash que asignan las claves de partición de los registros de datos. Sin embargo, los registros de datos que estaban en los fragmentos principales antes de los cambios permanecerán en dichos fragmentos. En otras palabras, los fragmentos principales no desaparecen cuando se producen los cambios. Persistirán junto con los datos que contenían antes de los cambios. Se puede acceder a los registros de datos de las particiones principales mediante las operaciones `getShardIterator` y `getRecords` de la API de Kinesis Data Streams, o a través de Kinesis Client Library.

> **Note**
>
> Se puede obtener acceso a los registros de datos a partir del momento en el que se agregan a la secuencia y hasta el periodo de retención actual. Esto es así independientemente de los cambios en los fragmentos de la secuencia durante ese periodo. Para obtener más información acerca del periodo de retención de una secuencia, consulte Cambiar el periodo de retención de datos.

En el proceso de realizar cambios en los fragmentos, un fragmento principal cambia de un estado `OPEN` a un estado `CLOSED`, y después a un estado `EXPIRED`.

- OPEN: antes de una operación de cambio en los fragmentos, un fragmento principal tiene el estado `OPEN`, lo que significa que los registros de datos pueden tanto agregarse al fragmento como recuperarse desde él.
- CLOSED: tras una operación de cambio en los fragmentos, el fragmento principal cambia a un estado `CLOSED`. Esto implica que ya no se agregan registros de datos al fragmento. Los registros de datos que se habrían añadido a este fragmento se agregarán a un fragmento secundario en su lugar. Sin embargo, se pueden seguir recuperando registros de datos desde el fragmento durante un tiempo limitado.
- EXPIRED: tras vencer el periodo de retención de la secuencia, todos los registros de datos del fragmento principal habrán caducado y ya no se podrá obtener acceso a ellos. En este punto, el propio fragmento cambia a un estado `EXPIRED`. Las llamadas a

`getStreamDescription().getShards` para enumerar los fragmentos de la secuencia no incluyen los fragmentos `EXPIRED` en la lista de fragmentos que devuelve. Para obtener más información acerca del periodo de retención de una secuencia, consulte Cambiar el periodo de retención de datos.

Tras realizar los cambios en los fragmentos, y cuando la secuencia ha recuperado el estado `ACTIVE`, podría comenzar inmediatamente a leer datos en los fragmentos secundarios. Sin embargo, los fragmentos principales que permanecen tras realizar los cambios podrían seguir conteniendo datos que aún no se han leído y que se habrían agregado a la secuencia antes de los cambios. Si lee datos de los fragmentos secundarios antes de haber leído todos los datos de los fragmentos principales, podría leer datos de una clave hash determinada fuera del orden determinado por los números secuenciales de los registros de datos. Por lo tanto, suponiendo que el orden de los datos sea importante, después de un cambio en los fragmentos siempre se deben seguir leyendo datos de los fragmentos principales hasta que se agoten. Solo entonces debería empezar a leer datos de los fragmentos secundarios. Cuando `getRecordsResult.getNextShardIterator` devuelve `null`, indica que ya ha leído todos los datos del fragmento principal. Si lee datos mediante Kinesis Client Library, la biblioteca garantiza que reciba los datos en orden, incluso si se produce una redistribución.

Cambiar el periodo de retención de datos

Amazon Kinesis Data Streams admite cambios en el periodo de retención de los registros de datos del flujo de datos. Un flujo de datos de Kinesis es una secuencia ordenada de registros de datos pensada para que se pueda escribir y leer en ella en tiempo real. Por lo tanto, los registros de datos se almacenan temporalmente en fragmentos de su secuencia. El periodo de tiempo desde que se agrega un registro hasta que ya no se puede obtener acceso a él se denomina periodo de retención. Un flujo de datos de Kinesis almacena registros desde 24 horas de forma predeterminada, hasta 8760 horas (365 días).

Puede actualizar el período de retención mediante la consola de Kinesis Data Streams o mediante IncreaseStreamRetentionPeriodlas operaciones y DecreaseStreamRetentionPeriod. Con la consola de Kinesis Data Streams, puede editar en bloque el periodo de retención de más de un flujo de datos al mismo tiempo. Puede aumentar el período de retención hasta un máximo de 8760 horas (365 días) mediante la IncreaseStreamRetentionPeriodoperación o la consola de Kinesis Data Streams. Puede reducir el período de retención hasta un mínimo de 24 horas mediante la DecreaseStreamRetentionPeriodoperación o la consola de Kinesis Data Streams. La sintaxis de la solicitud para ambas operaciones incluye el nombre de la secuencia y el periodo de retención en

horas. Por último, puede comprobar el periodo de retención actual de una secuencia si llama a la operación DescribeStream.

A continuación, se muestra un ejemplo de cambio del periodo de retención usando la AWS CLI:

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

Kinesis Data Streams deja de producir registros inaccesibles en el antiguo período de retención en unos minutos tras aumentar el periodo de retención. Por ejemplo, cambiar el periodo de retención de 24 horas a 48 horas implica que los registros añadidos a la secuencia 23 horas y 55 minutos antes seguirán estando disponibles después de que hayan transcurrido 24 horas.

Kinesis Data Streams hace que los registros más antiguos que el nuevo periodo de retención sean inaccesibles casi inmediatamente tras reducir el periodo de retención. Por tanto, debe tener mucho cuidado al llamar a la operación DecreaseStreamRetentionPeriod.

Establezca el periodo de retención de datos para garantizar que sus consumidores puedan leer los datos antes de que estos venzan, por si surge algún problema. Debe tener en cuenta detenidamente todas las posibilidades, como, por ejemplo, un problema con la lógica de procesamiento de registros o que una dependencia posterior esté inactiva durante un periodo prolongado de tiempo. Piense en el periodo de retención como si fuera una red de seguridad que proporciona un mayor tiempo de recuperación para los consumidores de datos. Las operaciones de la API para el periodo de retención le permiten realizar esta configuración de forma proactiva o responder a eventos operativos de forma reactiva.

Para las secuencias con periodos de retención mayores a 24 horas se aplican cargos adicionales. Para obtener más información, consulte los precios de Amazon Kinesis Data Streams.

Etiquetado de flujos de Amazon Kinesis Data Streams

Puede asignar sus propios metadatos a los flujos que cree en Amazon Kinesis Data Streams en forma de etiquetas. Una etiqueta es un par clave-valor definido por el usuario para un flujo. El uso de las etiquetas es una forma sencilla y potente para administrar los recursos de AWS y organizar los datos, incluidos los datos de facturación.

Contenido

- Restricciones de las etiquetas
- Etiquetado de flujos mediante la consola de Kinesis Data Streams
- Etiquetado de secuencias con la AWS CLI
- Etiquetado de flujos mediante la API de Kinesis Data Streams

Conceptos básicos de etiquetas

Utilice la consola de Kinesis Data Streams, la AWS CLI o la API de Kinesis Data Streams para realizar las siguientes tareas:

- Agregar etiquetas a una secuencia
- Listar las etiquetas para sus secuencias
- Eliminar etiquetas de una secuencia

Puede utilizar las etiquetas para categorizar sus secuencias. Por ejemplo, puede clasificar las secuencias en categorías por objetivo, propietario o entorno. Dado que define la clave y el valor de cada etiqueta, puede crear un conjunto de categorías personalizadas para satisfacer sus necesidades específicas. Por ejemplo, podría definir un conjunto de etiquetas que le ayude a realizar un seguimiento de las secuencias por propietario y aplicaciones asociadas. Estos son algunos ejemplos de etiquetas:

- Proyecto: nombre del proyecto
- Propietario: nombre
- Objetivo: pruebas de carga
- Aplicación: nombre de aplicación
- Entorno: producción

Seguimiento de costos utilizando el etiquetado

Puede utilizar etiquetas para categorizar y hacer un seguimiento de los costos de AWS. Cuando se aplican etiquetas a los recursos de AWS, incluidos los flujos, el informe de asignación de costos de AWS incluye el uso y los costos agregados por etiquetas. Puede aplicar etiquetas que representen categorías de negocio (por ejemplo, centros de costos, nombres de aplicación o propietarios) para organizar los costos entre diferentes servicios. Para obtener más información, consulte Utilizar

etiquetas de asignación de costos para informes de facturación personalizados en la Guía del usuario de AWS Billing.

Restricciones de las etiquetas

Se aplican las siguientes restricciones a las etiquetas.

Restricciones básicas

- El número máximo de etiquetas por recurso (flujo) es 50.
- Las claves y los valores de las etiquetas distinguen entre mayúsculas y minúsculas.
- No se pueden cambiar ni editar etiquetas de un flujo eliminado.

Restricciones de clave de etiqueta

- Cada clave de etiqueta debe ser única. Si agrega una etiqueta con una clave que ya está en uso, la nueva etiqueta sobrescribe el par clave-valor existente.
- Una clave de etiqueta no puede comenzar por `aws:` porque este prefijo está reservado para su utilización por AWS. AWS crea etiquetas cuyo nombre comienza por este prefijo por usted, pero usted no puede editarlas ni eliminarlas.
- Las claves de etiqueta deben tener entre 1 y 128 caracteres Unicode de longitud.
- Las claves de etiquetas deben constar de los siguientes caracteres: letras Unicode, números, espacios en blanco y los siguientes caracteres especiales: `_ . / = + - @`.

Restricciones de valor de etiqueta

- Los valores de etiqueta deben tener entre 0 y 255 caracteres Unicode de longitud.
- Los valores de etiqueta pueden estar en blanco. De lo contrario, deben constar de los siguientes caracteres: letras Unicode, números, espacios en blanco y cualquiera de los siguientes caracteres especiales: `_ . / = + - @`.

Etiquetado de flujos mediante la consola de Kinesis Data Streams

Puede agregar, enumerar y eliminar etiquetas con la consola de Kinesis Data Streams.

Para ver las etiquetas de una secuencia

1. Abra la consola de Kinesis Data Streams. En la barra de navegación, expanda el selector de regiones y seleccione una región.
2. En la página Stream-List (Lista de secuencias), seleccione una secuencia.
3. En la página Stream Details (Detalles de la secuencia) haga clic en la pestaña Tags (Etiquetas).

Para agregar una etiqueta a una secuencia

1. Abra la consola de Kinesis Data Streams. En la barra de navegación, expanda el selector de regiones y seleccione una región.
2. En la página Stream-List (Lista de secuencias), seleccione una secuencia.
3. En la página Stream Details (Detalles de la secuencia) haga clic en la pestaña Tags (Etiquetas).
4. Especifique la clave de etiqueta en el campo Key (Clave), escriba un valor de etiqueta en el campo Value (Valor) (opcional) y haga clic en Add Tag (Añadir etiqueta).

   Si el botón Add Tag (Añadir etiqueta) no está habilitado, la clave o el valor de etiqueta que ha especificado no cumplen las restricciones para etiquetas. Para obtener más información, consulte Restricciones de las etiquetas.
5. Para ver la nueva etiqueta en la pestaña Tags (Etiquetas), haga clic en el icono de actualización.

Para eliminar una etiqueta de una secuencia

1. Abra la consola de Kinesis Data Streams. En la barra de navegación, expanda el selector de regiones y seleccione una región.
2. En la página Stream-List, seleccione una secuencia.
3. En la página Stream Details (Detalles de la secuencia), haga clic en la pestaña Tags (Etiquetas) y, a continuación, haga clic en el icono Remove (Quitar) de la etiqueta.
4. En el cuadro de diálogo Delete Tag (Eliminar etiqueta), haga clic en Yes, Delete (Sí, eliminar).

Etiquetado de secuencias con la AWS CLI

Puede agregar, enumerar y eliminar etiquetas mediante la AWS CLI. Para ver ejemplos, consulte la documentación siguiente.

add-tags-to-stream

Añade o actualiza las etiquetas para la secuencia especificada.

list-tags-for-stream

Enumera las etiquetas de la secuencia especificada.

remove-tags-from-stream

Elimina las etiquetas de la secuencia especificada.

Etiquetado de flujos mediante la API de Kinesis Data Streams

Puede agregar, enumerar y eliminar etiquetas con la API de Kinesis Data Streams. Para ver ejemplos, consulte la documentación siguiente:

AddTagsToStream

Añade o actualiza las etiquetas para la secuencia especificada.

ListTagsForStream

Enumera las etiquetas de la secuencia especificada.

RemoveTagsFromStream

Elimina las etiquetas de la secuencia especificada.

Escritura de datos en Amazon Kinesis Data Streams

Un productor es una aplicación que escribe datos en Amazon Kinesis Data Streams. Puede crear productores para Kinesis Data Streams mediante AWS SDK for Java y Kinesis Producer Library.

Si es la primera vez que utiliza Kinesis Data Streams, familiarícese antes con los conceptos y los términos que encontrará en ¿Qué es Amazon Kinesis Data Streams? y Introducción a Amazon Kinesis Data Streams.

> **Important**
>
> Kinesis Data Streams admite cambios en el periodo de retención de los registros de datos del flujo de datos. Para obtener más información, consulte Cambiar el periodo de retención de datos.

Para insertar datos en la secuencia, debe especificar el nombre de la secuencia, una clave de fragmento y el blob de datos que se han de agregar a la secuencia. La clave de partición se utiliza para determinar en qué fragmento de la secuencia se agregará el registro de datos.

Todos los datos del fragmento se envían al mismo proceso de trabajo que procesa el fragmento. La clave de partición que utilice dependerá de la lógica de su aplicación. El número de claves de partición normalmente debe ser mucho mayor que el número de fragmentos. Esto se debe a que la clave de partición se utiliza para determinar cómo asignar un registro de datos a un fragmento determinado. Si tiene suficientes claves de partición, los datos pueden distribuirse con uniformidad entre todos los fragmentos de una secuencia.

Contenido

- Temas avanzados para productores de Kinesis Data Streams

Desarrollo de productores con Amazon Kinesis Producer Library

Un productor de Amazon Kinesis Data Streams es una aplicación que coloca los registros de datos de usuario en un flujo de datos de Kinesis Data Streams (también denominado ingesta de datos). Kinesis Producer Library (KPL) simplifica el desarrollo de aplicaciones productoras y permite a los desarrolladores alcanzar un mayor rendimiento de escritura en un flujo de datos de Kinesis Data Streams.

Puedes monitorizar la KPL con Amazon CloudWatch. Para obtener más información, consulte Supervisión de Kinesis Producer Library con Amazon CloudWatch.

Contenido

> **Note**
>
> Se recomienda actualizar a la versión más reciente de KPL. KPL se actualiza periódicamente con versiones más recientes que incluyen los últimos parches de dependencia y seguridad,

correcciones de errores y nuevas características compatibles con versiones anteriores. Para obtener más información, consulte https://github.com/awslabs/ amazon-kinesis-producer / releases/.

Función de KPL

La KPL es una easy-to-use biblioteca altamente configurable que le ayuda a escribir en una transmisión de datos de Kinesis. Ejerce de intermediaria entre el código de la aplicación de productor y las acciones de la API de Kinesis Data Streams. KPL realiza las siguientes tareas principales:

- Escribe en uno o varios flujo de datos de Kinesis con un mecanismo de reintento automático y configurable
- Recopila registros y utiliza `PutRecords` para escribir varios registros en varios fragmentos por solicitud
- Acumula registros de usuario para aumentar el tamaño de la carga y mejorar el rendimiento
- Se integra perfectamente con Kinesis Client Library (KCL) para desagrupar registros en lotes en el consumidor
- Envía las CloudWatch estadísticas de Amazon en tu nombre para proporcionar visibilidad sobre el rendimiento del productor

Tenga en cuenta que KPL es diferente de la API de Kinesis Data Streams que está disponible en los SDK deAWS. La API de Kinesis Data Streams ayuda a administrar muchos aspectos de Kinesis Data Streams (como la creación de flujos, el cambio de las particiones de registros, así como la inserción y obtención de estos), mientras que KPL proporciona una capa de abstracción específica para la ingesta de datos. Para obtener información sobre la API de Kinesis Data Streams, consulte Referencia de la API de Amazon Kinesis.

Ventajas del uso de KPL

La siguiente lista representa algunas de las ventajas más importantes de utilizar KPL para el desarrollo de productores de Kinesis Data Streams.

KPL puede utilizarse en casos de uso síncronos o asíncronos. Se recomienda utilizar el mayor rendimiento de la interfaz asíncrona a menos que haya una razón específica para utilizar un comportamiento síncrono. Para obtener más información sobre estos dos casos de uso y ver código de muestra, consulte Escritura en los flujos de datos de Kinesis mediante KPL.

Beneficios de rendimiento

KPL puede ayudar a crear productores de alto rendimiento. Piense en una situación en la que las instancias de Amazon EC2 sirven como un proxy para recopilar eventos de 100 bytes de cientos o miles de dispositivos de baja potencia y para escribir registros en un flujo de datos de Kinesis. Cada una de estas instancias EC2 debe escribir miles de eventos por segundo en la secuencia de datos. Para conseguir el nivel de rendimiento necesario, los productores deben implementar lógica complicada como, por ejemplo, la creación de lotes o la ejecución de subprocesos, además de lógica de reintento y desagrupación de registros en el lado del consumidor. KPL realiza todas estas tareas.

Facilidad de uso en el lado del consumidor

En el caso de los desarrolladores del lado del consumidor que usan KCL en Java, KPL se integra sin mayor esfuerzo. Cuando KCL recupera un registro agregado de Kinesis Data Streams que consta de varios registros de usuario de KPL, llama automáticamente a KPL para que extraiga los registro de usuario individuales antes de devolverlos al usuario.

Para los desarrolladores del lado del consumidor que no utilicen KCL, sino que, en su lugar, utilicen la operación de la API `GetRecords` directamente, hay disponible una biblioteca de Java de KPL que pueden usar para extraer los registros de los usuarios individuales antes de devolverlos al usuario.

Monitoreo del productor

Puede recopilar, supervisar y analizar a sus productores de Kinesis Data Streams mediante CloudWatch Amazon y la KPL. El KPL emite el rendimiento, los errores y otras métricas CloudWatch en su nombre, y se puede configurar para supervisarlo a nivel de transmisión, fragmento o productor.

Arquitectura asíncrona

Debido a que KPL puede almacenar en búfer los registros antes de enviarlos a Kinesis Data Streams, en la aplicación intermediaria no se fuerza el bloqueo y la espera a la confirmación de que el registro ha llegado al servidor antes de continuar con la ejecución. Una llamada para insertar un registro en KPL siempre vuelve inmediatamente, y no se espera a que el registro se envíe o a que se reciba una respuesta desde el servidor. En su lugar, se crea un objeto `Future` que recibe el resultado de enviar el registro a Kinesis Data Streams en un momento posterior. Este comportamiento es el mismo que el de los clientes asíncronos del SDK. AWS

Cuándo no utilizar KPL

KPL puede ocasionar un retraso adicional en el procesamiento de hasta `RecordMaxBufferedTime` en la biblioteca (configurable por el usuario). Los valores más altos de `RecordMaxBufferedTime` ofrecen un mayor eficacia de empaquetamiento y un mejor rendimiento. Es posible que las aplicaciones que no pueden tolerar este retraso tengan que utilizar el SDK de AWS directamente. Para obtener más información sobre el uso del AWS SDK con Kinesis Data Streams, Desarrollo de productores mediante la API de Amazon Kinesis Data Streams con AWS SDK for Java consulte. Para más información sobre `RecordMaxBufferedTime` y otras propiedades configurables por el usuario de KPL, consulte Configuración de Kinesis Producer Library.

Instalación de KPL

Amazon proporciona archivos binarios precompilados de Kinesis Producer Library (KPL) en C++ para macOS, Windows y distribuciones recientes de Linux (para información detallada sobre las plataformas compatibles, consulte la siguiente sección). Estos archivos binarios se empaquetan como parte de archivos .jar de Java y se invocan y utilizan automáticamente si se usa Maven para instalar el paquete. Para encontrar las versiones más recientes de KPL y KCL, utilice los siguientes enlaces de búsqueda de Maven:

- KPL
- KCL

Los archivos binarios de Linux han sido compilados con la recopilación del compilador GNU (GCC) y están vinculados estáticamente con libstdc++ en Linux. Lo esperable es que funcionen en cualquier distribución de Linux de 64 bits que incluya una versión de glibc 2.5 o superior.

Los usuarios de distribuciones Linux antiguas pueden compilar el KPL siguiendo las instrucciones de compilación incluidas en el código fuente. GitHub Para descargar el KPL desde GitHub, consulte la biblioteca de Kinesis Producer.

Transición a los certificados Amazon Trust Services (ATS) para Kinesis Producer Library

El 9 de febrero de 2018, a las 9:00 h (PST), Amazon Kinesis Data Streams instaló certificados ATS. Si desea continuar escribiendo registros en Kinesis Data Streams con Kinesis Producer Library (KPL) debe actualizar la instalación de KPL a la versión 0.12.6 o posterior. Este cambio afecta a todas las AWS regiones.

Para obtener información sobre el cambio a ATS, consulte Cómo prepararse para el AWStraslado a su propia autoridad de certificación.

Si tiene problemas y necesita soporte técnico, cree un caso en el Centro de AWS Support.

Plataformas compatibles con KPL

Kinesis Producer Library (KPL) está escrito en C++ y se ejecuta como un proceso secundario del proceso principal de usuario. Los binarios nativos precompilados de 64 bits se empaquetan con la versión de Java y vuelven a ser administrados por la clase contenedora Java.

El paquete de Java se ejecuta sin necesidad de instalar otras bibliotecas en los siguientes sistemas operativos:

- Distribuciones de Linux con kernel 2.6.18 (septiembre de 2006) y posteriores
- Apple OS X 10.9 y versiones posteriores
- Windows Server 2008 y posteriores

> **Important**
>
> Windows Server 2008 y las versiones posteriores son compatibles con todas las versiones de KPL hasta la versión 0.14.0.
> La plataforma Windows NO es compatible a partir de la versión 0.14.0 o superior de KPL.

Tenga en cuenta que KPL solo es compatible con sistemas de 64 bits.

Código fuente

Si los archivos binarios proporcionados en la instalación de KPL no son suficientes para el entorno, el núcleo de KPL se escribirá como un módulo C++. El código fuente del módulo C++ y la interfaz Java se publican bajo la licencia pública de Amazon y están disponibles GitHub en la biblioteca de productores de Kinesis. Aunque KPL se puede utilizar en cualquier plataforma en la que estén disponibles un compilador C++ y un JRE que cumplan los estándares recientes, Amazon no admite oficialmente ninguna plataforma que no figure en la lista de plataformas compatibles.

Conceptos clave de KPL

Las siguientes secciones contienen conceptos y términos necesarios para comprender y utilizar Kinesis Producer Library (KPL).

Temas

Registros

En esta guía, distinguimos entre registros de usuario de KPL y registros de Kinesis Data Streams. Cuando utilizamos el término registro sin ningún calificador, nos referimos a un registro de usuario de KPL. Cuando nos referimos a un registro de Kinesis Data Streams, decimos explícitamente registro de Kinesis Data Streams.

Un registro de usuario de KPL es un blob de datos que tiene un significado determinado para el usuario. Entre los ejemplos se incluyen un blob JSON que representa un evento de la interfaz de usuario en un sitio web, o una entrada de registro de un servidor web.

Un registro de Kinesis Data Streams es una instancia de la estructura de datos de `Record` definida por la API del servicio Kinesis Data Streams. Contiene una clave de partición, un número secuencial y un blob de datos.

Agrupación en lotes

La agrupación en lotes se refiere a la realización de una sola acción en varios elementos en lugar de realizar la acción repetidamente en cada elemento individual.

En este contexto, el “elemento” es un registro, y la acción es enviarlo a Kinesis Data Streams. En una situación sin agrupación en lotes, cada registro se insertaría en un registro de Kinesis Data Streams independiente y se hará una solicitud HTTP para enviarlo a Kinesis Data Streams. Al agrupar en lotes, cada solicitud HTTP puede tener varios registros en lugar de solo uno.

KPL admite dos tipos de agrupación en lotes:

- Agregación: almacenamiento de varios registros en un único registro de Kinesis Data Streams.
- Recopilación: uso de la operación de la API `PutRecords` para enviar varios registros de Kinesis Data Streams a una o varias particiones del flujo de datos de Kinesis.

Los dos tipos de agrupación en lote de KPL se diseñaron para coexistir y se pueden activar o desactivar de forma independiente entre sí. Ambos están activados de forma predeterminada.

Agregación

La agregación se refiere al almacenamiento de varios registros en un registro de Kinesis Data Streams. La agregación permite a los clientes aumentar el número de registros enviados por cada llamada a la API, lo que aumenta el rendimiento del productor de manera eficaz.

Las particiones de Kinesis Data Streams admiten hasta 1000 registros de Kinesis Data Streams por segundo o un rendimiento de 1 MB. El límite de registros de Kinesis Data Streams por segundo fuerza a los consumidores a usar registros de tamaño inferior a 1 KB. La agregación de registros en lote permite a los clientes combinar varios registros en un único registro de Kinesis Data Streams. Esto permite a los clientes mejorar su rendimiento por fragmento.

Piense en el caso de un fragmento en la región us-east-1 que se está ejecutando a una velocidad constante de 1000 registros por segundo, con registros de 512 bytes cada uno. Con la agregación de KPL puede empaquetar 1000 registros en solo 10 registros de Kinesis Data Streams, de manera que el número de registros por segundo se reduce a 10 (con 50 KB cada uno).

Recopilación

La recopilación se refiere a agrupar en lote varios registros de Kinesis Data Streams y enviarlos en una única solicitud HTTP con una llamada a la operación de la API `PutRecords`, en lugar de enviar cada registro de Kinesis Data Streams en su propia solicitud HTTP.

Esto aumenta el rendimiento, si lo comparamos con una situación en la que no se use recopilación, ya que reduce la sobrecarga que implica realizar muchas solicitudes de HTTP independientes. De hecho, `PutRecords` en sí se ha diseñado específicamente para este fin.

La recopilación es diferente de la agregación, ya que funciona con grupos de registros de Kinesis Data Streams. Los registros de Kinesis Data Streams que se recopilan pueden contener varios registros del usuario. La relación se puede visualizar así:

```
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--|
```

Integración de KPL con el código de productor

Kinesis Producer Library (KPL) se ejecuta en un proceso independiente y se comunica con el proceso principal de usuario mediante IPC. Esta arquitectura se denomina a veces microservicio, y se elige por dos razones principales:

1) El proceso de usuario no se bloqueará aunque KPL se bloquee

El proceso podría tener tareas no relacionadas con Kinesis Data Streams y así podría continuar la operación incluso si KPL se bloquea. También es posible que el proceso principal de usuario reinicie KPL y se recupere hasta alcanzar un estado totalmente funcional (esta funcionalidad se encuentra en los contenedores oficiales).

Un ejemplo es un servidor web que envía métricas a Kinesis Data Streams. El servidor puede continuar sirviendo páginas incluso si la parte de Kinesis Data Streams dejó de funcionar. El bloqueo de todo el servidor como consecuencia de un error en KPL provocaría una interrupción innecesaria del servicio.

2) Compatibilidad con clientes arbitrarios

Siempre hay consumidores que usan lenguajes diferentes a los que son oficialmente compatibles. Estos usuarios también deberían poder utilizar KPL fácilmente.

Matriz de uso recomendada

La siguiente matriz de uso enumera la configuración recomendada para diferentes usuarios y le recomienda si utilizar KPL y cómo hacerlo. Tenga en cuenta que si se habilita la agregación, también debe utilizar desagrupación para extraer sus registros en el lado del consumidor.

Lenguaje del lado del productor	Lenguaje del lado del consumidor	Versión de KCL	Lógica del punto de comprobación	¿Puede utilizar KPL?	Advertencias
Cualquiera excepto Java	*	*	*	No	N/A
Java	Java	Utiliza el SDK de Java directamente	N/A	Sí	Si se utiliza agregación, deberá usar la biblioteca de desagrupación facilitada tras la llamada de `GetRecords` .
Java	Cualquiera excepto Java	Utiliza SDK directamente	N/A	Sí	Debe deshabilitar la agregación.
Java	Java	1.3.x	N/A	Sí	Debe deshabilitar la agregación.
Java	Java	1.4.x	Llama al punto de comprobación sin argumentos	Sí	Ninguna

Lenguaje del lado del productor	Lenguaje del lado del consumidor	Versión de KCL	Lógica del punto de comprobación	¿Puede utilizar KPL?	Advertencias
Java	Java	1.4.x	Llama al punto de comprobación con un número secuencial explícito	Sí	Desactive la agregación o cambie el código para usar números secuenciales ampliados para la creación de puntos de control.
Java	Cualquiera excepto Java	1.3.x + daemon multilenguaje + contenedor específico del lenguaje	N/A	Sí	Debe deshabilitar la agregación.

Escritura en los flujos de datos de Kinesis mediante KPL

En las secciones siguientes, se muestra un código de muestra con una progresión que va desde el productor más básico y sencillo posible hasta un código completamente asíncrono.

Código de productor barebones

El siguiente código es todo lo que necesita para escribir un productor mínimamente funcional. Los registros de usuario de Kinesis Producer Library (KPL) se procesan en segundo plano.

```
// 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 ...
```

Respuesta síncrona a los resultados

En el ejemplo anterior, el código no comprobó si el procesamiento de registros de usuario de KPL finalizó correctamente. KPL efectúa los reintentos necesarios en caso de error. Sin embargo, si desea comprobar los resultados, puede examinarlos con los objetos `Future` que devuelve `addUserRecord`, como en el siguiente ejemplo (el ejemplo anterior se incluye para aportar contexto):

```
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
        }
    }
}
```

Respuesta asíncrona a los resultados

El ejemplo anterior llama a `get()` en un objeto `Future` que bloquea la ejecución. Si no desea bloquear la ejecución, puede utilizar una devolución de llamada asíncrona, tal y como se muestra en el ejemplo siguiente:

```
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);
}
```

Configuración de Kinesis Producer Library

Aunque la configuración predeterminada debería funcionar sin problemas para la mayoría de los casos de uso, es posible que desee cambiar algunos de los ajustes predeterminados para adaptar el comportamiento de `KinesisProducer` a sus necesidades. Para hacerlo, se puede pasar una instancia de la clase `KinesisProducerConfiguration` al constructor `KinesisProducer`, por ejemplo:

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

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

También puede cargar una configuración desde un archivo de propiedades:

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

Puede sustituir la ruta y el nombre de archivo a los que tiene acceso el proceso del usuario. También puede llamar a métodos fijos en la instancia `KinesisProducerConfiguration` que hayan sido creados así para personalizar la configuración.

El archivo de propiedades debe especificar los parámetros con sus nombres en PascalCase. Los nombres han de coincidir con los utilizados en los métodos fijos de la clase `KinesisProducerConfiguration`. Por ejemplo:

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

Para obtener más información sobre las reglas de uso de los parámetros de configuración y los límites de valores, consulte el ejemplo del archivo de propiedades de configuración en GitHub.

Tenga en cuenta que después de inicializar `KinesisProducer`, los cambios en la instancia `KinesisProducerConfiguration` no tienen efecto alguno. Actualmente, `KinesisProducer` no admite la reconfiguración dinámica.

Desagrupación del consumidor

A partir de la versión 1.4.0, KCL es compatible con la desagrupación automática de registros de usuario de KPL. El código de la aplicación de consumidor escrito con versiones anteriores de KCL se compilará sin ningún tipo de modificación después de actualizar KCL. Sin embargo, si la agregación de KPL se utiliza en el lado del productor, existe un matiz relacionado con la creación de puntos de control: todos los subregistros de un registro agregado tienen el mismo número de secuencia, de modo que los datos adicionales deben almacenarse con el punto de control si es necesario hacer una distinción entre subregistros. Estos datos adicionales se denominan números subsecuenciales.

Migrar desde versiones anteriores de KCL

No es necesario cambiar las llamadas existentes para crear puntos de control en conjunto con la agregación. La recuperación de todos los registros almacenados correctamente en Kinesis Data Streams seguirá estando garantizada. Ahora, KCL ofrece dos nuevas operaciones de puntos de control para casos de uso particulares que se describen a continuación.

En caso de que el código existente se hubiera escrito para KCL antes de que fuera compatible con KPL y se haga una llamada a la operación del punto de control sin argumentos, esto equivaldría a la creación de puntos de control para el número de secuencia del último registro de usuario de KPL en el lote. Si la operación de punto de comprobación se llama con una cadena de número secuencial, equivaldrá a la creación de un punto de comprobación para el número secuencial determinado del lote junto con el número subsecuencial implícito 0 (cero).

La llamada a la nueva operación de punto de control de KCL `checkpoint()` sin argumentos equivale semánticamente a hacer un punto de control del número de secuencia de la última llamada de `Record` en el lote, junto con el número subsecuencial implícito 0 (cero).

La llamada a la nueva operación de punto de control de KCL `checkpoint(Record record)` equivale semánticamente a hacer un punto de control del número secuencial del `Record` dado, junto con el número subsecuencial implícito 0 (cero). Si la llamada al `Record` es realmente un `UserRecord`, el número secuencial de `UserRecord` y el número subsecuencial se someten a un punto de comprobación.

La llamada a la nueva operación de punto de control de KCL `checkpoint(String sequenceNumber, long subSequenceNumber)` crea explícitamente un punto de control del número secuencial dado junto con el número subsecuencial dado.

En cualquiera de estos casos, después de que el punto de control se almacene en la tabla de Amazon DynamoDB, KCL puede reanudar correctamente la recuperación de los registros, incluso cuando la aplicación se bloquee y se reinicie. Si la secuencia contiene más registros, la recuperación se produce a partir del registro con el siguiente número subsecuencial dentro del registro con el número secuencial que se haya comprobado más recientemente. Si el punto de comprobación más reciente incluyó el último número subsecuencial del registro con el número secuencial anterior, la recuperación se produce a partir del registro con el siguiente número secuencial.

En la siguiente sección se explican los detalles de la comprobación secuencial y subsecuencial para los consumidores que deban evitar la omisión y la duplicación de registros. Si la omisión (o duplicación) de registros al detener y reiniciar el procesamiento de registros del consumidor no resulta importante, puede ejecutar su código existente sin modificaciones.

Extensiones de KCL para la desagrupación de KPL

Como se ha mencionado anteriormente, la desagrupación de KPL puede implicar la creación de puntos de control subsecuenciales. Para facilitar el uso de la creación de puntos de control subsecuenciales, se ha agregado una clase `UserRecord` a 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 */
    }
}
```

Esta categoría es la que se usa ahora en lugar de `Record`. Esto no afecta al código existente, porque es una subclase de `Record`. La clase `UserRecord` representa tanto subregistros reales como registros estándares no agregados. Se pueden describir los registros no agregados como registros agregados con solo un subregistro.

Además, se han añadido dos nuevas operaciones a `IRecordProcessorCheckpointer`:

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

Para empezar a utilizar la creación de puntos de comprobación de números subsecuenciales, puede realizar la siguiente conversión. Cambie el siguiente código de formulario:

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

Nuevo código de formulario:

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

Le recomendamos que use el formulario `checkpoint(Record record)` para la generación de puntos de comprobación subsecuenciales. Sin embargo, si ya está almacenando `sequenceNumbers` en cadenas para la creación de puntos de comprobación, ahora también deberá almacenar `subSequenceNumber`, tal y como se muestra en el ejemplo siguiente:

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

El cast de `Record` a `UserRecord` siempre se realiza correctamente, ya que en la implementación siempre se usa `UserRecord` internamente. A no ser que sea necesario realizar cálculos aritméticos en los números secuenciales, no recomendamos este enfoque.

Al procesar los registros de usuario de KPL, KCL escribe el número subsecuencial en Amazon DynamoDB como un campo adicional para cada fila. Las versiones anteriores de KCL utilizaban `AFTER_SEQUENCE_NUMBER` para obtener los registros al reanudar los puntos de control. KCL actual con compatibilidad con KPL utiliza `AT_SEQUENCE_NUMBER` en su lugar. Cuando se recupera el registro del número secuencial del punto de comprobación, se comprueba el número secuencial sometido al punto de comprobación, y los subregistros se descartan según proceda (podrían ser todos ellos, si el último subregistro es el del punto de comprobación). De nuevo, se pueden entender los registros no agregados como registros agregados con un único subregistro, de modo que el mismo algoritmo funciona tanto para los registros agregados como para los no agregados.

Utilizándolo GetRecords directamente

También puede optar por no utilizar KCL e invocar en su lugar la operación de la API `GetRecords` directamente para recuperar registros de Kinesis Data Streams. Para desempaquetar estos registros recuperados en el registro de usuario original de KPL, llame a una de las siguientes operaciones estáticas en `UserRecord.java`:

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

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

La primera operación utiliza el valor predeterminado `0` (cero) para `startingHashKey` y el valor predeterminado `2^128 -1` para `endingHashKey`.

Cada una de estas operaciones desagrupa la lista de registros de Kinesis Data Streams especificada y genera una lista de registros de usuario de KPL. Cualquier registro de usuario de KPL cuya clave hash explícita o clave de partición quede fuera del rango entre `startingHashKey` (incluida) y `endingHashKey` (incluida) se descarta de la lista de registros devuelta.

Uso del KPL con Firehose

Si utiliza Kinesis Producer Library (KPL) para escribir datos en un flujo de datos de Kinesis, puede utilizar la agregación para combinar los registros que escriba en ese flujo de datos de Kinesis. Si luego utilizas ese flujo de datos como fuente para tu flujo de entrega de Firehose, Firehose desagrega los registros antes de entregarlos al destino. Si configura el flujo de entrega para transformar los datos, Firehose desagrega los registros antes de entregarlos a ellos. AWS LambdaPara obtener más información, consulte Escritura en Amazon Firehose mediante Kinesis Data Streams.

Uso del KPL con el registro de esquemas de AWS Glue

Puede integrar sus flujos de datos de Kinesis con el registro de esquemas de AWS Glue. AWS Glue Schema Registry le permite descubrir, controlar y evolucionar de forma centralizada esquemas, además de garantizar que un esquema registrado valide de forma continua los datos generados. Un esquema define la estructura y el formato de un registro de datos. Un esquema es una especificación versionada para publicación, consumo o almacenamiento de confianza de datos. El registro de esquemas de AWS Glue le permite mejorar la end-to-end calidad y el gobierno de los datos en sus aplicaciones de streaming. Para obtener más información, consulte AWS Glue Schema Registry. Una de las maneras de configurar esta integración es a través de las bibliotecas KPL y Kinesis Client Library (KCL) en Java.

> **Important**
>
> Actualmente, la integración del registro de esquemas de Kinesis Data Streams y AWS Glue solo se admite para las transmisiones de datos de Kinesis que utilizan productores de KPL implementados en Java. No se proporciona soporte multilingüe.

Para obtener instrucciones detalladas sobre cómo configurar la integración de Kinesis Data Streams con Schema Registry mediante el KPL, consulte la sección «Interacción con los datos mediante las bibliotecas KPL/KCL» en Caso de uso: integración de Amazon Kinesis Data Streams con el registro de esquemas de Glue. AWS

Configuración del proxy de KPL

En el caso de las aplicaciones que no pueden conectarse directamente a Internet, todos los clientes del AWS SDK admiten el uso de proxies HTTP o HTTPS. En un entorno empresarial típico, todo el tráfico de red saliente tiene que pasar por servidores proxy. Si su aplicación utiliza la biblioteca de productores de Kinesis (KPL) para recopilar y enviar datos AWS en un entorno que utiliza servidores proxy, necesitará una configuración de proxy KPL. KPL es una biblioteca de alto nivel creada sobre el SDK de AWS Kinesis. Se divide en un proceso nativo y un contenedor. El proceso nativo realiza todos los trabajos de procesamiento y envío de registros, mientras que el contenedor administra el proceso nativo y se comunica con este. Para obtener más información, consulte Implementing Efficient and Reliable Producers with the Amazon Kinesis Producer Library.

El contenedor está escrito en Java y el proceso nativo está escrito en C++ con el uso del SDK de Kinesis. La versión 0.14.7 y posterior de KPL ahora admite la configuración del proxy en el contenedor de Java, que puede transferir todas las configuraciones del proxy al proceso nativo. Para obtener más información, consulte https://github.com/awslabs/ amazon-kinesis-producer /releases/ tag/v0.14.7.

Puede utilizar el código siguiente para agregar las configuraciones del proxy a las aplicaciones de 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);
```

Desarrollo de productores mediante la API de Amazon Kinesis Data Streams con AWS SDK for Java

Puede desarrollar productores mediante la API de Amazon Kinesis Data Streams con el SDK de AWS para Java. Si es la primera vez que utiliza Kinesis Data Streams, familiarícese antes con los

conceptos y los términos que encontrará en ¿Qué es Amazon Kinesis Data Streams? y Introducción a Amazon Kinesis Data Streams.

En estos ejemplos se aborda la API de Kinesis Data Streams y se utiliza el SDK de AWS para Java para insertar datos en un flujo. Sin embargo, en la mayoría de los casos de uso, es preferible optar por la biblioteca KPL de Kinesis Data Streams. Para obtener más información, consulte Desarrollo de productores con Amazon Kinesis Producer Library.

El código de ejemplo de Java de este capítulo demuestra cómo realizar operaciones básicas con la API de Kinesis Data Streams y está dividido lógicamente por tipo de operación. Estos ejemplos no representan códigos listos para producción, ya que no comprueban todas las excepciones posibles ni toman en cuenta todas las consideraciones de seguridad y desempeño posibles. Además, puede llamar a la API de Kinesis Data Streams mediante otros lenguajes de programación. Para más información acerca de todos los SDK de AWS disponibles, consulte Comience a crear con Amazon Web Services.

Cada tarea tiene requisitos previos. Por ejemplo, no se pueden agregar datos a una secuencia hasta que se haya creado una, para lo que se ha de crear un cliente. Para obtener más información, consulte Creación y administración de secuencias.

Temas

- Agregar datos a una secuencia
- Interacción con los datos mediante AWS Glue Schema Registry

Agregar datos a una secuencia

Una vez que se crea una secuencia, puede agregar datos a ella en forma de registros. Un registro es una estructura de datos que contiene los datos que se han de procesar en forma de un blob de datos. Después de almacenar los datos en el registro, Kinesis Data Streams no inspecciona, interpreta ni cambia los datos. Cada registro también tiene asociado un número secuencial y una clave de partición.

Existen dos operaciones diferentes en la API de Kinesis Data Streams que agregan datos a un flujo: `PutRecords` y `PutRecord`. La operación `PutRecords` envía varios registros a su secuencia por solicitud HTTP, y la operación única `PutRecord` envía registros a su secuencia, una por una (se necesita una solicitud HTTP independiente para cada registro). Es preferible utilizar `PutRecords` para la mayoría de las aplicaciones, ya que conseguirá un mayor rendimiento por productor de datos.

Para obtener más información sobre cada una de estas operaciones, consulte las subsecciones independientes que aparecen a continuación.

Temas

- Agregar varios registros con PutRecords
- Agregar un único registro con PutRecord

Siempre debe tener en mente que, a medida que la aplicación de origen agrega datos al flujo mediante la API de Kinesis Data Streams, muy probablemente haya una o varias aplicaciones de consumidor que procesan datos simultáneamente desde el flujo. Para obtener información sobre cómo los consumidores obtienen datos mediante la API de Kinesis Data Streams, consulte Obtención de datos de una secuencia.

> **Important**
>
> Cambiar el periodo de retención de datos

Agregar varios registros con PutRecords

La operación `PutRecords` envía varios registros a Kinesis Data Streams en una única solicitud. Al utilizar `PutRecords`, los productores pueden conseguir un mayor rendimiento cuando envían datos a los flujos de datos de Kinesis. Cada solicitud `PutRecords` puede admitir hasta 500 registros. Cada registro puede ser tan grande como 1 MB, hasta un límite de 5 MB para toda la solicitud, incluidas las claves de partición. Al igual que con la operación única `PutRecord` que se describe a continuación, `PutRecords` utiliza números secuenciales y claves de partición. Sin embargo, el parámetro de `PutRecord` `SequenceNumberForOrdering` no se incluye en una llamada `PutRecords`. La operación `PutRecords` intenta procesar todos los registros en el orden natural de la solicitud.

Cada registro de datos tiene un número secuencial único. Kinesis Data Streams asigna el número secuencial cuando se llama a `client.putRecords` para agregar los registros de datos al flujo. Por lo general, los números secuenciales de una misma clave de partición aumentan con el tiempo; cuanto más grande sea el periodo de tiempo transcurrido entre las solicitudes a `PutRecords`, más aumentan los números secuenciales.

> **Note**
>
> Los números secuenciales no se pueden utilizar como índices de conjuntos de datos dentro de la misma secuencia. Para separar lógicamente conjuntos de datos, utilice claves de partición o cree una secuencia independiente para cada conjunto de datos.

Una solicitud `PutRecords` puede incluir registros con diferentes claves de partición. El ámbito de la solicitud es una secuencia; cada solicitud puede incluir cualquier combinación de claves de partición y registros hasta alcanzar los límites de la solicitud. Las solicitudes realizadas con varias claves de partición a secuencias con muchos fragmentos diferentes suelen ser más rápidas que las solicitudes con un número reducido de claves de partición a un número pequeño de fragmentos. El número de claves de partición debe ser mucho mayor que el número de fragmentos a fin de reducir la latencia y maximizar el rendimiento.

Ejemplo de PutRecords

El siguiente código crea 100 registros de datos con claves de partición secuenciales y los inserta en una secuencia llamada `DataStream`.

