aws

Guia do Desenvolvedor

Amazon Kinesis Data Streams

Amazon Kinesis Data Streams: Guia do Desenvolvedor

Table of Contents

O que é o Amazon Kinesis Data Streams?

Você pode usar o Amazon Kinesis Data Streams para coletar e processar grandes fluxos de registros de dados em tempo real. Você pode criar aplicações de processamento de dados, conhecidos como aplicações do Kinesis Data Streams. Uma aplicação típica do Kinesis Data Streams lê dados de um fluxo de dados como registros de dados. Esses aplicativos podem usar a Kinesis Client Library e podem ser executados em instâncias da AmazonEC2. Você pode enviar os registros processados para painéis, usá-los para gerar alertas, alterar dinamicamente as estratégias de preços e publicidade ou enviar dados para vários outros serviços da AWS . Para obter informações sobre os recursos e preços do Kinesis Data Streams, consulte Amazon Kinesis Data Streams.

O Kinesis Data Streams faz parte da plataforma de dados de streaming Kinesis, junto com o Firehose, o Kinesis Video Streams e o Managed Service para Apache Flink.

Para obter mais informações sobre soluções de AWS big data, consulte Big Data on AWS. Para obter mais informações sobre as soluções de dados de streaming da AWS , consulte O que são dados em streaming?

Tópicos

- O que posso fazer com o Kinesis Data Streams?
- Benefícios do uso do Kinesis Data Streams
- Serviços relacionados

O que posso fazer com o Kinesis Data Streams?

Você pode usar o Kinesis Data Streams para entrada e agregação de dados de forma rápida e contínua. O tipo de dados usado pode incluir dados de log de infraestrutura de TI, logs de aplicativo, mídias sociais, feeds de dados de mercado e dados de sequência de cliques da web. Como o tempo de resposta para a entrada e o processamento de dados é em tempo real, o processamento geralmente é leve.

Estes são alguns cenários típicos de uso do Kinesis Data Streams:

Log acelerado e consumo e processamento de dados

> Você pode ter produtores que gerem dados diretamente em um stream. Por exemplo, gere logs de sistemas e aplicativos e eles estarão disponíveis para processamento em segundos. Isso

evitará que os dados de log sejam perdidos se o servidor de front-end ou de aplicações falhar. O Kinesis Data Streams fornece uma entrada acelerada de dados, pois você não organiza os dados em lotes nos servidores antes de enviá-los.

Métricas e relatórios em tempo real

Você pode usar dados coletados no Kinesis Data Streams para análise simples de dados e geração de relatórios em tempo real. Por exemplo, seu aplicativo de processamento de dados pode funcionar em métricas e geração de relatórios para logs do sistema e de aplicativos à medida que os dados passam por ele em vez de esperar receber lotes de dados.

Análise de dados em tempo real

Ela combina o poder do processamento paralelo com o valor de dados em tempo real. Por exemplo, processar clickstreams em tempo real e, em seguida, analisar o envolvimento da usabilidade do site usando várias aplicações diferentes do Kinesis Data Streams executadas em paralelo.

Complexo processamento de stream

Você pode criar gráficos acíclicos direcionados (DAGs) de aplicativos e fluxos de dados do Kinesis Data Streams. Normalmente, isso envolve colocar dados de várias aplicações do Kinesis Data Streams em outro fluxo para processamento downstream por outro aplicação desse serviço.

Benefícios do uso do Kinesis Data Streams

Embora você possa usar o Kinesis Data Streams para resolver vários problemas de dados em streaming, um uso comum é agregar dados em tempo real e carregar os dados agregados em um data warehouse ou cluster de redução de mapa.

Os dados são colocados em fluxos de dados do Kinesis, o que garante durabilidade e elasticidade. O atraso entre o momento em que um registro é colocado no stream e o tempo em que ele pode ser recuperado (put-to-get atraso) geralmente é inferior a 1 segundo. Em outras palavras, uma aplicação do Kinesis Data Streams pode começar a consumir os dados do fluxo quase que imediatamente após sua adição. Como é um serviço gerenciado, o Kinesis Data Streams libera você da carga operacional de criar e executar um pipeline de entrada de dados. Você pode criar aplicações de redução de mapa de streaming. A elasticidade do Kinesis Data Streams permite escalar o streaming. Assim, você nunca perde registros de dados porque eles expiraram.

Como várias aplicações do Kinesis Data Streams podem consumir dados de um fluxo, múltiplas ações como arquivamento e processamento podem ocorrer de maneira simultânea e independente.

Por exemplo, dois aplicativos podem ler dados do mesmo stream. A primeira aplicação calcula agregados em execução e atualiza uma tabela do Amazon DynamoDB, e a segunda compacta e arquiva dados em um datastore, como o Amazon Simple Storage Service (Amazon S3). A tabela do DynamoDB com agregados em execução é então lida por um painel para gerar relatórios. up-to-the-minute

A Kinesis Client Library permite o consumo de dados tolerante a falhas de fluxos e fornece suporte à escalabilidade para aplicações do Kinesis Data Streams.

Serviços relacionados

Para obter informações sobre o uso de EMR clusters da Amazon para ler e processar streams de dados do Kinesis diretamente, consulte Kinesis Connector.

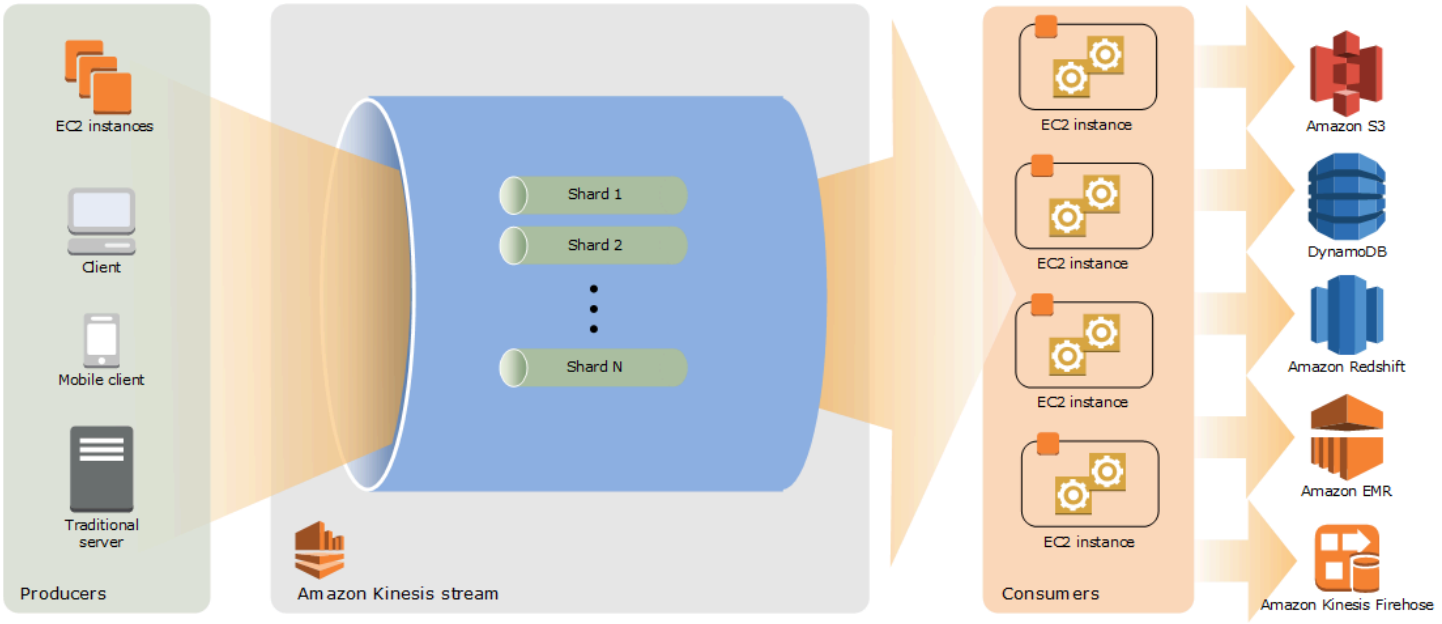
Terminologia e conceitos do Amazon Kinesis Data Streams

Antes de começar a usar o Amazon Kinesis Data Streams, conheça sua arquitetura e terminologia.

Tópicos

Analise a arquitetura de alto nível do Kinesis Data Streams

O diagrama a seguir ilustra a arquitetura de alto nível do Kinesis Data Streams. Os produtores enviam dados por push continuamente ao Kinesis Data Streams, que os consumidores processam em tempo real. Os consumidores (como um aplicativo personalizado executado na Amazon EC2 ou um stream de entrega do Amazon Data Firehose) podem armazenar seus resultados usando um AWS serviço como Amazon DynamoDB, Amazon Redshift ou Amazon S3.



Familiarize-se com a terminologia do Kinesis Data Streams

Fluxo de dados do Kinesis

Um fluxo de dados do Kinesis é um conjunto de fragmentos. Cada fragmento tem uma sequência de registros de dados. Cada registro de dados tem um número de sequência atribuído pelo Kinesis Data Streams.

Registro de dados

Um registro de dados é a unidade de dados armazenada em um fluxo de dados do Kinesis. Os registros de dados são compostos de um número de sequência, uma chave de partição e um blob de dados, que é uma sequência de bytes imutável. O Kinesis Data Streams não inspeciona, interpreta nem altera dados no blob. Um blob de dados pode ter até 1 MB.

Modo de capacidade

O modo de capacidade de um fluxo de dados determina como a capacidade é gerenciada e como são geradas cobranças pelo seu uso. Atualmente, no Kinesis Data Streams, você pode escolher entre um modo sob demanda e um modo provisionado para seus streams de dados. Para obter mais informações, consulte Escolha o modo de capacidade do fluxo de dados.