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

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

        AmazonKinesis kinesisClient = clientBuilder.build();

        PutRecordsRequest putRecordsRequest  = new PutRecordsRequest();
        putRecordsRequest.setStreamName(streamName);
        List <PutRecordsRequestEntry> putRecordsRequestEntryList  = new ArrayList<>();
        for (int i = 0; i < 100; i++) {
            PutRecordsRequestEntry putRecordsRequestEntry  = new
PutRecordsRequestEntry();

putRecordsRequestEntry.setData(ByteBuffer.wrap(String.valueOf(i).getBytes()));
            putRecordsRequestEntry.setPartitionKey(String.format("partitionKey-%d",
i));
            putRecordsRequestEntryList.add(putRecordsRequestEntry);
        }
```

```
        putRecordsRequest.setRecords(putRecordsRequestEntryList);
        PutRecordsResult putRecordsResult  =
 kinesisClient.putRecords(putRecordsRequest);
        System.out.println("Put Result" + putRecordsResult);
```

La respuesta `PutRecords` incluye una gama de `Records` de respuesta. Cada registro de la matriz de respuestas se correlaciona directamente con un registro en la matriz de solicitudes siguiendo el orden natural, de arriba abajo de la solicitud y la respuesta. La matriz de `Records` de respuesta siempre incluye el mismo número de registros que la matriz de solicitudes.

Control de errores al utilizar PutRecords

De forma predeterminada, el error de registros individuales en una solicitud no para el procesamiento de los registros siguientes en una solicitud `PutRecords`. Esto significa que una matriz de `Records` de respuesta incluye tanto los registros procesados correctamente como los que no. Debe detectar los registros procesados de forma incorrecta e incluirlos en una llamada posterior.

Los registros correctos incluyen los valores `SequenceNumber` y `ShardID`, y los registros incorrectos incluyen los valores `ErrorCode` y `ErrorMessage`. El parámetro `ErrorCode` refleja el tipo de error y puede tomar uno de los siguientes valores: `ProvisionedThroughputExceededException` o `InternalFailure`. `ErrorMessage` proporciona información más detallada sobre la excepción `ProvisionedThroughputExceededException`, e incluye el ID de la cuenta, el nombre de la secuencia y el ID del fragmento del registro al que se ha aplicado limitación. El ejemplo siguiente tiene tres registros en una solicitud `PutRecords`. El segundo registro ha generado un error y se refleja en la respuesta.

Example Sintaxis de la solicitud PutRecords

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

```
     "PartitionKey": "partitionKey3"
        }
    ],
    "StreamName": "myStream"
}
```

Example Sintaxis de la respuesta PutRecords

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

Los registros que se procesan sin éxito se pueden incluir en las solicitudes `PutRecords` posteriores. En primer lugar, compruebe el parámetro `FailedRecordCount` en `putRecordsResult` para confirmar si se hay registros con error en la solicitud. En caso afirmativo, cada `putRecordsEntry` que tenga un `ErrorCode` que no sea `null` se debe agregar a una solicitud posterior. Para un ejemplo de este tipo de controlador, consulte el siguiente código.

Example Administrador de errores de PutRecords

```
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);
}
```

Agregar un único registro con PutRecord

Cada llamada a `PutRecord` opera sobre un solo registro. Es preferible recurrir a la operación `PutRecords` que se describe en Agregar varios registros con PutRecords a menos que su aplicación necesite específicamente enviar siempre registros individuales en cada solicitud, o que por cualquier otro motivo no se pueda utilizar `PutRecords`.

Cada registro de datos tiene un número secuencial único. Kinesis Data Streams asigna el número secuencial cuando se llama a `client.putRecord` para agregar el registro de datos al flujo. Por lo general, los números secuenciales de una misma clave de partición aumentan con el tiempo; cuanto más grande sea el periodo de tiempo transcurrido entre las solicitudes a `PutRecord`, más aumentan los números secuenciales.

Cuando se producen inserciones (puts) en una sucesión rápida, no hay garantía de que los números secuenciales devueltos aumenten, ya que las operaciones put aparecen esencialmente de manera simultánea a Kinesis Data Streams. Para garantizar unos números secuenciales estrictamente en

aumento para la misma clave de partición, utilice el parámetro `SequenceNumberForOrdering` tal y como se muestra en el código de muestra de Ejemplo de PutRecord.

Independientemente de que use o no `SequenceNumberForOrdering`, los registros que recibe Kinesis Data Streams mediante una llamada a `GetRecords` están estrictamente ordenados por número secuencial.

> **Note**
>
> Los números secuenciales no se pueden utilizar como índices de conjuntos de datos dentro de la misma secuencia. Para separar lógicamente conjuntos de datos, utilice claves de partición o cree una secuencia independiente para cada conjunto de datos.

Una clave de partición se utiliza para agrupar datos dentro de una secuencia. Un registro de datos se asigna a un fragmento dentro de la secuencia en función de su clave de partición. En concreto, Kinesis Data Streams utiliza la clave de partición como entrada para una función hash que asigna la clave de partición (y los datos asociados) a una partición específica.

Como resultado de este mecanismo de hash, todos los registros de datos con la misma clave de partición se asignan al mismo fragmento dentro de la secuencia. Sin embargo, si el número de claves de partición supera el número de fragmentos, algunos fragmentos han de contener necesariamente registros con diferentes claves de partición. Desde el punto de vista del diseño, para garantizar que todos los fragmentos estén bien utilizados, el número de fragmentos (especificado mediante el método `setShardCount` de `CreateStreamRequest`) debe ser significativamente inferior al número de claves de partición únicas, y la cantidad de datos que entran en una única clave de partición debe ser significativamente inferior a la capacidad del fragmento.

Ejemplo de PutRecord

El siguiente código crea diez registros de datos distribuidos en dos claves de partición y los inserta en una secuencia llamada `myStreamName`.

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

```
  putRecordRequest.setPartitionKey( String.format( "partitionKey-%d", j/5 ));
  putRecordRequest.setSequenceNumberForOrdering( sequenceNumberOfPreviousRecord );
  PutRecordResult putRecordResult = client.putRecord( putRecordRequest );
  sequenceNumberOfPreviousRecord = putRecordResult.getSequenceNumber();
}
```

El código de muestra anterior utiliza `setSequenceNumberForOrdering` para garantizar un orden estrictamente creciente en cada clave de partición. Para utilizar este parámetro de forma eficaz, establezca como `SequenceNumberForOrdering` del registro actual (registro n) el número secuencial del registro anterior (registro n-1). Para obtener el número secuencial de un registro que se ha añadido a la secuencia, llame a `getSequenceNumber` en el resultado de `putRecord`.

El parámetro `SequenceNumberForOrdering` garantiza números de secuencia estrictamente crecientes para la misma clave de partición. `SequenceNumberForOrdering` no permite ordenar los registros en varias claves de partición.

Interacción con los datos mediante AWS Glue Schema Registry

Puede integrar los flujos de datos de Kinesis con AWS Glue Schema Registry. AWS Glue Schema Registry le permite descubrir, controlar y evolucionar de forma centralizada esquemas, además de garantizar que un esquema registrado valide de forma continua los datos generados. Un esquema define la estructura y el formato de un registro de datos. Un esquema es una especificación versionada para publicación, consumo o almacenamiento de confianza de datos. AWS Glue Schema Registry le permite mejorar la calidad de los datos de principio a fin y la gobernanza de los datos en las aplicaciones de streaming. Para obtener más información, consulte AWS Glue Schema Registry. Una de las formas de configurar esta integración es mediante la API de Kinesis Data Streams `PutRecords` y `PutRecord`, disponible en el SDK de AWS para Java.

Para obtener instrucciones detalladas sobre cómo configurar la integración de Kinesis Data Streams con Schema Registry mediante las API de Kinesis Data Streams PutRecords y PutRecord, consulte la sección “Interacción con datos mediante las API de Kinesis Data Streams” en Caso de uso: integración de Amazon Kinesis Data Streams con AWS Glue Schema Registry.

Escritura en Amazon Kinesis Data Streams mediante el agente de Kinesis

El agente de Kinesis es una aplicación de software Java independiente que ofrece una forma sencilla de recopilar y enviar datos a Kinesis Data Streams. El agente monitoriza constantemente un conjunto

de archivos y envía nuevos datos a su secuencia. El agente se encarga de rotar los archivos, crear puntos de restauración y realizar reintentos cuando se producen errores. El agente entrega todos los datos de manera confiable, puntual y sencilla. También emite CloudWatch métricas de Amazon para ayudarte a supervisar y solucionar mejor los problemas del proceso de streaming.

De forma predeterminada, los registros de cada archivo se analizan en función del carácter de nueva línea (`'\n'`). Sin embargo, el agente también se puede configurar para analizar registros multilínea (consulte Ajustes de la configuración del agente).

Puede instalar el agente en entornos de servidor basados en Linux, como servidores web, de registro o de base de datos. Después de instalar el agente, configúrelo especificando los archivos que desee monitorizar y la secuencia de los datos. Una vez configurado, el agente recopila datos de los archivos de forma duradera y los envía de forma confiable a la secuencia.

Temas

Requisitos previos

- Su sistema operativo debe ser la versión 2015.09 o posterior de Amazon Linux AMI o la versión 7 o posterior de Red Hat Enterprise Linux.
- Si utiliza Amazon EC2, lance la instancia de EC2 para ejecutar el agente.
- Gestiona tus AWS credenciales mediante uno de los siguientes métodos:
  - Especifique un rol de IAM al lanzar la instancia EC2.
  - Especifique AWS las credenciales al configurar el agente (consulte awsAccessKeyID y awsSecretAccessclave).
  - `/etc/sysconfig/aws-kinesis-agent`Edítelo para especificar su región y sus claves de AWS acceso.

- Si su instancia EC2 está en una AWS cuenta diferente, cree una función de IAM para proporcionar acceso al servicio Kinesis Data Streams y especifique esa función al configurar el agente (consulte AssumeroLearn and Id). assumeRoleExternal Utilice uno de los métodos anteriores para especificar AWS las credenciales de un usuario de la otra cuenta que tenga permiso para asumir este rol.
- El rol o AWS las credenciales de IAM que especifique deben tener permiso para realizar la operación de Kinesis Data PutRecordsStreams para que el agente envíe datos a su transmisión. Si habilita la CloudWatch supervisión del agente, también necesitará permiso para realizar la CloudWatch PutMetricDataoperación. Para obtener más información, consulte Control del acceso a los recursos de Amazon Kinesis Data Streams mediante IAMSupervisión del agente de estado de Kinesis Data Streams con Amazon CloudWatch, y Control de CloudWatch acceso.

Descargar e instalar el agente

Primero, conéctese a la instancia. Para obtener más información, consulte Conectarse a la instancia en la Guía del usuario de Amazon EC2 para instancias de Linux. Si tiene problemas para conectarse, consulte Solución de problemas de conexión a su instancia en la Guía del usuario de Amazon EC2 para instancias Linux.

Para configurar el agente a través de la AMI de Amazon Linux

Use el siguiente comando para descargar e instalar el agente:

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

Para configurar el agente en Red Hat Enterprise Linux

Use el siguiente comando para descargar e instalar el agente:

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

Para configurar el agente mediante GitHub

1. Descargue el agente desde awlabs/ amazon-kinesis-agent.
2. Instale el agente. Para ello, diríjase al directorio de descargas y ejecute el siguiente comando:

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

Configuración del agente en un contenedor de Docker

El agente de Kinesis también puede ejecutarse en un contenedor además de a través de la base de contenedores amazonlinux. Utilice el siguiente Dockerfile y ejecute `docker build`.

```
FROM amazonlinux

RUN yum install -y aws-kinesis-agent which findutils
COPY agent.json /etc/aws-kinesis/agent.json

CMD ["start-aws-kinesis-agent"]
```

Configuración e inicio del agente

Configuración e inicio del agente

1. Abra y edite el archivo de configuración (como superusuario si utiliza permisos de acceso de archivo predeterminado): `/etc/aws-kinesis/agent.json`

   En este archivo de configuración, especifique los archivos (`"filePattern"`) desde los que el agente deberá recopilar datos y el nombre de la secuencia (`"kinesisStream"`) a la que deberá enviarlos. Tenga en cuenta que el nombre de archivo es un patrón y que el agente reconoce rotaciones de archivos. No puede rotar más de un archivo ni crear más de uno nuevo por segundo. El agente utiliza la marca temporal de creación de archivos para determinar a qué archivos realizarles seguimiento y poner en fila en la secuencia. Crear nuevos archivos o rotarlos más de una vez por segundo evita que el agente pueda diferenciarlos correctamente.

   ```
   {
      "flows": [
           {
               "filePattern": "/tmp/app.log*",
               "kinesisStream": "yourkinesisstream"
           }
      ]
   }
   ```

2. Comience el agente de forma manual:

   ```
   sudo service aws-kinesis-agent start
   ```

3. Configure el agente para iniciarse al arrancar el sistema (opcional):

```
sudo chkconfig aws-kinesis-agent on
```

El agente ya está se ejecutando como un servicio de sistema en segundo plano. Monitoriza constantemente los archivos especificados y envía datos a la secuencia especificada. La auditoría de actividad se registra en `/var/log/aws-kinesis-agent/aws-kinesis-agent.log`.

Ajustes de la configuración del agente

El agente admite las dos opciones de configuración obligatorias, `filePattern` y `kinesisStream`, además de configuraciones opcionales para activar características adicionales. Las opciones de configuración obligatorias y opcionales se especifican en `/etc/aws-kinesis/agent.json`.

Cada vez que cambie el archivo de configuración, debe detener y comenzar el agente con los siguientes comandos:

```
sudo service aws-kinesis-agent stop
sudo service aws-kinesis-agent start
```

También puede hacerlo con el siguiente comando:

```
sudo service aws-kinesis-agent restart
```

Las opciones de configuración generales son las siguientes.

Opción de configuración	Descripción
`assumeRoleARN`	El ARN del rol que asumirá el usuario. Para obtener más información, consulte Delegar el acceso a todas AWS las cuentas mediante funciones de IAM en la Guía del usuario de IAM.
`assumeRoleExternalId`	Un identificador opcional que determina quién puede asumir el rol. Para obtener más información, consulte Cómo utilizar un ID externo en la Guía del usuario de IAM.

Opción de configuración	Descripción
`awsAccessKeyId`	AWS ID de clave de acceso que anula las credenciales predeterminadas. Este ajuste tiene prioridad sobre los demás proveedores de credenciales.
`awsSecretAccessKey`	AWS clave secreta que anula las credenciales predeterminadas. Este ajuste tiene prioridad sobre los demás proveedores de credenciales.
`cloudwatch.emitMetrics`	Permite que el agente emita métricas CloudWatch si se ha establecido (true). Predeterminado: true
`cloudwatch.endpoint`	El punto final regional de CloudWatch. Predeterminado: `monitoring.us-east-1.amazonaws.com`
`kinesis.endpoint`	El punto de conexión regional para Kinesis Data Streams. Predeterminado: `kinesis.us-east-1.amazonaws.com`

Las opciones de configuración de flujo son las siguientes.

Opción de configuración	Descripción
`dataProcessingOptions`	La lista de opciones de procesamiento aplicadas a cada registro analizado antes de enviarlo a la secuencia. Las opciones de procesamiento se realizan en el orden especificado. Para obtener más información, consulte Preprocesado de datos con el agente.
`kinesisStream`	[Obligatorio] El nombre de la secuencia.
`filePattern`	[Obligatorio] El patrón de directorio y archivo que debe coincidir para que el agente lo recoja. Se debe conceder permisos de lectura a `aws-kinesis-agent-user` para todos los archivos que coincidan con

Opción de configuración	Descripción
	este patrón. Se debe conceder permisos de lectura y ejecución a `aws-kinesis-agent-user` para el directorio que contiene los archivos.
`initialPosition`	La posición inicial desde la que el archivo comenzó a ser analizado. Los valores válidos son `START_OF_FILE` y `END_OF_FILE` . Predeterminado: `END_OF_FILE`
`maxBufferAgeMillis`	El tiempo máximo, en milisegundos, durante el cual el agente almacena los datos en búfer antes de enviarlos a la secuencia. Intervalo de valores: 1000 a 900 000 (1 segundo a 15 minutos) Valor predeterminado: 60 000 (1 minuto)
`maxBufferSizeBytes`	La cantidad de datos en bytes que el agente almacena en búfer antes de enviarlos a la secuencia. Intervalo de valores: 1 a 4 194 304 (4 MB) Valor predeterminado: 4 194 304 (4 MB)
`maxBufferSizeRecords`	La cantidad máxima de registros en datos que el agente almacena en búfer antes de enviarlos a la secuencia. Intervalo de valores: 1 a 500 Predeterminado: 500
`minTimeBetweenFilePollsMillis`	El intervalo de tiempo, en milisegundos, en el que el agente sondea y analiza los archivos monitorizados para identificar datos nuevos. Intervalo de valores: 1 o más Predeterminado: 100

Opción de configuración	Descripción
`multiLineStartPattern`	El patrón para identificar el comienzo de un registro. Un registro consta de una línea que coincide con el patrón y de líneas siguientes que no coinciden con el patrón. Los valores válidos son expresiones regulares . De forma predeterminada, cada línea en los archivos de registro se analiza como un registro.
`partitionKeyOption`	El método para generar la clave de partición. Los valores válidos son `RANDOM` (entero generado aleatoriamente) y `DETERMINISTIC` (un valor hash calculado a partir de los datos). Predeterminado: `RANDOM`
`skipHeaderLines`	La cantidad de líneas de los archivos monitorizados, a partir de la primera, que el agente debe omitir en el momento de analizarlos. Intervalo de valores: 0 o más Cantidad predeterminada: 0 (cero)
`truncatedRecordTerminator`	La cadena que utiliza el agente para truncar un registro analizado cuando su tamaño supera el límite de tamaño de registros de Kinesis Data Streams. (1000 KB) Valor predeterminado: `'\n'` (línea nueva)

Monitoreo de varios directorios de archivos y escritura en varias secuencias

Puede configurar el agente para que monitorice varios directorios de archivos y envíe datos a varias secuencias especificando varias opciones de configuración de secuencia. En el siguiente ejemplo de configuración, el agente supervisa dos directorios de archivos y envía datos a una transmisión de Kinesis y a una transmisión de entrega de Firehose, respectivamente. Tenga en cuenta que puede especificar puntos de enlace diferentes para Kinesis Data Streams y Firehose, de modo que la transmisión de Kinesis y la transmisión de entrega de Firehose no tengan que estar en la misma región.

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

Para obtener información más detallada sobre el uso del agente con Firehose, consulte Cómo escribir en Amazon Data Firehose con Kinesis Agent.

Preprocesado de datos con el agente

El agente puede preprocesar los registros previamente analizados de los archivos monitorizados antes de enviarlos a la secuencia. Para habilitar esta característica, añada la opción de configuración `dataProcessingOptions` al flujo de archivos. Se pueden agregar una o más opciones de procesamiento, que se ejecutarán en el orden especificado.

El agente es compatible con las siguientes opciones de procesamiento. Al ser de código abierto, puede desarrollar y ampliar las opciones de procesamiento del agente. Puede descargar el agente desde el agente de Kinesis.

Opciones de procesamiento

`SINGLELINE`

Elimina caracteres de nueva línea y espacios al principio y al final de las líneas para convertir un registro multilínea en un registro de una sola línea.

```
{
    "optionName": "SINGLELINE"
}
```

`CSVTOJSON`

Convierte un registro de formato con separaciones mediante delimitadores en un registro de formato JSON.

```
{
    "optionName": "CSVTOJSON",
    "customFieldNames": [ "field1", "field2", ... ],
    "delimiter": "yourdelimiter"
}
```

`customFieldNames`

[Obligatorio] Los nombres de campos utilizados como claves en cada par de valores de clave JSON. Por ejemplo, si especifica `["f1", "f2"]`, el registro "v1, v2" se convertirá en `{"f1":"v1","f2":"v2"}`.

`delimiter`

La cadena utilizada como delimitador en el registro. El valor predeterminado es una coma (,).

`LOGTOJSON`

Convierte un registro con un formato de registro en un registro con formato JSON. Los formatos de registro admitidos son Apache Common Log, Apache Combined Log, Apache Error Log y RFC3164 Syslog.

```
{
    "optionName": "LOGTOJSON",
    "logFormat": "logformat",
    "matchPattern": "yourregexpattern",
    "customFieldNames": [ "field1", "field2", … ]
}
```

`logFormat`

[Obligatorio] El formato de entrada del registro. Los valores posibles son los siguientes:

- `COMMONAPACHELOG`: formato común de registro de Apache. Cada entrada de registro sigue el siguiente patrón de forma predeterminada: "`%{host} %{ident} %{authuser} [%{datetime}] \"%{request}\" %{response} %{bytes}`".
- `COMBINEDAPACHELOG`: formato combinado de registro de Apache. Cada entrada de registro sigue el siguiente patrón de forma predeterminada: "`%{host} %{ident} %{authuser}`

`[%{datetime}] \"%{request}\" %{response} %{bytes} %{referrer} %{agent}"`.

- `APACHEERRORLOG`: formato de registro de errores de Apache. Cada entrada de registro sigue el siguiente patrón de forma predeterminada: "`[%{timestamp}] [%{module}: %{severity}] [pid %{processid}:tid %{threadid}] [client: %{client}] %{message}`".
- `SYSLOG`: formato RFC3164 de Syslog. Cada entrada de registro sigue el siguiente patrón de forma predeterminada: "`%{timestamp} %{hostname} %{program}[%{processid}]: %{message}`".

`matchPattern`

El patrón de expresiones regulares utilizado para extraer valores de entradas de registro. Este ajuste se utiliza si la entrada de registro no tiene uno de los formatos de registro predefinidos. Si se utiliza este ajuste, también tendrá que especificar `customFieldNames`.

`customFieldNames`

Los nombres de campos utilizados como claves en cada par de valores de clave JSON. Utilice esta opción para definir nombres de campos para valores extraídos de `matchPattern`, o sobrescriba los nombres de campos de los formatos de logs predefinidos.

Example : Configuración LOGTOJSON

Este es un ejemplo de configuración `LOGTOJSON` de una entrada de registro en Formato común de registro de Apache convertida a formato JSON:

```
{
    "optionName": "LOGTOJSON",
    "logFormat": "COMMONAPACHELOG"
}
```

Antes de la conversión:

```
64.242.88.10 - - [07/Mar/2004:16:10:02 -0800] "GET /mailman/listinfo/hsdivision
 HTTP/1.1" 200 6291
```

Después de la conversión:

```
{"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 : Configuración LOGTOJSON con campos personalizados

Este es otro ejemplo de configuración `LOGTOJSON`:

```
{
    "optionName": "LOGTOJSON",
    "logFormat": "COMMONAPACHELOG",
    "customFieldNames": ["f1", "f2", "f3", "f4", "f5", "f6", "f7"]
}
```

Con esta configuración, la misma entrada de registro con Formato común de registro de Apache del ejemplo anterior se convierte a formato JSON de la siguiente manera:

```
{"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 : Convertir una entrada de registro con Formato común de registro de Apache

La siguiente configuración de flujo convierte la entrada de registro con Formato común de registro de Apache en un registro de una línea en formato JSON:

```
{
    "flows": [
        {
            "filePattern": "/tmp/app.log*",
            "kinesisStream": "my-stream",
            "dataProcessingOptions": [
                {
                    "optionName": "LOGTOJSON",
                    "logFormat": "COMMONAPACHELOG"
                }
            ]
        }
    ]
}
```

Example : Convertir registros multilínea

La siguiente configuración de flujo analiza aquellos registros multilínea cuya primera línea comience por "`[SEQUENCE=`". Primero, el registro se convierte en un registro de una línea. Después, se extraen los valores del registro basándose en tabulaciones delimitadoras. Finalmente, los valores extraídos se asignan a valores `customFieldNames` específicos para formar un registro de una línea en formato JSON.

```
{
    "flows": [
        {
            "filePattern": "/tmp/app.log*",
            "kinesisStream": "my-stream",
            "multiLineStartPattern": "\\[SEQUENCE=",
            "dataProcessingOptions": [
                {
                    "optionName": "SINGLELINE"
                },
                {
                    "optionName": "CSVTOJSON",
                    "customFieldNames": [ "field1", "field2", "field3" ],
                    "delimiter": "\\t"
                }
            ]
        }
    ]
}
```

Example : Configuración LOGTOJSON con patrón de coincidencia

Este es un ejemplo de una configuración de entrada de registro con Formato común de registro de Apache `LOGTOJSON` convertida a formato JSON con el último campo (bytes) omitido:

```
{
    "optionName": "LOGTOJSON",
    "logFormat": "COMMONAPACHELOG",
    "matchPattern": "^([\\d.]+) (\\S+) (\\S+) \\[([\\w:/]+\\s[+\\-]\\d{4})\\] \"(.
+?)\" (\\d{3})",
    "customFieldNames": ["host", "ident", "authuser", "datetime", "request",
 "response"]
}
```

Antes de la conversión:

```
123.45.67.89 - - [27/Oct/2000:09:27:09 -0400] "GET /java/javaResources.html HTTP/1.0"
 200
```

Después de la conversión:

```
{"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"}
```

Comandos del agente de la CLI

Iniciar automáticamente al agente al arrancar el sistema:

```
sudo chkconfig aws-kinesis-agent on
```

Compruebe el estado del agente:

```
sudo service aws-kinesis-agent status
```

Detener el agente:

```
sudo service aws-kinesis-agent stop
```

Leer el archivo de registro del agente desde esta ubicación:

```
/var/log/aws-kinesis-agent/aws-kinesis-agent.log
```

Desinstalar el agente:

```
sudo yum remove aws-kinesis-agent
```

Preguntas frecuentes

¿Hay un agente de Kinesis para Windows?

El agente de Kinesis para Windows es un software diferente del agente de Kinesis para plataformas Linux.

¿Por qué se ralentiza el agente de Kinesis o aumenta el valor de **`RecordSendErrors`**?

Normalmente esto se debe a la limitación de Kinesis. Compruebe la `WriteProvisionedThroughputExceeded` métrica de Kinesis Data Streams o `ThrottledRecords` la métrica de Firehose Delivery Streams. Cualquier aumento a partir de 0 en estas métricas indica que es necesario aumentar los límites de flujo. Para obtener más información, consulte Límites de flujos de datos de Kinesis y flujos de entrega de Amazon Firehose.

Una vez que descarte la limitación de velocidad, compruebe si el agente de Kinesis está configurado para enviar a la cola una gran cantidad de archivos pequeños. Se produce un retraso cuando el agente de Kinesis sigue un archivo nuevo, por lo que el agente de Kinesis debería seguir una pequeña cantidad de archivos de mayor tamaño. Intente consolidar los archivos de registro en archivos más grandes.

¿Por qué se producen excepciones **`java.lang.OutOfMemoryError`** ?

El agente de Kinesis no tiene memoria suficiente para gestionar la carga de trabajo actual. Intente aumentar `JAVA_START_HEAP` y `JAVA_MAX_HEAP` en `/usr/bin/start-aws-kinesis-agent` y reinicie el agente.

¿Por qué se producen excepciones **`IllegalStateException : connection pool shut down`**?

El agente de Kinesis no tiene suficientes conexiones para gestionar la carga de trabajo actual. Intente aumentar `maxConnections` y `maxSendingThreads` en los ajustes generales de configuración del agente en `/etc/aws-kinesis/agent.json`. El valor predeterminado para estos campos es 12 veces los procesadores de tiempo de ejecución disponibles. Consulte AgentConfiguration.java para obtener más información sobre los ajustes de configuración avanzada de los agentes.

¿Cómo puedo depurar otro problema con el agente de Kinesis?

Los registros de nivel `DEBUG` pueden habilitarse en `/etc/aws-kinesis/log4j.xml`.

¿Cómo debo configurar el agente de Kinesis?

Cuanto menor sea el valor de `maxBufferSizeBytes`, más frecuentemente enviará datos el agente de Kinesis. Esto puede ser bueno ya que disminuye el tiempo de entrega de los registros, pero también aumenta las solicitudes por segundo a Kinesis.

¿Por qué el agente de Kinesis envía registros duplicados?

Esto ocurre debido a una mala configuración en el seguimiento de archivos. Asegúrese de que cada `fileFlow's filePattern` solo coincida con un archivo. Esto también puede ocurrir si el modo `logrotate` que se está utilizando está en modo `copytruncate`. Intente cambiar al modo predeterminado o al de creación para evitar la duplicación. Para obtener más información sobre la gestión de registros duplicados, consulte Handling Duplicate Records.

Escritura en flujos de datos de Kinesis mediante otros servicios de AWS

A continuación se muestra una lista de otros servicios de AWS que pueden integrarse directamente con Kinesis Data Streams para escribir datos en Kinesis Data Streams:

Temas

- AWS Amplify
- Amazon Aurora
- Amazon CloudFront
- Registros de Amazon CloudWatch
- Amazon Connect
- AWS Database Migration Service
- Amazon DynamoDB
- Amazon EventBridge
- AWS IoT Core
- Amazon Relational Database Service
- Amazon Pinpoint
- Amazon Quantum Ledger Database

AWS Amplify

Puede utilizar Amazon Kinesis Data Streams para transmitir fácilmente datos desde sus aplicaciones móviles creadas con AWS Amplify para su procesamiento en tiempo real. A continuación,

puede crear paneles de control en tiempo real, capturar excepciones y generar alertas, impulsar recomendaciones y tomar otras decisiones empresariales u operativas en tiempo real. También puede enviar datos fácilmente a otros servicios tales como Simple Amazon Simple Storage Service, Amazon DynamoDB y Amazon Redshift.

Para obtener más información, consulte Uso de Amazon Kinesis en el Centro para desarrolladores de AWS Amplify.

Amazon Aurora

Puede utilizar Amazon Kinesis Data Streams para supervisar las actividades de los clústeres de base de datos de Amazon Aurora. Con Database Activity Streams, el clúster de base de datos de Aurora envía actividades a Amazon Kinesis Data Streams en tiempo real. A continuación, puede crear aplicaciones para la administración de la conformidad que consuman estas actividades, las auditen y generen alertas. También puede utilizar Amazon Firehose para almacenar los datos.

Para obtener más información, consulte Database Activity Streams en la Guía para desarrolladores de Amazon Aurora.

Amazon CloudFront

Puede utilizar Amazon Kinesis Data Streams con registros en tiempo real de CloudFront y obtener información sobre las solicitudes realizadas a una distribución en tiempo real. A continuación, puede crear su propio consumidor de flujo de datos de Kinesis o utilizar Amazon Firehose para enviar los datos de logs a Amazon Simple Storage Service (Amazon S3), Amazon Redshift, Amazon OpenSearch Service (OpenSearch Service) o un servicio de procesamiento de registros de terceros.

Para obtener más información, consulte Registros en tiempo real en la guía del desarrollador de Amazon CloudFront.

Registros de Amazon CloudWatch

Puede utilizar las suscripciones de CloudWatch para obtener acceso a una fuente en tiempo real de eventos de registro de Registros de Amazon CloudWatch y enviarlos a Amazon Kinesis Data Streams para su procesamiento, análisis y carga en otros sistemas.

Para obtener más información, consulte Procesamiento en tiempo real de datos de registro con suscripciones en la Guía del usuario de Registros de Amazon CloudWatch.

Amazon Connect

Puede utilizar Kinesis Data Streams para exportar registros de contactos y eventos de agentes en tiempo real desde su instancia de Amazon Connect. También puede habilitar el flujo de datos desde Perfiles de clientes de Amazon Connect para recibir automáticamente actualizaciones en una transmisión de datos de Kinesis sobre la creación de nuevos perfiles o los cambios en los existentes.

A continuación, puede crear aplicaciones de consumo para procesar y analizar los datos en tiempo real. Por ejemplo, mediante los registros de contactos y los datos de perfiles de clientes, puede mantener actualizados los datos de sus sistemas de origen, como CRM y herramientas de automatización de marketing, con la información más reciente. Con los datos de eventos de los agentes, puede crear cuadros de mando que muestren la información y los eventos de los agentes, y activar notificaciones personalizadas de actividades específicas de los agentes.

Para obtener más información, consulte flujos de datos para su instancia, configuración de la exportación en tiempo real y flujos de eventos de agentes en la Guía del administrador de Amazon Connect.

AWS Database Migration Service

Puede utilizar AWS Database Migration Service para migrar datos a Amazon Kinesis Data Streams. A continuación, puede crear aplicaciones consumidoras que procesen los registros de datos en tiempo real. También puede enviar fácilmente datos a otros servicios como Amazon Simple Storage Service, Amazon DynamoDB y Amazon Redshift.

Para obtener más información, consulte Uso de Kinesis Data Streams en la Guía del usuario de AWS Database Migration Service.

Amazon DynamoDB

Puede utilizar Amazon Kinesis Data Streams para capturar cambios en Amazon DynamoDB. Kinesis Data Streams captura las modificaciones a nivel de elemento en cualquier tabla de DynamoDB y las replica en Kinesis Data Streams. Sus aplicaciones consumidoras pueden obtener acceso a esta transmisión para ver los cambios a nivel de elemento en tiempo real y entregar dichos cambios a los usuarios o tomar medidas en función del contenido.

Para obtener más información, consulte cómo funciona Kinesis Data Streams con DynamoDB en la Guía para desarrolladores de Amazon DynamoDB.

Amazon EventBridge

Con Kinesis Data Streams, puede enviar eventos de llamada a la API de AWS en EventBridge a una transmisión, crear aplicaciones de consumidor y procesar grandes cantidades de datos. También puede utilizar Kinesis Data Streams como destino en las canalizaciones de EventBridge y entregar registros de una transmisión de uno de los orígenes disponibles tras el filtrado y el enriquecimiento opcionales.

Para obtener más información, consulte Envío de eventos a un flujo de Amazon Kinesis y Canalizaciones de EventBridge en la Guía del usuario de Amazon EventBridge.

AWS IoT Core

Puede escribir datos en tiempo real desde mensajes MQTT en el núcleo de AWS IoT mediante acciones de reglas de AWS IoT. A continuación, puede crear aplicaciones que procesen los datos, analicen su contenido y generen alertas, y entregarlos en aplicaciones de análisis u otros servicios de AWS.

Para obtener más información, consulte Kinesis Data Streams en la Guía para desarrolladores de AWS IoT Core.

Amazon Relational Database Service

Puede utilizar Amazon Kinesis Data Streams para supervisar las actividades de sus instancias de Amazon RDS. Con Database Activity Streams, Amazon RDS envía actividades a Amazon Kinesis Data Streams en tiempo real. A continuación, puede crear aplicaciones para la administración de la conformidad que consuman estas actividades, las auditen y generen alertas. También puede utilizar Amazon Firehose para almacenar los datos.

Para obtener más información, consulte Database Activity Streams en la Guía para desarrolladores de Amazon RDS.

Amazon Pinpoint

Puede configurar Amazon Pinpoint para que envíe datos de eventos a Amazon Kinesis Data Streams. Amazon Pinpoint puede enviar datos de eventos para campañas, recorridos y mensajes transaccionales de email y SMS. A continuación, puede ingerir los datos en aplicaciones de análisis o crear sus propias aplicaciones de consumidor que realicen acciones basadas en el contenido de los eventos.

Para obtener más información, consulte Transmisión de eventos en la Guía para desarrolladores de Amazon Pinpoint.

Amazon Quantum Ledger Database

Puede crear una transmisión en QLDB que capture todas las revisiones de documentos consignadas en su diario y entregue estos datos a Amazon Kinesis Data Streams en tiempo real. Un flujo de QLDB es un flujo continuo de datos desde el diario de su libro mayor a un recurso de flujo de datos de Kinesis. A continuación, puede utilizar la plataforma de streaming de Kinesis o Kinesis Client Library para consumir la transmisión, procesar los registros de datos y analizar el contenido de los datos. Un flujo de QLDB escribe sus datos en flujos de datos de Kinesis en tres tipos de registros: `control`, `block summary` y `revision details`.

Para obtener más información, consulte Flujos en la Guía para desarrolladores de Amazon QLDB.

Uso de integraciones de terceros

Puede escribir datos en Kinesis Data Streans mediante una de las siguientes opciones de terceros que se integran con Kinesis Data Streams:

Temas

- Apache Flink
- Fluentd
- Debezium
- Oracle GoldenGate
- Conexión de Kafka
- Adobe Experience
- Striim

Apache Flink

Apache Flink es un marco y motor de procesamiento distribuido popular para computación con estado sobre flujos de datos ilimitados y delimitados. Para obtener más información sobre cómo escribir en Kinesis Data Streams desde Apache Flink, consulte Amazon Kinesis Data Streams Connector.

Fluentd

Fluentd es un recopilador de datos de código abierto para una capa de registro unificada. Para obtener más información sobre cómo escribir en Kinesis Data Streams desde Fluentd. Para obtener más información, consulte Stream processing with Kinesis.

Debezium

Debezium es una plataforma distribuida de código abierto para la captura de datos sobre cambios. Para obtener más información sobre cómo escribir en Kinesis Data Streams desde Debezium, consulte Streaming MySQL Data Changes to Amazon Kinesis.

Oracle GoldenGate

Oracle GoldenGate es un producto de software que le permite replicar, filtrar y transformar datos de una base de datos a otra. Para obtener más información sobre cómo escribir en Kinesis Data Streams desde Oracle GoldenGate, consulte Data replication to Kinesis Data Stream using Oracle GoldenGate.

Conexión de Kafka

Kafka Connect es una herramienta para transmitir datos de forma escalable y fiable entre Apache Kafka y otros sistemas. Para obtener más información sobre cómo escribir datos de Apache Kafka en Kinesis Data Streams, consulte kinesis-kafka-connector.

Adobe Experience

Adobe Experience Platform permite a las organizaciones centralizar y estandarizar los datos de los clientes de cualquier sistema. Luego, aplica la ciencia de datos y el machine learning para mejorar drásticamente el diseño y la entrega de experiencias enriquecidas y personalizadas. Para obtener más información sobre cómo escribir datos de Adobe Experience Platform en Kinesis Data Streams, consulte cómo crear una conexión de Amazon Kinesis.

Striim

Striim es una plataforma en memoria completa e integral para recopilar, filtrar, transformar, enriquecer, agregar, analizar y entregar datos en tiempo real. Para obtener más información sobre cómo escribir datos en Kinesis Data Streams desde Striim, consulte Kinesis Writer.

Solución de problemas de los productores de Amazon Kinesis Data Streams

En las secciones siguientes se ofrecen soluciones a algunos problemas comunes que puede encontrar al usar productores de Amazon Kinesis Data Streams.

- La aplicación productora escribe a una velocidad menor que lo esperado
- Error de permisos no autorizados para la clave maestra de KMS
- Problemas, preguntas e ideas de solución de problemas comunes para los productores

La aplicación productora escribe a una velocidad menor que lo esperado

Los motivos más comunes para que el rendimiento de escritura sea menor que lo esperado son los siguientes.

- Límites de servicio superados
- Optimización de productores

Límites de servicio superados

Para saber si se superan los límites de servicio, compruebe si el productor crea excepciones de rendimiento desde el servicio y valide qué operaciones de la API se están viendo limitadas. Tenga en cuenta que hay diferentes límites en función de la llamada, consulte Cuotas y límites. Por ejemplo, además de los límites de nivel de fragmento para escrituras y lecturas más conocidos, existen los siguientes límites para el nivel de la secuencia:

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

Las operaciones `CreateStream`, `DeleteStream`, `ListStreams`, `GetShardIterator` y `MergeShards` están limitadas a 5 llamadas por segundo. La operación `DescribeStream` está limitada a 10 llamadas por segundo. La operación `DescribeStreamSummary` está limitada a 20 llamadas por segundo.

Si estas llamadas no son el problema, asegúrese de que ha seleccionado una clave de partición que le permita distribuir de manera uniforme las operaciones put en todos los fragmentos, y que no tiene una clave de partición en particular que alcance los límites del servicio sin que el resto lo hagan. Esto requiere que mida los picos de rendimiento y que tenga en cuenta el número de fragmentos de su secuencia. Para obtener más información acerca de cómo administrar secuencias, consulte Creación y administración de secuencias.

> **Tip**
>
> Recuerde que debe redondear al kilobyte más cercano para calcular la limitación controlada del rendimiento al utilizar la operación de un solo registro PutRecord, mientras que la operación multirregistro PutRecords redondea a la suma acumulativa de los registros en cada llamada. Por ejemplo, una solicitud `PutRecords` con 600 registros de 1,1 KB no será objeto de limitación controlada.

Optimización de productores

Antes de empezar a optimizar el productor, hay algunas tareas clave que debe completar. En primer lugar, identifique el rendimiento máximo deseado en términos de tamaño del registro y registros por segundo. A continuación, descarte la capacidad de la secuencia como factor limitante (Límites de servicio superados). Si ha descartado la capacidad de la secuencia, recurra a los siguientes consejos para solucionar problemas y directrices de optimización para los dos tipos comunes de productores.

Productor grande

Un productor grande se suele ejecutar desde un servidor en las instalaciones o una instancia de Amazon EC2. Los clientes que necesitan un mayor rendimiento de un productor grande normalmente se preocupan de la latencia por registro. Entre las estrategias para afrontar los asuntos relacionados con la latencia se incluyen las siguientes: si el cliente puede crear microlotes o almacenar en memoria los registros, use Kinesis Producer Library (que cuenta con lógica de agregación avanzada), la operación multirregistro PutRecords o agregue registros en un archivo de mayor tamaño antes de utilizar la operación para un único registro PutRecord. Si no puede crear

lotes o almacenar en memoria, use varios subprocesos para escribir en un servicio de Kinesis Data Streams al mismo tiempo. El AWS SDK for Java y otros SDK incluyen clientes asíncronos que pueden hacer esto con una cantidad de código muy pequeña.

Productores pequeños

Un productor pequeño suele ser una aplicación móvil, un dispositivo de IoT o un cliente web. Si se trata de una aplicación móvil, le recomendamos que utilice la operación `PutRecords` o el grabador de Kinesis del SDK para móviles de AWS. Para obtener más información, consulte la Guía de introducción a AWS Mobile SDK for Android y la Guía de introducción a AWS Mobile SDK for iOS. Las aplicaciones móviles deben administrar conexiones intermitentes inherentemente y necesitan algún tipo de inserción por lotes, como por ejemplo `PutRecords`. Si no puede crear lotes por algún motivo, consulte la información sobre productores grandes que aparece anteriormente. Si su productor es un navegador, la cantidad de datos generados suele ser muy pequeña. Sin embargo, estará ubicando las operaciones put en la ruta crítica de la aplicación, lo que no es recomendable.

Error de permisos no autorizados para la clave maestra de KMS

Este error se produce cuando una aplicación productora escribe en una secuencia cifrada sin permisos para la clave maestra de KMS. Para asignar permisos a una aplicación de modo que pueda obtener acceso a una clave de KMS, consulte Uso de políticas de claves en AWS KMS y Uso de políticas de IAM con AWS KMS.

Problemas, preguntas e ideas de solución de problemas comunes para los productores

- ¿Por qué mi flujo de datos de Kinesis devuelve un error de servidor interno 500?
- ¿Cómo soluciono los errores de tiempo de espera al escribir desde Flink a Kinesis Data Streams?
- ¿Cómo puedo solucionar los errores de limitación en Kinesis Data Streams?
- ¿Por qué se limita mi flujo de datos de Kinesis?
- ¿Cómo puedo colocar registros de datos en un flujo de datos de Kinesis mediante KPL?

Temas avanzados para productores de Kinesis Data Streams

En esta sección se explica cómo optimizar sus productores de Amazon Kinesis Data Streams.

Temas

Reintentos y limitación de la velocidad en KPL

Cuando agrega registros de usuario de Kinesis Producer Library (KPL) mediante la operación `addUserRecord()` de KPL, cada registro recibe una marca de tiempo y se agrega a un búfer con un plazo fijado por el parámetro de configuración `RecordMaxBufferedTime`. La combinación de la marca temporal y el plazo establece la prioridad del búfer. Los registros se vacían del búfer en función de los siguientes criterios:

- Prioridad del búfer
- Configuración de agregación
- Configuración de recopilación

Los parámetros de configuración de agregación y recopilación que afectan al comportamiento del búfer son los siguientes:

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

A continuación, los registros vaciados se envían al flujo de datos de Kinesis como registros de Amazon Kinesis Data Streams a través de una llamada a la operación `PutRecords` de la API de Kinesis Data Streams. La operación `PutRecords` envía solicitudes a su secuencia que, de vez en cuando, presentan errores totales o parciales. Los registros con errores se agregan automáticamente al búfer de KPL. La nueva fecha límite se establece en función del mínimo de estos dos valores:

- La mitad del valor configurado `RecordMaxBufferedTime` actual
- El valor del tiempo de vida del registro

Esta estrategia permite que los registros de usuario de KPL reintentados se incluyan en las llamadas a la API de Kinesis Data Streams posteriores para mejorar el rendimiento y reducir la complejidad, a la vez que se aplica el valor del tiempo de vida del registro de Kinesis Data Streams. No hay ningún

algoritmo de retardo, por lo que esta estrategia de reintento resulta relativamente agresiva. El envío de spam debido a los reintentos excesivos se evita mediante una limitación de velocidad, que se aborda en la siguiente sección.

Limitación de la velocidad

KPL incluye una característica de limitación de velocidad, que limita el rendimiento por partición enviada desde un único productor. La limitación de velocidad se implementa a través de un algoritmo de bucket de token con buckets independientes para registros y bytes de Kinesis Data Streams. Cada operación de escritura correcta en un flujo de datos de Kinesis agrega un token (o varios) a cada bucket, hasta un límite determinado. Este límite se puede configurar, pero de forma predeterminada se establece un 50 por ciento superior al límite real del fragmento, para permitir la saturación del fragmento desde un único productor.

Puede reducir este límite para disminuir el envío de spam debido a los reintentos excesivos. Sin embargo, es recomendable que cada productor intente por todos los medios alcanzar el máximo rendimiento y que administre las limitaciones controladas resultantes determinadas como excesivas mediante la expansión de la capacidad de la secuencia y la implementación de una estrategia adecuada para la clave de partición.

Consideraciones para el uso de la agregación en KPL

Aunque el esquema de números secuenciales de los registros de Amazon Kinesis Data Streams resultantes sigue siendo el mismo, la agregación hace que la indexación de registros de usuario de Kinesis Producer Library (KPL) contenidos en un registro agregado de Kinesis Data Streams comience en 0 (cero); sin embargo, siempre y cuando no dependa de los números secuenciales para identificar exclusivamente sus registros de usuario de KPL, el código puede pasar por alto esta información, ya que la agregación (de los registros de usuario de KPL en un registro de Kinesis Data Streams) y la desagrupación posterior (del registro de Kinesis Data Streams en registros de usuario de KPL) se encarga de todo eso automáticamente. Esto se aplica independientemente de si el consumidor utiliza KCL o el SDK de AWS. Para utilizar esta funcionalidad de agregación, tendrá que extraer la parte en Java de KPL e insertarla en su compilación si el consumidor se ha escrito con la API proporcionada en el SDK de AWS.

Si va a utilizar números secuenciales como identificadores únicos para sus registros de usuario de KPL, se recomienda que utilice las operaciones persistentes `public int hashCode()` y `public boolean equals(Object obj)` proporcionadas en `Record` y `UserRecord` para permitir la comparación de sus registros de usuario de KPL. Además, si desea examinar el número

subsecuencial de su registro de usuario de KPL, puede transmitirlo a una instancia `UserRecord` y recuperar su número subsecuencial.

Para obtener más información, consulte Desagrupación del consumidor.

Lectura de datos de Amazon Kinesis Data Streams

Un consumidor es una aplicación que procesa todos los datos procedentes de un flujo de datos de Kinesis. Cuando un consumidor utiliza una distribución ramificada mejorada, obtiene su propia asignación de 2 MB/seg de rendimiento de lectura, lo que permite que varios consumidores lean datos del mismo flujo en paralelo, sin competir por el rendimiento de lectura con otros consumidores. Para utilizar la característica de distribución ramificada mejorada de los fragmentos, consulte Desarrollo y uso de consumidores personalizados con rendimiento dedicado (Distribución ramificada mejorada).

De manera predeterminada, los fragmentos en una secuencia proporcionan 2 MB/s de rendimiento de lectura por fragmento. Este rendimiento se comparte entre todos los consumidores que están leyendo en un fragmento determinado. En otras palabras, los 2 MB/s de rendimiento por fragmento son fijos, incluso si hay varios consumidores que leen el fragmento. Para utilizar este rendimiento predeterminado de los fragmentos, consulte Desarrollo de consumidores personalizados con rendimiento compartido.

En la tabla siguiente, se comparan el rendimiento predeterminado y la distribución ramificada mejorada. El retraso de propagación de los mensajes se define como el tiempo en milisegundos que tarda una carga útil enviada mediante las API de despacho de carga útil (como PutRecord y PutRecords) en llegar a la aplicación de consumo a través de las API que consumen carga útil (como y). GetRecords SubscribeToShard

Características	Consumidores no registrados sin distribución ramificada mejorada	Consumidores registrados con distribución ramificada mejorada
Rendimiento de lectura de los fragmentos	Se fija en un total de 2 MB/s por fragmento. Si hay varios consumidores que leen en un mismo fragmento, todos ellos comparten este rendimiento. La suma de rendimientos que reciben desde el fragmento no supera los 2 MB/s.	Se escala a medida que los consumidores se registran para utilizar la distribución ramificad a mejorada. Cada consumidor registrado para usar un despliegu e mejorado recibe su propio rendimiento de lectura por fragmento, hasta 2 MB/seg, independientemente de otros consumidores.

Características	Consumidores no registrados sin distribución ramificada mejorada	Consumidores registrados con distribución ramificada mejorada
Retraso de propagación de mensajes	Un promedio de alrededor de 200 ms si hay un consumido r leyendo la secuencia. Este promedio alcanza aproximad amente los 1000 ms si hay cinco consumidores.	Por lo general, el promedio es de 70 ms si hay un consumidor o cinco consumidores.
Costo	N/A	Existe un costo de recuperación de datos y un costo por hora y fragmento para los consumidores. Para obtener más información, consulte los precios de Amazon Kinesis Data Streams.
Modelo de entrega de registros	Utilice GetRecordsHTTP para sustituir el modelo por HTTP.	Kinesis Data Streams le envía los registros a través de HTTP/2 mediante. SubscribeToShard

Temas

- Uso del visor de datos en la consola de Kinesis
- Consulta de sus flujos de datos en la consola de Kinesis
- Desarrollar consumidores utilizando AWS Lambda
- Desarrollo de consumidores mediante Amazon Managed Service para Apache Flink
- Desarrollando consumidores con Amazon Data Firehose
- Uso de Kinesis Client Library
- Desarrollo de consumidores personalizados con rendimiento compartido
- Desarrollo y uso de consumidores personalizados con rendimiento dedicado (Distribución ramificada mejorada)
- Migración de consumidores de KCL 1.x a KCL 2.x
- Uso de otros servicios de AWS para leer datos de Kinesis Data Streams
- Uso de integraciones de terceros

- Solución de problemas de los consumidores de secuencias de datos de Kinesis
- Temas avanzados para consumidores de Amazon Kinesis Data Streams

Uso del visor de datos en la consola de Kinesis

El visor de datos de la consola de administración de Kinesis le permite ver los registros de datos dentro de la partición especificada de su flujo de datos sin tener que desarrollar una aplicación de consumo. Para utilizar el visor de datos, siga estos pasos:

1. Inicie sesión en la consola Kinesis AWS Management Console y ábrala en https://console.aws.amazon.com/kinesis.
2. Elija el flujo de datos activo cuyos registros desee ver con el visor de datos y, a continuación, elija la pestaña Visor de datos.
3. En la pestaña Visor de datos del flujo de datos activo seleccionado, elija la partición cuyos registros desee ver, elija la posición inicial y, a continuación, haga clic en Obtener registros. Puede establecer la posición de inicio en uno de los siguientes valores:

   - En el número secuencial: muestra los registros desde la posición indicada por el número secuencial especificado en el campo de números secuenciales.
   - Después del número secuencial: muestra los registros después de la posición indicada por el número secuencial especificado en el campo de números secuenciales.
   - En la marca de tiempo: muestra los registros desde la posición indicada por la marca de tiempo especificada en el campo de marca de tiempo.
   - Recortar horizonte: muestra los registros en el último registro sin recortar de la partición, que es el registro de datos más antiguo de esta.
   - Más reciente: muestra los registros justo después del registro más reciente de la partición, de modo que siempre se lean los datos más recientes de la partición.

     Los registros de datos generados que coinciden con el ID de partición y la posición inicial especificados se muestran entonces en una tabla de registros en la consola. Se muestran un máximo de 50 registros a la vez. Para ver el siguiente conjunto de registros, haga clic en el botón Siguiente.
4. Haga clic en cualquier registro individual para ver la carga de ese registro en datos sin procesar o en formato JSON en una ventana independiente.

Tenga en cuenta que al hacer clic en los botones Obtener registros o Siguiente del Visor de datos, se invoca la GetRecordsAPI y se aplica al límite de la GetRecordsAPI de 5 transacciones por segundo.

Consulta de sus flujos de datos en la consola de Kinesis

La pestaña Análisis de datos de la consola de Kinesis Data Streams le permite consultar sus flujos de datos mediante SQL. Para utilizar esta capacidad, siga estos pasos:

1. Inicie sesión en la consola Kinesis AWS Management Console y ábrala en https://console.aws.amazon.com/kinesis.
2. Elija el flujo de datos activo que desee consultar con SQL y, a continuación, elija la pestaña Análisis de datos.
3. En la pestaña Análisis de datos, puede inspeccionar y visualizar el flujo con un bloc de notas gestionado con Apache Flink Studio. Puede realizar consultas SQL ad hoc para inspeccionar su flujo de datos y ver los resultados en cuestión de segundos con Apache Zeppelin. En la pestaña Análisis de datos, selecciona Estoy de acuerdo y, a continuación, selecciona Crear bloc de notas para crear un bloc de notas.
4. Una vez creado el bloc de notas, selecciona Abrir en Apache Zeppelin. Se abrirá el bloc de notas en una pestaña nueva. Un cuaderno es una interfaz interactiva en la que puede enviar sus consultas SQL. Elige la nota que contiene el nombre de tu transmisión.
5. Verás una nota con un ejemplo de `SELECT` consulta para mostrar los datos de la transmisión que ya se está ejecutando. Esto le permite ver el esquema de su flujo de datos.
6. Para probar otras consultas, como ventanas plegables o deslizantes, selecciona Ver consultas de muestra en la pestaña Análisis de datos. Copia la consulta, modifícala para adaptarla a tu esquema de flujo de datos y, a continuación, ejecútala en un nuevo párrafo de tu nota de Zeppelin.

Desarrollar consumidores utilizando AWS Lambda

Puede utilizar una AWS Lambda función para procesar los registros de un flujo de datos. AWS Lambda es un servicio informático que permite ejecutar código sin aprovisionar ni administrar servidores. Ejecuta el código solo cuando es necesario y se escala de manera automática, pasando de pocas solicitudes al día a miles por segundo. Solo paga por el tiempo de computación que consume. No se aplican cargos cuando su código no se está ejecutando. Con AWS Lambdaél, puede ejecutar código para prácticamente cualquier tipo de aplicación o servicio de back-end,

todo sin necesidad de administración. Ejecuta el código en una infraestructura informática de alta disponibilidad y realiza todas las tareas de administración de los recursos informáticos, incluido el mantenimiento del servidor y del sistema operativo, el aprovisionamiento de capacidad y el escalado automático, así como la monitorización del código y las funciones de registro. Para obtener más información, consulte Uso AWS Lambda con Amazon Kinesis.

Para obtener información sobre la resolución de problemas, consulte ¿Por qué el desencadenador de Kinesis Data Streams no puede invocar mi función de Lambda?

Desarrollo de consumidores mediante Amazon Managed Service para Apache Flink

Puede utilizar una aplicación de Amazon Managed Service para Apache Flink a fin de procesar y analizar los datos de un flujo de Kinesis mediante SQL, Java o Scala. Las aplicaciones de Managed Service para Apache Flink pueden enriquecer los datos mediante orígenes de referencia, agregar datos a lo largo del tiempo o utilizar el machine learning para encontrar anomalías en los datos. A continuación, puede escribir los resultados del análisis en otra transmisión de Kinesis, una transmisión de entrega de Firehose o una función de Lambda. Para obtener más información, consulte la Guía para desarrolladores de Managed Service para Apache Flink para aplicaciones de SQL o la Guía para desarrolladores de Managed Service para Apache Flink para aplicaciones de Flink.

Desarrollando consumidores con Amazon Data Firehose

Puede usar una Firehose para leer y procesar registros de una transmisión de Kinesis. Firehose es un servicio totalmente gestionado para entregar datos de streaming en tiempo real a destinos como Amazon S3, Amazon Redshift, OpenSearch Amazon Service y Splunk. Firehose también es compatible con cualquier punto final HTTP personalizado o punto final HTTP propiedad de proveedores de servicios externos compatibles, incluidos Datadog, MongoDB y New Relic. También puede configurar Firehose para transformar sus registros de datos y convertir el formato de registro antes de entregar los datos a su destino. Para obtener más información, consulte Cómo escribir en Firehose mediante Kinesis Data Streams.

Uso de Kinesis Client Library

Uno de los métodos para desarrollar aplicaciones de consumo personalizadas que puedan procesar datos de flujos de datos de KDS consiste en utilizar Kinesis Client Library (KCL).

Temas

> **Note**
>
> Se recomienda actualizar a la última versión tanto KCL 1.x como KCL 2.x, según el escenario de uso. Tanto KCL 1.x como KCL 2.x se actualizan periódicamente con versiones más recientes que incluyen las últimas revisiones de dependencia y seguridad, correcciones de errores y nuevas características compatibles con versiones anteriores. Para obtener más información, consulte https://github.com/awslabs/ amazon-kinesis-client /releases.

¿Qué es Kinesis Client Library?

KCL ayuda a consumir y procesar los datos de un flujo de datos de Kinesis, ya que se encarga de muchas de las tareas complejas asociadas a la computación distribuida. Estas incluyen equilibrar la carga entre varias instancias de aplicaciones de consumo, responder a los errores de las instancias de aplicaciones de consumo, comprobar los registros procesados y reaccionar ante la repartición. KCL se encarga de todas estas subtareas para que pueda centrar sus esfuerzos en escribir una lógica de procesamiento de registros personalizada.

KCL es diferente de las API de Kinesis Data Streams que están disponibles en los SDK de AWS. La API de Kinesis Data Streams le ayuda a administrar numerosos aspectos de Kinesis Data Streams (como la creación de flujos, la repartición y la inserción y obtención de registros). KCL proporciona una capa de abstracción en torno a todas estas subtareas, específicamente para que pueda centrarse en la lógica de procesamiento de datos personalizada de su aplicación de consumo. Para obtener información sobre la API de Kinesis Data Streams, consulte la referencia de la API de Amazon Kinesis.

> **Important**
>
> KCL es una biblioteca de Java. Support para lenguajes distintos de Java se proporciona mediante una interfaz multilingüe llamada. MultiLangDaemon Este daemon está basado en Java y se ejecuta en segundo plano cuando se utiliza un lenguaje de KCL distinto de Java. Por ejemplo, si instala el KCL para Python y escribe su aplicación de consumo completamente en Python, seguirá necesitando instalar Java en su sistema debido a la MultiLangDaemon. Además, MultiLangDaemon tiene algunos ajustes predeterminados que puede que necesites personalizar para tu caso de uso, por ejemplo, la AWS región a la que se conecta. Para obtener más información MultiLangDaemon sobre esto GitHub, consulte el MultiLangDaemon proyecto KCL.

KCL ejerce de intermediaria entre su lógica de procesamiento de registros y Kinesis Data Streams. KCL realiza las siguientes tareas:

- Se conecta al flujo de datos.
- Enumera las particiones del flujo de datos.
- Utiliza los arrendamientos para coordinar las asociaciones de particiones con sus procesos de trabajo.
- Crea instancias de un procesador de registros para cada fragmento que administra
- Extrae registros de datos del flujo de datos.
- Inserta los registros en el procesador de registros correspondiente
- Genera puntos de comprobación para los registros procesados
- Equilibra las asociaciones entre procesos de trabajo y particiones (arrendamientos) cuando cambia el recuento de instancias de procesos de trabajo o cuando se vuelve a realizar la partición del flujo de datos (las particiones se dividen o fusionan).

Versiones disponibles de KCL

Actualmente, puede utilizar cualquiera de las siguientes versiones compatibles de KCL para crear sus aplicaciones de consumo personalizadas:

- KCL 1.x

  Para obtener más información, consulte Desarrollo de consumidores de KCL 1.x

- KCL 2.x

  Para obtener más información, consulte Desarrollo de consumidores de KCL 2.x

Puede usar KCL 1.x o KCL 2.x para crear aplicaciones de consumo que utilicen un rendimiento compartido. Para obtener más información, consulte Desarrollo de consumidores personalizados con rendimiento compartido mediante KCL.

Para crear aplicaciones de consumo que utilicen un rendimiento dedicado (consumidores con distribución mejorada), solo puede utilizar KCL 2.x. Para obtener más información, consulte Desarrollo y uso de consumidores personalizados con rendimiento dedicado (Distribución ramificada mejorada).

Para obtener información sobre las diferencias entre KCL 1.x y KCL 2.x e instrucciones sobre cómo migrar de KCL 1.x a KCL 2.x, consulte Migración de consumidores de KCL 1.x a KCL 2.x.

Conceptos de KCL

- Aplicación para consumidores de KCL: una aplicación creada a medida con KCL y diseñada para leer y procesar registros de flujos de datos.
- Instancia de aplicación de consumo: las aplicaciones de consumo de KCL suelen estar distribuidas y una o más instancias de aplicación se ejecutan simultáneamente para coordinar los fallos y equilibrar la carga de forma dinámica del procesamiento de registro de datos.
- Proceso de trabajo: clase de alto nivel que utiliza una instancia de aplicación de consumo de KCL para empezar a procesar datos.

  > **Important**
  >
  > Cada instancia de aplicación de consumo de KCL tiene un proceso de trabajo.

  El proceso de trabajo inicializa y supervisa diversas tareas, como la sincronización de la información sobre los arrendamientos y las particiones, el seguimiento de las asignaciones de las particiones y el procesamiento de los datos de las particiones. Un proceso de trabajo proporciona a KCL la información de configuración de la aplicación de consumo, como el nombre del flujo de datos cuyos registros de datos va a procesar la aplicación de consumo de KCL y las credenciales de AWS necesarias para acceder a este flujo de datos. El proceso de trabajo también pone en

marcha esa instancia específica de la aplicación de consumo de KCL para entregar los registros de datos del flujo de datos a los procesadores de registros.

> **Important**
>
> En KCL 1.x, esta clase se denomina Proceso de trabajo. Para obtener más información (estos son los repositorios KCL de Java), consulte https://github.com/awslabs/ /blob/ v1.x/src/main/java/com/Amazonaws/services/kinesis/clientlibrary/lib/worker/worker.java. amazon-kinesis-client En KCL 2.x, esta clase se denomina Programador. El propósito del programador en KCL 2.x es idéntico al propósito del proceso de trabajo en KCL 1.x. Para obtener más información sobre la clase Scheduler en KCL 2.x, amazon-kinesis-client consulte amazon-kinesis-client https://github.com/awslabs/ /blob/master/ /src/main/java/ software/amazon/kinesis/coordinator/Scheduler.java.

- Arrendamiento: datos que definen el enlace entre un proceso de trabajo y una partición. Las aplicaciones de consumo distribuidas de KCL utilizan los arrendamientos para dividir el procesamiento de registros de datos entre una flota de procesos de trabajo. En un momento dado, cada partición de registros de datos está vinculada a un proceso de trabajo en particular mediante un arrendamiento identificado por la variable leaseKey.

  De forma predeterminada, un trabajador puede tener uno o más contratos de arrendamiento (sujetos al valor de la variable Worker) al mismo tiempo. maxLeasesFor

  > **Important**
  >
  > Cada proceso de trabajo competirá por tener todos los arrendamientos disponibles para todas las particiones disponibles en un flujo de datos. Sin embargo, solo un proceso de trabajo podrá mantener satisfactoriamente cada arrendamiento a la vez.

  Por ejemplo, si tiene una instancia de aplicación de consumo A con el proceso de trabajo A que procesa un flujo de datos con 4 particiones, el proceso de trabajo A puede retener los arrendamientos de las particiones 1, 2, 3 y 4 al mismo tiempo. Sin embargo, si tiene dos instancias de aplicaciones de consumo: A y B con el proceso de trabajo A y el proceso de trabajo B, y estas instancias procesan un flujo de datos con 4 particiones, el proceso de trabajo A y el proceso de trabajo B no pueden retener el arrendamiento de la partición 1 al mismo tiempo. Un proceso de trabajo retiene el arrendamiento de una partición concreta hasta que esté listo para dejar de

procesar los registros de datos de esta partición o hasta que falle. Cuando un proceso de trabajo deja de ser titular del arrendamiento, otro proceso de trabajo lo acepta y lo retiene.

Para obtener más información (estos son los repositorios KCL de Java), consulte https://github.com/awslabs/ amazon-kinesis-client /blob/v1.x/src/main/java/com/Amazonaws/services/kinesis/leases/impl/lease.java para KCL 1.x y https://github.com/awslabs/ /blob/master/ /src/main/java/software/amazon/kinesis/leases/Lease.java para KCL 2.x. amazon-kinesis-client amazon-kinesis-client

- Tabla de arrendamiento: una tabla exclusiva de Amazon DynamoDB que se utiliza para realizar un seguimiento de las particiones de un flujo de datos de KDS que los procesos de trabajo de la aplicación de consumo de KCL están arrendando y procesando. La tabla de arrendamiento debe permanecer sincronizada (dentro de un proceso de trabajo y entre todos los procesos de trabajo) con la información más reciente sobre las particiones del flujo de datos mientras se ejecuta la aplicación de consumo de KCL. Para obtener más información, consulte Uso de una tabla de arrendamiento para realizar el seguimiento de las particiones procesadas por la aplicación de consumo de KCL.
- Procesador de registros: lógica que define la forma en que su aplicación de consumo de KCL procesa los datos que obtiene de los flujos de datos. En tiempo de ejecución, una instancia de una aplicación de consumo de KCL crea una instancia de un proceso de trabajo, y este proceso de trabajo crea una instancia de un procesador de registros por cada partición que tiene en arrendamiento.

Uso de una tabla de arrendamiento para realizar el seguimiento de las particiones procesadas por la aplicación de consumo de KCL

Temas

Qué es una tabla de arrendamiento

Para cada aplicación de Amazon Kinesis Data Streams, KCL utiliza una tabla de arrendamiento única (almacenada en una tabla de Amazon DynamoDB) para realizar un seguimiento de las particiones

de un flujo de datos de KDS que los procesos de trabajo de la aplicación de consumo de KCL están arrendando y procesando.

> **Important**
>
> KCL utiliza el nombre de la aplicación de consumo para crear el nombre de la tabla de arrendamiento que utiliza esta aplicación de consumo, por lo que el nombre de cada aplicación de consumo debe ser único.

Puede consultar la tabla con la consola de Amazon DynamoDB mientras se ejecuta la aplicación de consumo.

Si la tabla de arrendamiento de la aplicación de consumo de KCL no existe cuando se inicia la aplicación, uno de los procesos de trabajo crea la tabla de arrendamiento para esta aplicación.

> **Important**
>
> Se le realizará el cobro de los costos de su cuenta asociados a la tabla de DynamoDB, además de los costos propios asociados a Kinesis Data Streams.

Cada fila de la tabla de arrendamiento representa una partición que procesan los procesos de trabajo de la aplicación de consumo. Si la aplicación de consumo de KCL procesa solo un flujo de datos, la `leaseKey` que es la clave hash de la tabla de arrendamiento será el identificador de la partición. Si es Procesamiento de varios flujos de datos con el mismo KCL 2.x para aplicaciones de consumo de Java, la estructura de leaseKey tendrá el siguiente aspecto: `account-id:StreamName:streamCreationTimestamp:ShardId`. Por ejemplo, `111111111:multiStreamTest-1:12345:shardId-000000000336`.

Además de la ID del fragmento, cada fila incluye también los siguientes datos:

- checkpoint: el número secuencial de punto de comprobación más reciente del fragmento. Este valor es único en todas las particiones del flujo de datos.
- checkpointSubSequenceNúmero: cuando se utiliza la función de agregación de la biblioteca de productores de Kinesis, se trata de una extensión del punto de control que rastrea los registros de los usuarios individuales dentro del registro de Kinesis.

- leaseCounter: se utiliza para el control de versiones de las asignaciones, de modo que los procesos de trabajo puedan detectar que su asignación ha sido utilizada por otro proceso de trabajo.
- leaseKey: un identificador único para una asignación. Cada arrendamiento es específico de una partición del flujo de datos y solo lo retiene un proceso de trabajo cada vez.
- leaseOwner: el proceso de trabajo que tiene esta asignación.
- ownerSwitchesSincePunto de control: cuántas veces este contrato de arrendamiento ha cambiado de trabajadores desde la última vez que se emitió un punto de control.
- parentShardId: Se utiliza para garantizar que el fragmento principal se procese por completo antes de que comience el procesamiento en los fragmentos secundarios. Así, se garantiza que los registros se procesen en el mismo orden en el que se introdujeron en la secuencia.
- hashrange: lo utiliza `PeriodicShardSyncManager` para ejecutar sincronizaciones periódicas para encontrar las particiones que faltan en la tabla de arrendamiento y crear arrendamientos para ellas si es necesario.

  > **Note**
  >
  > Estos datos están presentes en la tabla de arrendamiento de todas las particiones a partir de KCL 1.14 y KCL 2.3. Para obtener más información sobre `PeriodicShardSyncManager` y la sincronización periódica entre los arrendamientos y las particiones, consulte Cómo se sincroniza una tabla de arrendamiento con las particiones de un flujo de datos de KDS.

- childshards: lo utiliza `LeaseCleanupManager` para revisar el estado de procesamiento de la partición secundaria y decidir si la partición principal se puede eliminar de la tabla de arrendamiento.

  > **Note**
  >
  > Estos datos están presentes en la tabla de arrendamiento de todas las particiones a partir de KCL 1.14 y KCL 2.3.

- shardID: ID de la partición.

> **Note**
>
> Estos datos solo están presentes en la tabla de arrendamiento si es Procesamiento de varios flujos de datos con el mismo KCL 2.x para aplicaciones de consumo de Java. Esto solo se admite en KCL 2.x para Java, a partir de KCL 2.3 para Java y versiones posteriores.

- nombre del flujo El identificador del flujo de datos en el siguiente formato: `account-id:StreamName:streamCreationTimestamp`.

> **Note**
>
> Estos datos solo están presentes en la tabla de arrendamiento si se dedica al Procesamiento de varios flujos de datos con el mismo KCL 2.x para aplicaciones de consumo de Java. Esto solo se admite en KCL 2.x para Java, a partir de KCL 2.3 para Java y versiones posteriores.

Rendimiento

Si la aplicación de Amazon Kinesis Data Streams recibe excepciones de rendimiento aprovisionado, debe aumentar el rendimiento aprovisionado para la tabla de DynamoDB. KCL crea la tabla con un rendimiento aprovisionado de 10 lecturas por segundo y 10 escrituras por segundo, pero esto podría no ser suficiente para su aplicación. Por ejemplo, si su aplicación de Amazon Kinesis Data Streams crea frecuentemente puntos de comprobación u opera en un flujo que se compone de muchas particiones, es posible que necesite más rendimiento.

Para obtener información sobre el rendimiento aprovisionado en DynamoDB, consulte Modo de capacidad de lectura y escritura y Uso de tablas y datos en la Guía para desarrolladores de Amazon DynamoDB.

Cómo se sincroniza una tabla de arrendamiento con las particiones de un flujo de datos de KDS

Los procesos de trabajo de las aplicaciones de consumo de KCL utilizan los arrendamientos para procesar particiones de un flujo de datos determinado. La información sobre qué proceso de trabajo está arrendando cada partición en un momento dado se almacena en una tabla de arrendamiento. La tabla de arrendamiento debe permanecer sincronizada con la información más

reciente sobre la partición del flujo de datos mientras se ejecuta la aplicación de consumo de KCL. KCL sincroniza la tabla de arrendamiento con la información de las particiones obtenida del servicio Kinesis Data Streams durante el arranque de la aplicación de consumo (ya sea al inicializar o reiniciar la aplicación de consumo) y también siempre que una partición que se esté procesando llegue a su fin (repartición). En otras palabras, los procesos de trabajo o una aplicación de consumo de KCL se sincronizan con el flujo de datos que están procesando durante el arranque inicial de la aplicación de consumo y siempre que la aplicación de consumo encuentra un evento de repartición del flujo de datos.

Temas

Sincronización en KCL 1.0 a 1.13 y KCL 2.0 a 2.2

En KCL 1.0 a 1.13 y KCL 2.0 a 2.2, durante el arranque de la aplicación de consumo y también durante cada evento de repartición del flujo de datos, KCL sincroniza la tabla de arrendamiento con la información de las particiones adquirida en el servicio Kinesis Data Streams invocando las `ListShards` o API de descubrimiento de `DescribeStream`. En todas las versiones de KCL enumeradas anteriormente, cada proceso de trabajo de una aplicación de consumo de KCL completa los siguientes pasos para realizar el proceso de sincronización entre arrendamientos y particiones durante el arranque de la aplicación de consumo y en cada evento de repartición del flujo:

- Obtiene todos las particiones de datos del flujo que se está procesando.
- Obtiene todos los arrendamientos de particiones de la tabla de arrendamiento.
- Filtra cada partición abierta que no tenga arrendamientos en la tabla de arrendamiento.
- Repite todas las particiones abiertas encontradas y para cada partición abierta sin un elemento principal abierto:
  - Recorre el árbol jerárquico siguiendo la ruta de sus antecesores para determinar si la partición es descendiente. Una partición se considera descendiente si se está procesando una partición anterior (en la tabla de arrendamiento se indica el arrendamiento de la partición anterior) o si se debe procesar una partición anterior (por ejemplo, si la posición inicial es `TRIM_HORIZON` o `AT_TIMESTAMP`).
  - Si la partición abierta en el contexto es descendiente, KCL comprueba la partición en función de su posición inicial y crea arrendamientos para sus elementos principales, si es necesario.

Sincronización en KCL 2.x, a partir de KCL 2.3 y versiones posteriores

A partir de las últimas versiones compatibles de KCL 2.x (KCL 2.3) y versiones posteriores, la biblioteca ahora admite los siguientes cambios en el proceso de sincronización. Estos cambios en la sincronización entre arrendamientos y particiones reducen significativamente el número de llamadas a la API que realizan las aplicaciones de consumo de KCL al servicio Kinesis Data Streams y optimizan la administración de arrendamientos en su aplicación de consumo de KCL.

- Durante el arranque de la aplicación, si la tabla de arrendamiento está vacía, KCL utiliza la opción de filtrado de la API `ListShard` (el parámetro de solicitud `ShardFilter` opcional) para recuperar y crear arrendamientos únicamente para una instantánea de las particiones abiertas en el momento especificado por el parámetro `ShardFilter`. El parámetro `ShardFilter` permite filtrar la respuesta de la API `ListShards`. La única propiedad obligatoria del parámetro `ShardFilter` es `Type`. KCL utiliza la propiedad de filtro `Type` y los siguientes valores válidos para identificar y devolver una instantánea de las particiones abiertas que podrían requerir nuevos arrendamientos:
  - `AT_TRIM_HORIZON`: la respuesta incluye todas las particiones que estaban abiertas en `TRIM_HORIZON`.
  - `AT_LATEST`: la respuesta incluye solo las particiones actualmente abiertas del flujo de datos.
  - `AT_TIMESTAMP`: la respuesta incluye todas las particiones cuya marca de tiempo de inicio es anterior o igual a la marca de tiempo dada y cuya marca de tiempo de finalización es posterior o igual que la marca de tiempo dada, o que aún están abiertas.

  `ShardFilter` se utiliza al crear arrendamientos para una tabla de arrendamiento vacía con el fin de inicializar los arrendamientos de una instantánea de las particiones especificadas en `RetrievalConfig#initialPositionInStreamExtended`.

  Para obtener más información acerca de `ShardFilter`, consulte https://docs.aws.amazon.com/kinesis/latest/APIReference/API_ShardFilter.html.
- En lugar de que todos los procesos de trabajo realicen la sincronización entre arrendamientos y particiones para mantener la tabla de arrendamiento actualizada con las particiones más recientes del flujo de datos, un único líder del proceso de trabajo elegido realiza la sincronización entre arrendamientos y particiones.
- KCL 2.3 utiliza el parámetro de devolución `ChildShards` de las API `GetRecords` y `SubscribeToShard` para realizar la sincronización entre arrendamientos y particiones que se produce en `SHARD_END` para las particiones cerradas, lo que permite a un proceso de trabajo de KCL crear arrendamientos solo para las particiones secundarias de la partición que ha

terminado de procesar. Para compartirla entre aplicaciones de consumo, esta optimización de la sincronización entre arrendamientos y particiones utiliza el parámetro `ChildShards` de la API `GetRecords`. En el caso de las aplicaciones de consumo dedicadas al rendimiento (distribución mejorada), esta optimización de la sincronización entre arrendamientos y particiones utiliza el parámetro `ChildShards` de la API `SubscribeToShard`. Para obtener más información, consulte GetRecordsSubscribeToShards, y. ChildShard

- Con los cambios anteriores, el comportamiento de KCL está pasando del modelo en el que todos los procesos de trabajo aprenden sobre todas las particiones existentes al modelo en el que los procesos de trabajo aprenden solo sobre las particiones secundarias de las particiones que son propiedad de cada proceso de trabajo. Por lo tanto, además de la sincronización que se produce durante el arranque de las aplicaciones de consumo y los eventos de repartición, KCL ahora también realiza exámenes periódicos adicionales de las particiones o arrendamientos para identificar cualquier posible laguna en la tabla de arrendamiento (en otras palabras, para obtener información sobre todas las particiones nuevas) a fin de garantizar que se procese todo el rango de hash del flujo de datos y crear arrendamientos para ellos si es necesario. `PeriodicShardSyncManager` es el componente responsable de ejecutar exámenes periódicos de arrendamientos o particiones.

  Para obtener más información sobre `PeriodicShardSyncManager` KCL 2.3, consulte https:// github.com/awslabs/ amazon-kinesis-client /blob/master/ /src/main/java/software/amazon/kinesis/ leases/ amazon-kinesis-client .java #L201 -L213. LeaseManagementConfig

  En KCL 2.3, hay nuevas opciones de configuración disponibles para configurar `PeriodicShardSyncManager` en `LeaseManagementConfig`:

Name	Valor predeterminado	Descripción
leasesRecoveryAuditorExecutionFrequencyMillis	120 000 (2 minutos)	Frecuencia (en milisegundos) del trabajo del auditor para buscar arrendamientos parciales en la tabla de arrendamiento. Si el auditor

Name	Valor predeterminado	Descripción
		detecta lagunas en los arrendamientos de un flujo, activará la sincronización de las particiones basándose en `leasesRecoveryAuditorInconsistencyConfidenceThreshold` .

Name	Valor predeterminado	Descripción
leasesRecoveryAuditorInconsistencyConfidenceThreshold	3	El umbral de confianza para el trabajo de auditor periódico permite determinar si los arrendamientos de un flujo de datos de la tabla de arrendamiento son incoherentes. Si el auditor encuentra varias veces el mismo conjunto de incoherencias consecutivas en un flujo de datos, se activará una sincronización de particiones.

CloudWatch Ahora también se emiten nuevas métricas para monitorear el estado del. `PeriodicShardSyncManager` Para obtener más información, consulte PeriodicShardSyncManager.

- Incluye una optimización de `HierarchicalShardSyncer` para crear solo arrendamientos para una capa de particiones.

Sincronización en KCL 1.x, a partir de KCL 1.14 y versiones posteriores

A partir de las últimas versiones compatibles de KCL 1.x (KCL 1.14) y versiones posteriores, la biblioteca ahora admite los siguientes cambios en el proceso de sincronización. Estos cambios

en la sincronización entre arrendamientos y particiones reducen significativamente el número de llamadas a la API que realizan las aplicaciones de consumo de KCL al servicio Kinesis Data Streams y optimizan la administración de arrendamientos en su aplicación de consumo de KCL.

- Durante el arranque de la aplicación, si la tabla de arrendamiento está vacía, KCL utiliza la opción de filtrado de la API `ListShard` (el parámetro de solicitud `ShardFilter` opcional) para recuperar y crear arrendamientos únicamente para una instantánea de las particiones abiertas en el momento especificado por el parámetro `ShardFilter`. El parámetro `ShardFilter` permite filtrar la respuesta de la API `ListShards`. La única propiedad obligatoria del parámetro `ShardFilter` es `Type`. KCL utiliza la propiedad de filtro `Type` y los siguientes valores válidos para identificar y devolver una instantánea de las particiones abiertas que podrían requerir nuevos arrendamientos:
  - `AT_TRIM_HORIZON`: la respuesta incluye todas las particiones que estaban abiertas en `TRIM_HORIZON`.
  - `AT_LATEST`: la respuesta incluye solo las particiones actualmente abiertas del flujo de datos.
  - `AT_TIMESTAMP`: la respuesta incluye todas las particiones cuya marca de tiempo de inicio es anterior o igual a la marca de tiempo dada y cuya marca de tiempo de finalización es posterior o igual que la marca de tiempo dada, o que aún están abiertas.

  `ShardFilter` se utiliza al crear arrendamientos para una tabla de arrendamiento vacía con el fin de inicializar los arrendamientos de una instantánea de las particiones especificadas en `KinesisClientLibConfiguration#initialPositionInStreamExtended`.

  Para obtener más información acerca de `ShardFilter`, consulte https://docs.aws.amazon.com/kinesis/latest/APIReference/API_ShardFilter.html.
- En lugar de que todos los procesos de trabajo realicen la sincronización entre arrendamientos y particiones para mantener la tabla de arrendamiento actualizada con las particiones más recientes del flujo de datos, un único líder del proceso de trabajo elegido realiza la sincronización entre arrendamientos y particiones.
- KCL 1.14 utiliza el parámetro de devolución `ChildShards` de las API `GetRecords` y `SubscribeToShard` para realizar la sincronización entre arrendamientos y particiones que se produce en `SHARD_END` para las particiones cerradas, lo que permite a un proceso de trabajo de KCL crear arrendamientos solo para las particiones secundarias de la partición que ha terminado de procesar. Para obtener más información, consulte GetRecords y ChildShard.
- Con los cambios anteriores, el comportamiento de KCL está pasando del modelo en el que todos los procesos de trabajo aprenden sobre todas las particiones existentes al modelo en el que

los procesos de trabajo aprenden solo sobre las particiones secundarias de las particiones que son propiedad de cada proceso de trabajo. Por lo tanto, además de la sincronización que se produce durante el arranque de las aplicaciones de consumo y los eventos de repartición, KCL ahora también realiza exámenes periódicos adicionales de las particiones o arrendamientos para identificar cualquier posible laguna en la tabla de arrendamiento (en otras palabras, para obtener información sobre todas las particiones nuevas) a fin de garantizar que se procese todo el rango de hash del flujo de datos y crear arrendamientos para ellos si es necesario. `PeriodicShardSyncManager` es el componente responsable de ejecutar exámenes periódicos de arrendamientos o particiones.

Cuando `KinesisClientLibConfiguration#shardSyncStrategyType` está establecido en `ShardSyncStrategyType.SHARD_END`, `PeriodicShardSync leasesRecoveryAuditorInconsistencyConfidenceThreshold` se utiliza para determinar el umbral del número de exámenes consecutivos que contienen lagunas en la tabla de arrendamiento, tras lo cual se exige la sincronización de las particiones. Cuando `KinesisClientLibConfiguration#shardSyncStrategyType` se establece en `ShardSyncStrategyType.PERIODIC`, `leasesRecoveryAuditorInconsistencyConfidenceThreshold` se ignora.

Para obtener más información sobre `PeriodicShardSyncManager` la versión 1.14 de KCL, consulte https://github.com/awslabs/ amazon-kinesis-client KinesisClientLibConfiguration /blob/ v1.x/src/main/java/com/amazonaws/services/kinesis/clientlibrary/lib/worker/ .java #L987 -L999.

En KCL 1.14, hay una nueva opción de configuración disponible para configurar `PeriodicShardSyncManager` en `LeaseManagementConfig`:

Name	Valor predeterminado	Descripción
leasesRecoveryAuditorInconsistencyConfidenceThreshold	3	El umbral de confianza para el trabajo de auditor periódico permite determinar si los arrendamientos de un flujo de datos de la tabla de arrendamiento son incoherentes. Si el auditor encuentra varias veces el mismo conjunto de incoherencias consecutivas en un flujo de datos, se activará una sincronización de particiones.

CloudWatch Ahora también se emiten nuevas métricas para monitorear el estado del. `PeriodicShardSyncManager` Para obtener más información, consulte PeriodicShardSyncManager.

- KCL 1.14 ahora también admite la limpieza de arrendamientos diferidos. `LeaseCleanupManager` elimina los arrendamientos de forma asíncrona al llegar a `SHARD_END`, cuando una partición ha caducado pasado el periodo de retención del flujo de datos o se ha cerrado como resultado de una operación de repartición.

  Están disponibles nuevas opciones de configuración para `LeaseCleanupManager`.

Name	Valor predeterminado	Descripción
leaseCleanupIntervalMillis	1 minuto	Intervalo en el que se ejecuta el subproceso de limpieza del arrendamiento.
completedLeaseCleanupIntervalMillis	5 minutos	Intervalo para comprobar si un arrendamiento se ha completado o no.
garbageLeaseCleanupIntervalMillis	30 minutos	Intervalo en el que se comprueba si un arrendamiento es un elemento no utilizado (es decir, si ha superado el periodo de retención del flujo de datos) o no.

- Incluye una optimización de `KinesisShardSyncer` para crear solo arrendamientos para una capa de particiones.

Procesamiento de varios flujos de datos con el mismo KCL 2.x para aplicaciones de consumo de Java

En esta sección se describen los siguientes cambios en KCL 2.x para Java, que permiten crear aplicaciones de consumo de KCL que pueden procesar más de un flujo de datos al mismo tiempo.

> **Important**
>
> El procesamiento de varios flujos solo se admite en KCL 2.x para Java, a partir de KCL 2.3 para Java y versiones posteriores.
> El procesamiento de varios flujos NO es compatible con ningún otro lenguaje en el que se pueda implementar KCL 2.x.
> El procesamiento de varios flujos NO es compatible con ninguna versión de KCL 1.x.

- MultistreamTracker interfaz

  Para crear una aplicación de consumo que pueda procesar múltiples transmisiones al mismo tiempo, debe implementar una nueva interfaz llamada MultistreamTracker. Esta interfaz incluye el método `streamConfigList` que devuelve la lista de flujos de datos y sus configuraciones para que los procese la aplicación de consumo de KCL. Tenga en cuenta que los flujos de datos que se procesan pueden cambiar durante el tiempo de ejecución de la aplicación de consumo. KCL llama a `streamConfigList` periódicamente para obtener información sobre los cambios en los flujos de datos que se van a procesar.

  El `streamConfigList` método rellena la StreamConfiglista.

```
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;
}
```

  Tenga en cuenta que los campos `StreamIdentifier` y `InitialPositionInStreamExtended` son obligatorios, aunque `consumerArn` es opcional.

Debe proporcionar `consumerArn` únicamente si utiliza KCL 2.x para implementar una aplicación de consumo con distribución mejorada.

Para obtener más información`StreamIdentifier`, consulte https://github.com/awslabs/amazon-kinesis-client /blob/0c5042dadf794fe988438436252a5a8fe70b6b0b/ amazon-kinesis-client /src/main/java/software/amazon/kinesis/common/ .java #L29. StreamIdentifier Puede crear una instancia de varios flujos para `StreamIdentifier` a partir del identificador de flujo serializado. El identificador de flujo serializado debe tener el siguiente formato: `account-id:StreamName:streamCreationTimestamp`.

```
 * @param streamIdentifierSer
    * @return StreamIdentifier
    */
   public static StreamIdentifier multiStreamInstance(String streamIdentifierSer) {
       if (PATTERN.matcher(streamIdentifierSer).matches()) {
           final String[] split = streamIdentifierSer.split(DELIMITER);
           return new StreamIdentifier(split[0], split[1],
Long.parseLong(split[2]));
       } else {
           throw new IllegalArgumentException("Unable to deserialize
StreamIdentifier from " + streamIdentifierSer);
       }
   }
```

`MultistreamTracker` también incluye una estrategia para eliminar los arrendamientos de flujos antiguos en la tabla de arrendamiento (`formerStreamsLeasesDeletionStrategy`). Tenga en cuenta que la estrategia NO SE PUEDE cambiar durante el tiempo de ejecución de la aplicación de consumo. Para obtener más información, consulta https://github.com/awslabs/ amazon-kinesis-client amazon-kinesis-client /blob/0c5042dadf794fe988438436252a5a8fe70b6b0b/ /src/main/java/software/amazon/kinesis/processor/ .java FormerStreamsLeasesDeletionStrategy

- ConfigsBuilderes una clase que abarca toda la aplicación y que se puede utilizar para especificar todos los ajustes de configuración de KCL 2.x que se utilizarán al crear una aplicación de consumo de KCL. `ConfigsBuilder`la clase ahora es compatible con la interfaz. `MultistreamTracker` Puede inicializar ConfigsBuilder cualquiera de las dos con el nombre del flujo de datos desde el que se van a consumir los registros:

```
/**
    * 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;
   }
```

O bien, puede inicializarlo ConfigsBuilder `MultiStreamTracker` si desea implementar una aplicación de consumo de KCL que procese varios flujos al mismo tiempo.

```
* Constructor to initialize ConfigsBuilder with MultiStreamTracker
    * @param multiStreamTracker
    * @param applicationName
    * @param kinesisClient
    * @param dynamoDBClient
    * @param cloudWatchClient
    * @param workerIdentifier
    * @param shardRecordProcessorFactory
    */
   public ConfigsBuilder(@NonNull MultiStreamTracker multiStreamTracker, @NonNull
String applicationName,
           @NonNull KinesisAsyncClient kinesisClient, @NonNull DynamoDbAsyncClient
dynamoDBClient,
```

```
            @NonNull CloudWatchAsyncClient cloudWatchClient, @NonNull String
workerIdentifier,
            @NonNull ShardRecordProcessorFactory shardRecordProcessorFactory) {
        this.appStreamTracker = Either.left(multiStreamTracker);
        this.applicationName = applicationName;
        this.kinesisClient = kinesisClient;
        this.dynamoDBClient = dynamoDBClient;
        this.cloudWatchClient = cloudWatchClient;
        this.workerIdentifier = workerIdentifier;
        this.shardRecordProcessorFactory = shardRecordProcessorFactory;
    }
```

- Con la compatibilidad con varios flujos implementada para la aplicación de consumo de KCL, cada fila de la tabla de arrendamiento de la aplicación ahora contiene el ID de la partición y el nombre del flujo de los varios flujos de datos que procesa esta aplicación.
- Cuando se implementa la compatibilidad con varios flujos para la aplicación de consumo de KCL, leaseKey adopta la siguiente estructura: `account-id:StreamName:streamCreationTimestamp:ShardId`. Por ejemplo, `111111111:multiStreamTest-1:12345:shardId-000000000336`.

> **Important**
>
> Cuando la aplicación de consumo de KCL existente está configurada para procesar solo un flujo de datos, leaseKey (que es la clave hash de la tabla de arrendamiento) es el ID de la partición. Si vuelve a configurar esta aplicación de consumo de KCL existente para procesar varios flujos de datos, se rompe la tabla de arrendamiento, ya que con la compatibilidad con varios flujos, la estructura de leaseKey debe ser la siguiente: `account-id:StreamName:StreamCreationTimestamp:ShardId`.

Uso de Kinesis Client Library con AWS Glue Schema Registry

Puede integrar los flujos de datos de Kinesis con AWS Glue Schema Registry. AWS Glue Schema Registry le permite descubrir, controlar y evolucionar de forma centralizada esquemas, además de garantizar que un esquema registrado valide de forma continua los datos generados. Un esquema define la estructura y el formato de un registro de datos. Un esquema es una especificación versionada para publicación, consumo o almacenamiento de confianza de datos. El registro de esquemas de AWS Glue le permite mejorar la end-to-end calidad y el gobierno de los datos en sus

aplicaciones de streaming. Para obtener más información, consulte AWS Glue Schema Registry. Una de las formas de configurar esta integración es a través de KCL en Java.

> **Important**
>
> Actualmente, la integración de Kinesis Data Streams y AWS Glue Schema Registry solo se admite para las instancias de Kinesis Data Streams que utilizan aplicaciones de consumo de KCL 2.3 implementadas en Java. No se proporciona soporte multilingüe. No se admiten las aplicaciones de consumo de KCL 1.0. No se admiten las aplicaciones de consumo de KCL 2.x anteriores a KCL 2.3.

Para obtener instrucciones detalladas sobre cómo configurar la integración de Kinesis Data Streams con Schema Registry mediante KCL, consulte la sección “Interacción con los datos mediante las bibliotecas de KPL/KCL” en Caso de uso: integración de Amazon Kinesis Data Streams con AWS Glue Schema Registry.

Desarrollo de consumidores personalizados con rendimiento compartido

Si no necesita un rendimiento específico al recibir datos de Kinesis Data Streams y tampoco necesita demoras de propagación de lectura inferiores a 200 ms, puede crear aplicaciones consumidoras tal y como se describe en los temas siguientes.

Temas

- Desarrollo de consumidores personalizados con rendimiento compartido mediante KCL
- Desarrollo de consumidores personalizados con rendimiento compartido con AWS SDK for Java

Para obtener más información sobre cómo crear consumidores que puedan recibir registros de flujos de datos de Kinesis con un rendimiento específico, consulte Desarrollo y uso de consumidores personalizados con rendimiento dedicado (Distribución ramificada mejorada).

Desarrollo de consumidores personalizados con rendimiento compartido mediante KCL

Uno de los métodos para desarrollar una aplicación de consumo personalizada con un rendimiento compartido consiste en utilizar Kinesis Client Library (KCL).

Temas

Desarrollo de consumidores de KCL 1.x

Puede desarrollar una aplicación de consumo para Amazon Kinesis Data Streams mediante Kinesis Client Library (KCL). Para obtener más información acerca de, consulte ¿Qué es Kinesis Client Library? .

Contenido

Desarrollo de un consumidor de Kinesis Client Library en Java

Puede utilizar Kinesis Client Library (KCL) para crear aplicaciones que procesen datos de los flujos de datos de Kinesis. Kinesis Client Library está disponible en varios idiomas. En este tema se habla de Java. Para ver la referencia de Javadoc, consulte el tema de AWSJavadoc para Class. AmazonKinesisClient

Para descargar el KCL de Java GitHub, vaya a la biblioteca de clientes de Kinesis (Java). Para localizar KCL para Java en Apache Maven, vaya a la página de resultados de la búsqueda de KCL. Para descargar un código de muestra para una aplicación de consumo de Java KCL desde GitHub, visite la página del proyecto de ejemplo de KCL para Java en. GitHub

La aplicación de muestra utiliza Apache Commons Logging. Puede cambiar la configuración de registro en el método estático `configure` definido en el archivo `AmazonKinesisApplicationSample.java`. Para más información acerca de cómo utilizar Apache Commons Logging con Log4j y aplicaciones de Java en AWS, consulte Logging with Log4j en la Guía para desarrolladores de AWS SDK for Java.

Debe completar las siguientes tareas a la hora de implementar una aplicación de consumo de KCL en Java:

Tareas

- Implemente los métodos I. RecordProcessor
- Implemente una fábrica de clases para la RecordProcessor interfaz I
- Creación de un proceso de trabajo
- Modificación de las propiedades de configuración
- Migración a la versión 2 de la interfaz del procesador de registros

Implemente los métodos I. RecordProcessor

KCL es compatible actualmente con dos versiones de la interfaz de `IRecordProcessor`: la interfaz original está disponible con la primera versión de KCL y la versión 2 está disponible a partir de la versión 1.5.0 de KCL. Ambas interfaces son totalmente compatibles. La elección dependerá de su situación específica. Consulte sus javadocs locales o el código fuente para ver todas las diferencias. En las siguientes secciones se describe la implementación mínima introductoria.

RecordProcessor Versiones I.

- Interfaz original (versión 1)
- Interfaz actualizada (versión 2)

Interfaz original (versión 1)

La interfaz original `IRecordProcessor` (`package com.amazonaws.services.kinesis.clientlibrary.interfaces`) expone los siguientes métodos del procesador de registros que el consumidor debe implementar. En la muestra se presentan implementaciones que puede utilizar como punto de partida (consulte `AmazonKinesisApplicationSampleRecordProcessor.java`).