No modo sob demanda, o Kinesis Data Streams gerencia automaticamente os fragmentos para fornecer a throughput necessária. Você só paga pela throughput real que usa, e o Kinesis Data Streams acomoda automaticamente as necessidades de throughput das cargas de trabalho à medida que elas aumentam ou diminuem. Para obter mais informações, consulte Recursos e casos de uso do modo sob demanda.

No modo provisionado, você precisa especificar o número de fragmentos para o fluxo de dados. A capacidade total de um fluxo de dados é a soma das capacidades de seus fragmentos. Você pode aumentar ou diminuir o número de fragmentos em um fluxo de dados conforme necessário e recebe uma cobrança pelo número de fragmentos a uma taxa horária. Para obter mais informações, consulte Recursos e casos de uso do modo provisionado.

Período de retenção

O período de retenção é o tempo em que os registros de dados permanecem acessíveis depois de serem adicionados ao streaming. O período de retenção de um stream é definido para um padrão de

24 horas após a criação. Você pode aumentar o período de retenção em até 8760 horas (365 dias) usando a IncreaseStreamRetentionPeriodoperação e diminuir o período de retenção para um mínimo de 24 horas usando a DecreaseStreamRetentionPeriodoperação. Encargos adicionais incidem sobre streams com período de retenção definido acima de 24 horas. Para obter mais informações, consulte Definição de preço do Amazon Kinesis Data Streams.

Produtor

Os produtores colocam registros no Amazon Kinesis Data Streams. Por exemplo, um servidor web que envia dados de log para um stream é um produtor.

Consumidor

Os consumidores obtêm registros do Amazon Kinesis Data Streams e os processam. Esses consumidores são conhecidos como Aplicativo do Amazon Kinesis Data Streams.

Aplicativo do Amazon Kinesis Data Streams

Um aplicativo Amazon Kinesis Data Streams é consumidor de um stream que normalmente é executado em uma EC2 frota de instâncias.

Há dois tipos de consumidores que você pode desenvolver: consumidores avançados compartilhados e consumidores avançados aprimorados. Para saber mais sobre as diferenças entre eles, e para ver como você pode criar cada tipo de consumidor, consulte Leia dados do Amazon Kinesis Data Streams.

A saída de uma aplicação do Kinesis Data Streams pode ser a entrada de outro fluxo, permitindo a criação de topologias complexas que processam dados em tempo real. Um aplicativo também pode enviar dados para uma variedade de outros AWS serviços. Pode haver vários aplicativos para um stream, e cada aplicativo pode consumir dados do stream de forma independente e simultaneamente.

Fragmento

Um estilhaço é uma sequência de registros de dados identificada de forma exclusiva em um streaming. Um stream é composto de um ou mais estilhaços, sendo que cada um deles fornece uma unidade fixa de capacidade. Cada fragmento é compatível com até 5 transações por segundo para leituras, até a taxa máxima total de leitura de dados de 2 MB por segundo, e até 1.000 registros por segundo para gravações, até a taxa máxima total de gravação de dados de 1 MB por segundo

(incluindo chaves de partição). A capacidade de dados do seu stream é uma função do número de estilhaços que você especifica para o stream. A capacidade total do stream é a soma das capacidades de seus estilhaços.

Se a taxa de dados aumenta, você pode aumentar ou diminuir o número de estilhaços alocados para seu stream. Para obter mais informações, consulte Refragmentar um stream.

Chave de partição

A chave de partição é usada para agrupar os dados por fragmento dentro de um fluxo. O Kinesis Data Streams segrega os registros de dados pertencentes a um fluxo em vários fragmentos. Ele usa a chave de partição associada a cada registro de dados para determinar a qual estilhaço um determinado registro de dados pertence. As chaves de partição são strings Unicode, com um limite de tamanho máximo de 256 caracteres para cada chave. Uma função MD5 hash é usada para mapear chaves de partição para valores inteiros de 128 bits e para mapear registros de dados associados a fragmentos usando os intervalos de chaves de hash dos fragmentos. Quando um aplicativo insere dados em um stream, ele deve especificar uma chave de partição.

Número de sequência

Cada registro de dados tem um número de sequência exclusivo por chave de partição dentro do fragmento. O Kinesis Data Streams atribuirá o número de sequência depois que você gravar um registro no fluxo com `client.putRecords` ou `client.putRecord`. Geralmente, os números de sequência da mesma chave de partição aumentam ao longo do tempo. Quanto maior for o período entre as solicitações de gravação, maiores serão os números de sequência.

> **Note**
>
> Os números de sequência não podem ser usados como índices para conjuntos de dados dentro do mesmo stream. Para separar logicamente conjuntos de dados, use chaves de partição ou crie um stream separado para cada conjunto de dados.

Kinesis Client Library

A Kinesis Client Library é compilada em sua aplicação para permitir o consumo de dados tolerante a falhas do fluxo. A Kinesis Client Library garante que cada fragmento tenha um processador de registros em execução e processando o fragmento. A biblioteca também simplifica a leitura de dados

do stream. A Kinesis Client Library usa uma tabela do Amazon DynamoDB para armazenar dados de controle. Ela cria uma tabela para cada aplicativo que está processando dados.

Há duas versões principais da Kinesis Client Library. Aquela que você usa depende do tipo de consumidor que deseja criar. Para obter mais informações, consulte Leia dados do Amazon Kinesis Data Streams.

Nome da aplicação

O nome identifica uma aplicação do Amazon Kinesis Data Streams. Cada um de seus aplicativos deve ter um nome exclusivo que tenha como escopo a AWS conta e a região usadas pelo aplicativo. Esse nome é usado como um nome para a tabela de controle no Amazon DynamoDB e o namespace para as métricas da Amazon. CloudWatch

Criptografia do lado do servidor

O Amazon Kinesis Data Streams pode criptografar automaticamente dados confidenciais à medida que um produtor os insere em um fluxo. O Kinesis Data Streams usa chaves mestras do AWS KMS para criptografia. Para obter mais informações, consulte Proteção de dados no Amazon Kinesis Data Streams.

> **Note**
>
> Para ler ou gravar um em um stream criptografado, aplicativos produtores e consumidores devem ter permissão para acessar a chave mestra. Para obter informações sobre a concessão de permissões para aplicativos produtores e consumidores, consulte the section called “Permissões para usar chaves geradas pelo usuário KMS”.

> **Note**
>
> O uso da criptografia do lado do servidor incorre AWS Key Management Service em custos ().AWS KMS Para obter mais informações, consulte AWS Key Management Service Pricing.

Cotas e limites

A tabela a seguir descreve as cotas e os limites de streams e fragmentos para o Amazon Kinesis Data Streams.

Quota	Modo sob demanda	Modo provisionado
Número de fluxos de dados	Não há cota máxima no número de streams em sua AWS conta. Por padrão, você pode criar até 50 fluxos de dados usando o modo de capacidade sob demanda. Se precisar de uma cota maior, crie um tíquete de suporte.	Não há uma cota máxima para o número de fluxos em uma conta usando o modo provisionado.
Número de fragmentos	Não há limite máximo. O número de fragmentos depende da quantidade de dados ingeridos e do nível de throughput necessário. O Kinesis Data Streams escala automaticamente o número de fragmentos em resposta a mudanças no volume e no tráfego de dados.	Não há limite máximo. A cota de fragmentos padrão é de 500 fragmentos por AWS conta para as seguintes AWS regiões: Leste dos EUA (Norte da Virgínia), Oeste dos EUA (Oregon) e Europa (Irlanda). Em todas as outras regiões, a cota padrão é de 200 fragmentos por conta da AWS . Para solicitar um aumento de cota de shards-per-data streaming, consulte Solicitando um aumento de cota.
Throughput do fluxo de dados	Por padrão, novos fluxos de dados criados com o modo de capacidade sob demanda têm 4 MB/s de throughput de gravação e	Não há limite máximo. A throughput máxima depende do número de fragmentos provisionados para o fluxo. Cada fragmento pode ter até

Quota	Modo sob demanda	Modo provisionado
	8 MB/s de throughput de leitura. À medida que o tráfego aumenta, os fluxos de dados com o modo de capacidad e sob demanda aumentam a escala verticalmente até 200 MB/s de throughput de gravação e 400 MB/s de throughput de leitura. Se precisar aumentar essa capacidade para 2 GB/s de gravação e 4 GB/s de leitura, envie um tíquete de suporte	1 MB/s ou 1.000 registros/s de throughput de gravação ou até 2 MB/s ou 2.000 registros/s de throughput de leitura. Se precisar de mais capacidad e de ingestão, você pode facilmente aumentar o número de fragmentos no stream usando o AWS Managemen t Console ou o. UpdateSha rdCountAPI
Tamanho da carga útil de dados	O tamanho máximo da carga útil de dados de um registro antes da `base64-encoding` é de até 1 MB.	
Tamanho da transação `GetRecords`	GetRecordspode recuperar até 10 MB de dados por chamada de um único fragmento e até 10.000 registros por chamada. Cada chamada para `GetRecords` é contada como uma transação de leitura. Cada estilhaço pode oferecer suporte a até cinco transações de leitura por segundo. Cada transação de leitura pode fornecer até 10.000 registros com uma cota máxima de 10 MB por transação.	
Taxa de leitura de dados por fragmento	Cada fragmento pode suportar até uma taxa máxima total de leitura de dados de 2 MB por segundo via GetRecords. Se uma chamada para `GetRecords` retornar 10 MB, as chamadas subsequentes feitas nos próximos 5 segundos lançarão uma exceção.	
Número de consumidores registrados por fluxo de dados	Você pode criar até 20 consumidores registrados (limite de distribuição avançada) para cada fluxo de dados.	

Quota	Modo sob demanda	Modo provisionado
Alternar entre os modos provisionado e sob demanda	Para cada fluxo de dados em sua AWS conta, você pode alternar entre os modos de capacidade sob demanda e provisionada duas vezes em 24 horas.	

APILimites

Como a maioria AWS APIs, as operações do Kinesis API Data Streams têm taxas limitadas. Os limites a seguir se aplicam por AWS conta e região. Para obter mais informações sobre o Kinesis APIs Data Streams, consulte a Referência do Amazon API Kinesis.

KDSAPILimites do plano de controle

A seção a seguir descreve os limites do plano KDS de controleAPIs. KDSAPIso plano de controle permite que você crie e gerencie seus fluxos de dados. Esses limites se aplicam por conta da AWS por região.

APILimites do plano de controle

API	APIlimite de chamadas	Por conta/fluxo	Descrição
AddTagsToStream	5 transações por segundo (TPS)	Por conta	50 tags por fluxo de dados
CreateStream	5 TPS	Por conta	Não há um cota máxima para o número de streaming s que você pode ter em uma conta. Você recebe um `LimitExceededException` ao fazer uma solicitação `CreateStream` tentando executar

API	APIlimite de chamadas	Por conta/fluxo	Descrição
			um dos seguintes procedimentos: • Ter mais de cinco streams no estado `CREATING` em qualquer momento. • Criar mais fragmentos do que o autorizado para sua conta.
DecreaseStreamRetentionPeriod	5 TPS	Por fluxo	O valor mínimo do período de retenção de um stream de dados é de 24 horas.
DeleteResourcePolicy	5 TPS	Por conta	Se precisar de um limite maior, crie um Tíquete de suporte.
DeleteStream	5 TPS	Por conta	
DeregisterStreamConsumer	5 TPS	Por fluxo	
DescribeLimits	1 TPS	Por conta	
DescribeStream	10 TPS	Por conta	
DescribeStreamConsumer	20 TPS	Por fluxo	
DescribeStreamSummary	20 TPS	Por conta	

API	APIlimite de chamadas	Por conta/fluxo	Descrição
DisableEnhancedMonitoring	5 TPS	Por fluxo	
EnableEnhancedMonitoring	5 TPS	Por fluxo	
GetResourcePolicy	5 TPS	Por conta	Se precisar de um limite maior, crie um Tíquete de suporte.
IncreaseStreamRetentionPeriod	5 TPS	Por fluxo	O valor máximo do período de retenção de um fluxo é de 8.760 horas (365 dias).
ListShards	1000 TPS	Por fluxo	
ListStreamConsumers	5 TPS	Por fluxo	
ListStreams	5 TPS	Por conta	
ListTagsForStream	5 TPS	Por fluxo	
MergeShards	5 TPS	Por fluxo	Aplicável somente no modo provisionado.
PutResourcePolicy	5 TPS	Por conta	Se precisar de um limite maior, crie um Tíquete de suporte.

API	APIlimite de chamadas	Por conta/fluxo	Descrição
RegisterStreamConsumer	5 TPS	Por fluxo	É possível registrar até 20 consumidores por stream de dados. Um consumidor só pode ser registrado em um stream de dados de cada vez. Apenas cinco consumidores podem ser criados simultaneamente. Em outras palavras, não é possível ter mais de cinco consumidores com status `CREATING` ao mesmo tempo. Registrar o sexto consumidor quando há cinco com status `CREATING`
RemoveTagsFromStream	5 TPS	Por fluxo	
SplitShard	5 TPS	Por fluxo	Aplicável somente no modo provisionado

API	APIlimite de chamadas	Por conta/fluxo	Descrição
StartStreamEncryption		Por fluxo	Você pode aplicar com sucesso uma nova AWS KMS chave para criptografia do lado do servidor 25 vezes em um período contínuo de 24 horas.
StopStreamEncryption		Por fluxo	É possível desabilitar com êxito a criptografia no lado do servidor 25 vezes em um período contínuo de 24 horas.
UpdateShardCount		Por fluxo	Aplicável somente no modo provisionado. O limite padrão do número de fragmentos é 10.000. Há limites adicionais para issoAPI. Para obter mais informações, consulte UpdateShardCount.

API	APIlimite de chamadas	Por conta/fluxo	Descrição
UpdateStreamMode		Por fluxo	Para cada fluxo de dados em sua AWS conta, você pode alternar entre os modos de capacidade sob demanda e provisionada duas vezes em 24 horas.

KDSAPILimites do plano de dados

A seção a seguir descreve os limites do plano KDS de dadosAPIs. KDSAPIso plano de dados permite que você use seus fluxos de dados para coletar e processar registros de dados em tempo real. Esses limites se aplicam por fragmento dentro dos streams de dados.

APILimites do plano de dados

API	APIlimite de chamadas	Limite de carga útil	Outros detalhes
GetRecords	5 TPS	O número máximo de registros que podem ser retornados por chamada é 10.000. O tamanho máximo de dados que `GetRecords` pode retornar é 10 MB.	Se uma chamada retornar essa quantidade de dados, as chamadas subsequentes feitas nos próximos cinco segundos gerarão `ProvisionedThroughputExceededException`. Se a throughpu

API	APIlimite de chamadas	Limite de carga útil	Outros detalhes
			t provisionada no fluxo for insuficiente, as chamadas subsequentes feitas no próximo segundo gerarão `ProvisionedThroughputExceededException` .
GetShardIterator	5 TPS		Um iterador de fragmentos expira cinco minutos depois que é retornado ao solicitante. Se uma GetShardIterator solicitação for feita com muita frequência, você recebe uma ProvisionedThroughputExceededException.
PutRecord	1000 TPS	Cada fragmento pode oferecer suporte a gravações de até 1.000 registros por segundo, até um total máximo de gravação de dados de 1 MB por segundo.	

API	APIlimite de chamadas	Limite de carga útil	Outros detalhes
PutRecords		Cada PutRecords solicitação pode suportar até 500 registros. Cada registro na solicitação pode ter no máximo 1 MB, até um limite de 5 MB para toda a solicitação, incluindo chaves de partição. Cada fragmento pode oferecer suporte a gravações de até 1.000 registros por segundo, até um total máximo de gravação de dados de 1 MB por segundo.	
SubscribeToShard	Você pode fazer uma chamada SubscribeToShard por segundo por consumidor registrado por fragmento.		Se você ligar SubscribeToShard novamente com o mesmo consumidor ARN e ShardId dentro de 5 segundos de uma chamada bem-sucedida, você receberá umaResourceInUseException.

Aumento de cotas

É possível usar o Service Quotas para solicitar um aumento para uma cota, se a cota for ajustável. Algumas solicitações são resolvidas automaticamente, enquanto outras são enviadas para o AWS Support. Você pode acompanhar o status de uma solicitação de aumento de cota enviada ao AWS Support. As solicitações para aumentar as cotas de serviço não recebem suporte prioritário. Se você tiver uma solicitação urgente, entre em contato com o AWS Support. Para obter mais informações, consulte What is Service Quotas?

Para solicitar um aumento de cota de serviço, siga o procedimento descrito em Solicitar um aumento de cota.

Pré-requisitos completos para configurar o Amazon Kinesis Data Streams

Antes de usar o Amazon Kinesis Data Streams pela primeira vez, conclua as tarefas a seguir para configurar seu ambiente.

Tarefas

- Inscreva-se para AWS
- Fazer download de bibliotecas e ferramentas
- Configure seu ambiente de desenvolvimento

Inscreva-se para AWS

Quando você se inscreve no Amazon Web Services (AWS), sua AWS conta é automaticamente inscrita em todos os serviços AWS, incluindo o Kinesis Data Streams. Você será cobrado apenas pelos serviços que usar.

Se você já tiver uma AWS conta, vá para a próxima tarefa. Se você ainda não possuir uma conta da AWS , use o procedimento a seguir para criar uma.

Para se inscrever em uma AWS conta

1. Abra a https://portal.aws.amazon.com/billing/inscrição.
2. Siga as instruções online.

   Parte do procedimento de inscrição envolve receber uma chamada telefônica e inserir um código de verificação no teclado do telefone.

   Quando você se inscreve em um Conta da AWS, um Usuário raiz da conta da AWSé criado. O usuário raiz tem acesso a todos os Serviços da AWS e atributos na conta. Como prática recomendada de segurança, atribua o acesso administrativo a um usuário e use somente o usuário-raiz para executar tarefas que exigem acesso de usuário-raiz.

Fazer download de bibliotecas e ferramentas

Estas bibliotecas e ferramentas ajudarão você a trabalhar com o Kinesis Data Streams:

- O Amazon Kinesis API Reference é o conjunto básico de operações que o Kinesis Data Streams suporta. Para obter mais informações sobre a execução de operações básicas usando código Java, consulte o seguinte:
  - Desenvolva produtores usando o Amazon Kinesis API Data Streams com o AWS SDK for Java
  - Desenvolva consumidores personalizados com taxa de transferência compartilhada usando o AWS SDK for Java
  - Crie e gerencie streams de dados do Kinesis
- O AWS SDKs para Go, Java, JavaScript,. NET, Node.js PHP, Python e Ruby incluem suporte e amostras do Kinesis Data Streams. Se sua versão do AWS SDK for Java não incluir amostras do Kinesis Data Streams, você também poderá baixá-las em. GitHub
- A Kinesis Client Library (KCL) fornece um modelo de easy-to-use programação para processamento de dados. Isso KCL pode ajudar você a começar rapidamente com o Kinesis Data Streams em Java, Node.js,. NET, Python e Ruby. Para obter mais informações, consulte Ler dados de streamings.
- A AWS Command Line Interface é compatível com o Kinesis Data Streams. AWS CLI Isso permite que você controle vários AWS serviços a partir da linha de comando e os automatize por meio de scripts.

Configure seu ambiente de desenvolvimento

Para usar oKCL, certifique-se de que seu ambiente de desenvolvimento Java atenda aos seguintes requisitos:

- Java 1.7 (Java SE 7JDK) ou posterior. Você pode fazer download do software Java mais recente em Java SE Downloads no site da Oracle.
- Pacote Apache Commons (código, HTTP cliente e registro)
- JSONProcessador Jackson

Observe que o AWS SDK for Java inclui o Apache Commons e o Jackson na pasta de terceiros. No entanto, o SDK for Java funciona com o Java 1.6, enquanto a Kinesis Client Library requer o Java 1.7.