```
public void initialize(String shardId)
public void processRecords(List<Record> records, IRecordProcessorCheckpointer
 checkpointer)
public void shutdown(IRecordProcessorCheckpointer checkpointer, ShutdownReason reason)
```

initialize

KCL llama al método `initialize` cuando se crea una instancia del procesador de registros y pasa un ID de partición específico como parámetro. Este procesador de registros procesa solo

este fragmento, y normalmente también se produce la situación contraria (este fragmento solo es procesado por este procesador de registros). Sin embargo, el consumidor debe contar con la posibilidad de que un registro de datos pueda ser procesado más de una vez. Kinesis Data Streams tiene una semántica de al menos una vez, lo que significa que cada registro de datos de una partición se procesa al menos una vez por parte de un proceso de trabajo del consumidor. Para obtener más información sobre los casos en los que un fragmento en particular puede ser procesado por más de un proceso de trabajo, consulte Cambio en los fragmentos, escalado y procesamiento paralelo.

```
public void initialize(String shardId)
```

processRecords

KCL llama al método `processRecords` y pasa una lista de registros de datos desde la partición especificada por el método `initialize(shardId)`. El procesador de registros procesa los datos en estos registros según la semántica del consumidor. Por ejemplo, el proceso de trabajo podría realizar una transformación de los datos y, a continuación, almacenar el resultado en un bucket de Amazon Simple Storage Service (Amazon S3).

```
public void processRecords(List<Record> records, IRecordProcessorCheckpointer
 checkpointer)
```

Además de los datos en sí, el registro contiene un número secuencial y una clave de partición. El proceso de trabajo puede utilizar estos valores al procesar los datos. Por ejemplo, el proceso de trabajo podría elegir el bucket de S3 en el que almacenar los datos en función del valor de la clave de partición. La clase `Record` expone los siguientes métodos que proporcionan acceso a los datos, el número secuencial y la clave de partición del registro.

```
record.getData()
record.getSequenceNumber()
record.getPartitionKey()
```

En el ejemplo, el método privado `processRecordsWithRetries` tiene un código que muestra cómo un proceso de trabajo puede obtener acceso a los datos, el número secuencial y la clave de partición del registro.

Kinesis Data Streams requiere que el procesador de registros realice un seguimiento de los registros que ya se han procesado en una partición. KCL se ocupa de este seguimiento pasando un

generador de puntos de verificación (`IRecordProcessorCheckpointer`) a `processRecords`. El procesador de registros llama al método `checkpoint` en esta interfaz para informar a KCL de su avance en el procesamiento de los registros de la partición. Si se produce un error en el proceso de trabajo, KCL utiliza esta información para reiniciar el procesamiento de la partición en el último registro procesado conocido.

En el caso de una operación de división o fusión, KCL no comenzará a procesar las particiones nuevas hasta que los procesadores de las particiones originales hayan llamado a `checkpoint` para indicar que se ha completado el procesamiento en las particiones originales.

Si no se pasa un parámetro, KCL supone que la llamada a `checkpoint` significa que todos los registros se han procesado, hasta el último registro pasado al procesador de registros. Por tanto, el procesador de registros solo debe llamar a `checkpoint` después de haber procesado todos los registros de la lista que se le ha pasado. Los procesadores de registros no necesitan llamar a `checkpoint` en cada llamada a `processRecords`. Un procesador podría, por ejemplo, llamar a `checkpoint` cada tercera vez que llame a `processRecords`. Puede especificar opcionalmente el número secuencial exacto de un registro como un parámetro para `checkpoint`. En este caso, KCL supone que todos los registros se han procesado exclusivamente hasta ese registro.

En el ejemplo, el método privado `checkpoint` muestra cómo llamar a `IRecordProcessorCheckpointer.checkpoint` mediante la administración de excepciones y la lógica de reintentos apropiadas.

KCL depende de `processRecords` para administrar cualquier excepción que surja del procesamiento de los registros de datos. Si `processRecords` genera una excepción, KCL omite los registros de datos que se pasaron antes de la excepción. Es decir, estos registros no se reenviarán al procesador de registros que generó la excepción ni a ningún otro procesador de registros en el consumidor.

shutdown

KCL llama al método `shutdown` cuando finaliza el procesamiento (el motivo del cierre es `TERMINATE`) o cuando el proceso de trabajo ya no responde (el motivo del cierre es `ZOMBIE`).

```
public void shutdown(IRecordProcessorCheckpointer checkpointer, ShutdownReason reason)
```

El procesamiento finaliza cuando el procesador de registros no recibe más registros desde el fragmento, ya sea porque el fragmento se ha dividido o fusionado o porque la secuencia se ha eliminado.

KCL también pasa una interfaz `IRecordProcessorCheckpointer` a `shutdown`. Si el motivo del shutdown es `TERMINATE`, el procesador de registros debería terminar de procesar los registros de datos y llamar al método `checkpoint` en esta interfaz.

Interfaz actualizada (versión 2)

La interfaz actualizada `IRecordProcessor` (`package com.amazonaws.services.kinesis.clientlibrary.interfaces.v2`) expone los siguientes métodos del procesador de registros que el consumidor debe implementar:

```
void initialize(InitializationInput initializationInput)
void processRecords(ProcessRecordsInput processRecordsInput)
void shutdown(ShutdownInput shutdownInput)
```

Se puede obtener acceso a todos los argumentos de la versión original de la interfaz mediante métodos "get" en los objetos del contenedor. Por ejemplo, para recuperar la lista de registros en `processRecords()`, puede utilizar `processRecordsInput.getRecords()`.

A partir de la versión 2 de esta interfaz (KCL 1.5.0 y posteriores), están disponibles las siguientes entradas nuevas, además de las entradas proporcionadas por la interfaz original:

starting sequence number

: En el objeto `InitializationInput` que se pasa a la operación `initialize()` el número secuencial inicial a partir del cual se facilitan los registros a la instancia del procesador de registros. Este es el último número secuencial objeto de un punto de comprobación por parte de la instancia del procesador de registros que procesara anteriormente el mismo fragmento. Estos datos se ofrecen por si su aplicación necesitara esta información.

pending checkpoint sequence number

: En el objeto `InitializationInput` que se pasa a la operación `initialize()`, el número secuencial pendiente de punto de comprobación (si hay alguno) que no se ha podido confirmar antes de que se detuviera la instancia anterior del procesador de registros.

Implemente una fábrica de clases para la RecordProcessor interfaz I

También necesitará implementar un generador para la clase que implementa los métodos del procesador de registros. Cuando el consumidor crea instancias del proceso de trabajo, pasa una referencia a este generador.

La muestra implementa el generador de clases en el archivo `AmazonKinesisApplicationSampleRecordProcessorFactory.java` mediante la interfaz del procesador de registros original. Si desea que el generador de clases cree procesadores de registros de la versión 2, utilice el nombre de paquete `com.amazonaws.services.kinesis.clientlibrary.interfaces.v2`.

```
public class SampleRecordProcessorFactory implements IRecordProcessorFactory {
    /**
    * Constructor.
    */
    public SampleRecordProcessorFactory() {
        super();
    }
    /**
    * {@inheritDoc}
    */
    @Override
    public IRecordProcessor createProcessor() {
        return new SampleRecordProcessor();
    }
}
```

Creación de un proceso de trabajo

Tal y como se ha explicado en Implemente los métodos I. RecordProcessor , hay dos versiones de la interfaz de procesador de registros de KCL para elegir, lo que afecta a la creación de un proceso de trabajo. La interfaz de procesador de registros original utiliza la siguiente estructura de código para crear un proceso de trabajo:

```
final KinesisClientLibConfiguration config = new KinesisClientLibConfiguration(...)
final IRecordProcessorFactory recordProcessorFactory = new RecordProcessorFactory();
final Worker worker = new Worker(recordProcessorFactory, config);
```

Con la versión 2 del procesador de registros, puede utilizar la `Worker.Builder` para crear un proceso de trabajo sin preocuparse por qué constructor utilizar ni por el orden de los argumentos. La interfaz de procesador de registros actualizada utiliza la siguiente estructura de código para crear un proceso de trabajo:

```
final KinesisClientLibConfiguration config = new KinesisClientLibConfiguration(...)
```

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

Modificación de las propiedades de configuración

En la muestra se proporcionan valores predeterminados para las propiedades de configuración. Este datos de configuración del proceso de trabajo se consolidan posteriormente en un objeto `KinesisClientLibConfiguration`. Este objeto y una referencia al generador de clases de `IRecordProcessor` se pasan en la llamada que crea la instancia del proceso de trabajo. Puede sobrescribir cualquiera de estas propiedades con sus propios valores a través de un archivo de propiedades de Java (consulte `AmazonKinesisApplicationSample.java`).

Nombre de la aplicación

KCL requiere un nombre de aplicación que sea único entre las aplicaciones y en las tablas de Amazon DynamoDB de la misma región. La biblioteca utiliza el valor del nombre de la aplicación de las siguientes formas:

- Se entiende que los procesos de trabajo asociados a este nombre de aplicación operan de forma conjunta en la misma secuencia. Estos procesos de trabajo pueden distribuirse en varias instancias. Si ejecuta otra instancia del mismo código de aplicación, pero con otro nombre de aplicación, KCL considera que la segunda instancia es una aplicación completamente independiente de la otra que opera en el mismo flujo.
- KCL crea una tabla de DynamoDB con el nombre de la aplicación y utiliza la tabla para actualizar la información de estado (como los puntos de verificación y el mapeo procesos de trabajo-particiones) para la aplicación. Cada aplicación tiene su propia tabla de DynamoDB. Para obtener más información, consulte Uso de una tabla de arrendamiento para realizar el seguimiento de las particiones procesadas por la aplicación de consumo de KCL.

Configuración de credenciales

Debe poner sus credenciales de AWS a disposición de uno de los proveedores de credenciales en la cadena de proveedores de credenciales predeterminada. Por ejemplo, si ejecuta su consumidor en una instancia de EC2, se recomienda que lance la instancia con un rol de IAM. Las credenciales de AWS que reflejan los permisos asociados a este rol de IAM se ponen a disposición de las

aplicaciones de la instancia a través de los metadatos de esta. Esta es la forma más segura de administrar las credenciales para un consumidor que se ejecute en una instancia EC2.

En primer lugar, la aplicación de muestra intenta recuperar las credenciales de IAM de los metadatos de la instancia:

```
credentialsProvider = new InstanceProfileCredentialsProvider();
```

Si la aplicación de muestra no puede obtener credenciales de los metadatos de la instancia, intenta recuperar las credenciales desde un archivo de propiedades:

```
credentialsProvider = new ClasspathPropertiesFileCredentialsProvider();
```

Para más información sobre los metadatos de instancia, consulte Metadatos de instancia en la Guía del usuario de Amazon EC2 para instancias de Linux.

Uso de ID de proceso de trabajo para varias instancias

El código de inicialización de muestra crea un ID para el proceso de trabajo, `workerId`, con el nombre del equipo local y un identificador global único anexo, tal y como se muestra en el siguiente fragmento de código. Este enfoque es compatible con un escenario con varias instancias de la aplicación consumidora ejecutándose en un único equipo.

```
String workerId = InetAddress.getLocalHost().getCanonicalHostName() + ":" +
 UUID.randomUUID();
```

Migración a la versión 2 de la interfaz del procesador de registros

Si desea migrar código que utilice la interfaz original, además de los pasos descritos anteriormente, tendrá que seguir estos pasos:

1. Cambie la clase de su procesador de registros para importar la versión 2 de la interfaz del procesador de registros:

   ```
   import com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor;
   ```

2. Cambiar las referencias a las entradas para usar métodos `get` en los objetos del contenedor. Por ejemplo, en la operación `shutdown()`, cambie "`checkpointer`" por "`shutdownInput.getCheckpointer()`".

3. Cambie la clase del generador de procesadores de registros para importar la interfaz del generador de procesadores de registros de la versión 2:

```
import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessorFactory;
```

4. Cambie la construcción del proceso de trabajo para usar `Worker.Builder`. Por ejemplo:

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

Desarrollo de un consumidor de Kinesis Client Library en Node.js

Puede utilizar Kinesis Client Library (KCL) para crear aplicaciones que procesen datos de los flujos de datos de Kinesis. Kinesis Client Library está disponible en varios idiomas. En este tema se habla de Node.js

La KCL es una biblioteca de Java; el soporte para lenguajes distintos de Java se proporciona mediante una interfaz multilingüe llamada. MultiLangDaemon Este daemon está basado en Java y se ejecuta en segundo plano cuando se utiliza un lenguaje de KCL distinto de Java. Por lo tanto, si instala el KCL para Node.js y escribe su aplicación para consumidores completamente en Node.js, seguirá necesitando instalar Java en su sistema debido a la. MultiLangDaemon Además, MultiLangDaemon tiene algunos ajustes predeterminados que puede que tengas que personalizar para tu caso de uso, por ejemplo, la AWS región a la que se conecta. Para obtener más información sobre MultiLangDaemon esto GitHub, visita la página del MultiLangDaemon proyecto KCL.

Para descargar el archivo KCL de Node.js GitHub, vaya a la biblioteca de clientes de Kinesis (Node.js).

Descargas de código de muestra

Hay dos códigos de muestra disponibles para la KCL en Node.js:

- basic-sample

  Se utiliza en las siguientes secciones para ilustrar los aspectos fundamentales de la creación de una aplicación de consumo de KCL en Node.js.
- click-stream-sample

Es ligeramente más avanzado y utiliza una situación real. Para cuando se haya familiarizado con el código de muestra básico. Esta muestra no se trata aquí, pero tiene un archivo README con más información.

Debe completar las siguientes tareas a la hora de implementar una aplicación de consumo de KCL en Node.js:

Tareas

- Implementar el procesador de registros
- Modificación de las propiedades de configuración

Implementar el procesador de registros

El consumidor más sencillo posible que utilice KCL para Node.js debe implementar una función `recordProcessor`, que a su vez contenga las funciones `initialize`, `processRecords` y `shutdown`. En la muestra se presenta una implementación que puede utilizar como punto de partida (consulte `sample_kcl_app.js`).

```
function recordProcessor() {
  // return an object that implements initialize, processRecords and shutdown
 functions.}
```

initialize

KCL llama a la función `initialize` cuando se inicia el procesador de registros. Este procesador de registros procesa solo la ID de fragmento que se haya pasado como `initializeInput.shardId`, y normalmente también se produce la situación contraria (este fragmento solo es procesado por este procesador de registros). Sin embargo, el consumidor debe contar con la posibilidad de que un registro de datos pueda ser procesado más de una vez. Esto se debe a que Kinesis Data Streams tiene una semántica de al menos una vez, lo que significa que cada registro de datos de una partición se procesa al menos una vez por parte de un proceso de trabajo del consumidor. Para obtener más información sobre los casos en los que un fragmento en particular puede ser procesado por varios procesos de trabajo, consulte Cambio en los fragmentos, escalado y procesamiento paralelo.

```
initialize: function(initializeInput, completeCallback)
```

processRecords

KCL llama a esta función con una entrada que contiene una lista de registros de datos de la partición especificados para la función `initialize`. El procesador de registros que implemente procesa los datos en estos registros según la semántica del consumidor. Por ejemplo, el proceso de trabajo podría realizar una transformación de los datos y, a continuación, almacenar el resultado en un bucket de Amazon Simple Storage Service (Amazon S3).

```
processRecords: function(processRecordsInput, completeCallback)
```

Además de los datos en sí, el registro también contiene un número secuencial y una clave de partición, que el proceso de trabajo puede utilizar al procesar los datos. Por ejemplo, el proceso de trabajo podría elegir el bucket de S3 en el que almacenar los datos en función del valor de la clave de partición. El diccionario `record` expone los siguientes pares clave-valor para obtener acceso a los datos, el número secuencial y la clave de partición del registro:

```
record.data
record.sequenceNumber
record.partitionKey
```

Tenga en cuenta que los datos se codifican en Base64.

En el ejemplo básico, la función `processRecords` tiene un código que muestra cómo un proceso de trabajo puede obtener acceso a los datos, el número secuencial y la clave de partición del registro.

Kinesis Data Streams requiere que el procesador de registros realice un seguimiento de los registros que ya se han procesado en una partición. KCL se ocupa de este seguimiento con un objeto `checkpointer` que pasa como `processRecordsInput.checkpointer`. Su procesador de registros llama a la función `checkpointer.checkpoint` para informar a KCL de su avance en el procesamiento de los registros de la partición. En el caso de que se produzca un error en el proceso de trabajo, KCL utiliza esta información al reiniciar el procesamiento de la partición para continuar desde el último registro procesado conocido.

En el caso de una operación de división o fusión, KCL no comenzará a procesar las particiones nuevas hasta que los procesadores de las particiones originales hayan llamado a `checkpoint` para indicar que se ha completado el procesamiento en las particiones originales.

Si no pasa el número secuencial a la función `checkpoint`, KCL supone que la llamada a `checkpoint` significa que todos los registros se han procesado, hasta el último registro pasado al procesador de registros. Por tanto, el procesador de registros debería llamar a `checkpoint` solo

después de haber procesado todos los registros de la lista que se le ha pasado. Los procesadores de registros no necesitan llamar a `checkpoint` en cada llamada a `processRecords`. Un procesador podría, por ejemplo, llamar a `checkpoint` en cada tercera llamada, o a algún evento externo a su procesador de registros, como un servicio de validación/verificación personalizado que haya implementado.

Puede especificar opcionalmente el número secuencial exacto de un registro como un parámetro para `checkpoint`. En este caso, KCL supone que todos los registros se han procesado exclusivamente hasta ese registro.

La aplicación de muestra básica muestra la llamada más sencilla posible a la función `checkpointer.checkpoint`. Puede agregar otra lógica de creación de puntos de comprobación que necesite para su consumidor en este punto en la función.

shutdown

KCL llama a la función `shutdown` cuando finaliza el procesamiento (`shutdownInput.reason` es `TERMINATE`) o cuando el proceso de trabajo ya no responde (`shutdownInput.reason` es `ZOMBIE`).

```
shutdown: function(shutdownInput, completeCallback)
```

El procesamiento finaliza cuando el procesador de registros no recibe más registros desde el fragmento, ya sea porque el fragmento se ha dividido o fusionado o porque la secuencia se ha eliminado.

KCL también pasa un objeto `shutdownInput.checkpointer` a `shutdown`. Si el motivo del cierre es `TERMINATE`, debe asegurarse de que el procesador de registros haya terminado de procesar los registros de datos y después llame a la función `checkpoint` en esta interfaz.

Modificación de las propiedades de configuración

En la muestra se proporcionan valores predeterminados para las propiedades de configuración. Puede sobrescribir cualquiera de estas propiedades con sus propios valores (consulte `sample.properties` en el ejemplo básico).

Nombre de la aplicación

KCL requiere una aplicación que sea única entre las aplicaciones y en las tablas de Amazon DynamoDB de la misma región. La biblioteca utiliza el valor del nombre de la aplicación de las siguientes formas:

- Se entiende que los procesos de trabajo asociados a este nombre de aplicación operan de forma conjunta en la misma secuencia. Estos procesos de trabajo pueden distribuirse en varias instancias. Si ejecuta otra instancia del mismo código de aplicación, pero con otro nombre de aplicación, KCL considera que la segunda instancia es una aplicación completamente independiente de la otra que opera en el mismo flujo.
- KCL crea una tabla de DynamoDB con el nombre de la aplicación y utiliza la tabla para actualizar la información de estado (como los puntos de verificación y el mapeo procesos de trabajo-particiones) para la aplicación. Cada aplicación tiene su propia tabla de DynamoDB. Para obtener más información, consulte Uso de una tabla de arrendamiento para realizar el seguimiento de las particiones procesadas por la aplicación de consumo de KCL.

Configuración de credenciales

Debe poner sus credenciales de AWS a disposición de uno de los proveedores de credenciales en la cadena de proveedores de credenciales predeterminada. Puede usar la propiedad `AWSCredentialsProvider` para configurar un proveedor de credenciales. El archivo `sample.properties` debe poner sus credenciales a disposición de uno de los proveedores de credenciales de la cadena de proveedores de credenciales predeterminada. Si ejecuta su consumidor en una instancia de Amazon EC2, se recomienda configurar la instancia con un rol de IAM. Las credenciales de AWS que reflejan los permisos asociados a este rol de IAM se ponen a disposición de las aplicaciones de la instancia a través de los metadatos de esta. Esta es la forma más segura de administrar las credenciales para una aplicación consumidora que se ejecute en una instancia EC2.

En el siguiente ejemplo, se configura KCL para procesar un flujo de datos de Kinesis denominado `kclnodejssample` utilizando el procesador de registros facilitado en `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
```

Desarrollo de un consumidor de Kinesis Client Library en .NET

Puede utilizar Kinesis Client Library (KCL) para crear aplicaciones que procesen datos de los flujos de datos de Kinesis. Kinesis Client Library está disponible en varios idiomas. En este tema se habla de .NET.

La KCL es una biblioteca de Java; la compatibilidad con otros lenguajes además de Java se proporciona mediante una interfaz multilingüe llamada. MultiLangDaemon Este daemon está basado en Java y se ejecuta en segundo plano cuando se utiliza un lenguaje de KCL distinto de Java. Por lo tanto, si instala la KCL para.NET y escribe la aplicación para el usuario en su totalidad en .NET, seguirá necesitando instalar Java en el sistema debido a la. MultiLangDaemon Además, MultiLangDaemon tiene algunos ajustes predeterminados que puede que tengas que personalizar para tu caso de uso, por ejemplo, la AWS región a la que se conecta. Para obtener más información sobre MultiLangDaemon esto GitHub, visita la página del MultiLangDaemon proyecto KCL.

Para descargar el KCL de.NET GitHub, vaya a la biblioteca de clientes de Kinesis (.NET). Para descargar un código de muestra para una aplicación para consumidores de KCL de.NET, visite la página de ejemplos de proyectos para consumidores de KCL para .NET en. GitHub

Debe completar las siguientes tareas a la hora de implementar una aplicación de consumo de KCL en .NET:

Tareas

- Implemente los métodos de clase I RecordProcessor
- Modificación de las propiedades de configuración

Implemente los métodos de clase I RecordProcessor

El consumidor debe implementar los siguientes métodos para `IRecordProcessor`. En el consumidor de muestra se presentan implementaciones que puede utilizar como punto de partida (consulte la clase `SampleRecordProcessor` en `SampleConsumer/AmazonKinesisSampleConsumer.cs`).

```
public void Initialize(InitializationInput input)
public void ProcessRecords(ProcessRecordsInput input)
public void Shutdown(ShutdownInput input)
```

Initialize

KCL llama a este método cuando se crea una instancia del procesador de registros y pasa un ID de partición específico en el parámetro `input` (`input.ShardId`). Este procesador de registros procesa solo este fragmento, y normalmente también se produce la situación contraria (este fragmento solo es procesado por este procesador de registros). Sin embargo, el consumidor debe contar con la posibilidad de que un registro de datos pueda ser procesado más de una vez. Esto se debe a que Kinesis Data Streams tiene una semántica de al menos una vez, lo que significa que cada registro de datos de una partición se procesa al menos una vez por parte de un proceso de trabajo del consumidor. Para obtener más información sobre los casos en los que un fragmento en particular puede ser procesado por varios procesos de trabajo, consulte Cambio en los fragmentos, escalado y procesamiento paralelo.

```
public void Initialize(InitializationInput input)
```

ProcessRecords

KCL llama a este método y pasa una lista de registros de datos en el parámetro `input` (`input.Records`) desde la partición especificada por el método `Initialize`. El procesador de registros que implemente procesa los datos en estos registros según la semántica del consumidor. Por ejemplo, el proceso de trabajo podría realizar una transformación de los datos y, a continuación, almacenar el resultado en un bucket de Amazon Simple Storage Service (Amazon S3).

```
public void ProcessRecords(ProcessRecordsInput input)
```

Además de los datos en sí, el registro contiene un número secuencial y una clave de partición. El proceso de trabajo puede utilizar estos valores al procesar los datos. Por ejemplo, el proceso de trabajo podría elegir el bucket de S3 en el que almacenar los datos en función del valor de la clave de partición. La clase `Record` expone lo siguiente para obtener acceso a los datos, el número secuencial y la clave de partición del registro:

```
byte[] Record.Data
string Record.SequenceNumber
string Record.PartitionKey
```

En el ejemplo, el método `ProcessRecordsWithRetries` tiene un código que muestra cómo un proceso de trabajo puede obtener acceso a los datos, el número secuencial y la clave de partición del registro.

Kinesis Data Streams requiere que el procesador de registros realice un seguimiento de los registros que ya se han procesado en una partición. KCL se ocupa de este seguimiento, pasando un objeto

`Checkpointer` a `ProcessRecords` (`input.Checkpointer`). El procesador de registros llama al método `Checkpointer.Checkpoint` para informar a KCL de su avance en el procesamiento de los registros de la partición. Si se produce un error en el proceso de trabajo, KCL utiliza esta información para reiniciar el procesamiento de la partición en el último registro procesado conocido.

En el caso de una operación de división o fusión, KCL no comenzará a procesar las particiones nuevas hasta que los procesadores de las particiones originales hayan llamado a `Checkpointer.Checkpoint` para indicar que se ha completado el procesamiento en las particiones originales.

Si no se pasa un parámetro, KCL supone que la llamada a `Checkpointer.Checkpoint` significa que todos los registros se han procesado, hasta el último registro pasado al procesador de registros. Por tanto, el procesador de registros solo debe llamar a `Checkpointer.Checkpoint` después de haber procesado todos los registros de la lista que se le ha pasado. Los procesadores de registros no necesitan llamar a `Checkpointer.Checkpoint` en cada llamada a `ProcessRecords`. Un procesador podría, por ejemplo, llamar a `Checkpointer.Checkpoint` en cada tercera o cuarta llamada. Puede especificar opcionalmente el número secuencial exacto de un registro como un parámetro para `Checkpointer.Checkpoint`. En este caso, KCL supone que los registros se han procesado exclusivamente hasta ese registro.

En el ejemplo, el método privado `Checkpoint(Checkpointer checkpointer)` muestra cómo llamar al método `Checkpointer.Checkpoint` mediante la administración de excepciones y la lógica de reintentos apropiadas.

KCL para .NET administra las excepciones de forma diferente a las bibliotecas de KCL para el resto de lenguajes, ya que no administra las excepciones que surgen del procesamiento de los registros de datos. Las excepciones no detectadas procedentes del código del usuario harán que el programa se bloquee.

Shutdown

KCL llama al método `Shutdown` cuando finaliza el procesamiento (el motivo del cierre es `TERMINATE`) o cuando el proceso de trabajo ya no responde (el valor de `input.Reason` del cierre es `ZOMBIE`).

```
public void Shutdown(ShutdownInput input)
```

El procesamiento finaliza cuando el procesador de registros no recibe más registros desde el fragmento, ya sea porque el fragmento se ha dividido o fusionado o porque la secuencia se ha eliminado.

KCL también pasa un objeto `Checkpointer` a `shutdown`. Si el motivo del shutdown es `TERMINATE`, el procesador de registros debería terminar de procesar los registros de datos y llamar al método `checkpoint` en esta interfaz.

Modificación de las propiedades de configuración

En el consumidor muestra se proporcionan valores predeterminados para las propiedades de configuración. Puede sobrescribir cualquiera de estas propiedades con sus propios valores (consulte `SampleConsumer/kcl.properties`).

Nombre de la aplicación

KCL requiere una aplicación que sea única entre las aplicaciones y en las tablas de Amazon DynamoDB de la misma región. La biblioteca utiliza el valor del nombre de la aplicación de las siguientes formas:

- Se entiende que los procesos de trabajo asociados a este nombre de aplicación operan de forma conjunta en la misma secuencia. Estos procesos de trabajo pueden distribuirse en varias instancias. Si ejecuta otra instancia del mismo código de aplicación, pero con otro nombre de aplicación, KCL considera que la segunda instancia es una aplicación completamente independiente de la otra que opera en el mismo flujo.
- KCL crea una tabla de DynamoDB con el nombre de la aplicación y utiliza la tabla para actualizar la información de estado (como los puntos de verificación y el mapeo procesos de trabajo-particiones) para la aplicación. Cada aplicación tiene su propia tabla de DynamoDB. Para obtener más información, consulte Uso de una tabla de arrendamiento para realizar el seguimiento de las particiones procesadas por la aplicación de consumo de KCL.

Configuración de credenciales

Debe poner sus credenciales de AWS a disposición de uno de los proveedores de credenciales en la cadena de proveedores de credenciales predeterminada. Puede usar la propiedad `AWSCredentialsProvider` para configurar un proveedor de credenciales. Las propiedades de muestra deben poner sus credenciales a disposición de uno de los proveedores de credenciales de la cadena de proveedores de credenciales predeterminada. Si ejecuta su aplicación de consumo en una instancia de EC2, se recomienda que configure la instancia con un rol de IAM. Las credenciales

de AWS que reflejan los permisos asociados a este rol de IAM se ponen a disposición de las aplicaciones de la instancia a través de los metadatos de esta. Esta es la forma más segura de administrar las credenciales para un consumidor que se ejecute en una instancia EC2.

El archivo de propiedades de ejemplo configura KCL para procesar un flujo de datos de Kinesis llamado “words” utilizando el procesador de registros facilitado en `AmazonKinesisSampleConsumer.cs`.

Desarrollo de un consumidor de Kinesis Client Library en Python

Puede utilizar Kinesis Client Library (KCL) para crear aplicaciones que procesen datos de los flujos de datos de Kinesis. Kinesis Client Library está disponible en varios idiomas. En este tema se habla de Python.

La KCL es una biblioteca de Java; el soporte para lenguajes distintos de Java se proporciona mediante una interfaz multilingüe llamada. MultiLangDaemon Este daemon está basado en Java y se ejecuta en segundo plano cuando se utiliza un lenguaje de KCL distinto de Java. Por lo tanto, si instala el KCL para Python y escribe su aplicación de consumo completamente en Python, seguirá necesitando instalar Java en su sistema debido a la MultiLangDaemon. Además, MultiLangDaemon tiene algunos ajustes predeterminados que puede que tengas que personalizar para tu caso de uso, por ejemplo, la AWS región a la que se conecta. Para obtener más información sobre MultiLangDaemon esto GitHub, visita la página del MultiLangDaemon proyecto KCL.

Para descargar la KCL de Python GitHub, vaya a la biblioteca de clientes de Kinesis (Python). Para descargar un código de muestra para una aplicación de consumo de KCL para Python, vaya a la página del proyecto de ejemplo de KCL para Python en. GitHub

Debe completar las siguientes tareas a la hora de implementar una aplicación de consumo de KCL en Python:

Tareas

- Implemente los métodos de clase RecordProcessor
- Modificación de las propiedades de configuración

Implemente los métodos de clase RecordProcessor

La clase `RecordProcess` debe ampliar la `RecordProcessorBase` para implementar los siguientes métodos. En la muestra se presentan implementaciones que puede utilizar como punto de partida (consulte `sample_kclpy_app.py`).

```
def initialize(self, shard_id)
def process_records(self, records, checkpointer)
def shutdown(self, checkpointer, reason)
```

initialize

KCL llama al método `initialize` cuando se crea una instancia del procesador de registros y pasa un ID de partición específico como parámetro. Este procesador de registros procesa solo este fragmento, y normalmente también se produce la situación contraria (este fragmento solo es procesado por este procesador de registros). Sin embargo, el consumidor debe contar con la posibilidad de que un registro de datos pueda ser procesado más de una vez. Esto se debe a que Kinesis Data Streams tiene una semántica de al menos una vez, lo que significa que cada registro de datos de una partición se procesa al menos una vez por parte de un proceso de trabajo del consumidor. Para obtener más información sobre los casos en los que un fragmento en particular puede ser procesado por más de un proceso de trabajo, consulte Cambio en los fragmentos, escalado y procesamiento paralelo.

```
def initialize(self, shard_id)
```

process_records

KCL llama a este método y pasa una lista de registros de datos de la partición especificada por el método `initialize`. El procesador de registros que implemente procesa los datos en estos registros según la semántica del consumidor. Por ejemplo, el proceso de trabajo podría realizar una transformación de los datos y, a continuación, almacenar el resultado en un bucket de Amazon Simple Storage Service (Amazon S3).

```
def process_records(self, records, checkpointer)
```

Además de los datos en sí, el registro contiene un número secuencial y una clave de partición. El proceso de trabajo puede utilizar estos valores al procesar los datos. Por ejemplo, el proceso de trabajo podría elegir el bucket de S3 en el que almacenar los datos en función del valor de la clave de partición. El diccionario `record` expone los siguientes pares clave-valor para obtener acceso a los datos, el número secuencial y la clave de partición del registro:

```
record.get('data')
record.get('sequenceNumber')
record.get('partitionKey')
```

Tenga en cuenta que los datos se codifican en Base64.

En el ejemplo, el método `process_records` tiene un código que muestra cómo un proceso de trabajo puede obtener acceso a los datos, el número secuencial y la clave de partición del registro.

Kinesis Data Streams requiere que el procesador de registros realice un seguimiento de los registros que ya se han procesado en una partición. KCL se ocupa de este seguimiento, pasando un objeto `Checkpointer` a `process_records`. El procesador de registros llama al método `checkpoint` en este objeto para informar a KCL de su avance en el procesamiento de los registros de la partición. Si se produce un error en el proceso de trabajo, KCL utiliza esta información para reiniciar el procesamiento de la partición en el último registro procesado conocido.

En el caso de una operación de división o fusión, KCL no comenzará a procesar las particiones nuevas hasta que los procesadores de las particiones originales hayan llamado a `checkpoint` para indicar que se ha completado el procesamiento en las particiones originales.

Si no se pasa un parámetro, KCL supone que la llamada a `checkpoint` significa que todos los registros se han procesado, hasta el último registro pasado al procesador de registros. Por tanto, el procesador de registros solo debe llamar a `checkpoint` después de haber procesado todos los registros de la lista que se le ha pasado. Los procesadores de registros no necesitan llamar a `checkpoint` en cada llamada a `process_records`. Un procesador podría, por ejemplo, llamar a `checkpoint` cada tercera vez que llame. Puede especificar opcionalmente el número secuencial exacto de un registro como un parámetro para `checkpoint`. En este caso, KCL supone que todos los registros se han procesado exclusivamente hasta ese registro.

En el ejemplo, el método privado `checkpoint` muestra cómo llamar al método `Checkpointer.checkpoint` mediante la administración de excepciones y la lógica de reintentos apropiadas.

KCL depende de `process_records` para administrar cualquier excepción que surja del procesamiento de los registros de datos. Si `process_records` genera una excepción, KCL omite los registros de datos que se pasaron a `process_records` antes de la excepción. Es decir, estos registros no se reenviarán al procesador de registros que generó la excepción ni a ningún otro procesador de registros en el consumidor.

shutdown

KCL llama al método `shutdown` cuando finaliza el procesamiento (el motivo del cierre es `TERMINATE`) o cuando el proceso de trabajo ya no responde (el `reason` del cierre es `ZOMBIE`).

```
def shutdown(self, checkpointer, reason)
```

El procesamiento finaliza cuando el procesador de registros no recibe más registros desde el fragmento, ya sea porque el fragmento se ha dividido o fusionado o porque la secuencia se ha eliminado.

KCL también pasa un objeto `Checkpointer` a `shutdown`. Si el `reason` del shutdown es `TERMINATE`, el procesador de registros debería terminar de procesar los registros de datos y llamar al método `checkpoint` en esta interfaz.

Modificación de las propiedades de configuración

En la muestra se proporcionan valores predeterminados para las propiedades de configuración. Puede sobrescribir cualquiera de estas propiedades con sus propios valores (consulte `sample.properties`).

Nombre de la aplicación

KCL requiere un nombre de aplicación que sea único entre las aplicaciones y en las tablas de Amazon DynamoDB de la misma región. La biblioteca utiliza el valor del nombre de la aplicación de las siguientes formas:

- Se supone que los procesos de trabajo que están asociados a este nombre de aplicación operan de forma conjunta en la misma secuencia. Estos procesos de trabajo pueden distribuirse en varias instancias. Si ejecuta otra instancia del mismo código de aplicación, pero con otro nombre de aplicación, KCL considera que la segunda instancia es una aplicación completamente independiente de la otra que opera en el mismo flujo.
- KCL crea una tabla de DynamoDB con el nombre de la aplicación y utiliza la tabla para actualizar la información de estado (como los puntos de verificación y el mapeo procesos de trabajo-particiones) para la aplicación. Cada aplicación tiene su propia tabla de DynamoDB. Para obtener más información, consulte Uso de una tabla de arrendamiento para realizar el seguimiento de las particiones procesadas por la aplicación de consumo de KCL.

Configuración de credenciales

Debe poner sus credenciales de AWS a disposición de uno de los proveedores de credenciales en la cadena de proveedores de credenciales predeterminada. Puede usar la propiedad `AWSCredentialsProvider` para configurar un proveedor de credenciales. Las propiedades de muestra deben poner sus credenciales a disposición de uno de los proveedores de credenciales de

la cadena de proveedores de credenciales predeterminada. Si ejecuta su aplicación de consumo en una instancia de Amazon EC2, se recomienda que configure la instancia con un rol de IAM. Las credenciales de AWS que reflejan los permisos asociados a este rol de IAM se ponen a disposición de las aplicaciones de la instancia a través de los metadatos de esta. Esta es la forma más segura de administrar las credenciales para una aplicación consumidora que se ejecute en una instancia EC2.

El archivo de propiedades de ejemplo configura KCL para procesar un flujo de datos de Kinesis llamado “words” utilizando el procesador de registros facilitado en `sample_kclpy_app.py`.

Desarrollo de un consumidor de Kinesis Client Library en Ruby

Puede utilizar Kinesis Client Library (KCL) para crear aplicaciones que procesen datos de los flujos de datos de Kinesis. Kinesis Client Library está disponible en varios idiomas. En este tema se habla de Ruby.

La KCL es una biblioteca de Java; el soporte para lenguajes distintos de Java se proporciona mediante una interfaz multilingüe llamada. MultiLangDaemon Este daemon está basado en Java y se ejecuta en segundo plano cuando se utiliza un lenguaje de KCL distinto de Java. Por lo tanto, si instala el KCL para Ruby y escribe su aplicación para consumidores completamente en Ruby, seguirá necesitando instalar Java en su sistema debido a la. MultiLangDaemon Además, MultiLangDaemon tiene algunos ajustes predeterminados que puede que tengas que personalizar para tu caso de uso, por ejemplo, la AWS región a la que se conecta. Para obtener más información sobre MultiLangDaemon esto GitHub, visita la página del MultiLangDaemon proyecto KCL.

Para descargar el KCL de Ruby GitHub, vaya a la biblioteca de clientes de Kinesis (Ruby). Para descargar un código de muestra para una aplicación de usuario de Ruby KCL, visite la página del proyecto de ejemplo de KCL para Ruby en. GitHub

Para más información acerca de la biblioteca de soporte de Ruby en KCL, consulte la documentación de Ruby Gems en KCL.

Desarrollo de consumidores de KCL 2.x

En este tema se muestra cómo utilizar la versión 2.0 de Kinesis Client Library (KCL). Para más información acerca de KCL, consulte la información general que se proporciona en Desarrollo de consumidores mediante Kinesis Client Library 1.x.

Contenido

- Desarrollo de un consumidor de Kinesis Client Library en Python

Desarrollo de un consumidor de Kinesis Client Library en Java

En el siguiente código, se muestra un ejemplo de implementación en Java de `ProcessorFactory` y `RecordProcessor`. Si desea aprovechar la característica de distribución ramificada mejorada, consulte Uso de consumidores con la distribución ramificada mejorada.

```
/*
 *  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);
            }
        }
    }

}
```

Desarrollo de un consumidor de Kinesis Client Library en Python

Puede utilizar Kinesis Client Library (KCL) para crear aplicaciones que procesen datos de los flujos de datos de Kinesis. Kinesis Client Library está disponible en varios idiomas. En este tema se habla de Python.

La KCL es una biblioteca de Java; el soporte para lenguajes distintos de Java se proporciona mediante una interfaz multilingüe llamada. MultiLangDaemon Este daemon está basado en Java y se ejecuta en segundo plano cuando se utiliza un lenguaje de KCL distinto de Java. Por lo tanto, si instala el KCL para Python y escribe su aplicación de consumo completamente en Python, seguirá necesitando instalar Java en su sistema debido a la MultiLangDaemon. Además, MultiLangDaemon tiene algunos ajustes predeterminados que puede que tengas que personalizar para tu caso de uso, por ejemplo, la AWS región a la que se conecta. Para obtener más información sobre MultiLangDaemon esto GitHub, visita la página del MultiLangDaemon proyecto KCL.

Para descargar la KCL de Python GitHub, vaya a la biblioteca de clientes de Kinesis (Python). Para descargar un código de muestra para una aplicación de consumo de KCL para Python, vaya a la página del proyecto de ejemplo de KCL para Python en. GitHub

Debe completar las siguientes tareas a la hora de implementar una aplicación de consumo de KCL en Python:

Tareas

- Implemente los métodos de clase RecordProcessor
- Modificación de las propiedades de configuración

Implemente los métodos de clase RecordProcessor

La clase `RecordProcess` debe ampliar la `RecordProcessorBase` para implementar los siguientes métodos:

```
initialize
process_records
shutdown_requested
```

Este ejemplo proporciona implementaciones que puede utilizar como punto de partida.

```
#!/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()
```

Modificación de las propiedades de configuración

En el ejemplo se proporcionan valores predeterminados para las propiedades de configuración, como los que se muestran en el siguiente script. Puede sobrescribir cualquiera de estas propiedades con sus propios valores.

```
# 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
```

Nombre de la aplicación

KCL requiere un nombre de aplicación que sea único entre las aplicaciones y en las tablas de Amazon DynamoDB de la misma región. La biblioteca utiliza el valor del nombre de la aplicación de las siguientes formas:

- Se supone que los procesos de trabajo que están asociados a este nombre de aplicación operan de forma conjunta en la misma secuencia. Estos procesos de trabajo pueden distribuirse entre varias instancias. Si ejecuta otra instancia del mismo código de aplicación, pero con otro nombre de aplicación, KCL considera que la segunda instancia es una aplicación completamente independiente de la otra que opera en el mismo flujo.
- KCL crea una tabla de DynamoDB con el nombre de la aplicación y utiliza la tabla para actualizar la información de estado (como los puntos de verificación y el mapeo procesos de trabajo-particiones) para la aplicación. Cada aplicación tiene su propia tabla de DynamoDB. Para obtener más información, consulte Uso de una tabla de arrendamiento para realizar el seguimiento de las particiones procesadas por la aplicación de consumo de KCL.

Credenciales

Debe poner sus credenciales de AWS a disposición de uno de los proveedores de credenciales en la cadena de proveedores de credenciales predeterminada. Puede usar la propiedad `AWSCredentialsProvider` para configurar un proveedor de credenciales. Si ejecuta su aplicación de consumo en una instancia de Amazon EC2, se recomienda que configure la instancia con un rol de IAM. Las credenciales de AWS que reflejan los permisos asociados a este rol de IAM se ponen a disposición de las aplicaciones de la instancia a través de los metadatos de esta. Esta es la forma más segura de administrar las credenciales para una aplicación consumidora que se ejecute en una instancia EC2.

Desarrollo de consumidores personalizados con rendimiento compartido con AWS SDK for Java

Uno de los métodos para desarrollar consumidores personalizados de Kinesis Data Streams con rendimiento compartido consiste en utilizar las API de Amazon Kinesis Data Streams. En esta sección se describe el uso de las API de Kinesis Data Streams con el SDK de AWS para Java. El código de muestra de Java de esta sección demuestra cómo hacer operaciones básicas con la API de KDS y está dividido de manera lógica por tipo de operación.

Estos ejemplos no representan código listo para producción. No comprueban todas las excepciones posibles ni tienen en cuenta todas las consideraciones de seguridad y de rendimiento.

Puede llamar a las API de Kinesis Data Streams mediante otros lenguajes de programación. Para más información acerca de todos los SDK de AWS disponibles, consulte Comience a crear con Amazon Web Services.

> **Important**
>
> El método recomendado para desarrollar consumidores personalizados de Kinesis Data Streams con rendimiento compartido es utilizar la Bibliotecta de clientes de Kinesis (KCL). KCL ayuda a consumir y procesar los datos de un flujo de datos de Kinesis, ya que se encarga de muchas de las tareas complejas asociadas a la computación distribuida. Para obtener más información, consulte Desarrollo de consumidores personalizados con rendimiento compartido mediante KCL.

Temas

Obtención de datos de una secuencia

Las API de Kinesis Data Streams incluyen los métodos `getShardIterator` y `getRecords` que se pueden invocar para recuperar los registros de un flujo de datos. Se trata del modelo de extracción, donde el código extrae registros de datos directamente de las particiones del flujo de datos.

> **Important**
>
> Se recomienda utilizar la funcionalidad de procesador de registros que proporciona KCL para recuperarlos de los flujos de datos. Se trata del modelo de inserción, en el que debe implementar el código que procesa los datos. KCL recupera los registros de datos del flujo de datos y los entrega al código de la aplicación. Además, KCL proporciona funciones de recuperación, conmutación por error y equilibrio de carga. Para obtener más información,

consulte Desarrollo de consumidores personalizados con rendimiento compartido mediante KCL.

Sin embargo, en algunos casos, es posible que prefiera utilizar las API de Kinesis Data Streams. Por ejemplo, para implementar herramientas personalizadas para la supervisión o la depuración de los flujos de datos.

> **Important**
>
> Kinesis Data Streams admite cambios en el periodo de retención de los registros de datos del flujo de datos. Para obtener más información, consulte Cambiar el periodo de retención de datos.

Uso de iteradores de fragmentos

Puede recuperar registros desde la secuencia por fragmentos. Para cada fragmento y cada lote de registros obtenido de ese fragmento debe conseguir un iterador de fragmentos. El iterador de fragmentos se utiliza en el objeto `getRecordsRequest` para especificar el fragmento a partir del cual deben recuperarse los registros. El tipo asociado con el iterador de fragmentos determina el punto del fragmento a partir del cual deben recuperarse los registros (consulte la información que se incluye más adelante en esta sección para obtener más detalles). Antes de trabajar con el iterador de fragmentos, tendrá que recuperar el fragmento tal como se describe en DescribeStream API: en desuso.

Obtenga el iterador de fragmentos inicial con el método `getShardIterator`. Obtenga iteradores de fragmentos para lotes adicionales de registros utilizando el método `getNextShardIterator` del objeto `getRecordsResult` que devuelve el método `getRecords`. Un iterador de fragmentos es válido durante 5 minutos. Si utiliza un iterador de fragmentos mientras sea válido, obtendrá uno nuevo. Cada iterador de fragmentos mantiene su validez durante 5 minutos, incluso después de utilizarlo.

Para obtener el iterador de fragmentos inicial, cree instancias de `GetShardIteratorRequest` y páselas al método `getShardIterator`. Para configurar la solicitud, especifique la secuencia y el ID del fragmento. Para más información sobre cómo obtener los flujos en su cuenta de AWS, consulte Visualización de secuencias. Para obtener información sobre cómo obtener los fragmentos en una secuencia, consulte DescribeStream API: en desuso.

```
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();
```

Este código de muestra especifica `TRIM_HORIZON` como el tipo de iterador que se utiliza para obtener el iterador de fragmentos inicial. Este tipo de iterador implica que se deben devolver los registros y comenzar por el primer registro agregado a la partición, en lugar de comenzar por el registro agregado más recientemente, también denominado extremo. Los tipos de iteradores posibles son los siguientes:

- `AT_SEQUENCE_NUMBER`
- `AFTER_SEQUENCE_NUMBER`
- `AT_TIMESTAMP`
- `TRIM_HORIZON`
- `LATEST`

Para obtener más información, consulte ShardIteratorType.

Algunos tipos de iteradores requieren que se especifique un número de secuencia además del tipo, por ejemplo:

```
getShardIteratorRequest.setShardIteratorType("AT_SEQUENCE_NUMBER");
getShardIteratorRequest.setStartingSequenceNumber(specialSequenceNumber);
```

Después de obtener un registro mediante `getRecords`, puede obtener el número de secuencia del registro si llama al método `getSequenceNumber` del registro.

```
record.getSequenceNumber()
```

Además, el código que añade registros a la secuencia de datos puede obtener el número de secuencia para un registro añadido llamando a `getSequenceNumber` en el resultado de `putRecord`.

```
lastSequenceNumber = putRecordResult.getSequenceNumber();
```

Puede utilizar números secuenciales para garantizar que los registros tengan un orden estrictamente ascendente. Para obtener más información, consulte el código de ejemplo en Ejemplo de PutRecord.

Uso de GetRecords

Una vez que haya obtenido el iterador de fragmentos, cree una instancia de un objeto `GetRecordsRequest`. Especifique el iterador para la solicitud con el método `setShardIterator`.

También puede establecer el número de registros que quiera recuperar mediante el método `setLimit`. El número de registros que devuelve `getRecords` es siempre igual o inferior a este límite. Si no especifica este límite, `getRecords` devuelve 10 MB de registros recuperados. El código de muestra que aparece a continuación establece este límite en 25 registros.

Si no se devuelven, significa que no hay registros de datos disponibles actualmente en este fragmento con el número de secuencia al que hace referencia el iterador de fragmentos. En una situación así, la aplicación debe esperar una cantidad de tiempo adecuada para los orígenes de datos del flujo. Intente obtener datos de nuevo a partir del fragmento mediante el iterador de fragmentos que ha devuelto la llamada anterior a `getRecords`.

Pase la `getRecordsRequest` al método `getRecords` y capture el valor devuelto como un objeto `getRecordsResult`. Para obtener los registros de datos, llame al método `getRecords` en el objeto `getRecordsResult`.

```
GetRecordsRequest getRecordsRequest = new GetRecordsRequest();
getRecordsRequest.setShardIterator(shardIterator);
getRecordsRequest.setLimit(25);

GetRecordsResult getRecordsResult = client.getRecords(getRecordsRequest);
List<Record> records = getRecordsResult.getRecords();
```

Para prepararse para otra llamada a `getRecords`, obtenga el siguiente iterador de fragmentos desde `getRecordsResult`.

```
shardIterator = getRecordsResult.getNextShardIterator();
```

Para obtener resultados óptimos, suspenda la actividad durante al menos 1 segundo (1000 milisegundos) entre las llamadas a `getRecords` para evitar que se supere el límite de frecuencia de `getRecords`.

```
try {
  Thread.sleep(1000);
}
catch (InterruptedException e) {}
```

Normalmente, debe llamar a `getRecords` en bucle, incluso cuando recupere un solo registro en un entorno de pruebas. Una única llamada a `getRecords` podría devolver una lista de registros vacía, incluso si el fragmento contiene más registros en números secuenciales posteriores. Si ocurre esto, el `NextShardIterator` que se devuelve junto con la lista de registros vacía hace referencia a un número de secuencia posterior en el fragmento, y las llamadas sucesivas a `getRecords` acabarán por devolver los registros. El siguiente ejemplo ilustra el uso de un bucle.

Ejemplo: getRecords

El siguiente ejemplo de código refleja las sugerencias sobre `getRecords` que hemos planteado en esta sección, incluidas la realización de llamadas en bucle.

```
// 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();
}
```

Si utiliza Kinesis Client Library, esta podría hacer varias llamadas antes de devolver los datos. Este es el comportamiento intencionado según el diseño y no indica ningún problema con KCL o los datos.

Adaptación a un cambio de los fragmentos

Si `getRecordsResult.getNextShardIterator` devuelve `null`, indica que se ha producido una división o combinación de una partición que ha implicado esta partición. Esta partición se encuentra ahora en un estado `CLOSED` y se han leído todos los registros de datos disponibles de esta partición.

En este escenario, puede utilizar `getRecordsResult.childShards` para obtener información sobre las nuevas particiones secundarias de la partición que se procesa y que se crearon mediante la división o la combinación. Para obtener más información, consulte ChildShard.

En el caso de una división, los dos nuevos fragmentos tienen un `parentShardId` igual al ID de fragmento del fragmento que estuviera procesando anteriormente. El valor de `adjacentParentShardId` para ambos fragmentos es `null`.

En el caso de una fusión, el único fragmento nuevo creado por la fusión tiene un `parentShardId` igual al ID del fragmento de uno de los fragmentos de origen y un `adjacentParentShardId` igual al ID de fragmento del otro fragmento de origen. La aplicación ya ha leído todos los datos de uno de estos fragmentos. Este es el fragmento para el que `getRecordsResult.getNextShardIterator` ha devuelto `null`. Si el orden de los datos es importante en la aplicación, debe asegurarse de que esta también lea todos los datos del otro fragmento principal antes de leer datos nuevos del fragmento secundario creado por la fusión.

Si utiliza varios procesadores para recuperar los datos de la secuencia (por ejemplo, un procesador por fragmento) y se produce una división o fusión de fragmentos, debe aumentar o disminuir el número de procesadores para adaptarse a los cambios en el número de fragmentos.

Para obtener más información acerca de cómo realizar cambios en los fragmentos, incluida una explicación de los estados de los fragmentos, como `CLOSED`, consulte Cambio de los fragmentos de una secuencia.

Interacción con los datos mediante AWS Glue Schema Registry

Puede integrar los flujos de datos de Kinesis con AWS Glue Schema Registry. AWS Glue Schema Registry le permite descubrir, controlar y evolucionar de forma centralizada esquemas, además de garantizar que un esquema registrado valide de forma continua los datos generados. Un esquema define la estructura y el formato de un registro de datos. Un esquema es una especificación versionada para publicación, consumo o almacenamiento de confianza de datos. AWS Glue Schema Registry le permite mejorar la calidad de los datos de principio a fin y la gobernanza de los datos en las aplicaciones de streaming. Para obtener más información, consulte AWS Glue Schema Registry. Una de las formas de configurar esta integración es mediante la API de `GetRecords` Kinesis Data Streams, disponible en el SDK de Java de AWS.

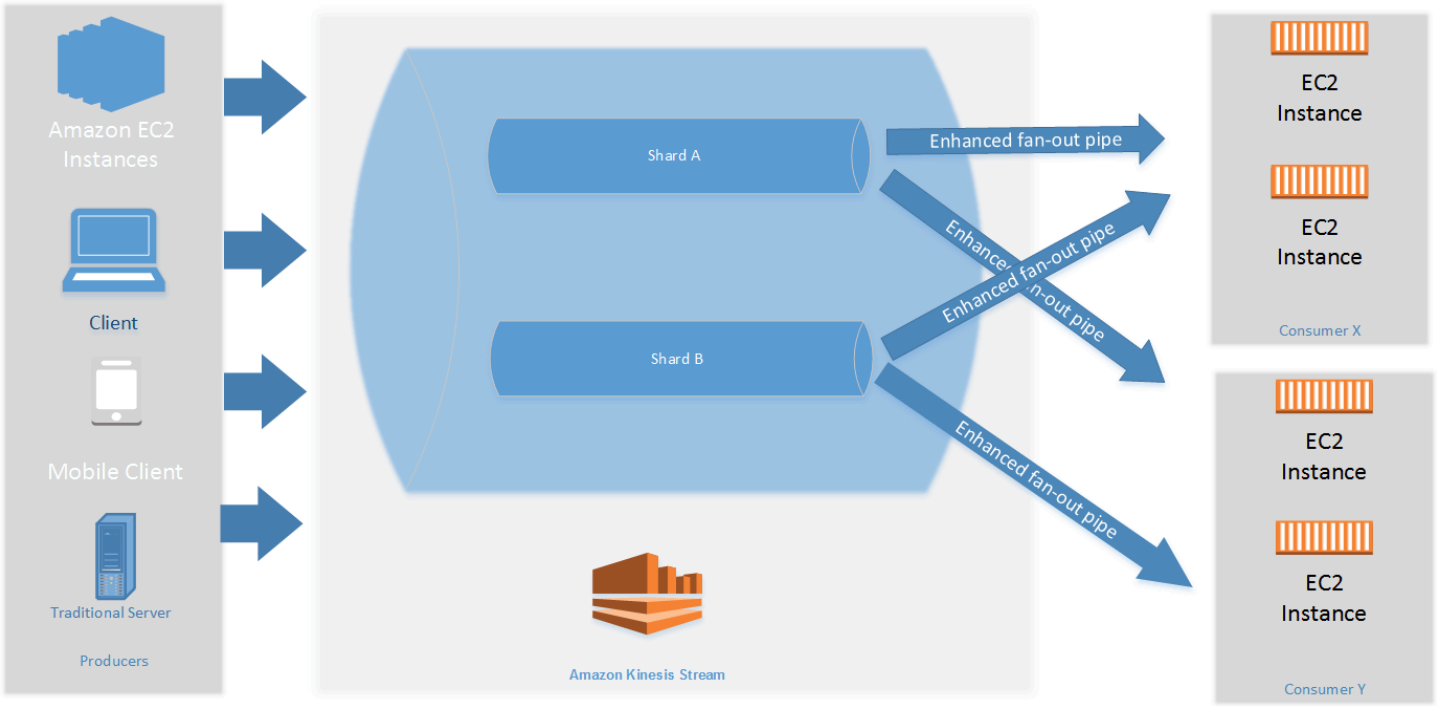
Para instrucciones detalladas sobre cómo configurar la integración de Kinesis Data Streams con Schema Registry mediante las API de Kinesis Data Streams `GetRecords`, consulte la sección “Interacción con datos mediante las API de Kinesis Data Streams” en Caso de uso: integración de Amazon Kinesis Data Streams con AWS Glue Schema Registry.

Desarrollo y uso de consumidores personalizados con rendimiento dedicado (Distribución ramificada mejorada)

En Amazon Kinesis Data Streams puede crear consumidores que utilicen una característica denominada distribución ramificada mejorada. Esta característica permite a los consumidores recibir registros de una transmisión con un rendimiento de hasta 2 MB de datos por segundo por fragmento. Este rendimiento está dedicado, lo que significa que los consumidores que utilizan la distribución ramificada mejorada no tienen que competir con otros consumidores que reciben datos del flujo. Kinesis Data Streams inserta registros de datos desde el flujo en los consumidores que utilizan la distribución ramificada mejorada. Por lo tanto, estos consumidores no necesitan sondear los datos.

> **Important**
>
> Es posible registrar hasta veinte consumidores por secuencia que utilicen la distribución ramificada mejorada.

En el siguiente diagrama, se muestra la arquitectura de distribución ramificada mejorada. Si utiliza la versión 2.0 o posterior de Amazon Kinesis Client Library (KCL) para crear un consumidor, KCL configura el consumidor para que utilice la distribución ramificada mejorada para recibir datos de todas las particiones del flujo. Si utiliza la API para crear un consumidor que utiliza la distribución ramificada mejorada, puede suscribirse a fragmentos individuales.



En el diagrama se muestra lo siguiente:

- Una secuencia con dos fragmentos.
- Dos consumidores que utilizan la distribución ramificada mejorada para recibir datos de la secuencia: Consumer X y Consumer Y. Ambos están suscritos a todos los fragmentos y a todos los registros de la secuencia. Si utiliza la versión 2.0 o posterior de KCL para crear un consumidor, KCL suscribe automáticamente ese consumidor a todas las particiones del flujo. Por otro lado, si utiliza la API para crear un consumidor, puede suscribirse a fragmentos individuales.
- Flechas que representan las canalizaciones de distribución ramificada mejorada que utilizan los consumidores para recibir datos de la secuencia. Una canalización de distribución ramificada mejorada proporciona hasta 2 MB/seg de datos por fragmento, independientemente de las otras canalizaciones o del número total de consumidores.

Temas

- Desarrollo de consumidores de distribución mejorados con KCL 2.x

- Desarrollo de consumidores de distribución ramificada mejorada con la API de Kinesis Data Streams
- Gestión de los consumidores de distribución mejorados con la AWS Management Console

Desarrollo de consumidores de distribución mejorados con KCL 2.x

Los consumidores que utilizan la distribución ramificada mejorada en Amazon Kinesis Data Streams pueden recibir los registros de un flujo de datos con un rendimiento dedicado de hasta 2 MB de datos por segundo por partición. Este tipo de consumidor no tiene que competir con otros consumidores que reciben datos de la secuencia. Para obtener más información, consulte Desarrollo y uso de consumidores personalizados con rendimiento dedicado (Distribución ramificada mejorada).

Puede utilizar la versión 2.0 o posterior de Kinesis Client Library (KCL) para desarrollar aplicaciones que utilicen la distribución ramificada mejorada para recibir datos de los flujos. KCL suscribe automáticamente la aplicación a todas las particiones de un flujo y garantiza que la aplicación de consumidor pueda leer con un valor de rendimiento de 2 MB/seg por partición. Si quiere utilizar KCL sin activar la distribución ramificada mejorada, consulte la página sobre desarrollo de consumidores mediante Kinesis Client Library 2.0.

Temas

- Desarrollo de consumidores de distribución mejorados con KCL 2.x en Java

Desarrollo de consumidores de distribución mejorados con KCL 2.x en Java

Puede utilizar la versión 2.0 o posterior de Kinesis Client Library (KCL) para desarrollar aplicaciones en Amazon Kinesis Data Streams para recibir datos de los flujos mediante la distribución ramificada mejorada. En el siguiente código, se muestra un ejemplo de implementación en Java de `ProcessorFactory` y `RecordProcessor`.

Es recomendable que utilice `KinesisClientUtil` para crear `KinesisAsyncClient` y para establecer `maxConcurrency` en `KinesisAsyncClient`.

> **Important**
>
> El cliente de Amazon Kinesis puede experimentar un importante aumento de la latencia, a menos que configure `KinesisAsyncClient` de forma que el valor de `maxConcurrency`

sea lo suficientemente alto para permitir todas las asignaciones, además de los usos adicionales de `KinesisAsyncClient`.

```
/*
 *  Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 *  Licensed under the Amazon Software License (the "License").
 *  You may not use this file except in compliance with the License.
 *  A copy of the License is located at
 *
 *  http://aws.amazon.com/asl/
 *
 *  or in the "license" file accompanying this file. This file is distributed
 *  on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
 *  express or implied. See the License for the specific language governing
 *  permissions and limitations under the License.
 */

/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 * Licensed under the Apache License, Version 2.0 (the "License").
 * You may not use this file except in compliance with the License.
 * A copy of the License is located at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * or in the "license" file accompanying this file. This file is distributed
 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
 * express or implied. See the License for the specific language governing
 * permissions and limitations under the License.
 */

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.UUID;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.ScheduledFuture;
```

```
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

import org.apache.commons.lang3.ObjectUtils;
import org.apache.commons.lang3.RandomStringUtils;
import org.apache.commons.lang3.RandomUtils;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.slf4j.MDC;

import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cloudwatch.CloudWatchAsyncClient;
import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;
import software.amazon.awssdk.services.kinesis.KinesisAsyncClient;
import software.amazon.awssdk.services.kinesis.model.PutRecordRequest;
import software.amazon.kinesis.common.ConfigsBuilder;
import software.amazon.kinesis.common.KinesisClientUtil;
import software.amazon.kinesis.coordinator.Scheduler;
import software.amazon.kinesis.exceptions.InvalidStateException;
import software.amazon.kinesis.exceptions.ShutdownException;
import software.amazon.kinesis.lifecycle.events.InitializationInput;
import software.amazon.kinesis.lifecycle.events.LeaseLostInput;
import software.amazon.kinesis.lifecycle.events.ProcessRecordsInput;
import software.amazon.kinesis.lifecycle.events.ShardEndedInput;
import software.amazon.kinesis.lifecycle.events.ShutdownRequestedInput;
import software.amazon.kinesis.processor.ShardRecordProcessor;
import software.amazon.kinesis.processor.ShardRecordProcessorFactory;

public class SampleSingle {

    private static final Logger log = LoggerFactory.getLogger(SampleSingle.class);

    public static void main(String... args) {
        if (args.length < 1) {
            log.error("At a minimum, the stream name is required as the first argument.
 The Region may be specified as the second argument.");
            System.exit(1);
        }

        String streamName = args[0];
        String region = null;
        if (args.length > 1) {
            region = args[1];
```

```
        }

        new SampleSingle(streamName, region).run();
    }

    private final String streamName;
    private final Region region;
    private final KinesisAsyncClient kinesisClient;

    private SampleSingle(String streamName, String region) {
        this.streamName = streamName;
        this.region = Region.of(ObjectUtils.firstNonNull(region, "us-east-2"));
        this.kinesisClient =
KinesisClientUtil.createKinesisAsyncClient(KinesisAsyncClient.builder().region(this.region));
    }

    private void run() {
        ScheduledExecutorService producerExecutor =
Executors.newSingleThreadScheduledExecutor();
        ScheduledFuture<?> producerFuture =
producerExecutor.scheduleAtFixedRate(this::publishRecord, 10, 1, TimeUnit.SECONDS);

        DynamoDbAsyncClient dynamoClient =
DynamoDbAsyncClient.builder().region(region).build();
        CloudWatchAsyncClient cloudWatchClient =
CloudWatchAsyncClient.builder().region(region).build();
        ConfigsBuilder configsBuilder = new ConfigsBuilder(streamName, streamName,
kinesisClient, dynamoClient, cloudWatchClient, UUID.randomUUID().toString(), new
SampleRecordProcessorFactory());

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

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

```
        System.out.println("Press enter to shutdown");
        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
        try {
            reader.readLine();
        } catch (IOException ioex) {
            log.error("Caught exception while waiting for confirm. Shutting down.",
ioex);
        }

        log.info("Cancelling producer, and shutting down executor.");
        producerFuture.cancel(true);
        producerExecutor.shutdownNow();

        Future<Boolean> gracefulShutdownFuture = scheduler.startGracefulShutdown();
        log.info("Waiting up to 20 seconds for shutdown to complete.");
        try {
            gracefulShutdownFuture.get(20, TimeUnit.SECONDS);
        } catch (InterruptedException e) {
            log.info("Interrupted while waiting for graceful shutdown. Continuing.");
        } catch (ExecutionException e) {
            log.error("Exception while executing graceful shutdown.", e);
        } catch (TimeoutException e) {
            log.error("Timeout while waiting for shutdown. Scheduler may not have
exited.");
        }
        log.info("Completed, shutting down now.");
    }

    private void publishRecord() {
        PutRecordRequest request = PutRecordRequest.builder()
                .partitionKey(RandomStringUtils.randomAlphabetic(5, 20))
                .streamName(streamName)
                .data(SdkBytes.fromByteArray(RandomUtils.nextBytes(10)))
                .build();
        try {
            kinesisClient.putRecord(request).get();
        } catch (InterruptedException e) {
            log.info("Interrupted, assuming shutdown.");
        } catch (ExecutionException e) {
            log.error("Exception while sending data to Kinesis. Will try again next
cycle.", e);
        }
    }
```

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


    private static class SampleRecordProcessor implements ShardRecordProcessor {

        private static final String SHARD_ID_MDC_KEY = "ShardId";

        private static final Logger log =
LoggerFactory.getLogger(SampleRecordProcessor.class);

        private String shardId;

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

        public void processRecords(ProcessRecordsInput processRecordsInput) {
            MDC.put(SHARD_ID_MDC_KEY, shardId);
            try {
                log.info("Processing {} record(s)",
processRecordsInput.records().size());
                processRecordsInput.records().forEach(r -> log.info("Processing record
pk: {} -- Seq: {}", r.partitionKey(), r.sequenceNumber()));
            } catch (Throwable t) {
                log.error("Caught throwable while processing records. Aborting.");
                Runtime.getRuntime().halt(1);
            } finally {
                MDC.remove(SHARD_ID_MDC_KEY);
            }
        }

        public void leaseLost(LeaseLostInput leaseLostInput) {
```

```
            MDC.put(SHARD_ID_MDC_KEY, shardId);
            try {
                log.info("Lost lease, so terminating.");
            } finally {
                MDC.remove(SHARD_ID_MDC_KEY);
            }
        }

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

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

}
```

Desarrollo de consumidores de distribución ramificada mejorada con la API de Kinesis Data Streams

La distribución ramificada mejorada es una característica de Amazon Kinesis Data Streams que permite a los consumidores recibir registros de un flujo de datos con un rendimiento dedicado de hasta 2 MB de datos por segundo por partición. Un consumidor que utiliza la distribución ramificada

mejorada no tiene que competir con otros consumidores que reciben datos de la secuencia. Para obtener más información, consulte Desarrollo y uso de consumidores personalizados con rendimiento dedicado (Distribución ramificada mejorada).

Puede utilizar las operaciones de la API para crear un consumidor que utilice la distribución ramificada mejorada en Kinesis Data Streams.

Para registrar un consumidor con distribución ramificada mejorada mediante la API de Kinesis Data Streams

1. Llame a RegisterStreamConsumer para registrar la aplicación como un consumidor que utiliza la distribución ramificada mejorada. Kinesis Data Streams genera un nombre de recurso de Amazon (ARN) para el consumidor y lo devuelve en la respuesta.
2. Para iniciar la escucha de una partición específica, traslade el ARN del consumidor en una llamada a SubscribeToShard. A continuación, Kinesis Data Streams comienza a enviarle los registros de esa partición en forma de eventos de tipo SubscribeToShardEvent a través de una conexión HTTP/2. La conexión permanece abierta durante un máximo de 5 minutos. Llame de nuevo a SubscribeToShard si desea continuar recibiendo registros del fragmento después de que el `future` que devuelve la llamada a SubscribeToShard finalice con normalidad o con excepciones.

   > **Note**
   >
   > La API `SubscribeToShard` también devuelve la lista de las particiones secundarias de la partición actual cuando se alcanza el final de la partición actual.

3. Para anular el registro de un consumidor que utiliza la distribución ramificada mejorada, llame a DeregisterStreamConsumer.

El código siguiente es un ejemplo de cómo suscribir el consumidor a un fragmento, renovar la suscripción de forma periódica y controlar los eventos.

```
    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);
        }
    }
```

Si `event.ContinuationSequenceNumber` devuelve `null`, indica que se ha producido una división o combinación de una partición que ha implicado esta partición. Esta partición se encuentra ahora en un estado `CLOSED`, y se han leído todos los registros de datos disponibles de esta partición. En este escenario, según el ejemplo anterior, puede utilizar `event.childShards` para obtener información sobre las nuevas particiones secundarias de la partición que se procesa y que se crearon mediante la división o la combinación. Para obtener más información, consulte ChildShard.

Interacción con los datos mediante AWS Glue Schema Registry

Puede integrar los flujos de datos de Kinesis con AWS Glue Schema Registry. AWS Glue Schema Registry le permite descubrir, controlar y evolucionar de forma centralizada esquemas, además de garantizar que un esquema registrado valide de forma continua los datos generados. Un esquema define la estructura y el formato de un registro de datos. Un esquema es una especificación versionada para publicación, consumo o almacenamiento de confianza de datos. AWS Glue Schema Registry le permite mejorar la calidad de los datos de principio a fin y la gobernanza de los datos en las aplicaciones de streaming. Para obtener más información, consulte AWS Glue Schema Registry. Una de las formas de configurar esta integración es mediante la API de `GetRecords` Kinesis Data Streams, disponible en el SDK de Java de AWS.

Para instrucciones detalladas sobre cómo configurar la integración de Kinesis Data Streams con Schema Registry mediante las API de Kinesis Data Streams `GetRecords`, consulte la sección “Interacción con datos mediante las API de Kinesis Data Streams” en Caso de uso: integración de Amazon Kinesis Data Streams con AWS Glue Schema Registry.

Gestión de los consumidores de distribución mejorados con la AWS Management Console

Los consumidores que utilizan la distribución ramificada mejorada en Amazon Kinesis Data Streams pueden recibir los registros de un flujo de datos con un rendimiento dedicado de hasta 2 MB de datos por segundo por partición. Para obtener más información, consulte Desarrollo y uso de consumidores personalizados con rendimiento dedicado (Distribución ramificada mejorada).

Puede utilizar la AWS Management Console para ver una lista de todos los consumidores que están registrados para utilizar la distribución ramificada mejorada con una secuencia específica. Podrá ver detalles como el ARN, el estado, la fecha de creación y las métricas de supervisión de estos consumidores.

Para ver los consumidores que están registrados para utilizar la distribución ramificada mejorada, así como su estado, fecha de creación y métricas en la consola

1. Inicie sesión en la AWS Management Console y abra la consola de Kinesis en https://console.aws.amazon.com/kinesis.
2. Elija Data Streams (Secuencias de datos) en el panel de navegación.
3. Elija un flujo de datos de Kinesis para ver sus detalles.
4. En la página de detalles de la secuencia, elija la pestaña Enhanced fan-out (Distribución ramificada mejorada).
5. Elija un consumidor para ver su nombre, su estado y su fecha de registro.

Para anular el registro de un consumidor

1. Abra la consola de Kinesis en https://console.aws.amazon.com/kinesis.
2. Elija Data Streams (Secuencias de datos) en el panel de navegación.
3. Elija un flujo de datos de Kinesis para ver sus detalles.
4. En la página de detalles de la secuencia, elija la pestaña Enhanced fan-out (Distribución ramificada mejorada).
5. Seleccione la casilla situada a la izquierda del nombre de cada uno de los consumidores cuyo registro desea anular.
6. Elija Deregister consumer (Anular registro del consumidor).

Migración de consumidores de KCL 1.x a KCL 2.x

En este tema se explican las diferencias entre las versiones 1.x y 2.x de Kinesis Client Library (KCL). También le muestra cómo migrar el consumidor de la versión 1.x a la versión 2.x de KCL. Después de migrar su cliente, comenzará el procesamiento de registros a partir de la última ubicación del punto de comprobación.

La versión 2.0 de KCL presenta los siguientes cambios en la interfaz:

Cambios en la interfaz de KCL

Interfaz de KCL 1.x	Interfaz de KCL 2.0
`com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor`	`software.amazon.kinesis.processor.ShardRecordProcessor`
`com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessorFactory`	`software.amazon.kinesis.processor.ShardRecordProcessorFactory`
`com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IShutdownNotificationAware`	Plegado en `software.amazon.kinesis.processor.ShardRecordProcessor`

Temas

Migración del procesador de registros

En el siguiente ejemplo, se muestra un procesador de registros implementado para la versión 1.x de KCL:

```
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();
        }
    }
}
```

Para migrar la clase del procesador de registros

1. Cambie las interfaces de `com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor` y `com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IShutdownNotificat` a `software.amazon.kinesis.processor.ShardRecordProcessor`, tal y como se indica a continuación:

   ```
   // 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. Actualice las instrucciones `import` para los métodos `initialize` y `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. Sustituya el método `shutdown` por los métodos nuevos siguientes: `leaseLost`, `shardEnded` y `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();
        }
    }
```

A continuación, se muestra la versión actualizada de la clase del procesador de registros.

```
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();
        }
    }
}
```

Migración del generador de procesadores de registros

El generador de procesadores de registros es responsable de la creación de procesadores de registros cuando se adquiere una asignación. A continuación, se muestra un ejemplo de un generador de la versión 1.x de KCL.

```
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();
    }
}
```

Para migrar el generador de procesadores de registros

1. Cambie la interfaz implementada de `com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessorFa` a `software.amazon.kinesis.processor.ShardRecordProcessorFactory`, tal y como se indica a continuación:

```
// 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. Cambie la firma de retorno de `createProcessor`.

```
// public IRecordProcessor createProcessor() {
public ShardRecordProcessor shardRecordProcessor() {
```

A continuación, se muestra un ejemplo de generador de procesadores de registros de la versión 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();
    }
}
```

Migración del proceso de trabajo

En la versión 2.0 de KCL, una nueva clase, llamada `Scheduler`, sustituye a la clase `Worker`. A continuación, se muestra un ejemplo de proceso de trabajo de la versión 1.x de KCL.

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

Para migrar el proceso de trabajo

1. Cambie la instrucción `import` para la clase `Worker` por las instrucciones de importación para las clases `Scheduler` y `ConfigsBuilder`.

   ```
   // import com.amazonaws.services.kinesis.clientlibrary.lib.worker.Worker;
   import software.amazon.kinesis.coordinator.Scheduler;
   import software.amazon.kinesis.common.ConfigsBuilder;
   ```

2. Cree un `ConfigsBuilder` y un `Scheduler` como se muestra en el ejemplo siguiente.

   Es recomendable que utilice `KinesisClientUtil` para crear `KinesisAsyncClient` y para establecer `maxConcurrency` en `KinesisAsyncClient`.

   > **Important**
   >
   > El cliente de Amazon Kinesis puede experimentar un importante aumento de la latencia, a menos que configure `KinesisAsyncClient` de forma que el valor de `maxConcurrency` sea lo suficientemente alto para permitir todas las asignaciones, además de los usos adicionales de `KinesisAsyncClient`.