Use o AWS CLI para realizar operações do Amazon Kinesis Data Streams

Esta seção mostra como realizar operações básicas do Amazon Kinesis Data Streams usando o. AWS Command Line Interface Você conhecerá os princípios fundamentais do fluxo de dados do Kinesis Data Streams e as etapas necessárias para colocar e obter dados de um fluxo de dados do Kinesis.

Se você nunca usou o Kinesis Data Streams, comece familiarizando-se com os conceitos e a terminologia apresentados em Terminologia e conceitos do Amazon Kinesis Data Streams.

Tópicos

- Tutorial: Instalar e configurar o AWS CLI para Kinesis Data Streams
- Tutorial: Execute operações básicas do Kinesis Data Streams usando o AWS CLI

Para CLI acessar, você precisa de um ID de chave de acesso e uma chave de acesso secreta. Use credenciais temporárias em vez de chaves de acesso de longo prazo quando possível. As credenciais temporárias incluem um ID de acesso, uma chave de acesso secreta e um token de segurança que indica quando as credenciais expiram. Para obter mais informações, consulte Usando credenciais temporárias com AWS recursos no Guia do IAM usuário.

Você pode encontrar instruções detalhadas step-by-step IAM e de configuração da chave de segurança em Criar um IAM usuário.

Nesta seção, os comandos específicos discutidos são fornecidos textualmente, exceto onde valores específicos são necessariamente diferentes para cada execução. Além disso, os exemplos estão usando a região Oeste dos EUA (Oregon), mas as etapas desta seção funcionam em qualquer região onde o Kinesis Data Streams é compatível.

Tutorial: Instalar e configurar o AWS CLI para Kinesis Data Streams

Instale o AWS CLI

Para obter etapas detalhadas sobre como instalar o AWS CLI para Windows e para os sistemas operacionais Linux, OS X e Unix, consulte Instalando o. AWS CLI

Use o comando a seguir para listar as opções e os serviços disponíveis:

```
aws help
```

Você usará o serviço Kinesis Data Streams para poder AWS CLI revisar os subcomandos relacionados ao Kinesis Data Streams usando o seguinte comando:

```
aws kinesis help
```

Esse comando gera uma saída que inclui os comandos disponíveis do Kinesis Data Streams:

```
AVAILABLE COMMANDS

       o add-tags-to-stream

       o create-stream

       o delete-stream

       o describe-stream

       o get-records

       o get-shard-iterator

       o help

       o list-streams

       o list-tags-for-stream

       o merge-shards

       o put-record

       o put-records

       o remove-tags-from-stream

       o split-shard

       o wait
```

Essa lista de comandos corresponde aos Kinesis API Data Streams documentados no Amazon Kinesis Service Reference. API Por exemplo, o `create-stream` comando corresponde à `CreateStream` API ação.

O agora AWS CLI está instalado com sucesso, mas não está configurado. Isso é mostrado na próxima seção.

Configurar o AWS CLI

Para uso geral, o `aws configure` comando é a maneira mais rápida de configurar sua AWS CLI instalação. Para obter mais informações, consulte Configurar a AWS CLI.

Tutorial: Execute operações básicas do Kinesis Data Streams usando o AWS CLI

Esta seção descreve o uso básico de um fluxo de dados do Kinesis na linha de comando usando a AWS CLI. Certifique-se de estar familiarizado com os conceitos discutidos em Terminologia e conceitos do Amazon Kinesis Data Streams.

> **Note**
>
> Depois de criar um stream, sua conta incorre em cobranças nominais pelo uso do Kinesis Data Streams porque o Kinesis Data Streams não está qualificado para o nível gratuito. AWS Ao concluir este tutorial, exclua seus AWS recursos para parar de incorrer em cobranças. Para ter mais informações, consulte Etapa 4: limpar.

Tópicos

Etapa 1: criar um stream

Sua primeira etapa é criar um stream e verificar se ele foi criado com êxito. Use o comando a seguir para criar um stream denominado "Foo":

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

Em seguida, emita o comando a seguir para verificar o andamento da criação do stream:

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

Você deve obter uma saída semelhante ao exemplo a seguir:

```
{
    "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
    }
}
```

Neste exemplo, o stream tem um statusCREATING, o que significa que ele ainda não está pronto para uso. Verifique novamente em alguns instantes para ver uma saída semelhante ao exemplo a seguir:

```
{
    "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
    }
}
```

Há informações nessa saída que você não precisa para este tutorial. Por enquanto`"StreamStatus": "ACTIVE"`, a informação importante é que indica que o stream está pronto para ser usado e as informações sobre o único fragmento que você solicitou. Você também pode verificar a existência do novo stream usando o comando `list-streams`, como mostrado aqui:

```
aws kinesis list-streams
```

Saída:

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

Etapa 2: colocar um registro

Agora que tem um stream ativo, você está pronto para colocar alguns dados. Neste tutorial, você usará o comando mais simples possível, `put-record`, que coloca um único registro de dados contendo o texto "testdata" no stream:

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

Se bem-sucedido, esse comando gerará uma saída semelhante ao seguinte exemplo:

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

Parabéns, você adicionou dados a um stream! Em seguida, você verá como obter dados do stream.

Etapa 3: obter o registro

GetShardIterator

Antes de obter dados do stream, você deve obter o iterador de fragmento para o fragmento em que está interessado. Um iterador de estilhaços representa a posição do stream e do estilhaço da qual o consumidor (neste caso, o comando `get-record`) lerá. Você usará o `get-shard-iterator` comando da seguinte forma:

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

Lembre-se de que os `aws kinesis` comandos têm um Kinesis API Data Streams por trás deles; portanto, se você estiver curioso sobre algum dos parâmetros mostrados, poderá ler sobre eles `GetShardIterator`APIno tópico de referência. A execução bem-sucedida resultará em uma saída semelhante ao exemplo a seguir:

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

A string longa de caracteres aparentemente aleatórios é o iterador de estilhaços (a sua será diferente). Você deve copiar/colar o iterador de fragmento no comando get, mostrado a seguir. Os iteradores de estilhaços têm um ciclo de vida de 300 segundos, tempo que deve ser suficiente para você copiar/colar o iterador de estilhaços no próximo comando. Você deve remover todas as novas linhas do iterador de fragmentos antes de colar no próximo comando. Se você receber uma mensagem de erro informando que o iterador de fragmento não é mais válido, execute o `get-shard-iterator` comando novamente.

GetRecords

O `get-records` comando obtém dados do stream e é resolvido em uma chamada `GetRecords`no Kinesis Data Streams. API O iterador de estilhaços especifica a posição, no estilhaço, de onde você quer começar a ler os registros de dados em sequência. Se não houver registros disponíveis na parte do estilhaço para onde o iterador aponta, `GetRecords` retornará uma lista vazia. Pode ser necessário fazer várias chamadas para chegar a uma parte do fragmento que contém registros.

No exemplo a seguir do `get-records` comando:

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

Se você estiver executando este tutorial a partir de um processador de comandos do tipo Unix, como o bash, poderá automatizar a aquisição do iterador de fragmentos usando um comando aninhado, como este:

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

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

Se você estiver executando este tutorial em um sistema que oferece suporte PowerShell, você pode automatizar a aquisição do iterador de fragmentos usando um comando como este:

```
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])
```

O resultado bem-sucedido do `get-records` comando solicitará registros do seu stream para o fragmento que você especificou ao obter o iterador de fragmento, como no exemplo a seguir:

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

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

Observe que isso `get-records` está descrito acima como uma solicitação, o que significa que você pode receber zero ou mais registros, mesmo que haja registros em seu stream. Os registros retornados podem não representar todos os registros atualmente em seu stream. Isso é normal, e o código de produção pesquisará o stream em busca de registros em intervalos apropriados. Essa velocidade de pesquisa variará dependendo dos requisitos específicos de design do aplicativo.

Em seu registro nesta parte do tutorial, você notará que os dados parecem ser lixo — e não são o texto claro `testdata` que enviamos. Isso ocorre devido ao modo como `put-record` usa a codificação Base64 para permitir que você envie dados binários. No entanto, o suporte do Kinesis Data Streams no não fornece decodificação AWS CLI em Base64 porque a decodificação em Base64 em conteúdo binário bruto impresso em stdout pode causar comportamentos indesejados e possíveis problemas de segurança em determinadas plataformas e terminais. Se você usar um decodificador Base64 (por exemplo, https://www.base64decode.org/) para decodificar `dGVzdGRhdGE=` manualmente, verá que ele é, na verdade, `testdata`. Isso é suficiente para fins deste tutorial porque, na prática, raramente AWS CLI é usado para consumir dados. Mais frequentemente, ele é usado para monitorar o estado do fluxo e obter informações, conforme mostrado anteriormente (`describe-streamelist-streams`). Para obter mais informações sobre oKCL, consulte Desenvolvendo consumidores personalizados com taxa de transferência compartilhada usando KCL.

`get-records`nem sempre retorna todos os registros no stream/fragmento especificado. Quando isso acontecer, use o `NextShardIterator` a partir do último resultado para obter o próximo conjunto de registros. Se mais dados estivessem sendo colocados no stream, o que é a situação normal em aplicativos de produção, você poderia continuar pesquisando os dados usados a `get-records` cada vez. No entanto, se você não chamar `get-records` usando o próximo iterador de fragmento dentro da vida útil de 300 segundos do iterador de fragmento, receberá uma mensagem de erro e deverá usar o `get-shard-iterator` comando para obter um novo iterador de fragmento.

Também é fornecido nesta saída`MillisBehindLatest`, que é o número de milissegundos em que a resposta da GetRecordsoperação vem da ponta do fluxo, indicando o quão atrasado o consumidor está no tempo atual. Um valor zero indica que o processamento de registros foi alcançado e não há nenhum registro novo para processar no momento. No caso deste tutorial, se você está lendo tudo conforme avança, poderá ver um número muito grande. Por padrão, os registros de dados permanecem em um fluxo por 24 horas esperando que você os recupere. Esse período, chamado de período de retenção, pode ser configurado para até 365 dias.

Um `get-records` resultado bem-sucedido sempre terá um resultado, `NextShardIterator` mesmo que não haja mais registros atualmente no stream. Este é um modelo de sondagem que

assume que um produtor colocará mais registros no stream em um determinado momento. Embora você possa escrever suas próprias rotinas de pesquisa, se você usar as mencionadas anteriormente KCL para desenvolver aplicativos de consumo, essa pesquisa é feita para você.

Se você ligar `get-records` até que não haja mais registros no stream e no fragmento dos quais você está extraindo, você verá uma saída com registros vazios semelhante ao exemplo a seguir:

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

Etapa 4: limpar

Exclua sua transmissão para liberar recursos e evitar cobranças não intencionais em sua conta. Faça isso sempre que você tiver criado um stream e não quiser usá-lo, pois as cobranças são acumuladas por stream, independentemente de você estar inserindo e obtendo dados com ele ou não. O comando de limpeza é o seguinte:

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

O sucesso resulta em nenhuma saída. Use `describe-stream` para verificar o progresso da exclusão:

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

Se você executar esse comando imediatamente após o comando delete, verá uma saída semelhante ao exemplo a seguir:

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

Após a exclusão total do stream, `describe-stream` gerará um erro "não encontrado":

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

Tutoriais de introdução ao Amazon Kinesis Data Streams

O Amazon Kinesis Data Streams fornece várias soluções diferentes para ingerir e consumir dados dos streams de dados do Kinesis. Os tutoriais desta seção foram elaborados para ajudá-lo ainda mais a entender os conceitos e a funcionalidade do Amazon Kinesis Data Streams e a identificar a solução que atenda às suas necessidades.

Tópicos

Tutorial: Processe dados de estoque em tempo real usando KPL e KCL 2.x

O cenário deste tutorial envolve a ingestão de negociações de ações em um stream de dados e a criação de um aplicativo básico do Amazon Kinesis Data Streams que realiza cálculos no stream. Você aprenderá a enviar um stream de registros para o Kinesis Data Streams e a implementar um aplicativo que consome e processa os registros quase em tempo real.

> **Important**
>
> Depois de criar um stream, sua conta incorre em cobranças nominais pelo uso do Kinesis Data Streams porque o Kinesis Data Streams não está qualificado para o nível gratuito. AWS Depois de iniciada, a aplicação de consumo também incorre em cobranças nominais pelo uso do Amazon DynamoDB. A aplicação de consumo usa o DynamoDB para monitorar o estado do processamento. Ao terminar de usar esta aplicação, exclua seus recursos da AWS para parar de gerar cobranças. Para obter mais informações, consulte Limpar os recursos.

O código não acessa os dados reais da bolsa de valores, ele simula o stream de negociações de ações. Isso é feito com o uso de um gerador de negociações de ações aleatórias cujo ponto de

partida são dados do mercado real referente às 25 principais ações por capitalização de mercado em fevereiro de 2015. Se tiver acesso a um streaming de negociações de ações em tempo real, você poderá se interessar em derivar estatísticas úteis e em tempo hábil desse streaming. Por exemplo, talvez convenha executar uma análise de janela deslizante na qual você determina a ação mais popular que foi adquirida nos últimos 5 minutos. Ou talvez convenha uma notificação sempre que uma ordem de venda for muito grande (ou seja, tenha muitas quotas). Você pode estender o código nesta série para oferecer essa funcionalidade.

É possível executar as etapas deste tutorial no desktop ou laptop e executar o código de produtor e de consumidor na mesma máquina ou em qualquer plataforma que ofereça suporte aos requisitos definidos.

Os exemplos mostrados usam a região Oeste dos EUA (Oregon), mas funcionam em qualquer região da AWS que oferece suporte ao Kinesis Data Streams.

Tarefas

- Concluir os pré-requisitos
- Crie um fluxo de dados
- Crie uma IAM política e um usuário
- Baixe e crie o código
- Implemente o produtor
- Implemente o consumidor
- (Opcional) Estenda o consumidor
- Limpar os recursos

Concluir os pré-requisitos

Você deve atender aos seguintes requisitos para concluir este tutorial:

Crie e use uma conta da Amazon Web Services

Antes de começar, certifique-se de estar familiarizado com os conceitos discutidos emTerminologia e conceitos do Amazon Kinesis Data Streams, principalmente com fluxos, fragmentos, produtores e consumidores. Também é útil concluir as etapas no seguinte guia: Tutorial: Instalar e configurar o AWS CLI para Kinesis Data Streams.

Você deve ter uma AWS conta e um navegador da web para acessar AWS Management Console o.

Para acessar o console, use seu nome de IAM usuário e senha para entrar na página AWS Management Consolede IAM login. Para obter informações sobre credenciais AWS de segurança, incluindo acesso programático e alternativas às credenciais de longo prazo, consulte as credenciais de AWS segurança no Guia do usuário. IAM Para obter detalhes sobre como fazer login no seu Conta da AWS, consulte Como fazer login AWS no Guia do Início de Sessão da AWS usuário.

Para obter mais informações IAM e instruções de configuração da chave de segurança, consulte Criar um IAM usuário.

Atenda aos requisitos de software do sistema

O sistema usado para executar o aplicativo deve ter o Java 7 ou posterior instalado. Para baixar e instalar o Java Development Kit mais recente (JDK), acesse o site de instalação do Java SE da Oracle.

Você precisa da versão mais recente do AWS SDK for Java.

O aplicativo consumidor requer a Kinesis Client Library (KCL) versão 2.2.9 ou superior, que você pode obter em /tree/master. GitHub https://github.com/awslabs/ amazon-kinesis-client

Próximas etapas

Crie um fluxo de dados

Crie um fluxo de dados

Primeiro, é necessário criar o fluxo de dados que você usará nas etapas seguintes deste tutorial.

Para criar um fluxo

1. Faça login no AWS Management Console e abra o console do Kinesis em https://console.aws.amazon.com /kinesis.
2. Selecione Data Streams (Fluxos de dados) no painel de navegação.
3. Na barra de navegação, expanda o seletor de região e escolha uma região.
4. Escolha Create Kinesis stream (Criar streaming do Kinesis).
5. Insira um nome para seu fluxo de dados (por exemplo, **`StockTradeStream`**).

6. Insira **1** o número de fragmentos, mas mantenha Estimar o número de fragmentos que você precisará reduzir.
7. Escolha Create Kinesis stream (Criar streaming do Kinesis).

Na página de lista Fluxos do Kinesis, o status do fluxo aparece como `CREATING` enquanto ele está sendo criado. Quando o stream fica pronto para uso, o status é alterado para `ACTIVE`.

Se você escolher o nome do fluxo, na página exibida, a guia Details (Detalhes) exibirá um resumo da configuração do fluxo de dados. A seção Monitoring (Monitoramento) exibe informações de monitoramento do streaming.

Próximas etapas

Crie uma IAM política e um usuário

Crie uma IAM política e um usuário

Práticas recomendadas de segurança para AWS ditar o uso de permissões refinadas para controlar o acesso a diferentes recursos. AWS Identity and Access Management (IAM) permite gerenciar usuários e permissões de usuário no AWS. Uma IAMpolítica lista explicitamente as ações que são permitidas e os recursos aos quais as ações são aplicáveis.

Veja a seguir as permissões mínimas normalmente necessárias para produtores e consumidores do Kinesis Data Streams.

Produtor

Ações	Recurso	Finalidade
`DescribeStream` , `DescribeStreamSummary` , `DescribeStreamConsumer`	Fluxo de dados do Kinesis	Antes de tentar ler registros, o consumidor verifica se o flux está ativo e se os estilhaços estão contidos no fluxo de dad
`SubscribeToShard` , `RegisterStreamConsumer`	Fluxo de dados do Kinesis	Assina e registra os consumidores em um estilhaço.
`PutRecord` , `PutRecords`	Fluxo de dados do Kinesis	Grava registros no Kinesis Data Streams.

Consumidor

Ações	Recurso	Finalidade
`DescribeStream`	Fluxo de dados do Kinesis	Antes de tentar ler registros, o consumidor verifica se o flux está ativo e se os estilhaços estão contidos no fluxo de dad
`GetRecords` , `GetShardIterator`	Fluxo de dados do Kinesis	Lê registros de um estilhaço.
`CreateTable` , `DescribeTable` , `GetItem`, `PutItem`, `Scan`, `UpdateItem`	Tabela do Amazon DynamoDB	Se o consumidor for desenvolvido usando a Kinesis Client 1.x ou 2.x), ele precisará de permissões para uma tabela d rastrear o estado de processamento do aplicativo.
`DeleteItem`	Tabela do Amazon DynamoDB	Para operações de divisão/mesclagem que o consumidor e do Kinesis Data Streams.
`PutMetricData`	CloudWatch Registro da Amazon	Ele KCL também carrega métricas para CloudWatch, que s o aplicativo.

Para este tutorial, você criará uma única IAM política que concede todas as permissões anteriores. Na produção, talvez você queira criar duas políticas, uma para produtores e outra para consumidores.

Para criar uma política do IAM

1. Localize o Amazon Resource Name (ARN) para o novo fluxo de dados que você criou na etapa anterior. Você pode encontrar isso ARN listado como Stream ARN na parte superior da guia Detalhes. O ARN formato é o seguinte:

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

   região

   O código AWS da região; por exemplo,`us-west-2`. Para obter mais informações, consulte Conceitos de região e zona de disponibilidade.

conta

O ID da AWS conta, conforme mostrado nas Configurações da conta.

name

O nome do fluxo de dados que você criou na etapa anterior, que é`StockTradeStream`.