   ```
   import java.util.UUID;

   import software.amazon.awssdk.regions.Region;
   import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;
   import software.amazon.awssdk.services.cloudwatch.CloudWatchAsyncClient;
   ```

```
import software.amazon.awssdk.services.kinesis.KinesisAsyncClient;
import software.amazon.kinesis.common.ConfigsBuilder;
import software.amazon.kinesis.common.KinesisClientUtil;
import software.amazon.kinesis.coordinator.Scheduler;

...

Region region = Region.AP_NORTHEAST_2;
KinesisAsyncClient kinesisClient =
 KinesisClientUtil.createKinesisAsyncClient(KinesisAsyncClient.builder().region(region));
DynamoDbAsyncClient dynamoClient =
 DynamoDbAsyncClient.builder().region(region).build();
CloudWatchAsyncClient cloudWatchClient =
 CloudWatchAsyncClient.builder().region(region).build();

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

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

Configuración del cliente de Amazon Kinesis

Con el lanzamiento de la versión 2.0 de Kinesis Client Library, la configuración del cliente pasa de tener una única clase de configuración (`KinesisClientLibConfiguration`) a tener seis clases de configuración. En la tabla siguiente, se describe la migración.

Campos de configuración y sus clases nuevas

Campo original	Clase de configuración nueva	Descripción
`applicationName`	`ConfigsBuilder`	El nombre de la aplicación de KCL. Se utiliza de forma predeterminada para `tableName` y `consumerName` .
`tableName`	`ConfigsBuilder`	Permite sustituir el nombre de la tabla que se utiliza para la tabla de asignaciones de Amazon DynamoDB.
`streamName`	`ConfigsBuilder`	El nombre de la secuencia cuyos registros procesa de esta aplicación.
`kinesisEndpoint`	`ConfigsBuilder`	Esta opción se ha eliminado. Consulte Eliminación de los parámetros de configuración de clientes.
`dynamoDBEndpoint`	`ConfigsBuilder`	Esta opción se ha eliminado. Consulte Eliminación de los parámetros de configuración de clientes.
`initialPositionInStreamExtended`	`RetrievalConfig`	La ubicación de la partición desde la que KCL comienza a recuperar registros, empezando por la ejecución inicial de la aplicación.
`kinesisCredentialsProvider`	`ConfigsBuilder`	Esta opción se ha eliminado. Consulte Eliminación de los parámetros de configuración de clientes.
`dynamoDBCredentialsProvider`	`ConfigsBuilder`	Esta opción se ha eliminado. Consulte Eliminación de los parámetros de configuración de clientes.
`cloudWatchCredentialsProvider`	`ConfigsBuilder`	Esta opción se ha eliminado. Consulte Eliminación de los parámetros de configuración de clientes.

Campo original	Clase de configuración nueva	Descripción
`failoverTimeMillis`	LeaseManagementConfig	El número de milisegundos que deben transcurrir antes de que se pueda considerar que se ha producido un error en el propietario de una asignación.
`workerIdentifier`	`ConfigsBuilder`	Identificador único que representa esta instancia del procesador de aplicaciones. Deben ser único.
`shardSyncIntervalMillis`	LeaseManagementConfig	El tiempo entre llamadas de sincronización del fragmento.
`maxRecords`	`PollingConfig`	Permite establecer el número máximo de registros que devuelve Kinesis.
`idleTimeBetweenReadsInMillis`	`CoordinatorConfig`	Esta opción se ha eliminado. Consulte Eliminación del tiempo de inactividad.
`callProcessRecordsEvenForEmptyRecordList`	`ProcessorConfig`	Cuando se establece, se llama al procesador de registros incluso cuando no se proporciona ningún registro de Kinesis.
`parentShardPollIntervalMillis`	`CoordinatorConfig`	Determina la frecuencia con que debería sondear un procesador de registros para ver si el fragmento principal se ha completado.
`cleanupLeasesUponShardCompletion`	`LeaseManagementConfig`	Cuando se establece, se eliminan las asignaciones tan pronto como se inicia el procesamiento del fragmento secundario.

Campo original	Clase de configuración nueva	Descripción
`ignoreUnexpectedChildShards`	`LeaseManagementConfig`	Cuando se establece, los fragmentos secundarios que tienen un fragmento abierto se pasan por alto. Esto es principalmente para DynamoDB Streams.
`kinesisClientConfig`	`ConfigsBuilder`	Esta opción se ha eliminado. Consulte Eliminación de los parámetros de configuración de clientes.
`dynamoDBClientConfig`	`ConfigsBuilder`	Esta opción se ha eliminado. Consulte Eliminación de los parámetros de configuración de clientes.
`cloudWatchClientConfig`	`ConfigsBuilder`	Esta opción se ha eliminado. Consulte Eliminación de los parámetros de configuración de clientes.
`taskBackoffTimeMillis`	`LifecycleConfig`	El tiempo que se debe esperar para reintentar la tareas con errores.
`metricsBufferTimeMillis`	`MetricsConfig`	Controla la publicación de métricas de CloudWatch.
`metricsMaxQueueSize`	`MetricsConfig`	Controla la publicación de métricas de CloudWatch.
`metricsLevel`	`MetricsConfig`	Controla la publicación de métricas de CloudWatch.
`metricsEnabledDimensions`	`MetricsConfig`	Controla la publicación de métricas de CloudWatch.
`validateSequenceNumberBeforeCheckpointing`	`CheckpointConfig`	Esta opción se ha eliminado. Consulte Validación del número de secuencia del punto de comprobación.

Campo original	Clase de configuración nueva	Descripción
`regionName`	`ConfigsBuilder`	Esta opción se ha eliminado. Consulte Eliminación de los parámetros de configuración de clientes.
`maxLeasesForWorker`	`LeaseManagementConfig`	El número máximo de asignaciones que debería aceptar una sola instancia de la aplicación.
`maxLeasesToStealAtOneTime`	`LeaseManagementConfig`	El número máximo de asignaciones del que debería intentar apropiarse una aplicación al mismo tiempo.
`initialLeaseTableReadCapacity`	`LeaseManagementConfig`	El número de IOPS de lectura de DynamoDB que se utiliza cuando Kinesis Client Library debe crear una tabla de asignaciones de DynamoDB.
`initialLeaseTableWriteCapacity`	`LeaseManagementConfig`	El número de IOPS de lectura de DynamoDB que se utiliza cuando Kinesis Client Library debe crear una tabla de asignaciones de DynamoDB.
`initialPositionInStreamExtended`	LeaseManagementConfig	La posición inicial de la secuencia en la que debería comenzar la aplicación. Esto solo se utiliza durante la creación inicial de la asignación.
`skipShardSyncAtWorkerInitializationIfLeasesExist`	`CoordinatorConfig`	Deshabilita la sincronización de los datos de los fragmentos si la tabla de asignaciones todavía contiene entradas. TODO: KinesisEco-438
`shardPrioritization`	`CoordinatorConfig`	La priorización de fragmentos que se va a utilizar.
`shutdownGraceMillis`	N/A	Esta opción se ha eliminado. Consulte Eliminaciones de MultiLang.

Campo original	Clase de configuración nueva	Descripción
`timeoutInSeconds`	N/A	Esta opción se ha eliminado. Consulte Eliminaciones de MultiLang.
`retryGetRecordsInSeconds`	`PollingConfig`	Configura el retraso entre los intentos de GetRecords cuando se producen errores.
`maxGetRecordsThreadPool`	`PollingConfig`	El tamaño del grupo de subprocesos utilizando para GetRecords.
`maxLeaseRenewalThreads`	`LeaseManagementConfig`	Controla el tamaño del grupo de subprocesos del renovador de asignaciones. Cuanto más grande sea el número de asignaciones que puede tomar la aplicación, más grande debe ser este grupo.
`recordsFetcherFactory`	`PollingConfig`	Permite sustituir el generador que se utiliza para crear capturadores que recuperan datos de las secuencias.
`logWarningForTaskAfterMillis`	`LifecycleConfig`	Tiempo que se debe esperar antes de registrar una advertencia si una tarea no ha finalizado.
`listShardsBackoffTimeInMillis`	`RetrievalConfig`	El número de milisegundos que se debe esperar entre llamadas a `ListShards` cuando se producen errores.
`maxListShardsRetryAttempts`	`RetrievalConfig`	El número máximo de veces que se reintenta `ListShards` antes de desistir.

Eliminación del tiempo de inactividad

En la versión 1.x de KCL, `idleTimeBetweenReadsInMillis` correspondía a dos cantidades:

- La cantidad de tiempo entre envíos de tareas. Ahora puede configurar este tiempo entre tareas estableciendo `CoordinatorConfig#shardConsumerDispatchPollIntervalMillis`.
- La cantidad de tiempo en reposo cuando no se devuelven registros desde Kinesis Data Streams. En la versión 2.0, con la distribución ramificada mejorada, los registros se envían desde sus respectivos recuperadores. Solo se produce actividad en el consumidor del fragmento cuando se recibe una solicitud enviada.

Eliminación de los parámetros de configuración de clientes

En la versión 2.0, KCL ya no crea los clientes. El usuario es el responsable de suministrar un cliente válido. Con este cambio, se han eliminado todos los parámetros de configuración que controlaban la creación de clientes. Si necesita estos parámetros, puede establecerlos en los clientes antes de proporcionar clientes a `ConfigsBuilder`.

Campo eliminado	Configuración equivalente
`kinesisEndpoint`	Configure el SDK `KinesisAsyncClient` con el punto de enlace preferido: `KinesisAsyncClient.builder().endpointOverride(URI.create("https://<kinesis endpoint>")).build()` .
`dynamoDBEndpoint`	Configure el SDK `DynamoDbAsyncClient` con el punto de enlace preferido: `DynamoDbAsyncClient.builder().endpointOverride(URI.create("https://<dynamodb endpoint>")).build()` .
`kinesisClientConfig`	Configure el SDK `KinesisAsyncClient` con la configuración necesaria: `KinesisAsyncClient.builder().overrideConfiguration(<your configuration>).build()` .
`dynamoDBClientConfig`	Configure el SDK `DynamoDbAsyncClient` con la configuración necesaria: `DynamoDbAsyncClient.builder().overrideConfiguration(<your configuration>).build()` .
`cloudWatchClientConfig`	Configure el SDK `CloudWatchAsyncClient` con la configuración necesaria: `CloudWatchAsyncClient.builder().overrideConfiguration(<your configuration>).build()` .

Campo eliminado	Configuración equivalente
`regionName`	Configure el SDK con la región preferida. Es la misma para todos los clientes del SDK. Por ejemplo, `KinesisAsyncClient.builder().region(Region.US_WEST_2).build()` .

Uso de otros servicios de AWS para leer datos de Kinesis Data Streams

A continuación se muestra una lista de otros servicios de AWS que pueden integrarse directamente con Kinesis Data Streams para leer datos de Kinesis Data Streams:

Temas

- Uso de Amazon EMR
- Uso de Amazon EventBridge Pipes
- Uso de AWS Glue
- Uso de Amazon Redshift

Uso de Amazon EMR

Los clústeres de Amazon EMR pueden leer y procesar las transmisiones de Amazon Kinesis directamente mediante herramientas conocidas del ecosistema de Hadoop, como Hive, Pig MapReduce, la API de streaming de Hadoop y Cascading. También puede unir los datos en tiempo real de Amazon Kinesis con los datos existentes en Amazon S3, Amazon DynamoDB y HDFS en un clúster en ejecución. Puede cargar datos directamente de Amazon EMR en Amazon S3 o en DynamoDB para actividades posteriores al procesamiento.

Para obtener más información, consulte Amazon Kinesis en la Guía de publicación de Amazon EMR.

Uso de Amazon EventBridge Pipes

Amazon EventBridge Pipes admite Amazon Kinesis Data Streams como fuente. Amazon EventBridge Pipes te ayuda a crear point-to-point integraciones entre los productores de eventos y los consumidores con pasos opcionales de transformación, filtrado y enriquecimiento. Puede usar

EventBridge Pipes para recibir registros en un Kinesis Data Stream y, si lo desea, filtrar o mejorar estos registros antes de enviarlos a uno de los destinos disponibles para su procesamiento, incluido Kinesis Data Streams.

Para obtener más información, consulte Amazon Kinesis stream como fuente en la Amazon EventBridge Release Guide.

Uso de AWS Glue

Con AWS Glue streaming ETL, puede crear trabajos de extracción, transformación y carga (ETL) de streaming que se ejecutan de forma continua y consumen datos de Amazon Kinesis Data Streams. Los trabajos limpian y transforman los datos y, a continuación, cargan los resultados en los lagos de datos de Amazon S3 o en los almacenes de datos JDBC.

Para obtener más información, consulte Streaming de trabajos ETL en AWS Glue en la Guía de versiones de AWS Glue.

Uso de Amazon Redshift

Amazon Redshift admite la ingesta de streaming desde Amazon Kinesis Data Streams. La característica de ingesta de streaming de Amazon Redshift proporciona una ingesta de datos de streaming de baja latencia y alta velocidad desde Amazon Kinesis Data Streams a una vista materializada de Amazon Redshift. La ingesta de streaming de Amazon Redshift elimina la necesidad de organizar los datos en Amazon S3 antes de ingerirlos en Amazon Redshift.

Para obtener más información, consulte Ingesta de streaming en la Guía de versiones de Amazon Redshift.

Uso de integraciones de terceros

Puede leer los datos de las transmisiones de datos de Amazon Kinesis Data Streams mediante una de las siguientes opciones de terceros que se integran con Kinesis Data Streams:

Temas

- Apache Flink
- Adobe Experience Platform
- Apache Druid
- Apache Spark

- Databricks
- Kafka Confluent Platform
- Kinesumer
- Talend

Apache Flink

Apache Flink es un marco y motor de procesamiento distribuido popular para computación con estado sobre flujos de datos ilimitados y delimitados. Para obtener más información sobre el consumo de Kinesis Data Streams mediante Apache Flink, consulte Amazon Kinesis Data Streams Connector.

Adobe Experience Platform

Adobe Experience Platform permite a las organizaciones centralizar y estandarizar los datos de los clientes de cualquier sistema. Luego, aplica la ciencia de datos y el machine learning para mejorar drásticamente el diseño y la entrega de experiencias enriquecidas y personalizadas. Para obtener más información sobre el consumo de transmisiones de datos de Kinesis mediante Adobe Experience Platform, consulte el conector de Amazon Kinesis.

Apache Druid

Druid es una base de datos de análisis en tiempo real de alto rendimiento que permite hacer consultas en fracciones de segundo a datos en streaming y por lotes a escala y bajo carga. Para obtener más información sobre la ingesta de transmisiones de datos de Kinesis mediante Apache Druid, consulte Amazon Kinesis Ingestion.

Apache Spark

Apache Spark es un motor de análisis unificado para el procesamiento de datos a gran escala. Proporciona API de alto nivel en Java, Scala, Python y R, y un motor optimizado que admite gráficos de ejecución general. Puede usar Apache Spark para crear aplicaciones de procesamiento de transmisiones que consuman los datos de sus transmisiones de datos de Kinesis.

Para consumir transmisiones de datos de Kinesis mediante Apache Spark Structured Streaming, utilice el conector Amazon Kinesis Data Streams. Este conector admite el consumo gracias a la función de distribución mejorada, que proporciona a la aplicación un rendimiento de lectura específico de hasta 2 MB de datos por segundo por fragmento. Para obtener más información,

consulte Desarrollo de consumidores personalizados con un rendimiento específico (expansión mejorada).

Para consumir transmisiones de datos de Kinesis mediante Spark Streaming, consulte Spark Streaming + Kinesis Integration.

Databricks

Databricks es una plataforma basada en la nube que proporciona un entorno colaborativo para la ingeniería de datos, la ciencia de datos y el machine learning. Para obtener más información sobre el consumo de transmisiones de datos de Kinesis mediante Databricks, consulte Connect to Amazon Kinesis.

Kafka Confluent Platform

Confluent Platform se basa en Kafka y ofrece funciones y características adicionales que ayudan a las empresas a crear y administrar flujos de datos y aplicaciones de streaming en tiempo real. Para obtener más información sobre el consumo de transmisiones de datos de Kinesis mediante la plataforma Confluent, consulte Amazon Kinesis Source Connector for Confluent Platform.

Kinesumer

Kinesumer es un cliente de Go que implementa un cliente de grupo de consumidores distribuido del lado del cliente para las transmisiones de datos de Kinesis. Para obtener más información, consulte el repositorio de GitHub de Kinesumer.

Talend

Talend es un software de integración y administración de datos que permite a los usuarios recopilar, transformar y conectar datos de diversos orígenes de forma escalable y eficiente. Para obtener más información sobre el consumo de transmisiones de datos de Kinesis mediante Talend, consulte Conectar talend a una transmisión de Amazon Kinesis.

Solución de problemas de los consumidores de secuencias de datos de Kinesis

En las secciones siguientes se ofrecen soluciones a algunos problemas comunes que puede encontrar al usar consumidores de Amazon Kinesis Data Streams.

Algunos registros de Kinesis Data Streams se omiten al usar Kinesis Client Library

La causa más frecuente de la omisión de registros es que haya una excepción de `processRecords` no administrada. Kinesis Client Library (KCL) depende del código `processRecords` para administrar cualquier excepción que surja del procesamiento de los registros de datos. Cualquier excepción que se origine en `processRecords` se absorbe en KCL. Para evitar los reintentos infinitos sobre un error recurrente, KCL no reenvía el lote de registros procesados en el momento de la excepción. A continuación, KCL llama a `processRecords` para solicitar el siguiente lote de registros de datos sin reiniciar el procesador de registros. Esto da como resultado que en las aplicaciones consumidoras se observen registros omitidos. Para evitar que se omitan registros, administre todas las excepciones de `processRecords` convenientemente.

Registros que pertenecen al mismo fragmento se procesan en distintos procesadores de registros a la vez

Para cualquier aplicación de Kinesis Client Library (KCL) en ejecución, una partición solo tiene un propietario. Sin embargo, varios procesadores de registros podrían procesar el mismo fragmento temporalmente. En el caso de una instancia de un proceso de trabajo que pierda la conectividad de red, KCL asume que el proceso de trabajo inaccesible ya no está procesando registros, después de que pase el tiempo de conmutación por error, y da instrucciones a otras instancias de procesos de trabajo para sustituirla. Durante un breve periodo, los nuevos procesadores de registros y los procesadores de registros del proceso de trabajo inaccesible pueden procesar datos procedentes del mismo fragmento.

Debe definir un tiempo de conmutación por error que sea adecuado para su aplicación. En el caso de las aplicaciones de baja latencia, el valor predeterminado de 10 segundos puede representar el tiempo máximo que desee esperar. Sin embargo, en aquellos casos en los que prevea que se producirán problemas de conectividad, como al hacer llamadas en zonas geográficas en las que la conectividad se podría perder con más frecuencia, puede que este número sea demasiado bajo.

Su aplicación debe anticiparse a esta situación y administrarla, especialmente debido a que la conectividad de red normalmente se restaura al proceso de trabajo previamente inaccesible. Si los fragmentos de un procesador de registros son ocupados por otro procesador de registros, debe afrontar los dos casos siguientes para cerrarse sin ocasionar problemas:

1. Después de que se complete la llamada actual a `processRecords`, KCL invoca el método de cierre en el procesador de registros con el motivo "ZOMBIE". Cabe esperar que sus procesadores de registros eliminen los recursos según corresponda y, a continuación, se cierren.
2. Si intenta crear un punto de comprobación en un proceso de trabajo "zombie", KCL producirá una `ShutdownException`. Tras recibir esta excepción, lo normal es que el código salga del método actual sin ocasionar problemas.

Para obtener más información, consulte Administración de registros duplicados.

La aplicación consumidora lee a una velocidad menor que lo esperado

Los motivos más comunes para que el rendimiento de lectura sea menor que lo esperado son los siguientes:

1. Varias aplicaciones consumidoras tienen lecturas totales que superan los límites por fragmento. Para obtener más información, consulte Cuotas y límites. En este caso, puede aumentar el número de particiones en el flujo de datos de Kinesis.
2. El límite que especifica el número máximo de comandos GetRecords por llamada puede haberse configurado con un valor bajo. Si utiliza KCL, es posible que haya configurado el proceso de trabajo con un valor bajo para la propiedad `maxRecords`. En general, recomendamos que utilice los valores predeterminados del sistema para esta propiedad.
3. La lógica de la llamada `processRecords` puede tardar más de lo previsto por una serie de posibles razones; la lógica puede requerir un uso intensivo de la CPU, bloquear la E/S o provocar un efecto embudo en la sincronización. Para probar si alguno de estos supuestos es cierto, realice ejecuciones de prueba de procesadores de registros vacíos y compare el rendimiento de lectura.

Para obtener información sobre cómo mantener la entrada de datos, consulte Cambio en los fragmentos, escalado y procesamiento paralelo.

Si tiene una única aplicación consumidora, siempre puede leer al menos dos veces más rápido que la velocidad de inclusión. Esto se debe a que puede escribir hasta 1000 registros por segundo para escrituras, hasta un máximo de escritura de datos de 1 MB por segundo (incluidas las claves de partición). Cada partición abierta admite hasta 5 transacciones por segundo en el caso de las lecturas, con una velocidad máxima total de lectura de datos de 2 MB por segundo. Tenga en cuenta que con cada lectura (llamada a GetRecords) se obtiene un lote de registros. El tamaño de los datos devueltos por GetRecords varía en función del uso del fragmento. El volumen máximo de datos que GetRecords puede devolver es de 10 MB. Si una llamada devuelve ese límite, las llamadas posteriores realizadas en los siguientes 5 segundos provocan una `ProvisionedThroughputExceededException`.

GetRecords Devuelve una matriz de registros vacía incluso cuando hay datos en la transmisión

El consumo o la obtención de registros se basa en un modelo de extracción. Se espera que los desarrolladores GetRecordsrealicen llamadas en un bucle continuo sin interrupciones. Cada llamada a GetRecords devuelve también un valor `ShardIterator` que debe utilizarse en la siguiente iteración del bucle.

La operación GetRecords no se bloquea. En su lugar, se devuelve de inmediato, con registros de datos relevantes o con un elemento `Records` vacío. Un elemento `Records` vacío se devuelve con dos condiciones:

1. No hay más datos actualmente en el fragmento.
2. No hay datos cerca de la parte del fragmento a la que apunta el `ShardIterator`.

La última condición es sutil, pero supone un equilibrio de diseño necesario para evitar el tiempo de búsqueda ilimitado (latencia) al recuperar registros. Por lo tanto, la aplicación que consume la secuencia debe proceder en bucle y llamar a GetRecords, ocupándose también de los registros vacíos.

En un escenario de producción, la única vez que se debería salir del bucle continuo es cuando el valor `NextShardIterator` es `NULL`. Cuando `NextShardIterator` es `NULL`, significa que el fragmento actual se ha cerrado y el valor `ShardIterator`, de lo contrario, apuntaría más allá del

último registro. Si la aplicación consumidora nunca llama a SplitShard o a MergeShards, el fragmento permanece abierto y las llamadas a GetRecords nunca devuelven para `NextShardIterator` el valor `NULL`.

Si utiliza Kinesis Client Library (KCL), el patrón de consumo anterior se abstrae automáticamente. Esto incluye la administración automática de un conjunto de fragmentos que cambian de forma dinámica. Con KCL, el desarrollador solo suministra la lógica para procesar registros de entrada. Esto es posible porque la biblioteca realiza automáticamente llamadas continuas a GetRecords.

El iterador de fragmentos caduca de forma inesperada

Con cada nueva solicitud GetRecords se devuelve un nuevo iterador de fragmentos (como `NextShardIterator`) que debe usar entonces en la siguiente solicitud GetRecords (como `ShardIterator`). Normalmente, este iterador de fragmentos no caduca antes de utilizarlo. Sin embargo, los iteradores de fragmentos pueden caducar por no haber llamado a GetRecords durante más de 5 minutos, o porque haber reiniciado la aplicación consumidora.

Si el iterador de particiones caduca inmediatamente, antes de poder utilizarlo, esto podría indicar que la tabla de DynamoDB que utiliza Kinesis no tiene capacidad suficiente para almacenar los datos arrendados. Es más probable que se dé esta situación si tiene un gran número de fragmentos. Para solucionar este problema, aumente la capacidad de escritura asignada a la tabla de fragmentos. Para obtener más información, consulte Uso de una tabla de arrendamiento para realizar el seguimiento de las particiones procesadas por la aplicación de consumo de KCL.

El procesamiento de registros del consumidor se queda atrás

En la mayoría de casos de uso, las aplicaciones consumidoras leen los datos más recientes de la secuencia. En determinadas circunstancias, puede que las lecturas del consumidor se queden atrás, lo que no es deseable. Tras identificar el retraso con el que están realizando las lecturas sus consumidores, consulte los motivos más comunes por los que estos se retrasan.

Comience por la métrica `GetRecords.IteratorAgeMilliseconds`, que controla la posición de lectura de todos los fragmentos y los consumidores de la secuencia. Tenga en cuenta que si la antigüedad de un iterador supera el 50 % del periodo de retención (con un valor predeterminado de 24 horas pero configurable hasta 365 días), existe el riesgo de pérdida de datos debido a la caducidad del registro. Una parche rápido es aumentar el periodo de retención. Así se detiene la pérdida de datos importantes mientras se realizan los pasos para solucionar el problema. Para obtener más información, consulte Supervisión del servicio Amazon Kinesis Data Streams con

Amazon CloudWatch. A continuación, identifique el retraso con el que su aplicación de consumo lee cada fragmento mediante una CloudWatch métrica personalizada emitida por la Biblioteca de clientes de Kinesis (KCL). `MillisBehindLatest` Para obtener más información, consulte Supervisión de Kinesis Client Library con Amazon CloudWatch.

Estos son los motivos más comunes por los que los consumidores se pueden retrasar:

- Los grandes aumentos repentinos de `GetRecords.IteratorAgeMilliseconds` o `MillisBehindLatest` suelen indicar un problema transitorio, como que la operación de la API provoca un error en una aplicación descendente. Debe investigar estos aumentos repentinos si alguna de las métricas muestra constantemente este comportamiento.
- Un incremento gradual de estas métricas indica que un consumidor no mantiene el ritmo de la secuencia porque no procesa los registros lo suficientemente rápido. Las causas más comunes para este comportamiento son la insuficiencia de recursos físicos o una lógica de procesamiento de registros que no está ajustada a un aumento en el rendimiento de la secuencia. Para comprobar este comportamiento, consulte las demás CloudWatch métricas personalizadas que la KCL emite asociadas a la `processTask` operación, incluidas, y. `RecordProcessor.processRecords.Time` `Success` `RecordsProcessed`
  - Si percibe un aumento en la métrica `processRecords.Time` que se correlaciona con una mejora en el rendimiento, debe analizar su lógica de procesamiento de registros para identificar por qué no se ajusta al aumento de rendimiento.
  - Si percibe un incremento de los valores `processRecords.Time` que no está correlacionado con el aumento de rendimiento, compruebe si está realizando llamadas de bloqueo en la ruta crítica, ya que suelen ser la causa de la reducción de velocidad en el procesamiento de registros. Otro enfoque consiste en aumentar el paralelismo incrementando el número de fragmentos. Por último, confirme que tiene una cantidad suficiente de recursos físicos (memoria, uso de CPU, etc.) en los nodos de procesamiento subyacentes durante los picos de demanda.

Error de permisos no autorizados para la clave maestra de KMS

Este error se produce cuando una aplicación consumidora lee desde una secuencia cifrada sin permisos para la clave maestra de KMS. Para asignar permisos a una aplicación de modo que pueda obtener acceso a una clave de KMS, consulte Uso de políticas de claves en AWS KMS y Uso de políticas de IAM con AWS KMS.

Problemas, preguntas e ideas de solución de problemas comunes para los consumidores

- ¿Por qué el desencadenador de Kinesis Data Streams no puede invocar mi función de Lambda?
- ¿Cómo puedo detectar y solucionar las ReadProvisionedThroughputExceeded excepciones en Kinesis Data Streams?
- ¿Por qué tengo problemas de alta latencia con Kinesis Data Streams?
- ¿Por qué mi flujo de datos de Kinesis devuelve un error de servidor interno 500?
- ¿Cómo puedo solucionar los problemas de una aplicación de KCL bloqueada para Kinesis Data Streams?
- ¿Puedo usar diferentes aplicaciones de Amazon Kinesis Client Library con la misma tabla de Amazon DynamoDB?

Temas avanzados para consumidores de Amazon Kinesis Data Streams

Aprenda a optimizar sus consumidores de Amazon Kinesis Data Streams.

Contenido

Procesamiento de baja latencia

El retraso de propagación se define como la latencia de un punto a otro desde el momento en que se escribe un registro en la secuencia hasta que la aplicación consumidora lo lee. Este retraso varía en función de una serie de factores, pero principalmente se ve afectado por el intervalo de sondeo de las aplicaciones consumidoras.

Para la mayoría de las aplicaciones, le recomendamos que se sondee cada fragmento una vez por segundo y aplicación. Esto le permite tener varias aplicaciones de consumo que procesen un

flujo simultáneamente sin alcanzar los límites de Amazon Kinesis Data Streams de cinco llamadas `GetRecords` por segundo. Además, el procesamiento de lotes grandes de datos suele ser más eficaz en la reducción de la latencia de red y otras latencias descendentes en su aplicación.

Los valores predeterminados en KCL se establecen para seguir la práctica recomendada de sondear cada segundo. Este valor predeterminado tiene como consecuencia retrasos promedios en la propagación que normalmente están por debajo de 1 segundo.

Los registros de Kinesis Data Streams están disponibles para su lectura inmediatamente después de que se escriben. Existen algunos casos de uso en los que es necesario aprovecharse de este factor y requieren que se consuman los datos de la secuencia en cuanto estén disponibles. Puede reducir significativamente el retraso de propagación si anula la configuración predeterminada de KCL para sondear con mayor frecuencia, tal y como se muestra en los siguientes ejemplos.

Código de configuración de KCL en Java:

```
kinesisClientLibConfiguration = new
        KinesisClientLibConfiguration(applicationName,
        streamName,
        credentialsProvider,

 workerId).withInitialPositionInStream(initialPositionInStream).withIdleTimeBetweenReadsInMilli
```

Configuración del archivo de propiedades para KCL en Python y Ruby:

```
idleTimeBetweenReadsInMillis = 250
```

> **Note**
>
> Dado que Kinesis Data Streams tiene un límite de cinco llamadas a `GetRecords` por segundo y por partición, si se establece en la propiedad `idleTimeBetweenReadsInMillis` un valor inferior a 200 ms puede ocurrir que la aplicación encuentre la excepción `ProvisionedThroughputExceededException`. Si se producen demasiadas de estas excepciones, pueden darse retrocesos exponenciales y, por lo tanto, causar latencias importantes e inesperadas en el procesamiento. Si configura esta propiedad para que sea de 200 ms o esté por encima de este valor y tiene más de una aplicación de procesamiento, experimentará una limitación controlada similar.

Uso de AWS Lambda con Kinesis Client Library

Kinesis Producer Library (KPL) agrega pequeños registros formateados por el usuario en registros de mayor tamaño, hasta 1 MB, para utilizar mejor el rendimiento de Amazon Kinesis Data Streams. Aunque KCL para Java admite la desagrupación de estos registros, tendrá que utilizar un módulo especial para desagrupar registros al utilizar AWS Lambda como el consumidor de los flujos. Puede obtener el código de proyecto necesario e instrucciones de GitHub en Módulos de desagrupación de Kinesis Producer Library para AWS Lambda. Los componentes de este proyecto le ofrecen la capacidad de procesar datos serializados por KPL dentro de AWS Lambda, en Java, Node.js y Python. Estos componentes también se pueden utilizar como parte de una aplicación KCL multilenguaje.

Cambio en los fragmentos, escalado y procesamiento paralelo

Los cambios en los fragmentos le permiten aumentar o reducir el número de fragmentos en una secuencia para adaptarse a los cambios en la velocidad del flujo de datos en la secuencia. Los cambios en los fragmentos los suele realizar una aplicación administrativa que monitorea las métricas de administración de datos en los fragmentos. Aunque la propia KCL no inicia las operaciones de cambios en las particiones, está diseñada para adaptarse a los cambios en el número de particiones que resultan de las mismas.

Como se indica en Uso de una tabla de arrendamiento para realizar el seguimiento de las particiones procesadas por la aplicación de consumo de KCL, KCL hace un seguimiento de las particiones en el flujo mediante una tabla de Amazon DynamoDB. Cuando se crean nuevas particiones como resultado de estos cambios, KCL descubre las nuevas particiones y rellena nuevas filas en la tabla. Los procesos de trabajo descubren automáticamente los nuevos fragmentos y crean procesadores para administrar los datos de los mismos. KCL también distribuye las particiones en el flujo, por todos los procesos de trabajo y procesadores de registros disponibles.

KCL garantiza que los datos existentes en particiones antes de que se produjeran cambios en los mismos se procesen primero. Después de que los datos se hayan procesado, los datos de los nuevos fragmentos se envían a los procesadores de registros. De esta forma, KCL conserva el orden en que los registros de datos se han agregado al flujo para una clave de partición en particular.

Ejemplo: Cambio en los fragmentos, escalado y procesamiento paralelo

El siguiente ejemplo ilustra cómo KCL ayuda a administrar el escalado y los cambios en las particiones:

- Por ejemplo, si su aplicación se ejecuta en una instancia de EC2 y está procesando un flujo de datos de Kinesis que tiene cuatro particiones. Esta instancia tiene un proceso de trabajo de KCL y cuatro procesadores de registros (un procesador de registros por partición). Estos cuatro procesadores de registros se ejecutan en paralelo en el mismo proceso.
- A continuación, si escala la aplicación para utilizar otra instancia, tendrá dos instancias procesando una secuencia con cuatro fragmentos. Cuando el proceso de trabajo de KCL se inicia en la segunda instancia, realiza un equilibrio de la carga con la primera instancia, de modo que cada instancia ahora procesará dos particiones.
- Si entonces decide dividir los cuatro fragmentos en cinco fragmentos. La nueva KCL vuelve a coordinar el procesamiento en las instancias: una instancia procesa tres particiones y la otra procesa dos particiones. Se produce una coordinación similar al fusionar fragmentos.

Normalmente, cuando utilice KCL, debe asegurarse de que el número de instancias no supere el número de particiones (excepto si quiere esperar a errores). Cada partición se procesa exactamente en un proceso de trabajo de KCL y tiene un procesador de registros correspondientes, por lo que no necesitará varias instancias para procesar una partición. Sin embargo, un proceso de trabajo puede procesar cualquier número de fragmentos, por lo que no hay problema si el número de fragmentos supera el número de instancias.

Para aumentar el procesamiento de su aplicación, debe probar una combinación de estos enfoques:

- Aumentar el tamaño de la instancia (porque todos los procesadores de registros se ejecutan en paralelo dentro de un proceso)
- Al aumentar el número de instancias hasta el número máximo de fragmentos abiertos (porque los fragmentos se pueden procesar de manera independiente)
- Al aumentar el número de fragmentos (lo que aumenta el nivel de paralelismo)

Tenga en cuenta que puede utilizar Auto Scaling para escalar automáticamente las instancias en función de las métricas adecuadas. Para obtener más información, consulte la Guía del usuario de Amazon EC2 Auto Scaling.

Cuando los cambios en los fragmentos aumentan el número de fragmentos en la secuencia, el aumento correspondiente en el número de procesadores de registros aumenta la carga de las instancias EC2 que los alojan. Si las instancias forman parte de un grupo de Auto Scaling y la carga aumenta lo suficiente, el grupo de Auto Scaling añade más instancias para administrar el aumento de la carga. Debe configurar las instancias para lanzar su aplicación de Amazon Kinesis Data Streams

en el inicio, de forma que los procesos de trabajo y los procesadores de registros adicionales se activen en la nueva instancia de inmediato.

Para obtener más información acerca de los cambios en los fragmentos, consulte Cambio de los fragmentos de una secuencia.

Administración de registros duplicados

Hay dos principales motivos por los que se podrían entregar registros más de una vez en la aplicación de Amazon Kinesis Data Streams: los reintentos de los productores y de los consumidores. Su aplicación debe prever y administrar de forma adecuada el procesamiento de registros individuales varias veces.

Reintentos en los productores

Piense en un productor que experimenta un tiempo de inactividad relacionado con un problema en la red después de hacer una llamada a `PutRecord`, pero antes de que pueda recibir la confirmación desde Amazon Kinesis Data Streams. El productor no puede asegurar que se ha entregado el registro a Kinesis Data Streams. Suponiendo que cada registro es importante para la aplicación, el productor habría sido escrito para volver a intentar la llamada con los mismos datos. Si ambas llamadas a `PutRecord` sobre los mismos datos se enviaron correctamente a Kinesis Data Streams, habrá dos registros de Kinesis Data Streams. Aunque los dos registros tendrán datos idénticos, también tendrán números secuenciales únicos. Las aplicaciones que necesitan garantías estrictas deben tener incrustada una clave maestra en el registro para eliminar los duplicados en etapas posteriores del procesamiento. Tenga en cuenta que el número de duplicados debidos a reintentos del productor suele ser bajo en comparación con el número de duplicados debidos a reintentos del consumidor.

> **Note**
>
> Si utiliza `PutRecord` del SDK de AWS, la configuración predeterminada volverá a intentar una llamada errónea a `PutRecord` un máximo de tres veces.

Reintentos en los consumidores

Los reintentos en los consumidores (aplicaciones de procesamiento de datos) ocurren cuando se reinician los procesadores de registros. Los procesadores de registros para un mismo fragmento se reinician en los siguientes casos:

1. Si un proceso de trabajo termina de forma inesperada
2. Si las instancias de los procesos de trabajo se agregan o eliminan
3. Si se fusionan o dividen fragmentos
4. Si se implementa la aplicación

En todos estos casos, el mapeo entre los fragmentos, los procesos de trabajo y los procesadores de registros se actualiza constantemente para lograr un procesamiento del balanceo de la carga. Los procesadores de fragmentos que se migraron a otras instancias reinician el procesamiento de registros a partir del último punto de comprobación. Esto se traduce en un procesamiento de registros duplicado, tal y como se muestra en el ejemplo siguiente. Para obtener más información sobre el balanceo de carga, consulte Cambio en los fragmentos, escalado y procesamiento paralelo.

Ejemplo: Reintentos de consumidores que dan como resultado la entrega doble de registros

En este ejemplo tiene una aplicación que lee continuamente los registros de un flujo, agrega registros en un archivo local y carga el archivo en Amazon S3. Para simplificar, supongamos que solo hay 1 fragmento y un proceso de trabajo que lo procesa. Fíjese en el siguiente ejemplo de secuencia de eventos, suponiendo que el último punto de comprobación se realizó en el número de registro 10 000:

1. Un proceso de trabajo lee el siguiente lote de registros del fragmento, los registros de 10 001 a 20 000.
2. A continuación, el proceso de trabajo transmite el lote de registros al procesador de registros asociado.
3. El procesador de registros agrega los datos, crea un archivo de Amazon S3 y carga el archivo a Amazon S3 correctamente.
4. El proceso de trabajo se cierra de forma inesperada antes de que pueda crearse un nuevo punto de comprobación.
5. La aplicación, el proceso de trabajo y el procesador de registros se reinician.
6. A partir de ahora, el proceso de trabajo comienza a leer desde el último punto de comprobación correcto, en este caso el 10 001.

Por lo tanto, los registros 10 001-20 000 se consumen más de una vez.

Resistencia a los reintentos en los consumidores

Aunque puede que los registros se procesen más de una vez, la aplicación podría presentar los efectos adversos como si los registros se hubieran procesado solo una vez (procesamiento idempotente). Las soluciones a este problema varían en cuanto a su complejidad y precisión. Si el destino final de los datos puede administrar bien los duplicados, le recomendamos que deje que dicho destino final se encargue del procesamiento idempotente. Por ejemplo, con Opensearch puede utilizar una combinación de control de versiones e ID únicos para evitar el procesamiento duplicado.

En el ejemplo de la aplicación de la sección anterior, lee continuamente los registros de un flujo, agrega los registros en un archivo local y carga el archivo en Amazon S3. Como se ha indicado, los registros 10 001-20 000 se consumen más de una vez, lo que da como resultado que haya varios archivos en Amazon S3 con los mismos datos. Una forma de mitigar los duplicados en este ejemplo consiste en garantizar que en el paso 3 se utilice el siguiente esquema:

1. El procesador de registros utiliza un número fijo de registros por archivo de Amazon S3; por ejemplo, 5000.
2. El nombre de archivo utiliza este esquema: prefijo de Amazon S3, ID de partición y `First-Sequence-Num`. En este caso, podría ser algo parecido a `sample-shard000001-10001`.
3. Después de cargar el archivo de Amazon S3, cree un punto de verificación especificando `Last-Sequence-Num`. En este caso, debería establecer puntos de comprobación en el número de registro 15 000.

Con este esquema, incluso si los registros se procesan más de una vez, el archivo de Amazon S3 resultante tendrá el mismo nombre y los mismos datos. Los reintentos solo dan como resultado la escritura de los mismos datos en el mismo archivo más de una vez.

En el caso de una operación de cambio en los fragmentos, el número de registros que quedan en el fragmento podría ser menor que el número fijo necesario. En este caso, el método `shutdown()` tiene que volcar el archivo a Amazon S3 y establecer puntos de verificación en el último número secuencial. El esquema anterior también es compatible con las operaciones de cambios en los fragmentos.

Administración del startup, el shutdown y la limitación controlada

A continuación, se muestran otras consideraciones que podría incorporar en el diseño de su aplicación de Amazon Kinesis Data Streams.

Contenido

Inicio de productores y consumidores de datos

De forma predeterminada, KCL comienza a leer registros desde el extremo del flujo, que es el registro agregado más recientemente. Con esta configuración, si una aplicación productora de datos agrega registros a la secuencia antes de que se ejecuten los procesadores de registros receptores, los procesadores de registros no podrán leer los registros tras el inicio.

Para cambiar el comportamiento de los procesadores de registros de forma que siempre lean los datos desde el inicio del flujo, defina el siguiente valor en el archivo de propiedades de su aplicación de Amazon Kinesis Data Streams:

```
initialPositionInStream = TRIM_HORIZON
```

De forma predeterminada, Amazon Kinesis Data Streams almacena todos los datos durante 24 horas. También admite la retención prolongada de hasta 7 días y la retención a largo plazo de hasta 365 días. Este periodo de tiempo se denomina el periodo de retención. Si se establece la posición inicial en `TRIM_HORIZON`, el procesador de registros se iniciará con los datos más antiguos de la secuencia, según defina el periodo de retención. Incluso con la configuración `TRIM_HORIZON`, si un procesador de registros se iniciara tras transcurrir mucho más tiempo del que determina el periodo de retención, algunos de los registros de la secuencia dejarán de estar disponibles. Por este motivo, debería tener siempre aplicaciones de consumo leyendo desde el flujo y utilizar la métrica `GetRecords.IteratorAgeMilliseconds` de CloudWatch para supervisar que las aplicaciones puedan estar al tanto de los datos de entrada.

En algunos casos, puede ser una buena opción para los procesadores de registros omitir los primeros registros de la secuencia. Por ejemplo, puede ejecutar algunos registros iniciales en la secuencia para comprobar que la secuencia esté funcionando de extremo a extremo, como cabría esperar. Tras realizar la verificación inicial, a continuación iniciaría sus procesos de trabajo y empezaría a introducir datos de producción en la secuencia.

Para obtener más información acerca de la opción `TRIM_HORIZON`, consulte Uso de iteradores de fragmentos.

Cierre de una aplicación de Amazon Kinesis Data Streams

Cuando su aplicación de Amazon Kinesis Data Streams haya completado la tarea correspondiente, debe cerrar mediante la terminación de las instancias de EC2 en las que se está ejecutando. Puede terminar las instancias usando la AWS Management Console o la AWS CLI.

Después de cerrar la aplicación de Amazon Kinesis Data Streams, debe eliminar la tabla de Amazon DynamoDB que utilizó KCL para realizar un seguimiento del estado de la aplicación.

Limitación controlada de lectura

El rendimiento de una secuencia se facilita en el nivel del fragmento. Cada partición tiene un rendimiento de lectura de hasta cinco transacciones por segundo, con una velocidad máxima total de lectura de datos de 2 MB por segundo. Si una aplicación (o grupo de aplicaciones que operan en el mismo flujo) intenta obtener datos de una partición a una velocidad superior, Kinesis Data Streams limita las operaciones Get correspondientes.

En una aplicación de Amazon Kinesis Data Streams, si un procesador de registros procesa datos más rápido que el límite (por ejemplo, en el caso de una conmutación por error), se produce una limitación. Dado que KCL administra las interacciones entre la aplicación y Kinesis Data Streams, las excepciones por limitación se producen en el código de KCL y no en el código de la aplicación. No obstante, puesto que KCL registra estas excepciones, las verá en los registros.

Si cree que su aplicación es objeto de una limitación controlada de forma constante, debería pensar en aumentar el número de fragmentos para la secuencia.

Supervisión de Amazon Kinesis Data Streams

Es posible supervisar los flujos de datos en Amazon Kinesis Data Streams usando las siguientes características:

- Métricas de CloudWatch: Kinesis Data Streams envía métricas personalizadas a Amazon CloudWatch con una supervisión detallada de cada flujo.
- Agente de Kinesis: el agente de Kinesis publica métricas personalizadas que ayudan a evaluar si el agente funciona según lo previsto.
- Registro de la API: Kinesis Data Streams utiliza AWS CloudTrail para registrar las llamadas a la API y almacenar los datos en un bucket de Amazon S3.
- Kinesis Client Library: Kinesis Client Library (KCL) proporciona métricas por partición, proceso de trabajo y aplicación de KCL.
- Kinesis Producer Library: Kinesis Producer Library (KPL) proporciona métricas por partición, proceso de trabajo y aplicación de KPL.

Para obtener más información sobre problemas de supervisión, preguntas y solución de problemas comunes, consulte lo siguiente:

- Which metrics should I use to monitor and troubleshoot Kinesis Data Streams issues?
- ¿Por qué el valor IteratorAgeMilliseconds en Kinesis Data Streams sigue aumentando?

Supervisión del servicio Amazon Kinesis Data Streams con Amazon CloudWatch

Amazon Kinesis Data Streams y Amazon CloudWatch están integrados, por lo que puede recopilar, ver y analizar métricas de CloudWatch de los flujos de datos de Kinesis. Por ejemplo, para realizar un seguimiento del uso de los fragmentos, puede monitorizar las métricas `IncomingBytes` y `OutgoingBytes` y compararlas con el número de fragmentos de la secuencia.

Las métricas configuradas para sus flujos se recopilan y envían automáticamente a CloudWatch cada minuto. Las métricas se archivan durante dos semanas; después de ese periodo, los datos se descartan.

En la siguiente tabla, se describen la supervisión básica de los flujos y la supervisión mejorada de las particiones de los flujos de datos de Kinesis.

Type	Descripción
Básica (en el nivel de la secuencia)	Los datos en el nivel de la secuencia se envían automáticamente cada minuto sin costo alguno.
Mejorada (en el nivel de fragmento)	Los datos en el nivel de fragmento se envían cada minuto por un costo adicional. Para obtener este nivel de datos, debe habilitar lo específicamente para el flujo mediante la operación EnableEnhancedMonitoring. Para obtener más información acerca de los precios, consulte la página Precios de Amazon CloudWatch.

Métricas y dimensiones de Amazon Kinesis Data Streams

Kinesis Data Streams envía métricas a CloudWatch en dos niveles: en el de flujo y, opcionalmente, en el nivel de partición. Las métricas a nivel de secuencia son para casos de uso de monitorización más comunes en condiciones normales. Las métricas de partición se han creado para tareas de supervisión específicas, normalmente relacionadas con la resolución de problemas, y se habilitan mediante la operación EnableEnhancedMonitoring.

Para obtener una explicación de las estadísticas recopiladas a partir de las métricas de CloudWatch, consulte Estadísticas de CloudWatch en la Guía del usuario de Amazon CloudWatch.

Temas

- Métricas básicas de nivel de secuencia
- Métricas ampliadas de nivel de fragmento
- Dimensiones de las métricas de Amazon Kinesis Data Streams
- Métricas recomendadas de Amazon Kinesis Data Streams

Métricas básicas de nivel de secuencia

El espacio de nombres `AWS/Kinesis` incluye las siguientes métricas de nivel de secuencia.

Kinesis Data Streams envía estas métricas de los flujos a CloudWatch cada minuto. Estas métricas están disponibles siempre.

Métrica	Descripción
`GetRecords.Bytes`	El número de bytes recuperados del flujo de Kinesis, calculado de acuerdo al periodo de tiempo especificado. Las estadísticas Minimum, Maximum y Average represent an los bytes de una sola operación `GetRecords` de la secuencia en el periodo de tiempo especificado. Nombre de métrica de nivel de fragmento: `OutgoingBytes` Dimensiones: StreamName Estadísticas: Minimum, Maximum, Average, Sum, Samples Unidades: bytes
`GetRecords.IteratorAge`	Esta métrica ya no está disponible. Utilice `GetRecords.IteratorAgeMilliseconds` .
`GetRecords.IteratorAgeMilliseconds`	La antigüedad del último registro en todas las llamadas `GetRecords` realizadas en un flujo de Kinesis, calculada de acuerdo al periodo de tiempo especific ado. La antigüedad es la diferencia entre la hora actual y el momento en que el último registro de la llamada `GetRecords` se escribió en la secuencia. Las estadísti cas Mínimo y Máximo se pueden usar para realizar un seguimiento del progreso de las aplicaciones consumido ras de Kinesis. Un valor de cero indica que los registros que se leen están totalmente al día con respecto a la secuencia.

Métrica	Descripción
	Nombre de métrica de nivel de fragmento: `IteratorAgeMilliseconds` Dimensiones: StreamName Estadísticas: Minimum, Maximum, Average, Samples Unidades: milisegundos
`GetRecords.Latency`	El tiempo que tarda cada operación `GetRecords` , medido durante el periodo de tiempo especificado. Dimensiones: StreamName Estadísticas: Minimum, Maximum, Average Unidades: milisegundos
`GetRecords.Records`	El número de registros recuperados del fragmento medidos durante el periodo de tiempo especificado. Las estadísticas Minimum, Maximum y Average representan los registros de una sola operación `GetRecords` de la secuencia en el periodo de tiempo especificado. Nombre de métrica de nivel de fragmento: `OutgoingRecords` Dimensiones: StreamName Estadísticas: Minimum, Maximum, Average, Sum, Samples Unidades: recuento

Métrica	Descripción
GetRecords.Success	El número de registros GetRecords correctos de cada secuencia medidos durante el periodo de tiempo especificado. Dimensiones: StreamName Estadísticas: Average, Sum, Samples Unidades: recuento
IncomingBytes	El número de bytes insertados correctamente en el flujo de Kinesis de acuerdo al periodo de tiempo especific ado. Esta métrica incluye los bytes de las operacion es PutRecord y PutRecords . Las estadísticas Minimum, Maximum y Average representan los bytes de una sola operación put del flujo en el periodo de tiempo especificado. Nombre de métrica de nivel de fragmento: IncomingB ytes Dimensiones: StreamName Estadísticas: Minimum, Maximum, Average, Sum, Samples Unidades: bytes

Métrica	Descripción
IncomingRecords	El número de registros insertados correctamente en el flujo de Kinesis de acuerdo al periodo de tiempo especific ado. Esta métrica incluye el número de registros de las operaciones PutRecord y PutRecords . Las estadísti cas Minimum, Maximum y Average representan los registros de una sola operación put del flujo en el periodo de tiempo especificado. Nombre de métrica de nivel de fragmento: IncomingR ecords Dimensiones: StreamName Estadísticas: Minimum, Maximum, Average, Sum, Samples Unidades: recuento
PutRecord.Bytes	El número de bytes insertados en el flujo de Kinesis mediante la operación PutRecord de acuerdo al periodo de tiempo especificado. Dimensiones: StreamName Estadísticas: Minimum, Maximum, Average, Sum, Samples Unidades: bytes
PutRecord.Latency	El tiempo que tarda cada operación PutRecord , medido durante el periodo de tiempo especificado. Dimensiones: StreamName Estadísticas: Minimum, Maximum, Average Unidades: milisegundos

Métrica	Descripción
PutRecord.Success	El número de operaciones PutRecord correctas de cada flujo de Kinesis, calculado de acuerdo al periodo de tiempo especificado. La media refleja el porcentaje de operaciones de escritura correctas en una secuencia. Dimensiones: StreamName Estadísticas: Average, Sum, Samples Unidades: recuento
PutRecords.Bytes	El número de bytes insertados en el flujo de Kinesis mediante la operación PutRecords de acuerdo al periodo de tiempo especificado. Dimensiones: StreamName Estadísticas: Minimum, Maximum, Average, Sum, Samples Unidades: bytes
PutRecords.Latency	El tiempo que tarda cada operación PutRecords , medido durante el periodo de tiempo especificado. Dimensiones: StreamName Estadísticas: Minimum, Maximum, Average Unidades: milisegundos

Métrica	Descripción
PutRecords.Records	Esta métrica ya no está disponible. Utilice PutRecords.SuccessfulRecords . Dimensiones: StreamName Estadísticas: Minimum, Maximum, Average, Sum, Samples Unidades: recuento
PutRecords.Success	El número de operaciones PutRecords en las que al menos un registro se realizó correctamente de cada flujo de Kinesis, calculado de acuerdo al periodo de tiempo especificado. Dimensiones: StreamName Estadísticas: Average, Sum, Samples Unidades: recuento
PutRecords.TotalRecords	El número total de registros enviados en una operación PutRecords de cada flujo de datos de Kinesis, calculado de acuerdo al periodo de tiempo especificado. Dimensiones: StreamName Estadísticas: Minimum, Maximum, Average, Sum, Samples Unidades: recuento

Métrica	Descripción
PutRecords.SuccessfulRecords	El número de registros correctos en una operación PutRecords de cada flujo de datos de Kinesis, calculado de acuerdo al periodo de tiempo especificado. Dimensiones: StreamName Estadísticas: Minimum, Maximum, Average, Sum, Samples Unidades: recuento
PutRecords.FailedRecords	El número de registros rechazados debido a errores internos en una operación PutRecords de cada flujo de datos de Kinesis, calculado de acuerdo al periodo de tiempo especificado. Es de esperar que se produzcan errores internos ocasionales, por lo que se debe volver a intentar. Dimensiones: StreamName Estadísticas: Minimum, Maximum, Average, Sum, Samples Unidades: recuento
PutRecords.ThrottledRecords	El número de registros rechazados debido a la limitación en una operación PutRecords de cada flujo de datos de Kinesis, calculado de acuerdo al periodo de tiempo especificado. Dimensiones: StreamName Estadísticas: Minimum, Maximum, Average, Sum, Samples Unidades: recuento

Métrica	Descripción
`ReadProvisionedThroughputExceeded`	El número de llamadas `GetRecords` limitadas para la secuencia medidas durante el periodo de tiempo especificado. La estadística usada con más frecuencia para esta métrica es Average. Cuando la estadística Minimum tiene un valor de 1, se limitan todos los registros del flujo durante el periodo de tiempo especificado. Cuando la estadística Maximum tiene un valor de 0 (cero), no se limita ningún registro del flujo durante el periodo de tiempo especificado. Nombre de métrica de nivel de fragmento: `ReadProvisionedThroughputExceeded` Dimensiones: StreamName Estadísticas: Minimum, Maximum, Average, Sum, Samples Unidades: recuento
`SubscribeToShard.RateExceeded`	Esta métrica se emite cuando un nuevo intento de suscripción produce un error porque ya existe una suscripción activa para el mismo consumidor o si se supera el número de llamadas por segundo permitidas para esta operación. Dimensiones: StreamName, ConsumerName
`SubscribeToShard.Success`	Esta métrica registra si la suscripción SubscribeToShard se ha establecido correctamente. La suscripción solo dura un máximo de 5 minutos. Por lo tanto, esta métrica se emite al menos una vez cada 5 minutos. Dimensiones: StreamName, ConsumerName

Métrica	Descripción
`SubscribeToShardEvent.Bytes`	El número de bytes recibidos del fragmento, medidos durante el periodo de tiempo especificado. Las estadísticas Minimum, Maximum y Average representan los bytes publicados en un solo evento durante el periodo de tiempo especificado. Nombre de métrica de nivel de fragmento: `OutgoingBytes` Dimensiones: StreamName, ConsumerName Estadísticas: Minimum, Maximum, Average, Sum, Samples Unidades: bytes
`SubscribeToShardEvent.MillisBehindLatest`	La diferencia entre la hora actual y el momento en que el último registro del evento SubscribeToShard se escribió en la secuencia. Dimensiones: StreamName, ConsumerName Estadísticas: Minimum, Maximum, Average, Samples Unidades: milisegundos

Métrica	Descripción
`SubscribeToShardEvent.Records`	El número de registros recibidos del fragmento, medidos durante el periodo de tiempo especificado. Las estadísticas Minimum, Maximum y Average representan los registros de un solo evento durante el periodo de tiempo especificado. Nombre de métrica de nivel de fragmento: `OutgoingRecords` Dimensiones: StreamName, ConsumerName Estadísticas: Minimum, Maximum, Average, Sum, Samples Unidades: recuento
`SubscribeToShardEvent.Success`	Esta métrica se emite cada vez que un evento se publica correctamente. Solo emite cuando hay una suscripción activa. Dimensiones: StreamName, ConsumerName Estadísticas: Minimum, Maximum, Average, Sum, Samples Unidades: recuento

Métrica	Descripción
`WriteProvisionedThroughputExceeded`	El número de registros rechazados debido a una limitación del flujo durante el periodo de tiempo especificado. Esta métrica incluye la limitación de las operaciones `PutRecord` y `PutRecords`. La estadística usada con más frecuencia para esta métrica es Average. Cuando la estadística Minimum tiene un valor distinto de 0, se limitan los registros del flujo durante el periodo de tiempo especificado. Cuando la estadística Maximum tiene un valor de 0 (cero), no se limita ningún registro del flujo durante el periodo de tiempo especificado. Nombre de métrica de nivel de fragmento: `WriteProvisionedThroughputExceeded` Dimensiones: StreamName Estadísticas: Minimum, Maximum, Average, Sum, Samples Unidades: recuento

Métricas ampliadas de nivel de fragmento

El espacio de nombres `AWS/Kinesis` incluye las siguientes métricas de nivel de fragmento.

Kinesis envía las métricas de particiones siguientes a CloudWatch cada minuto. Cada dimensión de la métrica crea una métrica de CloudWatch y realiza aproximadamente 43 200 llamadas mensuales a la API `PutMetricData`. Estas métricas no están habilitadas de forma predeterminada. Se aplica un cargo para las métricas mejoradas emitidas desde Kinesis. Para obtener más información, consulte Precios de Amazon CloudWatch en la sección Métricas personalizadas de Amazon CloudWatch. Los precios son por fragmento por métrica y por mes.

Métrica	Descripción
`IncomingBytes`	El número de bytes insertados correctamente en el fragmento durante el periodo de tiempo especific ado. Esta métrica incluye los bytes de las operacion es `PutRecord` y `PutRecords` . Las estadísticas Minimum, Maximum y Average representan los bytes de una sola operación put del fragmento en el periodo de tiempo especificado. Nombre de métrica de nivel de secuencia: `IncomingB ytes` Dimensiones: StreamName, ShardId Estadísticas: Minimum, Maximum, Average, Sum, Samples Unidades: bytes
`IncomingRecords`	El número de registros insertados correctamente en el fragmento durante el periodo de tiempo especific ado. Esta métrica incluye el número de registros de las operaciones `PutRecord` y `PutRecords` . Las estadísti cas Minimum, Maximum y Average representan los registros de una sola operación put del fragmento en el periodo de tiempo especificado. Nombre de métrica de nivel de secuencia: `IncomingR ecords` Dimensiones: StreamName, ShardId Estadísticas: Minimum, Maximum, Average, Sum, Samples Unidades: recuento

Métrica	Descripción
IteratorAgeMilliseconds	La antigüedad del último registro en todas las llamadas GetRecords realizadas en un fragmento, medida en el periodo de tiempo especificado. La antigüedad es la diferencia entre la hora actual y el momento en que el último registro de la llamada GetRecords se escribió en la secuencia. Las estadísticas Mínimo y Máximo se pueden usar para realizar un seguimiento del progreso de las aplicaciones consumidoras de Kinesis. Un valor de 0 (cero) indica que los registros que se leen están totalment e al día con respecto a la secuencia. Nombre de métrica de nivel de secuencia: GetRecord s.IteratorAgeMilliseconds Dimensiones: StreamName, ShardId Estadísticas: Minimum, Maximum, Average, Samples Unidades: milisegundos
OutgoingBytes	El número de bytes recuperados del fragmento medidos durante el periodo de tiempo especificado. Las estadísti cas Minimum, Maximum y Average representan los bytes devueltos por una sola operación GetRecords o publicados en un solo evento SubscribeToShard para el fragmento en el periodo de tiempo especificado. Nombre de métrica de nivel de secuencia: GetRecord s.Bytes Dimensiones: StreamName, ShardId Estadísticas: Minimum, Maximum, Average, Sum, Samples Unidades: bytes

Métrica	Descripción
`OutgoingRecords`	El número de registros recuperados del fragmento medidos durante el periodo de tiempo especificado. Las estadísticas Minimum, Maximum y Average representan los registros devueltos por una sola operación `GetRecords` o publicados en un solo evento `SubscribeToShard` para el fragmento en el periodo de tiempo especificado. Nombre de métrica de nivel de secuencia: `GetRecords.Records` Dimensiones: StreamName, ShardId Estadísticas: Minimum, Maximum, Average, Sum, Samples Unidades: recuento

Métrica	Descripción
`ReadProvisionedThroughputExceeded`	El número de llamadas `GetRecords` limitadas para el fragmento medidas durante el periodo de tiempo especificado. Este recuento de excepciones abarca todas las dimensiones de los siguientes límites: 5 lecturas por fragmento y segundo o 2 MB por segundo y fragmento . La estadística usada con más frecuencia para esta métrica es Average. Cuando la estadística Minimum tiene un valor de 1, se limitan todos los registros del fragmento durante el periodo de tiempo especificado. Cuando la estadística Maximum tiene un valor de 0 (cero), no se limita ningún registro del fragmento durante el periodo de tiempo especificado. Nombre de métrica de nivel de secuencia: `ReadProvisionedThroughputExceeded` Dimensiones: StreamName, ShardId Estadísticas: Minimum, Maximum, Average, Sum, Samples Unidades: recuento

Métrica	Descripción
`WriteProvisionedThroughputExceeded`	El número de registros rechazados debido a una limitación del fragmento durante el periodo de tiempo especificado. Esta métrica incluye las limitaciones de las operaciones `PutRecord` y `PutRecords` y abarca todas las dimensiones de los siguientes límites: 1 000 registros por segundo y fragmento o 1 MB por segundo y fragmento. La estadística usada con más frecuencia para esta métrica es Average. Cuando la estadística Minimum tiene un valor distinto de 0, se limitan los registros del fragmento durante el periodo de tiempo especificado. Cuando la estadística Maximum tiene un valor de 0 (cero), no se limita ningún registro del fragmento durante el periodo de tiempo especificado. Nombre de métrica de nivel de secuencia: `WriteProvisionedThroughputExceeded` Dimensiones: StreamName, ShardId Estadísticas: Minimum, Maximum, Average, Sum, Samples Unidades: recuento

Dimensiones de las métricas de Amazon Kinesis Data Streams

Dimensión	Descripción
`StreamName`	El nombre de la secuencia de Kinesis. Todas las estadísticas disponibles se filtran por `StreamName` .

Métricas recomendadas de Amazon Kinesis Data Streams

Varias métricas de Amazon Kinesis Data Streams podrían resultar de especial interés para los clientes de Kinesis Data Streams. La siguiente lista aparecen algunas métricas recomendadas y sus usos.

Métrica	Notas de uso
`GetRecords.IteratorAgeMilliseconds`	Controla la posición de lectura de todos los fragmentos y los consumidores de la secuencia. Si la edad de un iterador supera el 50% del periodo de retención (con un valor predeterminado de 24 horas, pero configurable hasta 7 días), existe el riesgo de pérdida de datos debido al vencimiento del registro. Recomendamos que utilice alarmas de CloudWatch para la estadística Máximo para recibir avisos antes de que esta pérdida suponga un riesgo. Para ver un escenario de ejemplo en el que se use esta métrica, consulte El procesamiento de registros del consumidor se queda atrás.
`ReadProvisionedThroughputExceeded`	Si el procesamiento de registros en el lado del consumidor no alcanza el rendimiento necesario, a veces es difícil saber dónde está el cuello de botella. Utilice esta métrica para determinar si sus lecturas se están viendo limitadas debido a que se superan los límites de rendimiento de lectura. La estadística usada con más frecuencia para esta métrica es Average.
`WriteProvisionedThroughputExceeded`	Tiene el mismo objetivo que la métrica `ReadProvisionedThroughputExceeded` , pero en el extremo de productor (put) de la secuencia. La estadística usada con más frecuencia para esta métrica es Average.
`PutRecord.Success` , `PutRecords.Success`	Recomendamos el uso de alarmas de CloudWatch para la estadística Average para indicar la ausencia de registros entrantes en el flujo. Seleccione uno o ambos tipos de métrica "put" en función de lo que el productor utilice. En caso de que use Kinesis Producer Library (KPL), utilice `PutRecords.Success` .
`GetRecords.Success`	Recomendamos el uso de alarmas de CloudWatch para la estadística Average para indicar la ausencia de registros salientes en el flujo.

Acceso a las métricas de Amazon CloudWatch para Kinesis Data Streams

Las métricas de Kinesis Data Streams se pueden supervisar con la consola de CloudWatch, la línea de comandos o la API de CloudWatch. Los siguientes procedimientos le muestran cómo obtener acceso a las métricas a través de los distintos métodos descritos a continuación.

Acceso a las métricas a través de la consola de CloudWatch

1. Abra la consola de CloudWatch en https://console.aws.amazon.com/cloudwatch/.
2. Seleccione una región en la barra de navegación.
3. En el panel de navegación, seleccione Métricas.
4. En el panel CloudWatch Metrics by Category (Métricas de CloudWatch por categoría), elija Kinesis Metrics (Métricas de Kinesis).
5. Haga clic en la fila correspondiente para ver las estadísticas para el MetricName (Nombre de la métrica) y el StreamName (Nombre de la secuencia) especificados.

   Nota: la mayoría de los nombres de estadísticas en la consola coinciden con los nombres de las métricas correspondientes de CloudWatch que aparecen en la lista anterior, con la excepción de Rendimiento de lectura y Rendimiento de escritura. Estas estadísticas se calculan en intervalos de 5 minutos: Rendimiento de escritura supervisa la métrica `IncomingBytes` de CloudWatch, mientras que Rendimiento de lectura supervisa `GetRecords.Bytes`.
6. (Opcional) En el panel de gráficos, seleccione una estadística y un periodo de tiempo y, a continuación, cree una alarma de CloudWatch utilizando estos ajustes.

Obtención del acceso a las métricas mediante la AWS CLI

Utilice los comandos list-metrics y get-metric-statistics.

Obtención del acceso a las métricas mediante la CLI de CloudWatch

Utilice los comandos mon-list-metrics y mon-get-stats.

Obtención del acceso a las métricas mediante la API de CloudWatch

Utilice las operaciones ListMetrics y GetMetricStatistics.

Supervisión del agente de estado de Kinesis Data Streams con Amazon CloudWatch

El agente publica métricas de CloudWatch personalizadas con un espacio de nombres de AWSKinesisAgent. Estas métricas le permiten evaluar si el agente está enviando datos a Kinesis Data Streams como se ha especificado, y si se encuentra en buen estado y consume la cantidad adecuada de recursos de CPU y de memoria en el productor de datos. Las métricas como el número de registros y bytes enviados resultan útiles para comprender la velocidad a la que el agente está enviando datos a la secuencia. Cuando estas métricas caen por debajo de los umbrales previstos en determinado porcentaje o pasan a ser cero, esto podría indicar que existen problemas de configuración, errores de red o problemas con el estado del agente. Las métricas como, por ejemplo, el consumo de CPU y memoria de host y los contadores de errores del agente indican el uso de los recursos por parte del productor y proporcionan información útil sobre posibles errores de host o de configuración. Por último, el agente también registra excepciones de servicio para ayudar a investigar los problemas del agente. Estas métricas se notifican en la región especificada en la opción de configuración del agente `cloudwatch.endpoint`. Las métricas de Cloudwatch publicadas desde varios agentes de Kinesis se agregan o combinan. Para obtener más información acerca de la configuración del agente, consulte Ajustes de la configuración del agente.

Supervisión con CloudWatch

El agente de Kinesis Data Streams envía las siguientes métricas a CloudWatch.

Métrica	Descripción
`BytesSent`	El número de bytes enviados a Kinesis Data Streams durante el periodo de tiempo especificado. Unidades: bytes
`RecordSendAttempts`	El número de registros que se ha intentado grabar (como primer intento o como repetición) en una llamada a `PutRecords` durante el periodo de tiempo especificado. Unidades: recuento

Métrica	Descripción
`RecordSendErrors`	El número de registros que han devuelto un estado de error en una llamada a `PutRecords` , incluidos los intentos repetidos, durante el periodo de tiempo especificado. Unidades: recuento
`ServiceErrors`	El número de llamadas a `PutRecords` que ocasionaron un error de servicio (distinto de un error de limitación controlada) durante el periodo especificado. Unidades: recuento

Registros de las llamadas a la API de Amazon Kinesis Data Streams mediante AWS CloudTrail

Amazon Kinesis Data Streams está integrada con AWS CloudTrail, un servicio que proporciona un registro de las acciones del usuario, el rol o un servicio de AWS en Kinesis Data Streams. CloudTrail captura las llamadas a la API de Kinesis Data Streams como eventos. Las llamadas capturadas incluyen las llamadas desde la consola de Kinesis Data Streams y las llamadas desde el código a las operaciones de la API de Kinesis Data Streams. Si crea un registro de seguimiento, puede habilitar la entrega continua de eventos de CloudTrail a un bucket de Amazon S3, incluidos los eventos de Kinesis Data Streams. Si no configura un registro de seguimiento, puede ver los eventos más recientes en la consola de CloudTrail en el Historial de eventos. Con la información recopilada por CloudTrail, puede determinar la solicitud que se ha realizado a Kinesis Data Streams, la dirección IP desde la que se ha realizado la solicitud, quién la ha realizado, cuándo la ha realizado y los detalles adicionales.

Para obtener más información acerca de CloudTrail, incluso cómo configurarlo y habilitarlo, consulte la Guía del usuario de AWS CloudTrail.

Información de Kinesis Data Streams en CloudTrail

CloudTrail se habilita en su cuenta de AWS cuando la crea. Cuando se produce una actividad de eventos compatible en Kinesis Data Streams, la actividad se registra en un evento de CloudTrail junto con otros eventos de servicios de AWS en Historial de eventos. Puede ver, buscar y descargar

los últimos eventos de la cuenta de AWS. Para obtener más información, consulte Ver eventos con el historial de eventos de CloudTrail.

Para mantener un registro continuo de los eventos de la cuenta de AWS, incluido los eventos de Kinesis Data Streams, cree un registro de seguimiento. Un registro de seguimiento permite a CloudTrail enviar archivos de registro a un bucket de Amazon S3. De manera predeterminada, cuando se crea un registro de seguimiento en la consola, el registro de seguimiento se aplica a todas las regiones de AWS. El registro de seguimiento registra los eventos de todas las regiones de la partición de AWS y envía los archivos de registro al bucket de Amazon S3 especificado. También es posible configurar otros servicios de AWS para analizar en profundidad y actuar en función de los datos de eventos recopilados en los registros de CloudTrail. Para más información, consulte los siguientes temas:

- Introducción a la creación de registros de seguimiento
- Servicios e integraciones compatibles con CloudTrail
- Configurar notificaciones de Amazon SNS para CloudTrail
- Recepción de archivos de registro de CloudTrail de varias regiones y Recepción de archivos de registro de CloudTrail de varias cuentas

Kinesis Data Streams admite el registro de las siguientes acciones como eventos en archivos de registros de CloudTrail:

- AddTagsToStream
- CreateStream
- DecreaseStreamRetentionPeriod
- DeleteStream
- DeregisterStreamConsumer
- DescribeStream
- DescribeStreamConsumer
- DisableEnhancedMonitoring
- EnableEnhancedMonitoring
- IncreaseStreamRetentionPeriod
- ListStreamConsumers
- ListStreams

- ListTagsForStream
- MergeShards
- RegisterStreamConsumer
- RemoveTagsFromStream
- SplitShard
- StartStreamEncryption
- StopStreamEncryption
- UpdateShardCount
- UpdateStreamMode

Cada entrada de registro o evento contiene información sobre quién generó la solicitud. La información de identidad del usuario lo ayuda a determinar lo siguiente:

- Si la solicitud se realizó con credenciales de usuario AWS Identity and Access Management (IAM) o credenciales de usuario raíz.
- Si la solicitud se realizó con credenciales de seguridad temporales de un rol o fue un usuario federado.
- Si la solicitud la realizó otro servicio de AWS.

Para obtener más información, consulte el Elemento userIdentity de CloudTrail.

Ejemplo: entradas del archivo de registro de Kinesis Data Streams

Un registro de seguimiento es una configuración que permite la entrega de eventos como archivos de registro en un bucket de Amazon S3 que especifique. Los archivos de registro de CloudTrail pueden contener una o varias entradas de registro. Un evento representa una solicitud específica realizada desde un código fuente y contiene información sobre la acción solicitada, la fecha y la hora de la acción, los parámetros de la solicitud, etc. Los archivos de registro de CloudTrail no rastrean el orden en la pila de las llamadas públicas a la API, por lo que estas no aparecen en ningún orden específico.

En el ejemplo siguiente, se muestra una entrada de registro de CloudTrail que ilustra las acciones `CreateStream`, `DescribeStream`, `ListStreams`, `DeleteStream`, `SplitShard` y `MergeShards`.

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

Supervisión de Kinesis Client Library con Amazon CloudWatch

Kinesis Client Library (KCL) para Amazon Kinesis Data Streams publica métricas personalizadas de Amazon CloudWatch en su nombre. Para ello, utiliza el nombre de su aplicación de KCL como espacio de nombres. Para ver estas métricas, vaya a la consola de CloudWatch y seleccione Métricas personalizadas. Para obtener más información, consulte Publicación de métricas personalizadas en la Guía del usuario de Amazon CloudWatch.

Existe un cargo nominal para las métricas cargadas en CloudWatch por KCL, específicamente, se aplican los cargos por métricas personalizadas de Amazon CloudWatch y solicitudes a la API de Amazon CloudWatch. Para obtener más información, consulte Precios de Amazon CloudWatch.

Temas

- Métricas y espacio de nombres
- Niveles y dimensiones de las métricas
- Configuración de métricas
- Lista de métricas

Métricas y espacio de nombres

El espacio de nombres que se utiliza para cargar las métricas es el nombre de aplicación que se ha especificado al lanzar KCL.

Niveles y dimensiones de las métricas

Existen dos opciones para controlar qué métricas se cargan en CloudWatch:

Niveles de métricas

A cada métrica se le asigna un nivel individual. Al establecer un nivel de informe de métricas, las métricas con un nivel individual por debajo del nivel de informe no se envían a CloudWatch. Los niveles son: `NONE`, `SUMMARY` y `DETAILED`. El valor predeterminado es `DETAILED`; es decir, se envían todas las métricas a CloudWatch. El nivel de informe `NONE` se utiliza cuando no se desea enviar ninguna métrica. Para obtener información sobre qué niveles se asignan a las métricas, consulte Lista de métricas.

Dimensiones habilitadas

Cada métrica de KCL tiene asociadas determinadas dimensiones que también se envían a CloudWatch. En KCL 2.x, si KCL se configura para procesar un solo flujo de datos, todas las dimensiones de las métricas (`Operation`, `ShardId` y `WorkerIdentifier`) están habilitadas de forma predeterminada. Además, en KCL 2.x, si KCL se configura para procesar un único flujo de datos, la dimensión `Operation` no se puede deshabilitar. En KCL 2.x, si KCL se configura para procesar varios flujos de datos, todas las dimensiones de las métricas (`Operation`, `ShardId`, `StreamId` y `WorkerIdentifier`) están habilitadas de forma predeterminada. Además, en KCL 2.x, si KCL se configura para procesar varios flujos de datos, las dimensiones `Operation` y `StreamId` no se pueden deshabilitar. La dimensión `StreamId` solo está disponible para métricas por partición.

En KCL 1.x, solo las dimensiones `Operation` y `ShardId` están habilitadas de forma predeterminada. La dimensión `WorkerIdentifier` está deshabilitada. En KCL 1.x, la dimensión `Operation` no se puede deshabilitar.

Para obtener más información acerca de las dimensiones de las métricas de CloudWatch, consulte la sección Dimensiones en el tema Conceptos de Amazon CloudWatch en la Guía del usuario de Amazon CloudWatch.

Cuando la dimensión `WorkerIdentifier` está habilitada, si se utiliza un valor diferente para la propiedad de ID de proceso de trabajo cada vez que se reinicia un proceso de trabajo concreto de KCL, se enviarán a CloudWatch nuevos conjuntos de métricas con nuevos valores en la dimensión `WorkerIdentifier`. Si necesita que el valor de la dimensión `WorkerIdentifier` sea el mismo cada vez que un proceso de trabajo de KCL determinado se reinicie, debe especificar explícitamente el mismo valor del ID del proceso de trabajo durante la inicialización

para cada proceso de trabajo. Tenga en cuenta que el valor del ID de proceso de trabajo para cada proceso de trabajo de KCL activo deberá ser único entre todos los procesos de trabajo de KCL.

Configuración de métricas

Las dimensiones y los niveles habilitados de las métricas se pueden configurar mediante la instancia KinesisClientLibConfiguration, que se transfiere al proceso de trabajo al lanzar la aplicación de KCL. En el caso de MultiLangDaemon, las propiedades `metricsLevel` y `metricsEnabledDimensions` se pueden especificar en el archivo .properties utilizado para lanzar la aplicación MultiLangDaemon de KCL.

Los niveles de las métricas pueden tener asignado uno de estos tres valores: NONE, SUMMARY o DETAILED. Los valores de las dimensiones habilitadas deben ser valores de cadena separados por comas con la lista de dimensiones permitidas para las métricas de CloudWatch. Las dimensiones utilizadas por la aplicación de KCL son `Operation`, `ShardId` y `WorkerIdentifier`.

Lista de métricas

En las siguientes tablas se enumeran las métricas de KCL agrupadas por ámbito y operación.

Temas

Métricas por aplicación de KCL

Estas métricas se agrupan para todos los procesos de trabajo de KCL en el ámbito de la aplicación, según defina el espacio de nombres de Amazon CloudWatch.

Temas

- PeriodicShardSyncManager
- MultistreamTracker

InitializeTask

La operación `InitializeTask` es responsable de inicializar el procesador de registros de la aplicación de KCL. La lógica de esta operación incluye la obtención de un iterador de particiones de Kinesis Data Streams y la inicialización del procesador de registros.

Métrica	Descripción
KinesisDataFetcher .getIterator.Success	Número de operaciones `GetShardIterator` correctas por aplicación de KCL. Nivel de métrica: Detailed Unidades: recuento
KinesisDataFetcher .getIterator.Time	Tiempo que tarda la operación `GetShardIterator` para la aplicación de KCL en cuestión. Nivel de métrica: Detailed Unidades: milisegundos
RecordProcessor.in itialize.Time	Tiempo que tarda el método de inicialización del procesador de registros. Nivel de métrica: Summary Unidades: milisegundos
Success	Número de inicializaciones correctas del procesador de registros. Nivel de métrica: Summary Unidades: recuento
Time	Tiempo que tarda el proceso de trabajo de KCL en inicializar el procesador de registros. Nivel de métrica: Summary

Métrica	Descripción
	Unidades: milisegundos

ShutdownTask

La operación `ShutdownTask` inicia la secuencia de cierre para el procesamiento de fragmentos. Esto puede ocurrir porque un fragmento esté dividido o fusionado, o cuando se pierde la asignación del fragmento desde el proceso de trabajo. En ambos casos se invoca la función `shutdown()` del procesador de registros. También se descubren fragmentos nuevos en caso de que un fragmento se divida o se fusione, lo que da como resultado la creación de uno o varios fragmentos nuevos.

Métrica	Descripción
CreateLease.Success	Número de veces que se agregan correctamente nuevas particiones secundarias a la tabla de DynamoDB de la aplicación de KCL después del cierre de la partición principal. Nivel de métrica: Detailed Unidades: recuento
CreateLease.Time	Tiempo que se tarda en agregar la información de la nueva partición secundaria a la tabla DynamoDB de la aplicación de KCL. Nivel de métrica: Detailed Unidades: milisegundos
UpdateLease.Success	Número de puntos de comprobación finales correctos durante el cierre del procesador de registros. Nivel de métrica: Detailed Unidades: recuento
UpdateLease.Time	Tiempo que tarda la operación de puntos de comprobación durante el cierre del procesador de registros. Nivel de métrica: Detailed

Métrica	Descripción
	Unidades: milisegundos
RecordProcessor.shutdown.Time	Tiempo que tarda el método de cierre del procesador de registros. Nivel de métrica: Summary Unidades: milisegundos
Success	Número de tareas cerradas correctamente. Nivel de métrica: Summary Unidades: recuento
Time	Tiempo que tarda el proceso de trabajo de KCL en la tarea de cierre. Nivel de métrica: Summary Unidades: milisegundos

ShardSyncTask

La operación `ShardSyncTask` detecta los cambios en la información de la partición del flujo de datos de Kinesis, de modo que las nuevas particiones se puedan procesar en la aplicación de KCL.

Métrica	Descripción
CreateLease.Success	Número de intentos correctos para agregar nueva información sobre la partición a la tabla de DynamoDB de la aplicación de KCL. Nivel de métrica: Detailed Unidades: recuento
CreateLease.Time	Tiempo que se tarda en agregar la información de la nueva partición a la tabla DynamoDB de la aplicación de KCL. Nivel de métrica: Detailed

Métrica	Descripción
	Unidades: milisegundos
Success	Número de operaciones de sincronización del fragmento correctas. Nivel de métrica: Summary Unidades: recuento
Time	Tiempo necesario para la operación de sincronización del fragmento. Nivel de métrica: Summary Unidades: milisegundos

BlockOnParentTask

Si el fragmento está dividido o fusionado con otros fragmentos, entonces se crean nuevos fragmentos secundarios. La operación `BlockOnParentTask` garantiza que el procesamiento de registros de las nuevas particiones no se inicie hasta que KCL procese por completo las particiones principales.

Métrica	Descripción
Success	Número de comprobaciones correctas para completar el fragmento principal. Nivel de métrica: Summary Unidades: recuento
Time	Tiempo necesario para completar los fragmentos principales. Nivel de métrica: Summary Unidad: milisegundos

PeriodicShardSyncManager

`PeriodicShardSyncManager` es responsable de examinar los flujos de datos que procesa la aplicación de consumo de KCL, identificar los flujos de datos con asignaciones parciales y transferirlos para su sincronización.

Las siguientes métricas están disponibles cuando KCL se configura para procesar un único flujo de datos (entonces el valor de NumStreamsToSync y NumStreamsWithPartialLeases se establece en 1) y también cuando KCL se configura para procesar varios flujos de datos.

Métrica	Descripción
NumStreamsToSync	El número de flujos de datos (por cuenta de AWS) que procesa la aplicación de consumo que contienen asignaciones parciales y que deben transferirse para su sincronización. Nivel de métrica: Summary Unidades: recuento
NumStreamsWithPartialLeases	El número de flujos de datos (por cuenta de AWS) que la aplicación de consumo está procesando y que contienen asignaciones parciales. Nivel de métrica: Summary Unidades: recuento
Success	El número de veces que `PeriodicShardSyncManager` pudo identificar correctamente las asignaciones parciales en los flujos de datos que la aplicación de consumo está procesando. Nivel de métrica: Summary Unidades: recuento
Time	El tiempo (en milisegundos) que `PeriodicShardSyncManager` tarda en examinar los flujos de datos que la aplicación de consumo está procesando, a fin de determinar qué flujos de datos requieren una sincronización de particiones. Nivel de métrica: Summary

Métrica	Descripción
	Unidades: milisegundos

MultistreamTracker

La interfaz de `MultistreamTracker` le permite crear aplicaciones de consumo de KCL que pueden procesar varios flujos de datos al mismo tiempo.

Métrica	Descripción
DeletedStreams.Count	El número de flujos de datos eliminados en este periodo de tiempo. Nivel de métrica: Summary Unidades: recuento
ActiveStreams.Count	El número de flujos de datos activos que se están procesando. Nivel de métrica: Summary Unidades: recuento
StreamsPendingDeletion.Count	El número de flujos de datos pendientes de eliminación en función de `FormerStreamsLeasesDeletionStrategy` . Nivel de métrica: Summary Unidades: recuento

Métricas preproceso de trabajo

Estas métricas se agrupan para todos los procesadores de registro que consuman datos de un flujo de datos de Kinesis, por ejemplo una instancia de Amazon EC2.

Temas

- RenewAllLeases
- TakeLeases

RenewAllLeases

La operación `RenewAllLeases` renueva periódicamente las asignaciones de un fragmento propiedad de una instancia de proceso de trabajo en particular.

Métrica	Descripción
RenewLease.Success	Número de renovaciones correctas de asignaciones por parte del proceso de trabajo. Nivel de métrica: Detailed Unidades: recuento
RenewLease.Time	Tiempo necesario para la operación de renovación de la asignación. Nivel de métrica: Detailed Unidades: milisegundos
CurrentLeases	Número de asignaciones de fragmentos propiedad del proceso de trabajo tras la renovación de todas las asignaciones. Nivel de métrica: Summary Unidades: recuento
LostLeases	Número de asignaciones de fragmentos perdidos tras un intento de renovar todas las asignaciones propiedad del proceso de trabajo. Nivel de métrica: Summary Unidades: recuento
Success	Número de veces que se realiza correctamente la operación de renovación para el proceso de trabajo. Nivel de métrica: Summary Unidades: recuento

Métrica	Descripción
Time	Tiempo necesario para renovar todas las asignaciones del proceso de trabajo. Nivel de métrica: Summary Unidades: milisegundos

TakeLeases

La operación `TakeLeases` equilibra el procesamiento de registros entre todos los procesos de trabajo de KCL. Si el proceso de trabajo actual de KCL tiene menos asignaciones de particiones de lo necesario, asume asignaciones de particiones de otro proceso de trabajo sobrecargado.

Métrica	Descripción
ListLeases.Success	Número de veces que se han recuperado correctamente todas las concesiones de particiones de la tabla de DynamoDB de la aplicación de KCL. Nivel de métrica: Detailed Unidades: recuento
ListLeases.Time	Tiempo que se tarda en recuperar todas las concesiones de particiones de la tabla de DynamoDB de la aplicación de KCL. Nivel de métrica: Detailed Unidades: milisegundos
TakeLease.Success	Número de veces que el proceso de trabajo ha asumido correctamente asignaciones de particiones de otros procesos de trabajo de KCL. Nivel de métrica: Detailed Unidades: recuento

Métrica	Descripción
TakeLease.Time	Tiempo necesario para actualizar la tabla de asignaciones con las asignaciones asumidas por el proceso de trabajo. Nivel de métrica: Detailed Unidades: milisegundos
NumWorkers	El número total de procesos de trabajo identificado por un proceso de trabajo específico. Nivel de métrica: Summary Unidades: recuento
NeededLeases	Número de asignaciones de fragmentos que necesita el proceso de trabajo actual para una carga de procesamiento de fragmentos equilibrada. Nivel de métrica: Detailed Unidades: recuento
LeasesToTake	Número de asignaciones que el proceso de trabajo intentará asumir. Nivel de métrica: Detailed Unidades: recuento
TakenLeases	Número de asignaciones asumidas correctamente por el proceso de trabajo. Nivel de métrica: Summary Unidades: recuento
TotalLeases	Número total de particiones que procesa la aplicación de KCL. Nivel de métrica: Detailed Unidades: recuento

Métrica	Descripción
ExpiredLeases	Número total de fragmentos que no están siendo procesados por ningún proceso de trabajo, según lo identifica el proceso de trabajo específico. Nivel de métrica: Summary Unidades: recuento
Success	Número de veces que se completa correctamente la operación `TakeLeases` . Nivel de métrica: Summary Unidades: recuento
Time	Tiempo necesario para la operación `TakeLeases` para un proceso de trabajo. Nivel de métrica: Summary Unidades: milisegundos

Métricas por fragmento

Las métricas se acumulan en un único procesador de registros.

ProcessTask

La operación `ProcessTask` llama a GetRecords con la posición actual del iterador para recuperar los registros del flujo e invoca la función `processRecords` del procesador de registros.

Métrica	Descripción
KinesisDataFetcher .getRecords.Success	Número de operaciones `GetRecords` correctas por partición de flujo de datos de Kinesis. Nivel de métrica: Detailed Unidades: recuento

Métrica	Descripción
KinesisDataFetcher.getRecords.Time	Tiempo que tarda cada operación `GetRecords` para la partición de flujo de datos de Kinesis. Nivel de métrica: Detailed Unidades: milisegundos
UpdateLease.Success	Número de puntos de comprobación correctos realizados por el procesador de registros para un fragmento determinado. Nivel de métrica: Detailed Unidades: recuento
UpdateLease.Time	Tiempo necesario para cada operación de punto de comprobación para el fragmento determinado. Nivel de métrica: Detailed Unidades: milisegundos
DataBytesProcessed	Tamaño total en bytes de los registros procesados en cada invocación de `ProcessTask` . Nivel de métrica: Summary Unidades: bytes
RecordsProcessed	Número de registros procesados en cada invocación de `ProcessTask` . Nivel de métrica: Summary Unidades: recuento

Métrica	Descripción
ExpiredIterator	Número de ExpiredIteratorException recibidas al llamar a `GetRecords`. Nivel de métrica: Summary Unidades: recuento
MillisBehindLatest	Tiempo de que el iterador actual permanece por detrás del último registro (el extremo) del fragmento. Este valor es inferior o igual a la diferencia de tiempo entre el último registro en una respuesta y la hora actual. Se trata de una representación más precisa de la distancia entre un fragmento y el extremo que la comparación de las marcas temporales del último registro de respuesta. Este valor se aplica al último lote de registros, no a una media de todas las marcas temporales de cada registro. Nivel de métrica: Summary Unidades: milisegundos
RecordProcessor.processRecords.Time	Tiempo que tarda el método `processRecords` del procesador de registros. Nivel de métrica: Summary Unidades: milisegundos
Success	Número de operaciones de tareas de proceso correctas. Nivel de métrica: Summary Unidades: recuento
Time	Tiempo necesario para la operación de tarea de proceso. Nivel de métrica: Summary Unidades: milisegundos

Supervisión de Kinesis Producer Library con Amazon CloudWatch

Kinesis Producer Library (KPL) para Amazon Kinesis Data Streams publica métricas personalizadas de Amazon CloudWatch en su nombre. Para ver estas métricas, vaya a la consola de CloudWatch y seleccione Métricas personalizadas. Para obtener más información, consulte Publicación de métricas personalizadas en la Guía del usuario de Amazon CloudWatch.

Existe un cargo nominal para las métricas cargadas en CloudWatch por KPL, específicamente, se aplican los cargos por métricas personalizadas de Amazon CloudWatch y solicitudes a la API de Amazon CloudWatch. Para obtener más información, consulte Precios de Amazon CloudWatch. La recopilación de métricas locales no incurrirá en cargos de CloudWatch.

Temas

Métricas, dimensiones y espacios de nombres

Puede especificar un nombre de aplicación al lanzar KPL, que se utilizará como parte del espacio de nombres al cargar las métricas. Esto es opcional; KPL proporciona un valor predeterminado si no se establece el nombre de una aplicación.

También puede configurar KPL para que agregue dimensiones adicionales arbitrarias para las métricas. Esto resulta útil si desea datos de alta precisión en sus métricas de CloudWatch. Por ejemplo, puede añadir el nombre de host como dimensión, lo que le permite identificar distribuciones de carga irregulares en la flota. La configuración de KPL es inmutable, por lo que no se pueden modificar estas dimensiones adicionales después de inicializar la instancia de KPL.

Nivel de métricas y grado de detalle

Existen dos opciones para controlar el número de métricas que se cargan en CloudWatch:

Nivel de métricas

> Es una calibración aproximada de la importancia de una métrica. A cada métrica se le asigna un nivel. Al establecer un nivel, las métricas con niveles por debajo de este no se envían

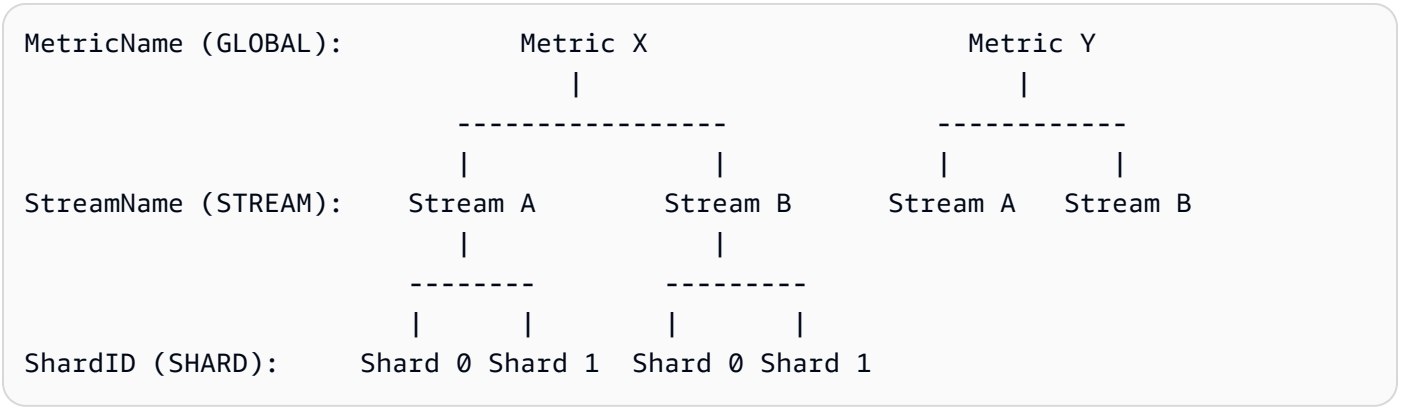
a CloudWatch. Los niveles son: `NONE`, `SUMMARY` y `DETAILED`. El valor predeterminado es `DETAILED`; es decir, todas las métricas. `NONE` significa ninguna métrica, de modo no se asigna ninguna a ese nivel.

Grado de detalle

Controla si una misma métrica se emite con grados de detalle adicionales. Los niveles son: `GLOBAL`, `STREAM` y `SHARD`. El valor predeterminado es `SHARD`, que contiene las métricas con el mayor grado de detalle.

Cuando se elige `SHARD`, las métricas se emiten con el nombre de la secuencia y la ID del fragmento como dimensiones. Además, la misma métrica también se emitida únicamente con la dimensión del nombre de la secuencia y la métrica, sin el nombre de la secuencia. Esto significa que, para una determinada métrica, dos flujos con dos particiones cada uno producen siete métricas de CloudWatch: una para cada partición, una para cada flujo y una general, y todas ellas describen las mismas estadísticas pero en diferentes grados de detalle. Para ver una ilustración, consulte el diagrama siguiente.

Los distintos niveles de granularidad forman una jerarquía y todas las métricas del sistema forman árboles enraizados en los nombres de las métricas:



No todas las métricas están disponibles en el nivel de fragmento, algunas pertenecen en el nivel de la secuencia o son de naturaleza global. Estas no se producen en el nivel de fragmento, ni siquiera si ha activado las métricas en el nivel de fragmento (`Metric Y` en el diagrama anterior).

Al especificar una dimensión adicional, debe proporcionar valores `tuple:<DimensionName, DimensionValue, Granularity>`. El grado de detalle se utiliza para determinar dónde se inserta la dimensión personalizada en la jerarquía: `GLOBAL` implica que la dimensión adicional se inserta después del nombre de la métrica, `STREAM` implica que se inserta después del nombre de la secuencia y `SHARD` implica que se inserta después del ID de fragmento. Si se dan varias dimensiones adicionales por grado de detalle, se insertan en el orden determinado.

Acceso local y carga a Amazon CloudWatch

Las métricas de la instancia de KPL actual están disponibles localmente en tiempo real. Puede enviar consultas a KPL en cualquier momento para obtenerlas. KPL calcula localmente la suma, el promedio, el mínimo, el máximo y el recuento de cada métrica, como en CloudWatch.

Puede obtener estadísticas que son acumulativas desde el principio del programa hasta el momento actual, o mediante una ventana continua durante los últimos N segundos, donde N es un número entero entre 1 y 60.

Todas las métricas están disponibles para cargarlas en CloudWatch. Esto resulta especialmente útil para agregar datos entre varios hosts, monitorizaciones y alarmas. Esta funcionalidad no está disponible a nivel local.

Como se ha descrito previamente, puede seleccionar qué métricas cargar con los ajustes de nivel y grado de detalle de las métricas. Las métricas que no están cargadas están disponibles localmente.

La carga individual de puntos de datos es insostenible, ya que podría producir millones de cargas por segundo, si el tráfico es alto. Por este motivo, KPL agrupa métricas localmente en buckets de 1 minuto y carga un objeto de estadísticas en CloudWatch una vez por minuto por cada métrica habilitada.

Lista de métricas

Métrica	Descripción
`UserRecordsReceived`	Recuento de la cantidad de registros de usuarios lógicas recibidos por el núcleo de KPL para operaciones put. No disponible en el nivel de fragmento. Nivel de métrica: Detailed Unidad: recuento
`UserRecordsPending`	Muestra periódica del número de registros de usuario pendientes actualmente. Un registro está pendiente si se encuentra actualmente en la memoria intermedia y está en espera para enviarse, o se ha enviado y está en tránsito al servicio de backend. No disponible en el nivel de fragmento.

Métrica	Descripción
	KPL proporciona un método específico para recuperar esta métrica de forma global para que los clientes administren la velocidad de la operación put. Nivel de métrica: Detailed Unidad: recuento
`UserRecordsPut`	Recuento del número de registros de usuario lógicos producidos correctamente. KPL no cuenta los registros incorrectos para esta métrica. Así se permite que la media ofrezca la tasa de corrección y que el recuento exprese los intentos totales y la diferencia entre el recuento y suma para ver la cuenta de errores. Nivel de métrica: Summary Unidad: recuento
`UserRecordsDataPut`	Bytes producidos correctamente en los registros de los usuarios lógicos. Nivel de métrica: Detailed Unidades: bytes
`KinesisRecordsPut`	Recuento de cuántos registros de Kinesis Data Streams se produjeron correctamente (cada registro de Kinesis Data Streams puede contener varios registros de usuario). KPL da como resultado un cero para los registros incorrectos. Así se permite que la media ofrezca la tasa de corrección y que el recuento exprese los intentos totales y la diferencia entre el recuento y suma para ver la cuenta de errores. Nivel de métrica: Summary Unidad: recuento

Métrica	Descripción
`KinesisRecordsDataPut`	Bytes en los registros de Kinesis Data Streams. Nivel de métrica: Detailed Unidades: bytes
`ErrorsByCode`	Recuento de cada tipo de código de error. Esto introduce una dimensión adicional de `ErrorCode` , además de las dimensiones normales como, por ejemplo, `StreamName` y `ShardId`. No todos los errores pueden localizarse en un fragmento. Los errores que no se puede localizar solo se emiten en niveles globales o de secuencias. Esta métrica recopila información sobre la limitación controlada, los cambios en el mapa de fragmentos, los errores internos, la indisponibilidad del servicio, los tiempos de espera, etcétera. Los errores de la API de Kinesis Data Streams se cuentan una vez por registro de Kinesis Data Streams. Si hay varios registros de usuario dentro de un registro de Kinesis Data Streams, no se generan varios recuentos. Nivel de métrica: Summary Unidad: recuento
`AllErrors`	Esto se activa cuando se producen los mismos errores los mismos errores que en Errors by Code (Errores por código), pero sin distinguir entre tipos. Resulta útil como monitorización general de la tasa de errores sin necesidad de sumar manualmente los recuentos de todos los diferentes tipos de errores. Nivel de métrica: Summary Unidad: recuento

Métrica	Descripción
`RetriesPerRecord`	Número de reintentos realizados por registro de usuario. Se produce un cero para los registros que resultan correctos con un solo intento. Los datos se emiten en el momento en que el registro de un usuario termina (si termina correctamente no se puede volver a intentar). Si el tiempo de vida del registro es un valor grande, esta métrica puede retrasarse significativamente. Nivel de métrica: Detailed Unidad: recuento
`BufferingTime`	El tiempo entre la llegada de un registro de usuario a KPL y la salida hacia el backend. Esta información se transmite de vuelta al usuario en cada registro, pero también está disponible como una estadística acumulada. Nivel de métrica: Summary Unidad: milisegundos
`Request Time`	El tiempo que se tarda en realizar `PutRecordsRequests` . Nivel de métrica: Detailed Unidad: milisegundos
`User Records per Kinesis Record`	El número de registros de usuarios lógicos agregados en un solo registro de Kinesis Data Streams. Nivel de métrica: Detailed Unidad: recuento

Métrica	Descripción
`Amazon Kinesis Records per PutRecord sRequest`	El número de registros de Kinesis Data Streams agrupados en una sola `PutRecordsRequest` . No disponible en el nivel de fragmento. Nivel de métrica: Detailed Unidad: recuento
`User Records per PutRecord sRequest`	El número total de registros de usuario almacenados en un `PutRecord sRequest` . Esto es aproximadamente equivalente al producto de las dos métricas anteriores. No disponible en el nivel de fragmento. Nivel de métrica: Detailed Unidad: recuento

Seguridad en Amazon Kinesis Data Streams

La seguridad en la nube AWS es la máxima prioridad. Como AWS cliente, se beneficiará de una arquitectura de centro de datos y red diseñada para cumplir con los requisitos de las organizaciones más sensibles a la seguridad.

La seguridad es una responsabilidad compartida entre usted AWS y usted. El modelo de responsabilidad compartida la describe como seguridad de la nube y seguridad en la nube:

- Seguridad de la nube: AWS es responsable de proteger la infraestructura que ejecuta AWS los servicios en la AWS nube. AWS también le proporciona servicios que puede utilizar de forma segura. Auditores externos prueban y verifican periódicamente la eficacia de nuestra seguridad en el marco de los programas de conformidad deAWS. Para más información acerca de los programas de conformidad que se aplican a Kinesis Data Streams, consulte Servicios deAWS en el ámbito del programa de conformidad.
- Seguridad en la nube: su responsabilidad viene determinada por el AWS servicio que utilice. Usted también es responsable de otros factores, incluida la confidencialidad de los datos, los requisitos de la empresa y la legislación y los reglamentos aplicables.

Esta documentación le ayuda a comprender cómo aplicar el modelo de responsabilidad compartida cuando se utiliza Kinesis Data Streams. En los siguientes temas, se le mostrará el proceso para configurar Kinesis Data Streams para satisfacer sus objetivos de seguridad y conformidad. También aprenderá a utilizar otros AWS servicios que pueden ayudarle a supervisar y proteger sus recursos de Kinesis Data Streams.

Temas

Protección de datos en Amazon Kinesis Data Streams

El cifrado del lado del servidor mediante claves AWS Key Management Service (AWS KMS) le permite cumplir con facilidad los estrictos requisitos de administración de datos al cifrar los datos en reposo en Amazon Kinesis Data Streams.

> **Note**
>
> Si necesita módulos criptográficos validados por FIPS 140-2 para acceder a AWS través de una interfaz de línea de comandos o una API, utilice un punto de conexión FIPS. Para obtener más información sobre los puntos de conexión de FIPS disponibles, consulte Estándar de procesamiento de la información federal (FIPS) 140-2.

Temas

¿Qué es el cifrado del lado del servidor para Kinesis Data Streams?

El cifrado del lado del servidor es una función de Amazon Kinesis Data Streams que cifra automáticamente los datos antes de que estén en reposo mediante la clave maestra del AWS KMS cliente (CMK) que especifique. Los datos se cifran antes de escribirlos en la capa de almacenamiento del flujo de Kinesis y se descifran después de recuperarlos del almacenamiento. Como resultado, los datos se cifran en reposo en el servicio de Kinesis Data Streams. Esto le permite cumplir requisitos normativos estrictos y mejorar la seguridad de sus datos.

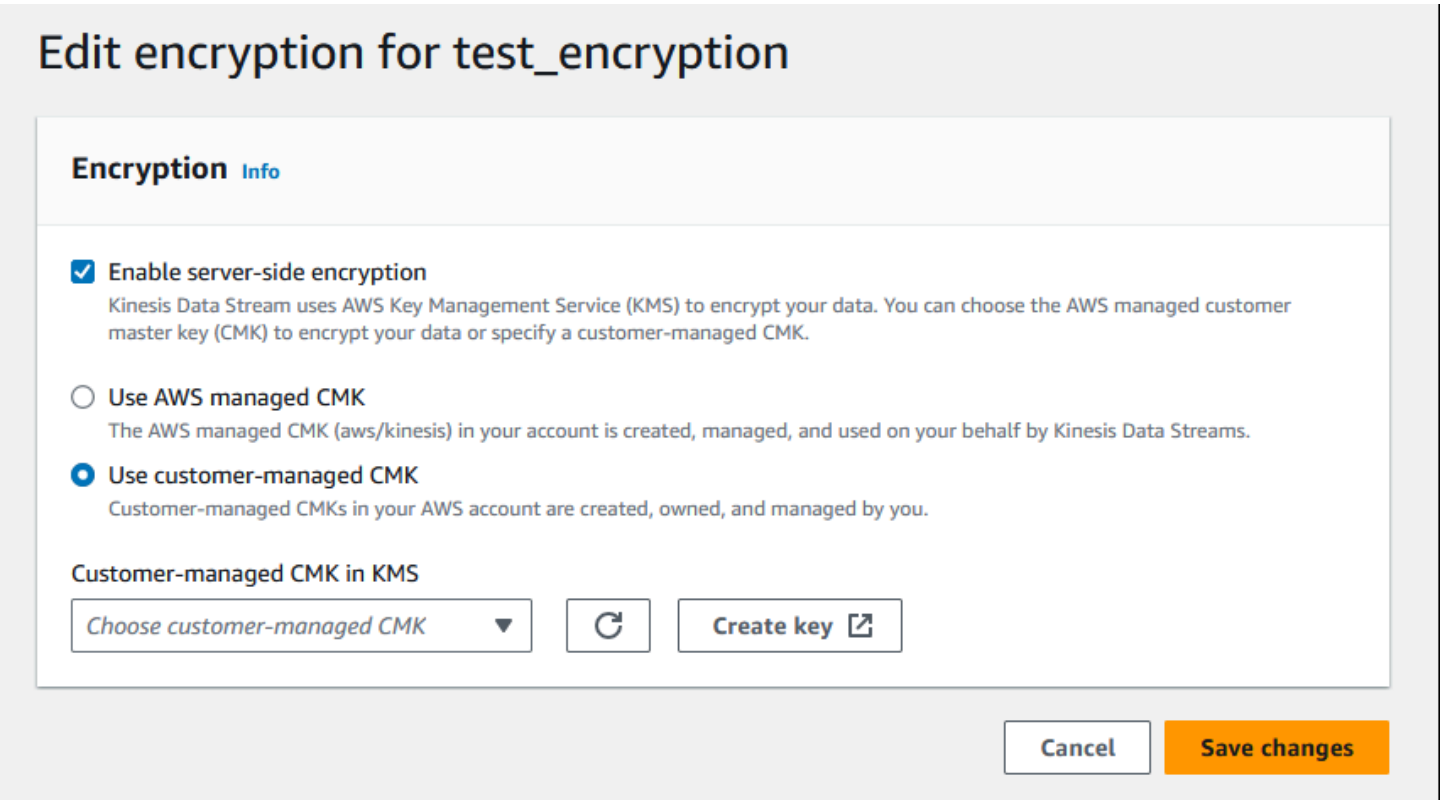
Con el cifrado del lado del servidor, sus consumidores y productores del flujo de Kinesis no tendrán que ocuparse de administrar claves principales ni realizar operaciones criptográficas. Los datos se cifran automáticamente a medida que entran y salen del servicio Kinesis Data Streams, por lo que los

datos en reposo se cifran. AWS KMS proporciona todas las claves maestras que utiliza la función de cifrado del lado del servidor. AWS KMS facilita el uso de una CMK para Kinesis gestionada por una CMK AWS KMS especificada AWSpor el usuario o una clave maestra importada al servicio. AWS KMS

> **Note**
>
> El cifrado en el servidor cifra los datos entrantes solo después de que se habilite el cifrado. Los datos preexistentes de una secuencia sin cifrar no se cifran tras activar el cifrado en el servidor.

Al cifrar sus flujos de datos y compartir el acceso con otras entidades principales, debe conceder permisos tanto a la política clave de la clave como a las políticas de IAM de la cuenta externa. AWS KMS Para obtener más información, consulte Allowing users in other accounts to use a KMS key (Permitir que los usuarios de otras cuentas utilicen una clave KMS).

Si ha habilitado el cifrado del lado del servidor para un flujo de datos con una clave KMS AWS administrada y desea compartir el acceso mediante una política de recursos, debe cambiar a la clave administrada por el cliente (CMK), como se muestra a continuación:



Además, debe permitir que las entidades principales compartidas tengan acceso a su CMK mediante las capacidades de uso compartido entre cuentas de KMS. Asegúrese de realizar también el cambio en las políticas de IAM para las entidades principales compartidas. Para obtener más información, consulte Allowing users in other accounts to use a KMS key (Permitir que los usuarios de otras cuentas utilicen una clave KMS).

Cuestiones sobre costos, regiones y rendimiento

Al aplicar el cifrado del lado del servidor, está sujeto al uso de la API y a los costes de las claves. AWS KMS A diferencia de las claves maestras de KMS personalizadas, la clave maestra del cliente (CMK) `(Default) aws/kinesis` se ofrece sin cargo alguno. No obstante, debe pagar los costos de uso de la API que Amazon Kinesis Data Streams genera en su nombre.

Los costos de uso de la API se aplican a todas las CMK, incluidas las personalizadas. Kinesis Data Streams llama a AWS KMS aproximadamente cada cinco minutos cuando rota la clave de datos. En un mes de 30 días, el coste total de las llamadas a la AWS KMS API iniciadas por una transmisión de Kinesis debería ser inferior a unos pocos dólares. Este coste varía en función del número de credenciales de usuario que utilice en sus productores y consumidores de datos, ya que cada credencial de usuario requiere una llamada única a la API. AWS KMSCuando utiliza un rol de IAM para la autenticación, cada llamada al rol genera credenciales de usuario únicas. Para ahorrar costos de KMS, tal vez le resulte conveniente almacenar las credenciales de usuario devueltas por la llamada al rol.

A continuación se describen los costos por recurso:

Claves

- La CMK para Kinesis gestionada AWS por (alias `aws/kinesis` =) es gratuita.
- Las claves de KMS generadas por el usuario están sujetas a los costos por clave de KMS. Para obtener más información, consulte Precios deAWS Key Management Service.

Los costos de uso de la API se aplican a todas las CMK, incluidas las personalizadas. Kinesis Data Streams llama a KMS aproximadamente cada cinco minutos cuando rota la clave de datos. En un mes de 30 días, el costo total de las llamadas a la API de KMS iniciadas por un flujo de datos de Kinesis debería ser inferior a unos pocos dólares. Tenga en cuenta que este costo varía en función del número de credenciales de usuario que utilice para generar y consumir datos, ya que cada credencial de usuario requiere una llamada de API única a KMS. AWS Al utilizar la función de IAM para la autenticación, cada una de ellas assume-role-call generará credenciales de usuario únicas,

por lo que puede que prefiera almacenar en caché las credenciales de usuario devueltas por esa función assume-role-call para ahorrar costes de KMS.

Uso de las API de KMS

Por cada flujo cifrado, al leer información de TIP y utilizar una única clave de acceso de usuario o cuenta de IAM entre lectores y escritores, el servicio Kinesis llama al servicio AWS KMS aproximadamente 12 veces cada cinco minutos. No leer el TIP podría provocar un aumento de las llamadas al AWS KMS servicio. Las solicitudes de API para generar nuevas claves de cifrado de datos están sujetas a costes AWS KMS de uso. Para obtener más información, consulte Precios deAWS Key Management Service: Uso.

Disponibilidad del cifrado en el servidor por región

Actualmente, el cifrado del lado del servidor de las transmisiones de Kinesis está disponible en todas las regiones compatibles con Kinesis Data Streams, AWS GovCloud incluidas las regiones de EE. UU. y China. Para más información sobre las regiones compatibles con Kinesis Data Streams, consulte https://docs.aws.amazon.com/general/latest/gr/ak.html.

Consideraciones sobre el rendimiento

Debido a la sobrecarga de servicio que supone aplicar el cifrado, el cifrado en el servidor aumenta la latencia típica de `PutRecord`, `PutRecords` y `GetRecords` en menos de 100 µs.

¿Cómo puedo comenzar a usar el cifrado en el servidor?

La forma más sencilla de empezar a utilizar el cifrado del lado del servidor es utilizar la clave de AWS Management Console servicio Amazon Kinesis KMS,. `aws/kinesis`

El siguiente procedimiento demuestra cómo habilitar el cifrado del lado del servidor para un flujo de Kinesis.

Para habilitar el cifrado del lado del servidor para un flujo de Kinesis

1. Inicie sesión en la consola de Amazon Kinesis Data Streams AWS Management Console y ábrala.
2. Cree o seleccione un flujo de Kinesis en la AWS Management Console.
3. Elija la pestaña Details (Detalles).
4. En Server-side encryption (Cifrado en el servidor), elija Edit (Editar).

5. A no ser que quiera utilizar una clave maestra de KMS generada por el usuario, asegúrese de que la clave maestra de KMS (Default) aws/kinesis esté seleccionada. Esta es la clave principal de KMS generada por el servicio de Kinesis. Seleccione Enabled (Habilitada) y, a continuación, elija Save (Guardar).

   > **Note**
   >
   > La clave maestra del servicio de Kinesis predeterminada es gratuita; sin embargo, las llamadas a la API que Kinesis realiza al AWS KMS servicio están sujetas a los costes de uso de KMS.

6. La secuencia realiza una transición a través del estado pendiente. Cuando la secuencia vuelve a un estado activo con el cifrado habilitado, todos los datos entrantes que se escriben en la secuencia se cifran con la clave maestra de KMS que haya seleccionado.
7. Para deshabilitar el cifrado del lado del servidor, seleccione Desactivar el cifrado del lado del servidor en el AWS Management Consoley, a continuación, seleccione Guardar.

Creación y uso de claves maestras de KMS generadas por el usuario

Esta sección describe cómo crear y utilizar sus propias claves principales de KMS en lugar de utilizar la clave principal administrada por Amazon Kinesis.

Creación de claves maestras de KMS generadas por el usuario

Para obtener instrucciones sobre cómo crear sus propias claves principales, consulte Creación de claves en la Guía para desarrolladores deAWS Key Management Service . Después de crear las claves para su cuenta, el servicio de Kinesis Data Streams devuelve estas claves en la lista de claves principales de KMS.

Uso de claves maestras de KMS generadas por el usuario

Una vez que se hayan aplicado los permisos correctos a sus consumidores, productores y administradores, podrá utilizar las claves maestras de KMS personalizadas en su propia AWS cuenta o en otra cuenta. AWS Todas las claves maestras de KMS de su cuenta aparecerán en la lista KMS Master Key (Clave maestra de KMS) de la AWS Management Console.

Para utilizar las claves maestras de KMS personalizadas en otra cuenta, necesitará permisos para utilizar las claves. También debe especificar el ARN de la clave maestra de KMS en el cuadro de introducción del ARN de la AWS Management Console.

Permisos de para utilizar claves maestras de KMS generadas por el usuario

Para poder utilizar el cifrado del lado del servidor con una clave maestra de KMS generada por el usuario, debe configurar políticas de AWS KMS claves que permitan el cifrado de las transmisiones y el cifrado y descifrado de los registros de las transmisiones. Para ver ejemplos y más información sobre AWS KMS los permisos, consulte Permisos de la API deAWS KMS: referencia sobre acciones y recursos.

> **Note**
>
> El uso de las claves de servicio predeterminadas para el cifrado no requiere que se apliquen permisos de IAM personalizados.

Antes de utilizar las claves principales de KMS generadas por el usuario, asegúrese de que sus productores y consumidores de flujos de Kinesis (entidades principales de IAM) sean usuarios en la política de claves principales de KMS. De lo contrario, las labores de escritura y lectura de una secuencia producirán un error, lo que, en definitiva, podría resultar en pérdida de datos, retrasos en el procesamiento, aplicaciones colgadas. Puede administrar los permisos para las claves de KMS con las políticas de IAM. Para obtener más información, consulte Uso de políticas de IAM con AWS KMS.

Ejemplo de permisos para productores

Sus productores de flujos de Kinesis deben tener el permiso `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"
    }
  ]
}
```