2. Determine ARN a tabela do DynamoDB a ser usada pelo consumidor (e criada pela primeira instância do consumidor). Ele deve estar no seguinte formato:

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

A região e o ID da conta são idênticos aos valores ARN do stream de dados que você está usando neste tutorial, mas o nome é o nome da tabela do DynamoDB criada e usada pelo aplicativo consumidor. KCLusa o nome do aplicativo como nome da tabela. Nesta etapa, use `StockTradesProcessor` como o nome da tabela do DynamoDB, pois esse é o nome da aplicação usada nas etapas posteriores do tutorial.

3. No IAM console, em Políticas (https://console.aws.amazon.com/iam/home #policies), escolha Criar política. Se for a primeira vez que você trabalha com IAM políticas, escolha Começar, Criar política.
4. Escolha Select (Selecionar) ao lado de Policy Generator (Gerador de políticas).
5. Escolha o Amazon Kinesis como serviço. AWS
6. Selecione `DescribeStream`, `GetShardIterator`, `GetRecords`, `PutRecord`e `PutRecords` como ações permitidas.
7. Insira o ARN do fluxo de dados que você está usando neste tutorial.
8. Use Add Statement (Adicionar instrução) para cada um dos seguintes:

AWS Serviço	Ações	ARN
Amazon DynamoDB	`CreateTable` , `DeleteItem` , `DescribeTable` , `GetItem`, `PutItem`, `Scan`, `UpdateItem`	A ARN da tabela do DynamoDB que você criou na Etapa 2 deste procedimento.
Amazon CloudWatch	`PutMetricData`	*

O asterisco (*) usado ao especificar um não ARN é obrigatório. Nesse caso, é porque não há nenhum recurso específico CloudWatch no qual a `PutMetricData` ação seja invocada.

9. Escolha Next Step.
10. Altere Policy Name (Nome da política) para `StockTradeStreamPolicy`, analise o código e escolha Create Policy (Criar política).

O documento de política resultante deve ser semelhante a:

```
{
  "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 criar um usuário do IAM

1. Abra o IAM console em https://console.aws.amazon.com/iam/.
2. Na página Usuários, selecione Adicionar usuário.
3. Para User name, digite `StockTradeStreamUser`.
4. Em Access type (Tipo de acesso), escolha Programmatic access (Aceso programático) e, em seguida, escolha Next: Permissions (Próximo: permissões).
5. Escolha Anexar políticas existentes diretamente.
6. Pesquise por nome a política que você criou no procedimento anterior (`StockTradeStreamPolicy`. Selecione a caixa à esquerda do nome da política e escolha Next: Review (Próximo: revisão).
7. Revise os detalhes e o resumo e, em seguida, escolha Create user (Criar usuário).
8. Copie o Access key ID (ID da chave de acesso) e salve-o de forma privada. Em Secret access key (Chave de acesso secreta), escolha Show (Mostrar) e salve a chave de forma privada também.

9. Cole as chaves de acesso e chaves secretas em um arquivo local em lugar seguro que somente você possa acessar. Para esse aplicativo, crie um arquivo denominado `~/.aws/credentials` (com permissões restritas). O arquivo deverá estar no seguinte formato:

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

Para anexar uma IAM política a um usuário

1. No IAM console, abra Políticas e escolha Ações de política.
2. Escolha `StockTradeStreamPolicy` e Attach (Anexar).
3. Escolha `StockTradeStreamUser` e Attach Policy (Anexar política).

Próximas etapas

Baixe e crie o código

Baixe e crie o código

Este tópico fornece um exemplo de código de implementação para o exemplo de ingestão de transações de ações no fluxo de dados (produtor) e o processamento desses dados (consumidor).

Como fazer download e criar o código

1. Baixe o código-fonte do https://github.com/aws-samples/amazon-kinesis-learning GitHub repositório para o seu computador.
2. Crie um projeto em seu IDE com o código-fonte, seguindo a estrutura de diretórios fornecida.
3. Adicione as seguintes bibliotecas ao projeto:

   - Biblioteca de cliente do Amazon Kinesis () KCL
   - AWS SDK
   - Apache HttpCore
   - Apache HttpClient
   - Apache Commons Lang
   - Apache Commons Logging

- Guava (Google Core Libraries For Java)
- Jackson Annotations
- Jackson Core
- Jackson Databind
- Formato de dados Jackson: CBOR
- Joda Time

4. Dependendo do seuIDE, o projeto pode ser construído automaticamente. Caso contrário, crie o projeto usando as etapas apropriadas para o seuIDE.

Se concluiu essas etapas com êxito, você agora está pronto para ir para a próxima seção, the section called “Implemente o produtor”.

Próximas etapas

Implemente o produtor

Este tutorial usa o cenário do mundo real de monitoramento de transações da bolsa de valores. Os princípios a seguir explicam brevemente como este cenário é mapeado para o produtor e a estrutura de código de suporte.

Consulte o código-fonte e analise as informações a seguir.

StockTrade classe

Uma negociação de ações individual é representada por uma instância da StockTrade classe. Essa instância contém atributos como o símbolo ticker, o preço, o número de ações, o tipo da negociação (compra ou venda) e um ID que identifica a negociação com exclusividade. Essa classe é implementada para você.

Registro de stream

Um stream é uma sequência de registros. Um registro é uma serialização de uma `StockTrade` instância em JSON formato. Por exemplo:

```
{
  "tickerSymbol": "AMZN",
  "tradeType": "BUY",
  "price": 395.87,
```

```
  "quantity": 16,
  "id": 3567129045
}
```

StockTradeGenerator classe

StockTradeGenerator tem um método chamado `getRandomTrade()` que retorna uma nova negociação de ações gerada aleatoriamente toda vez que é invocada. Essa classe é implementada para você.

StockTradesWriter classe

O `main` método do produtor recupera StockTradesWriter continuamente uma negociação aleatória e a envia para o Kinesis Data Streams executando as seguintes tarefas:

1. Lê o nome do fluxo de dados e o nome da região como entrada.
2. Usa o `KinesisAsyncClientBuilder` para definir a região, as credenciais e a configuração do cliente.
3. Verifica se o stream existe e está ativo (se não, ele será encerrado com um erro).
4. Em um loop contínuo, chama o método `StockTradeGenerator.getRandomTrade()` e o método `sendStockTrade` para enviar a negociação ao stream a cada 100 milissegundos.

O método `sendStockTrade` da classe `StockTradesWriter` tem o seguinte 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 o desmembramento do código a seguir:

- O `PutRecord` API espera uma matriz de bytes e você deve converter a negociação em JSON formato. Essa única linha de código executa a seguinte operação:

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

- Antes de enviar a transação, crie uma nova instância de `PutRecordRequest` (chamada solicitação neste caso). Cada `request` exige o nome do fluxo, uma chave de partição e um blob de dados.

  ```
  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();
  ```

  O exemplo usa um ticker de ações como chave de partição, que mapeia o registro para um fragmento específico. Na prática, você deve ter centenas ou milhares de chaves de partição por estilhaço, de forma que os registros sejam uniformemente disseminados no seu stream. Para obter mais informações sobre como adicionar dados a um stream, consulte Grave dados no Amazon Kinesis Data Streams.

  Agora, `request` está pronto para enviar para o cliente (operação put):

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

- A verificação e o registro de erros são sempre inclusões úteis. Este código registra condições de erro:

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

Adicione o bloco try/catch ao redor da operação `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);
}
```

Isso ocorre porque uma operação put do Kinesis Data Streams pode falhar devido a erro de rede ou porque o fluxo de dados pode atingir o limite de throughput e ficar limitado. É recomendável que você considere cuidadosamente sua política de repetição de `put` operações para evitar perda de dados, como usar uma nova tentativa.

- O registro de status é útil mas opcional:

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

O produtor mostrado aqui usa a funcionalidade de registro único do Kinesis API Data Streams,. `PutRecord` Na prática, se um produtor individual gerar muitos registros, costuma ser mais

eficiente usar a funcionalidade de vários registros de `PutRecords` e enviar lotes de registros por vez. Para obter mais informações, consulte Grave dados no Amazon Kinesis Data Streams.

Para executar o produtor

1. Verifique se a chave de acesso e o par de chaves secretas recuperados em Crie uma IAM política e um usuário estão salvos no arquivo `~/.aws/credentials`.
2. Execute a classe `StockTradeWriter` com os seguintes argumentos:

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

   Se você criou o fluxo em uma região diferente de `us-west-2`, será necessário especificar essa região aqui.

Você deve ver saída semelhante a:

```
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
```

Suas negociações de ações agora estão sendo ingeridas pelo Kinesis Data Streams.

Próximas etapas

Implemente o consumidor

Implemente o consumidor

O aplicativo consumidor neste tutorial processa continuamente as transações de ações em seu fluxo de dados. Em seguida, ele produz as ações mais populares compradas e vendidas a cada minuto. O aplicativo é construído com base na Kinesis Client Library (KCL), que faz grande parte do trabalho pesado comum aos aplicativos de consumo. Para obter mais informações, consulte Use a biblioteca de cliente Kinesis.

Consulte o código-fonte e analise as informações a seguir.

StockTradesProcessor classe

A classe principal do consumidor, fornecida para você, que executa as seguintes tarefas:

- Lê os nomes do aplicativo, do fluxo de dados e da região, transmitidos como argumentos.
- Cria uma `KinesisAsyncClient` instância com o nome da região.
- Cria uma instância de `StockTradeRecordProcessorFactory` que veicula instâncias de `ShardRecordProcessor`, implementadas por uma instância de `StockTradeRecordProcessor`.
- Cria uma `ConfigsBuilder` instância com a `StockTradeRecordProcessorFactory` instância `KinesisAsyncClient` `StreamNameApplicationName`,, e. Isso é útil para criar todas as configurações com valores padrão.
- Cria um KCL agendador (anteriormente, nas KCL versões 1.x, era conhecido como KCL trabalhador) com a `ConfigsBuilder` instância.
- O programador cria um novo thread para cada estilhaço (atribuído a essa instância de consumidor), que faz loop continuamente para ler registros do fluxo de dados. Em seguida, ele invoca a instância de `StockTradeRecordProcessor` para processar cada lote de registros recebidos.