Ejemplo de permisos para consumidores

Sus consumidores de flujos de Kinesis deben tener el permiso `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 para Apache Flink y AWS Lambda utilice roles para consumir transmisiones de Kinesis. Asegúrese de agregar el permiso `kms:Decrypt` a los roles que usen estos consumidores.

Permisos de administrador de secuencias

Los administradores de flujos de Kinesis deben tener autorización para llamar a `kms:List*` y `kms:DescribeKey*`.

Comprobación y solución de problemas de permisos de claves de KMS

Tras habilitar el cifrado en una transmisión de Kinesis, le recomendamos que supervise el éxito de sus `getRecords` llamadas y de sus `putRecord` llamadas utilizando las siguientes métricas de Amazon CloudWatch : `putRecords`

- `PutRecord.Success`
- `PutRecords.Success`
- `GetRecords.Success`

Para obtener más información, consulte Supervisión de Amazon Kinesis Data Streams

Uso de Amazon Kinesis Data Streams con puntos de conexión de VPC de interfaz

Puede utilizar un punto de conexión de VPC de interfaz para evitar que el tráfico entre Amazon VPC y Kinesis Data Streams abandone la red de Amazon. Los puntos finales de la interfaz VPC no requieren una puerta de enlace a Internet, un dispositivo NAT, una conexión VPN o una conexión. AWS Direct Connect Los puntos de enlace de la interfaz VPC funcionan con una AWS tecnología que permite la comunicación privada entre AWS servicios mediante una interfaz de red elástica con direcciones IP privadas en su Amazon VPC. AWS PrivateLink Para obtener más información, consulte Amazon Virtual Private Cloud e Interface VPC Endpoints ().AWS PrivateLink

Temas

- Uso de puntos de conexión de VPC de interfaz para Kinesis Data Streams
- Control del acceso a puntos de conexión de VPCE para Kinesis Data Streams
- Disponibilidad de las políticas de punto de conexión de VPC para Kinesis Data Streams

Uso de puntos de conexión de VPC de interfaz para Kinesis Data Streams

Para empezar, no es necesario cambiar la configuración de las transmisiones, los productores ni los consumidores. Solo tiene que crear un punto de conexión de VPC de interfaz para que el tráfico de Kinesis Data Streams con origen y destino en los recursos de Amazon VPC comience a circular por el punto de conexión de VPC de interfaz. Para obtener más información, consulte Creación de un punto de conexión de interfaz.

La biblioteca de productores de Kinesis (KPL) y la biblioteca de consumidores de Kinesis (KCL) llaman a servicios como AWS Amazon y Amazon CloudWatch DynamoDB mediante puntos de enlace públicos o puntos de enlace de VPC de interfaz privada, según se utilicen. Por ejemplo, si su aplicación de KCL se ejecuta en una VPC con interfaz de DynamoDB con puntos de conexión de VPC habilitados, las llamadas entre DynamoDB y su aplicación de KCL fluyen a través del punto de conexión de VPC de interfaz.

Control del acceso a puntos de conexión de VPCE para Kinesis Data Streams

Las políticas de punto de enlace de la VPC le permiten controlar el acceso asociando una política a un punto de enlace de la VPC o utilizando campos adicionales en una política asociada a un usuario, grupo o rol de IAM para restringir el acceso para que solo se produzca a través del punto de enlace especificado de la VPC. Estas políticas se pueden utilizar para restringir el acceso a transmisiones específicas a un punto de enlace de la VPC especificado cuando se utilizan junto con las políticas de IAM para conceder acceso únicamente a las acciones de transmisión de datos de Kinesis a través del punto de enlace especificado de la VPC.

A continuación, se muestran ejemplos de políticas de punto de enlace para obtener acceder a las transmisiones de datos de Kinesis.

- Ejemplo de política de VPC: acceso de solo lectura: esta política de ejemplo se puede asociar a un punto de enlace de la VPC. (Para obtener más información, consulte Control de acceso a recursos de Amazon VPC). Limita las acciones a solo enumerar y describir una transmisión de datos de Kinesis a través del punto de enlace de la VPC al que está asociada.

```
{
  "Statement": [
    {
      "Sid": "ReadOnly",
      "Principal": "*",
      "Action": [
        "kinesis:List*",
        "kinesis:Describe*"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

- Ejemplo de política de VPC: restringe el acceso a una transmisión de datos de Kinesis específica ; esta política de ejemplo se puede asociar a un punto de enlace de la VPC. Limita el acceso a una secuencia de datos específica a través del punto de enlace de la VPC al que está asociada.

```
{
  "Statement": [
    {
      "Sid": "AccessToSpecificDataStream",
      "Principal": "*",
      "Action": "kinesis:*",
      "Effect": "Allow",
      "Resource": "arn:aws:kinesis:us-east-1:123456789012:stream/MyStream"
    }
  ]
}
```

- Ejemplo de política de IAM: restringe el acceso a una transmisión específica solo desde un punto de enlace específico de la VPC ; esta política de ejemplo se puede asociar a un usuario, rol o grupo de IAM. Restringe el acceso a una transmisión de datos de Kinesis especificada para que solo se produzca desde un punto de enlace especificado de la VPC.

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

Disponibilidad de las políticas de punto de conexión de VPC para Kinesis Data Streams

Los puntos de conexión de VPC de interfaz de Kinesis Data Streams con políticas se admiten en las siguientes regiones:

- Europa (París)
- Europa (Irlanda)
- Este de EE. UU. (Norte de Virginia)
- Europa (Estocolmo)
- Este de EE. UU. (Ohio)
- Europa (Fráncfort)
- América del Sur (São Paulo)
- Europa (Londres)
- Asia-Pacífico (Tokio)
- Oeste de EE. UU. (Norte de California)
- Asia-Pacífico (Singapur)
- Asia-Pacífico (Sídney)
- China (Pekín)
- China (Ningxia)
- Asia-Pacífico (Hong Kong)
- Medio Oriente (Baréin)
- Medio Oriente (EAU)
- Europa (Milán)
- África (Ciudad del Cabo)
- Asia-Pacífico (Bombay)
- Asia-Pacífico (Seúl)
- Canadá (Centro)
- Oeste de EE. UU. (Oregón) excepto usw2-az4
- AWS GovCloud (Este de EE. UU.)
- AWS GovCloud (Estados Unidos-Oeste)

- Asia-Pacífico (Osaka)
- Europa (Zúrich)
- Asia-Pacífico (Hyderabad)

Control del acceso a los recursos de Amazon Kinesis Data Streams mediante IAM

AWS Identity and Access Management (IAM) le permite hacer lo siguiente:

- Cree usuarios y grupos en su cuenta AWS
- Asigna credenciales de seguridad únicas a cada usuario de tu AWS cuenta
- Controle los permisos de cada usuario para realizar tareas utilizando AWS los recursos
- Permite que los usuarios de otra AWS cuenta compartan tus AWS recursos
- Cree roles para su AWS cuenta y defina los usuarios o servicios que pueden asumirlos
- Utilice las identidades existentes de su empresa a fin de conceder permisos para realizar tareas con AWS recursos

Al utilizar IAM con Kinesis Data Streams, puede controlar si los usuarios de su organización pueden realizar una tarea mediante acciones específicas de la API de Kinesis Data Streams y si pueden utilizar recursos específicos de AWS .

Si está desarrollando una aplicación con la biblioteca de clientes de Kinesis (KCL), su política debe incluir permisos para Amazon DynamoDB y Amazon; la KCL utiliza DynamoDB para realizar un seguimiento de CloudWatch la información de estado de la aplicación y para enviar las métricas de KCL en su nombre. CloudWatch CloudWatch Para obtener más información sobre KCL, consulte Desarrollo de consumidores de KCL 1.x.

Para obtener más información sobre IAM, consulte lo siguiente:

- AWS Identity and Access Management (IAM)
- Introducción
- Guía del usuario de IAM

Para más información acerca de IAM y DynamoDB, consulte Uso de IAM para controlar el acceso a recursos de Amazon DynamoDB en la Guía para desarrolladores de Amazon DynamoDB.

Para obtener más información sobre IAM y Amazon CloudWatch, consulta Cómo controlar el acceso de los usuarios a tu AWS cuenta en la Guía del CloudWatch usuario de Amazon.

Contenido

Sintaxis de la política

Una política de IAM es un documento JSON que contiene una o varias instrucciones. Cada instrucción tiene la estructura siguiente:

```
{
  "Statement":[{
    "Effect":"effect",
    "Action":"action",
    "Resource":"arn",
    "Condition":{
      "condition":{
        "key":"value"
        }
      }
    }
  ]
}
```

Una instrucción está compuesta por varios elementos:

- Effect: el valor de effect puede ser `Allow` o `Deny`. De forma predeterminada, los usuarios de IAM no tienen permiso para utilizar los recursos y las acciones de la API, por lo que se deniegan todas las solicitudes. Si se concede un permiso explícito se anula el valor predeterminado. Una denegación explícita invalida cualquier permiso concedido.
- Action: el valor de action es la acción de la API para la que concede o deniega permisos.

- Resource: el recurso al que afecta la acción. Para especificar un recurso en la instrucción, debe usar el nombre de recurso de Amazon (ARN).
- Condition: las condiciones son opcionales. Se pueden usar para controlar cuándo entrará en vigor la política.

Al crear y administrar las políticas de IAM, es posible que quiera utilizar el generador de políticas de IAM y el simulador de política de IAM.

Acciones para Kinesis Data Streams

En una instrucción de política de IAM, puede especificar cualquier acción de API de cualquier servicio que sea compatible con IAM. Para Kinesis Data Streams, utilice el prefijo siguiente con el nombre de la acción de la API: `kinesis:`. Por ejemplo, `kinesis:CreateStream`, `kinesis:ListStreams` y `kinesis:DescribeStreamSummary`.

Para especificar varias acciones en una única instrucción, sepárelas con comas del siguiente modo:

```
"Action": ["kinesis:action1", "kinesis:action2"]
```

También puede utilizar caracteres comodín para especificar varias acciones. Por ejemplo, puede especificar todas las acciones cuyo nombre comience por la palabra "Get" del siguiente modo:

```
"Action": "kinesis:Get*"
```

Para especificar todas las acciones de Kinesis Data Streams, utilice el carácter comodín * del siguiente modo:

```
"Action": "kinesis:*"
```

Para obtener la lista completa de las acciones de la API de Kinesis Data Streams, consulte Referencia de la API de Amazon Kinesis.

Nombres de recursos de Amazon (ARN) para Kinesis Data Streams

Cada instrucción de política de IAM se aplica a los recursos especificados utilizando sus ARN.

Utilice el siguiente formato de recurso de ARN para los flujos de datos de Kinesis:

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

Por ejemplo:

```
"Resource": arn:aws:kinesis:*:111122223333:stream/my-stream
```

Ejemplo de políticas para Kinesis Data Streams

Los siguientes ejemplos de políticas demuestran cómo puede controlar el acceso de los usuarios a sus flujos de datos Kinesis.

Example 1: Allow users to get data from a stream

Example

Esta política permite a un usuario o grupo realizar las operaciones `DescribeStreamSummary`, `GetShardIterator` y `GetRecords` en la secuencia especificada y `ListStreams` en cualquier secuencia. Esta política podría aplicarse a los usuarios que puedan obtener los datos de una determinada secuencia.

```
{
    "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

Esta política permite a un usuario o grupo usar la operación `PutRecord` con cualquiera de las secuencias de la cuenta. Esta política podría aplicarse a usuarios que puedan agregar registros de datos a todas las secuencias de una cuenta.

```
{
    "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

Esta política permite a un usuario o grupo utilizar cualquier operación de Kinesis Data Streams en el flujo especificado. Esta política podría aplicarse a usuarios que deberían tener control administrativo en una secuencia específica.

```
{
    "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

Esta política permite a un usuario o grupo utilizar cualquier operación de Kinesis Data Streams en cualquier flujo de la cuenta. Dado que esta política concede acceso completo a todas las secuencias, debe restringirla solo a los administradores.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": [
                "arn:aws:kinesis:*:111122223333:stream/*"
            ]
        }
    ]
}
```

Cómo compartir su flujo de datos con otra cuenta

Adjunte una política basada en recursos a su flujo de datos para conceder acceso a otra cuenta, usuario de IAM o rol de IAM. Las políticas basadas en recursos son documentos de políticas JSON que puede adjuntar a un recurso como un flujo de datos. Estas políticas conceden a la entidad principal especificada el permiso para ejecutar acciones concretas en el recurso y definen en qué condiciones son aplicables. Una política puede tener varias declaraciones. Debe especificar una entidad principal en una política basada en recursos. Los principales pueden incluir cuentas, usuarios, roles, usuarios federados o servicios. AWS Puede configurar políticas en la consola, la API o el SDK de Kinesis Data Streams.

Tenga en cuenta que compartir el acceso con consumidores registrados, como Enhanced Fan Out, requiere una política tanto en el ARN del flujo de datos como en el ARN del consumidor.

Habilitar el acceso entre cuentas

Para habilitar el acceso entre cuentas, puede especificar toda una cuenta o entidades de IAM de otra cuenta como la entidad principal de una política en función de recursos. Añadir a una política en función de recursos una entidad principal entre cuentas es solo una parte del establecimiento de una

relación de confianza. Cuando el principal y el recurso están en AWS cuentas distintas, también debe utilizar una política basada en la identidad para conceder al principal acceso al recurso. Sin embargo, si la política en función de recursos concede el acceso a una entidad principal de la misma cuenta, no es necesaria una política basada en identidad adicional.

Para obtener más información sobre el uso de políticas basadas en recursos para acceso entre cuentas, consulte Acceso entre cuentas por recurso en IAM.

Los administradores del flujo de datos pueden usar AWS Identity and Access Management políticas para especificar quién tiene acceso a qué. Es decir, qué entidad principal puede realizar acciones en qué recursos y en qué condiciones. El elemento `Action` de una política JSON describe las acciones que puede utilizar para conceder o denegar el acceso en una política. Las acciones políticas suelen tener el mismo nombre que la operación de AWS API asociada.

Acciones de Kinesis Data Streams que se pueden compartir:

Acción de	Nivel de acceso
DescribeStreamConsumer	Consumidor
DescribeStreamSummary	Flujo de datos
GetRecords	Flujo de datos
GetShardIterator	Flujo de datos
ListShards	Flujo de datos
PutRecord	Flujo de datos
PutRecords	Flujo de datos
SubscribeToShard	Consumidor

A continuación, se muestran ejemplos de uso de una política basada en recursos para conceder acceso entre cuentas a su flujo de datos o a un consumidor registrado.

Para realizar una acción entre cuentas, debe especificar el ARN del flujo para el acceso al flujo de datos y el ARN del consumidor para el acceso de los consumidores registrados.

Ejemplo de políticas basadas en recursos para Kinesis Data Streams

Compartir un consumidor registrado implica tanto una política de flujo de datos como una política de consumidor, debido a las medidas necesarias.

> **Note**
>
> Los siguientes son ejemplos de valores válidos de `Principal`:
>
> - `{"AWS": "123456789012"}`
> - Usuario de IAM: `{"AWS": "arn:aws:iam::123456789012:user/user-name"}`
> - Rol de IAM: `{"AWS":["arn:aws:iam::123456789012:role/role-name"]}`
> - Múltiples entidades principales (puede ser una combinación de cuenta, usuario o rol): `{"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

Declaración de política de flujo de datos:

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

Declaración de política de consumidor:

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

No se admite el comodín (*) para las acciones o el campo principal a fin de mantener el principio del privilegio mínimo.

Administrar la política del flujo de datos de forma programática

Además de eso AWS Management Console, Kinesis Data Streams tiene tres API para administrar su política de transmisión de datos:

- PutResourcePolicy

- GetResourcePolicy
- DeleteResourcePolicy

Use `PutResourePolicy` para adjuntar o sobrescribir una política para un flujo de datos o un consumidor. Use `GetResourcePolicy` para comprobar y ver una política para el flujo de datos o el consumidor especificado. Use `DeleteResourcePolicy` para eliminar una política para el flujo de datos o el consumidor especificado.

Límites de políticas

Las políticas de recursos de Kinesis Data Streams tienen las siguientes restricciones:

- No se admite el comodín (*) para las acciones o las secciones principales a fin de mantener el principio del privilegio mínimo.
- AWS No se admiten directores de servicio como directores para evitar posibles confusiones entre los diputados.
- No se admiten entidades principales federadas.
- No se admiten los ID de usuario canónicos.
- El tamaño de la política no puede superar los 20 KB.

Compartir el acceso a datos cifrados

Si ha habilitado el cifrado del lado del servidor para un flujo de datos con una clave KMS AWS administrada y desea compartir el acceso mediante una política de recursos, debe cambiar a la clave administrada por el cliente (CMK). Para obtener más información, consulte ¿Qué es el cifrado del lado del servidor para Kinesis Data Streams?. Además, debe permitir que las entidades principales compartidas tengan acceso a su CMK mediante las capacidades de uso compartido entre cuentas de KMS. Asegúrese de realizar también el cambio en las políticas de IAM para las entidades principales compartidas. Para obtener más información, consulte Allowing users in other accounts to use a KMS key (Permitir que los usuarios de otras cuentas utilicen una clave KMS).

Configurar una AWS Lambda función para leer contenido de Kinesis Data Streams en otra cuenta

Si necesita ver un ejemplo de cómo configurar una función de Lambda para leer contenido de Kinesis Data Streams en otra cuenta, consulte Compartir el acceso con funciones multicuenta AWS Lambda.

Acceso compartido mediante políticas basadas en recursos

> **Note**
>
> Actualizar una política basada en recursos existente implica reemplazarla, así que asegúrese de incluir toda la información necesaria en su nueva política.

Compartir el acceso con funciones multicuenta AWS Lambda

Operador Lambda

1. Vaya a la consola de IAM para crear una función de IAM que se utilizará como función de ejecución de Lambda para su función. AWS Lambda Añada la política de IAM gestionada `AWSLambdaKinesisExecutionRole` que tenga los permisos de invocación de Kinesis Data Streams y Lambda necesarios. Esta política también concede acceso a todos los recursos potenciales de Kinesis Data Streams a los que pueda tener acceso.
2. En la AWS Lambda consola, cree una AWS Lambda función para procesar los registros de una transmisión de datos de Kinesis Data Streams y, durante la configuración de la función de ejecución, elija la función que creó en el paso anterior.
3. Proporcione la función de ejecución al propietario del recurso de Kinesis Data Streams para configurar la política de recursos.
4. Terminar de configurar la función de Lambda.

Propietario de recursos de Kinesis Data Streams

1. Obtenga la función de ejecución de Lambda entre cuentas que invocará la función de Lambda.
2. En la consola de Amazon Kinesis Data Streams, seleccione el flujo de datos. Seleccione la pestaña Compartir el flujo de datos y, a continuación, el botón Crear política de uso compartido para iniciar el editor visual de políticas. Para compartir un consumidor registrado dentro de un flujo de datos, elija el consumidor y, a continuación, elija Crear política de uso compartido. También puede escribir la política de JSON directamente.
3. Especifique la función de ejecución entre cuentas de Lambda como la entidad principal y las acciones exactas de Kinesis Data Streams a las que comparte el acceso. Asegúrese de incluir la acción `kinesis:DescribeStream`. Para obtener más información sobre ejemplos de políticas

de recursos para Kinesis Data Streams, consulte Ejemplo de políticas basadas en recursos para Kinesis Data Streams.

4. Elija Crear política o utilice la PutResourcePolicypara adjuntar la política a su recurso.

Compartir el acceso con consumidores de KCL entre cuentas

- Si utiliza KCL 1.x, asegúrese de utilizar KCL 1.15.0 o superior.
- Si utiliza KCL 2.x, asegúrese de utilizar KCL 2.5.3 o superior.

Operador de KCL

1. Proporcione al propietario del recurso el usuario de IAM o el rol de IAM que ejecutará la aplicación KCL.
2. Pida al propietario del recurso el flujo de datos o el ARN del consumidor.
3. Asegúrese de especificar el ARN del flujo proporcionado como parte de la configuración de KCL.
   - Para KCL 1.x: utilice el KinesisClientLibConfigurationconstructor y proporcione el ARN de flujo.
   - Para KCL 2.x: puede proporcionar solo el ARN de la transmisión o la biblioteca de clientes de StreamTrackerKinesis. ConfigsBuilder Para ello StreamTracker, proporcione el ARN de la transmisión y la época de creación de la tabla de arrendamiento de DynamoDB que genera la biblioteca. Si quieres leer información de un consumidor registrado compartido, como Enhanced Fan-Out, utiliza StreamTracker y proporciona también el ARN del consumidor.

Propietario de recursos de Kinesis Data Streams

1. Obtenga el usuario de IAM entre cuentas o el rol de IAM que ejecutará la aplicación KCL.
2. En la consola de Amazon Kinesis Data Streams, seleccione el flujo de datos. Seleccione la pestaña Compartir el flujo de datos y, a continuación, el botón Crear política de uso compartido para iniciar el editor visual de políticas. Para compartir un consumidor registrado dentro de un flujo de datos, elija el consumidor y, a continuación, elija Crear política de uso compartido. También puede escribir la política de JSON directamente.
3. Especifique el usuario de IAM o el rol de IAM de la aplicación KCL entre cuentas como las entidades principales y las acciones exactas de Kinesis Data Streams a las que comparte el acceso. Para obtener más información sobre ejemplos de políticas de recursos para Kinesis Data Streams, consulte Ejemplo de políticas basadas en recursos para Kinesis Data Streams.

4. Elija Crear política o utilice la PutResourcePolicypara adjuntar la política a su recurso.

Compartir el acceso a datos cifrados

Si ha habilitado el cifrado del lado del servidor para un flujo de datos con una clave KMS AWS administrada y desea compartir el acceso mediante una política de recursos, debe cambiar a la clave administrada por el cliente (CMK). Para obtener más información, consulte ¿Qué es el cifrado del lado del servidor para Kinesis Data Streams?. Además, debe permitir que las entidades principales compartidas tengan acceso a su CMK mediante las capacidades de uso compartido entre cuentas de KMS. Asegúrese de realizar también el cambio en las políticas de IAM para las entidades principales compartidas. Para obtener más información, consulte Allowing users in other accounts to use a KMS key (Permitir que los usuarios de otras cuentas utilicen una clave KMS).

Validación de conformidad para Amazon Kinesis Data Streams

Los auditores externos evalúan la seguridad y la conformidad de Amazon Kinesis Data Streams como parte de AWS varios programas de conformidad. Estos incluyen SOC, PCI, FedRAMP, HIPAA y otros.

Para obtener una lista de AWS los servicios incluidos en el ámbito de los programas de conformidad específicos, consulte AWS Servicios incluidos en el ámbito de aplicación por programa de conformidad. Para obtener información general, consulte Programas de conformidad deAWS.

Puede descargar informes de auditoría de terceros utilizando AWS Artifact. Para obtener más información, consulta Descarga de informes en AWS Artifact.

Su responsabilidad con la conformidad al utilizar Kinesis Data Streams se determina en función de la confidencialidad de los datos, los objetivos de conformidad de su empresa, así como de la legislación y los reglamentos aplicables. Si el uso de Kinesis Data Streams está sujeto al cumplimiento de normas como HIPAA, PCI o FedRAMP AWS , proporciona recursos que le ayudarán a:

- Guías de inicio rápido sobre seguridad y conformidad: estas guías de implementación analizan las consideraciones arquitectónicas y proporcionan los pasos para implementar entornos básicos centrados en la seguridad y el cumplimiento. AWS
- Documento técnico sobre cómo diseñar una arquitectura para la seguridad y el cumplimiento de la HIPAA: este documento técnico describe cómo las empresas pueden utilizar para crear aplicaciones que cumplan con la HIPAA. AWS

- AWS Recursos de cumplimiento: esta colección de libros de trabajo y guías que pueden aplicarse a su sector y ubicación
- AWS Config— Este AWS servicio que evalúa en qué medida las configuraciones de sus recursos cumplen con las prácticas internas, las directrices del sector y las normativas.
- AWS Security Hub— Este AWS servicio proporciona una visión integral del estado de su seguridad AWS que le ayuda a comprobar el cumplimiento de los estándares y las mejores prácticas del sector de la seguridad.

Resiliencia de Amazon Kinesis Data Streams

La infraestructura AWS global se basa en AWS regiones y zonas de disponibilidad. AWS Las regiones proporcionan varias zonas de disponibilidad aisladas y separadas físicamente, que están conectadas mediante redes de baja latencia, alto rendimiento y alta redundancia. Con las zonas de disponibilidad, puede diseñar y utilizar aplicaciones y bases de datos que realizan una conmutación por error automática entre zonas de disponibilidad sin interrupciones. Las zonas de disponibilidad tienen una mayor disponibilidad, tolerancia a errores y escalabilidad que las infraestructuras tradicionales de centros de datos únicos o múltiples.

Para obtener más información sobre AWS las regiones y las zonas de disponibilidad, consulte Infraestructura global.AWS

Además de la infraestructura AWS global, Kinesis Data Streams ofrece varias funciones que le ayudan a satisfacer sus necesidades de respaldo y resiliencia de datos.

Recuperación de desastres en Amazon Kinesis Data Streams

Cuando utiliza una aplicación de Amazon Kinesis Data Streams para procesar datos procedentes de un flujo, pueden producirse errores en los siguientes niveles:

- Error en un procesador de registros
- Error en un proceso de trabajo, o en la instancia de la aplicación que creó la instancia del proceso de trabajo
- Error en una instancia EC2 que aloja una o más instancias de la aplicación

Error del procesador de registros

El trabajador invoca los métodos del procesador de registros mediante tareas de Java. ExecutorService Si una tarea produce un error, el proceso de trabajo conserva el control del fragmento que estaba procesando el procesador de registros. El proceso de trabajo inicia un nuevo procesador de registros para procesar dicho fragmento. Para obtener más información, consulte Limitación controlada de lectura.

Error en el proceso de trabajo o la aplicación

Si un proceso de trabajo o una instancia de la aplicación de Amazon Kinesis Data Streams produce un error, debe definir y encargarse de la situación. Por ejemplo, si el método `Worker.run` crea una excepción, debe identificarla y administrarla.

Si el error se produce en la propia aplicación, debe detectarlo y reiniciarla. Cuando se inicia la aplicación, crea una instancia de un nuevo proceso de trabajo, que a su vez crea instancias de nuevos procesadores de registros a los que se asignan automáticamente fragmentos para procesar. Podrían ser los mismos fragmentos que procesaban estos procesadores de registros antes del error, o fragmentos nuevos para estos procesadores.

Si el proceso de trabajo o la aplicación tiene un error, pero no se detecta, y existen otras instancias de la aplicación en ejecución en otras instancias EC2, los procesos de trabajo en estas instancias administran el error. Crean procesadores de registros adicionales para procesar los fragmentos que ya no están siendo procesados por proceso de trabajo que experimentó el error. La carga sobre estas otras instancias EC2 aumenta en consecuencia.

La situación que se describe aquí supone que, aunque el proceso de trabajo o la aplicación haya experimentado un error, la instancia de EC2 que los aloja sigue en ejecución y, por lo tanto, no ha de reiniciarla un grupo de Auto Scaling.

Error en una instancia de Amazon EC2

Se recomienda que ejecute las instancias de EC2 para su aplicación en un grupo de Auto Scaling. De esta forma, si una de las instancias de EC2 experimenta un error, el grupo de Auto Scaling lanza automáticamente una instancia nueva para sustituirla. Debe configurar las instancias para lanzar la aplicación de Amazon Kinesis Data Streams al iniciar.

Seguridad de infraestructura en Kinesis Data Streams

Como servicio gestionado, Amazon Kinesis Data Streams está protegido por AWS los procedimientos de seguridad de red global que se describen en el documento técnico Amazon Web Services: Overview of Security Processes.

Utilice las llamadas a la API AWS publicadas para acceder a Kinesis Data Streams a través de la red. Los clientes deben ser compatibles con Transport Layer Security (TLS) 1.2 o una versión posterior. Los clientes también deben ser compatibles con conjuntos de cifrado con confidencialidad directa total (PFS) tales como Ephemeral Diffie-Hellman (DHE) o Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). La mayoría de los sistemas modernos como Java 7 y posteriores son compatibles con estos modos.

Además, las solicitudes deben estar firmadas mediante un ID de clave de acceso y una clave de acceso secreta que esté asociada a una entidad principal de IAM. También puede utilizar AWS Security Token Service (AWS STS) para generar credenciales de seguridad temporales para firmar solicitudes.

Prácticas recomendadas de seguridad para Kinesis Data Streams

Amazon Kinesis Data Streams proporciona una serie de características de seguridad que debe tener en cuenta a la hora de desarrollar e implementar sus propias políticas de seguridad. Las siguientes prácticas recomendadas son directrices generales y no suponen una solución de seguridad completa. Puesto que es posible que estas prácticas recomendadas no sean adecuadas o suficientes para el entorno, considérelas como consideraciones útiles en lugar de como normas.

Implementación del acceso a los privilegios mínimos

Cuando concede permisos, debe decidir a quién concede cada permiso y para qué recurso de Kinesis Data Streams se lo concede. Habilite las acciones específicas que desea permitir en dichos recursos. Por lo tanto, debe conceder únicamente los permisos obligatorios para realizar una tarea. La implementación del acceso con privilegios mínimos es esencial a la hora de reducir los riesgos de seguridad y el impacto que podrían causar los errores o los intentos malintencionados.

Uso de roles de IAM

Las aplicaciones de clientes y productores deben tener credenciales válidas para acceder a los flujos de datos de Kinesis. No debe almacenar AWS las credenciales directamente en una aplicación

cliente o en un bucket de Amazon S3. Estas son las credenciales a largo plazo que no rotan automáticamente y que podrían tener un impacto empresarial significativo si se comprometen.

En su lugar, tiene que utilizar un rol de IAM para administrar credenciales temporales de las aplicaciones de clientes y productores con el fin de acceder a los flujos de datos de Kinesis. Al utilizar un rol, no tiene que utilizar credenciales a largo plazo (como un nombre de usuario y una contraseña o claves de acceso) para acceder a otros recursos.

Para obtener más información, consulte los siguientes temas de la guía del usuario de IAM:

- Roles de IAM
- Situaciones habituales con los roles: usuarios, aplicaciones y servicios

Implementación del cifrado en el servidor en recursos dependientes

Los datos en reposo y los datos en tránsito se pueden cifrar en Kinesis Data Streams. Para obtener más información, consulte Protección de datos en Amazon Kinesis Data Streams.

Úselo CloudTrail para monitorear las llamadas a la API

Kinesis Data Streams está integrado AWS CloudTrailcon un servicio que proporciona un registro de las acciones realizadas por un usuario, un rol o AWS un servicio en Kinesis Data Streams.

Con la información recopilada por CloudTrail, puede determinar la solicitud que se realizó a Kinesis Data Streams, la dirección IP desde la que se realizó la solicitud, quién la realizó, cuándo se realizó y detalles adicionales.

Para obtener más información, consulte the section called “Registros de las llamadas a la API de Amazon Kinesis Data Streams mediante AWS CloudTrail”.

Historial del documento

En la siguiente tabla se describen los cambios importantes que se han hecho en la documentación de esta versión de Amazon Kinesis Data Streams.

Cambio	Descripción	Fecha de modificación
Se agregó soporte para compartir flujos de datos entre cuentas.	Cómo compartir su flujo de datos con otra cuenta añadido.	22 de noviembre de 2023
Se agregó compatibilidad con los modos de capacidad de flujos de datos aprovisionados y bajo demanda.	Elegir el modo de capacidad del flujo de datos añadido.	29 de noviembre de 2021
Nuevo contenido para el cifrado del lado del servidor.	Protección de datos en Amazon Kinesis Data Streams añadido.	7 de julio de 2017
Nuevo contenido para mejorar CloudWatch las métricas.	Actualizado Supervisión de Amazon Kinesis Data Streams.	19 de abril de 2016
Nuevo contenido para agente de Kinesis mejorado.	Actualizado Escritura en Amazon Kinesis Data Streams mediante el agente de Kinesis.	11 de abril de 2016
Nuevo contenido para utilizar agentes de Kinesis.	Escritura en Amazon Kinesis Data Streams mediante el agente de Kinesis añadido.	2 de octubre de 2015

Cambio	Descripción	Fecha de modificación
Actualización del contenido de KPL para la versión 0.10.0.	Desarrollo de productores con Amazon Kinesis Producer Library añadido.	15 de julio de 2015
Actualización del tema de métricas de KCL para las métricas configurables.	Supervisión de Kinesis Client Library con Amazon CloudWatch añadido.	9 de julio de 2015
Contenido reorganizado.	Reorganización significativa de los temas de contenido para ofrecer una vista esquemática más concisa y una agrupación más lógica.	01 de julio de 2015
Nuevo tema de la guía para desarrolladores de KPL.	Desarrollo de productores con Amazon Kinesis Producer Library añadido.	02 de junio de 2015
Nuevo tema de métricas de KCL.	Supervisión de Kinesis Client Library con Amazon CloudWatch añadido.	19 de mayo de 2015
Compatibilidad para .NET en la KCL	Desarrollo de un consumidor de Kinesis Client Library en .NET añadido.	1 de mayo de 2015
Compatibilidad para Node.js en la KCL	Desarrollo de un consumidor de Kinesis Client Library en Node.js añadido.	26 de marzo de 2015
Compatibilidad para Ruby en la KCL	Se han agregado enlaces a la biblioteca de Ruby de la KCL.	12 de enero de 2015

Cambio	Descripción	Fecha de modificación
Nueva API PutRecords	Se agregó información sobre la nueva PutRecords API athe section called “Agregar varios registros con PutRecords”.	15 de diciembre de 2014
Compatibilidad con el etiquetado	Etiquetado de flujos de Amazon Kinesis Data Streams añadido.	11 de septiembre de 2014
Nueva CloudWatch métrica	Se ha añadido la métrica `GetRecords.IteratorAgeMilliseconds` a Métricas y dimensiones de Amazon Kinesis Data Streams.	3 de septiembre de 2014
Nuevo capítulo de monitorización	Se añadieron Supervisión de Amazon Kinesis Data Streams y Supervisión del servicio Amazon Kinesis Data Streams con Amazon CloudWatch.	30 de julio de 2014
Límite de fragmentos predeterminado	Se ha actualizado Cuotas y límites: el límite de fragmentos predeterminado se ha aumentado de 5 a 10.	25 de febrero de 2014
Límite de fragmentos predeterminado	Se ha actualizado Cuotas y límites: el límite de fragmentos predeterminado se ha aumentado de 2 a 5.	28 de enero de 2014
Actualizaciones de versión de la API	Actualizaciones para la versión 2013-12-02 de la API de Kinesis Data Streams.	12 de diciembre de 2013
Lanzamiento inicial	Versión inicial de la Guía para desarrolladores de Amazon Kinesis.	14 de noviembre de 2013

Glosario de AWS

Para ver la terminología más reciente de AWS, consulte el Glosario de AWS en la Referencia de Glosario de AWS.

Las traducciones son generadas a través de traducción automática. En caso de conflicto entre la traducción y la version original de inglés, prevalecerá la version en inglés.