StockTradeRecordProcessor classe

Implementação da instância de `StockTradeRecordProcessor`, que, por sua vez, implementa cinco métodos necessários: `initialize`, `processRecords`, `leaseLost`, `shardEnded` e `shutdownRequested`.

Os `shutdownRequested` métodos `initialize` e são usados pelo KCL para informar ao processador de registros quando ele deve estar pronto para começar a receber registros e quando deve esperar parar de receber registros, respectivamente, para que ele possa

realizar qualquer tarefa de configuração e encerramento específica do aplicativo. `leaseLoste` `shardEnded` são usados para implementar qualquer lógica sobre o que fazer quando uma concessão é perdida ou um processamento chega ao fim de um fragmento. Neste exemplo, simplesmente registramos em log mensagens indicando esses eventos.

O código para esses métodos é fornecido para você. O processamento principal ocorre no método `processRecords`, que, por sua vez, usa `processRecord` para cada registro. Esse último método é fornecido como o código esqueleto quase todo vazio, para você implementar na próxima etapa, onde é explicado em mais detalhes.

Observe também a implementação dos métodos de suporte de `processRecord`: `reportStats` e `resetStats`, que estão vazios no código-fonte original.

O método `processRecords`, implementado para você, executa as seguintes etapas:

- Para cada registro passado, ele chama `processRecord`.
- Se tiver decorrido pelo menos 1 minuto após o último relatório, chamará `reportStats()`, que imprime as estatísticas mais recentes e, em seguida, `resetStats()`, que limpa as estatísticas para que o próximo intervalo inclua apenas registros novos.
- Define o próximo horário para geração de relatórios.
- Se tiver decorrido pelo menos 1 minuto após o último ponto de verificação, chamará `checkpoint()`.
- Define o próximo horário do ponto de verificação.

Este método usa intervalos de 60 segundos como taxa de geração de relatórios e definição de pontos de verificação. Para obter mais informações sobre pontos de verificação, consulte Using the Kinesis Client Library.

StockStats classe

Essa classe fornece retenção de dados e rastreamento de estatísticas em relação às ações mais populares ao longo do tempo. Esse código, fornecido para você, contém os seguintes métodos:

- `addStockTrade(StockTrade)`: injeta o `StockTrade` conhecido nas estatísticas correntes.
- `toString()`: retorna as estatísticas em uma string formatada.

Essa classe rastreia as ações mais populares mantendo uma contagem contínua do número total de negociações de cada ação e a contagem máxima. Ela atualiza essas contagens sempre que chega uma negociação de ação.

Adicione código aos métodos da classe `StockTradeRecordProcessor`, como mostrado nas etapas a seguir.

Para implementar o consumidor

1. Implemente o método `processRecord` instanciando um objeto `StockTrade` de tamanho correto e adicionando a ele os dados do registro, registrando um aviso caso ocorra 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 um `reportStats` método. Modifique o formato de saída de acordo com suas preferências.

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

3. Implemente o método `resetStats`, que cria uma nova instância de `stockStats`.

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

4. Implemente os seguintes métodos exigidos pela `ShardRecordProcessor` interface:

```
@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 executar o consumidor

1. Execute o produtor escrito em para injetar registros de negociações de ações no streaming.

2. Verifique se a chave de acesso e o par de chaves secretas recuperados anteriormente (ao criar o IAM usuário) estão salvos no arquivo`~/.aws/credentials`.
3. Execute a classe `StockTradesProcessor` com os seguintes argumentos:

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

   Observe que, se você criou o stream em uma região diferente de `us-west-2`, precisará especificar essa região aqui.

Depois de um minuto, deverá aparecer uma saída como a seguir, atualizada a cada minuto a partir de então:

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

Próximas etapas

[(Opcional) Estenda o consumidor]

(Opcional) Estenda o consumidor

Esta seção opcional mostra como estender o código de consumidor para um cenário um pouco mais elaborado.

Se quiser saber mais sobre os maiores pedidos de venda a cada minuto, você poderá modificar a classe `StockStats` em três locais para acomodar essa nova prioridade.

Para estender o consumidor

1. Adicione novas variáveis de instância:

   ```
   // 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. Adicione o seguinte código a `addStockTrade`:

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

3. Modifique o método `toString` para imprimir as informações adicionais:

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

Se você executar o consumidor agora (lembre-se de executar o produtor também), deverá aparecer uma saída semelhante a esta:

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

Próximas etapas

Limpar os recursos

Limpar os recursos

Como você está pagando para usar o fluxo de dados do Kinesis, certifique-se de excluí-lo e de excluir a tabela do Amazon DynamoDB correspondente ao concluir. As cobranças nominais ocorrerão em um stream ativo mesmo quando você não estiver enviando e recebendo registros. Isso ocorre porque um stream ativo usa recursos por meio da "escuta" contínua de registros recebidos e solicitações para obter registros.

Para excluir o stream e tabela

1. Feche todos os produtores e consumidores que você ainda possa ter administrando.
2. Abra o console do Kinesis em https://console.aws.amazon.com /kinesis.
3. Escolha o stream que você criou para este aplicativo (`StockTradeStream`).
4. Escolha Delete Stream (Excluir streaming).
5. Abra o console do DynamoDB em. https://console.aws.amazon.com/dynamodb/
6. Exclua a tabela `StockTradesProcessor`.

Resumo

O processamento de uma grande quantidade de dados quase em tempo real não exige escrever códigos complicados ou desenvolver uma grande infraestrutura. É tão básico quanto escrever uma lógica para processar uma pequena quantidade de dados (como escrever`processRecord(Record)`), mas usar o Kinesis Data Streams para escalar de forma que funcione para uma grande quantidade de dados de streaming. Você não precisa se preocupar com a escalabilidade do processamento, porque o Kinesis Data Streams cuida de tudo. Você só precisa enviar seus registros de streaming ao Kinesis Data Streams e escrever a lógica para processar cada novo registro recebido.

Veja aqui alguns aprimoramentos potenciais para este aplicativo.

Agregar em todos os estilhaços

Atualmente, você obtém estatísticas resultantes da agregação de registros de dados recebidos por um único operador proveniente de um único estilhaço. (Um estilhaço não pode ser

processado por mais de um operador em um aplicativo ao mesmo tempo). Naturalmente, quando escala e tem mais de um estilhaço, você pode agregar em todos os estilhaços. É possível fazer isso tendo uma arquitetura de pipeline em que a saída de cada operador é alimentada em outro fluxo com um único estilhaço, o qual é processado por um operador que agrega as saídas do primeiro estágio. Como os dados do primeiro estágio são limitados (um exemplo por minuto por estilhaço), eles podem ser facilmente tratados por um estilhaço.

Escalar o processamento

Quando o stream é expandido para ter muitos estilhaços (porque muitos produtores estão enviando dados), a maneira de escalar o processamento é adicionando mais operadores. Você pode executar os trabalhadores em EC2 instâncias da Amazon e usar grupos de Auto Scaling.

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

Como um fluxo é processado continuamente, sua saída pode ser enviada para outros destinos. AWS fornece conectores para integrar o Kinesis Data Streams com AWS outros serviços e ferramentas de terceiros.

Tutorial: Processe dados de estoque em tempo real usando KPL e KCL 1.x

O cenário deste tutorial envolve consumir negociações do mercado de ações em um fluxo de dados e criar uma aplicação simples do Amazon Kinesis Data Streams para realizar cálculos no fluxo. Você aprenderá a enviar um stream de registros para o Kinesis Data Streams e a implementar um aplicativo que consome e processa os registros quase em tempo real.

> **Important**
>
> Depois de criar um stream, sua conta incorre em cobranças nominais pelo uso do Kinesis Data Streams porque o Kinesis Data Streams não está qualificado para o nível gratuito. AWS Depois de iniciada, a aplicação de consumo também incorre em cobranças nominais pelo uso do Amazon DynamoDB. A aplicação de consumo usa o DynamoDB para monitorar o estado do processamento. Ao terminar de usar esta aplicação, exclua seus recursos da AWS para parar de gerar cobranças. Para obter mais informações, consulte Limpar os recursos.

O código não acessa os dados reais da bolsa de valores, ele simula o stream de negociações de ações. Isso é feito com o uso de um gerador de negociações de ações aleatórias cujo ponto de

partida são dados do mercado real referente às 25 principais ações por capitalização de mercado em fevereiro de 2015. Se tiver acesso a um streaming de negociações de ações em tempo real, você poderá se interessar em derivar estatísticas úteis e em tempo hábil desse streaming. Por exemplo, talvez convenha executar uma análise de janela deslizante na qual você determina a ação mais popular que foi adquirida nos últimos 5 minutos. Ou talvez convenha uma notificação sempre que uma ordem de venda for muito grande (ou seja, tenha muitas quotas). Você pode estender o código nesta série para oferecer essa funcionalidade.

Você pode seguir as etapas deste tutorial em seu computador desktop ou laptop e executar o código do produtor e do consumidor na mesma máquina ou em qualquer plataforma que suporte os requisitos definidos, como o Amazon Elastic Compute Cloud (AmazonEC2).

Os exemplos mostrados usam a região Oeste dos EUA (Oregon), mas funcionam em qualquer região da AWS que oferece suporte ao Kinesis Data Streams.

Tarefas

- Concluir os pré-requisitos
- Crie um fluxo de dados
- Crie uma IAM política e um usuário
- Baixe e crie o código de implementação
- Implemente o produtor
- Implemente o consumidor
- (Opcional) Estenda o consumidor
- Limpar os recursos

Concluir os pré-requisitos

Estes são os requisitos para concluir o Tutorial: Processe dados de estoque em tempo real usando KPL e KCL 1.x.

Crie e use uma conta da Amazon Web Services

Antes de começar, familiarize-se com os conceitos abordados em Terminologia e conceitos do Amazon Kinesis Data Streams, especialmente fluxos, estilhaços, produtores e consumidores. Também é útil ter concluído Tutorial: Instalar e configurar o AWS CLI para Kinesis Data Streams.

Você precisa de uma AWS conta e de um navegador da web para acessar AWS Management Console o.

Para acessar o console, use seu nome de IAM usuário e senha para entrar no a AWS Management Consolepartir da página de IAM login. Para obter informações sobre credenciais AWS de segurança, incluindo acesso programático e alternativas às credenciais de longo prazo, consulte as credenciais de AWS segurança no Guia do usuário. IAM Para obter detalhes sobre como fazer login no seu Conta da AWS, consulte Como fazer login AWS no Guia do Início de Sessão da AWS usuário.

Para obter mais informações IAM e instruções de configuração da chave de segurança, consulte Criar um IAM usuário.

Atenda aos requisitos de software do sistema

O sistema usado para executar o aplicativo precisa ter o Java 7 ou superior instalado. Para baixar e instalar o Java Development Kit mais recente (JDK), acesse o site de instalação do Java SE da Oracle.

Se você tiver um JavaIDE, como o Eclipse, poderá abrir o código-fonte, editá-lo, compilá-lo e executá-lo.

Você precisa da versão mais recente do AWS SDK for Java. Se você estiver usando o Eclipse como seuIDE, você pode instalar o AWS Toolkit for Eclipse em vez disso.

O aplicativo consumidor requer a Kinesis Client Library (KCL) versão 1.2.1 ou superior, que você pode obter na GitHub Kinesis Client Library (Java).

Próximos Passos

Crie um fluxo de dados

Crie um fluxo de dados

Na primeira etapa do Tutorial: Processe dados de estoque em tempo real usando KPL e KCL 1.x, você cria o streaming que usará em etapas subsequentes.

Para criar um fluxo

1. Faça login no AWS Management Console e abra o console do Kinesis em https://console.aws.amazon.com /kinesis.
2. Selecione Data Streams (Fluxos de dados) no painel de navegação.

3. Na barra de navegação, expanda o seletor de região e escolha uma região.
4. Escolha Create Kinesis stream (Criar streaming do Kinesis).
5. Insira um nome para seu streaming (por exemplo, **`StockTradeStream`**).
6. Insira **1** o número de fragmentos, mas mantenha Estimar o número de fragmentos que você precisará reduzir.
7. Escolha Create Kinesis stream (Criar streaming do Kinesis).

Na página de lista Fluxos do Kinesis, o status do fluxo é `CREATING` enquanto ele está sendo criado. Quando o stream fica pronto para uso, o status é alterado para `ACTIVE`. Escolha o nome do fluxo. Na página exibida, a guia Details (Detalhes) exibe um resumo da configuração do streaming. A seção Monitoring (Monitoramento) exibe informações de monitoramento do streaming.

Informações adicionais sobre fragmentos

Ao começar a usar o Kinesis Data Streams fora deste tutorial, você poderá precisar planejar o processo de criação de fluxos mais cuidadosamente. Você deve se planejar para a demanda máxima esperada ao provisionar estilhaços. Usando este cenário como exemplo, o tráfego de negociações da bolsa de valores dos EUA atinge o pico durante o dia (fuso horário do leste dos EUA) e, a partir desse horário, é preciso tirar amostras das estimativas de demanda. Em seguida, você tem a opção de provisionar para a máxima demanda esperada ou expandir e reduzir o stream em resposta às variações de demanda.

Um estilhaço é uma unidade de capacidade de throughput. Na página Criar fluxo do Kinesis, expanda Estime o número de fragmentos necessários. Digite o tamanho médio do registro, o máximo de registros gravados por segundo e o número de aplicativos de consumo usando as seguintes diretrizes:

Tamanho médio do registro

> Uma estimativa do tamanho médio calculado dos registros. Se você não sabe esse valor, use o tamanho de registro máximo estimado.

Máximo de registros gravados

> Considere o número de entidades que fornecem dados e o número aproximado de registros por segundo produzidos por cada uma. Por exemplo, se você recebe dados de negociações de ações provenientes de 20 servidores mercantis, com cada um gerando 250 negociações por segundo, o número total de transações (registros) por segundo é 5.000 por segundo.

Número de aplicativos de consumo

Número de aplicativos que fazem leitura do stream de forma independente para processar o stream de outro modo e produzir saída diferente. Cada aplicativo pode ter várias instâncias em execução em máquinas diferentes (ou seja, execução em um cluster) para dar conta de um streaming de alto volume.

Se o número estimado de estilhaços mostrado exceder o limite atual de estilhaços, poderá ser necessário enviar uma solicitação para aumentar esse limite para poder criar um streaming com esse número de estilhaços. Para solicitar um aumento do limite de fragmentos, use o formulário de limites do Kinesis Data Streams. Para obter mais informações sobre fluxos e fragmentos, consulte Crie e gerencie streams de dados do Kinesis.

Próximas etapas

Crie uma IAM política e um usuário

Crie uma IAM política e um usuário

Práticas recomendadas de segurança para AWS ditar o uso de permissões refinadas para controlar o acesso a diferentes recursos. AWS Identity and Access Management (IAM) permite gerenciar usuários e permissões de usuário no AWS. Uma IAMpolítica lista explicitamente as ações que são permitidas e os recursos aos quais as ações são aplicáveis.

Veja a seguir as permissões mínimas normalmente necessárias para um produtor e um consumidor do Kinesis Data Streams.

Produtor

Ações	Recurso	Finalidade
`DescribeStream` , `DescribeStreamSummary` , `DescribeStreamConsumer`	Fluxo de dados do Kinesis	Antes de tentar gravar registros, o produtor verifica se o str se os fragmentos estão contidos no stream e se o stream te
`SubscribeToShard` , `RegisterStreamConsumer`	Fluxo de dados do Kinesis	Faz a inscrição e registra um consumidor em um estilhaço Kinesis.

Ações	Recurso	Finalidade
`PutRecord` , `PutRecords`	Fluxo de dados do Kinesis	Gravar registros no Kinesis Data Streams.

Consumidor

Ações	Recurso	Finalidade
`DescribeStream`	Fluxo de dados do Kinesis	Antes de tentar ler registros, o consumidor verifica se o stre se os estilhaços estão contidos no stream.
`GetRecords` , `GetShardIterator`	Fluxo de dados do Kinesis	Ler registros em um fragmento do Kinesis Data Streams.
`CreateTable` , `DescribeTable` , `GetItem`, `PutItem`, `Scan`, `UpdateItem`	Tabela do Amazon DynamoDB	Se o consumidor for desenvolvido usando a Kinesis Client precisará de permissões para uma tabela do DynamoDB pa processamento do aplicativo. O primeiro consumidor iniciad
`DeleteItem`	Tabela do Amazon DynamoDB	Para operações de divisão/mesclagem que o consumidor e do Kinesis Data Streams.
`PutMetricData`	CloudWatch Registro da Amazon	Ele KCL também carrega métricas para CloudWatch, que s o aplicativo.

Para esse aplicativo, você cria uma única IAM política que concede todas as permissões anteriores. Na prática, talvez convenha considerar a criação de duas políticas, uma para produtores e uma para consumidores.

Para criar uma política do IAM

1. Localize o Amazon Resource Name (ARN) para o novo stream. Você pode encontrar isso ARN listado como Stream ARN na parte superior da guia Detalhes. O ARN formato é o seguinte:

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

região

Código da região; por exemplo, `us-west-2`. Para obter mais informações, consulte Conceitos de região e zona de disponibilidade.

conta

O ID da AWS conta, conforme mostrado nas Configurações da conta.

name

Nome do stream de Crie um fluxo de dados, que é `StockTradeStream`.

2. Determine ARN a tabela do DynamoDB a ser usada pelo consumidor (e criada pela primeira instância do consumidor). Ele deve estar no seguinte formato:

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

A região e a conta são do mesmo local que a etapa anterior, mas desta vez name é o nome da tabela criada e usada pelo aplicativo de consumidor. O KCL usado pelo consumidor usa o nome do aplicativo como nome da tabela. Use o nome do aplicativo que será usado mais tarde, `StockTradesProcessor`.

3. No IAM console, em Políticas (https://console.aws.amazon.com/iam/home #policies), escolha Criar política. Se for a primeira vez que você trabalha com IAM políticas, escolha Começar, Criar política.
4. Escolha Select (Selecionar) ao lado de Policy Generator (Gerador de políticas).
5. Escolha o Amazon Kinesis como serviço. AWS
6. Selecione `DescribeStream`, `GetShardIterator`, `GetRecords`, `PutRecord`e `PutRecords` como ações permitidas.
7. Insira o ARN que você criou na Etapa 1.
8. Use Add Statement (Adicionar instrução) para cada um dos seguintes:

AWS Serviço	Ações	ARN
Amazon DynamoDB	`CreateTable` , `DeleteItem` ,	O ARN que você criou na Etapa 2

AWS Serviço	Ações	ARN
	`DescribeTable` , `GetItem`, `PutItem`, `Scan`, `UpdateItem`	
Amazon CloudWatch	`PutMetricData`	*

O asterisco (*) usado ao especificar um não ARN é obrigatório. Nesse caso, é porque não há nenhum recurso específico CloudWatch no qual a `PutMetricData` ação seja invocada.

9. Escolha Next Step.
10. Altere Policy Name (Nome da política) para `StockTradeStreamPolicy`, analise o código e escolha Create Policy (Criar política).

O documento de política resultante deve ser algo como o exemplo a seguir:

```
{
  "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 criar um usuário do IAM

1. Abra o IAM console em https://console.aws.amazon.com/iam/.
2. Na página Usuários, selecione Adicionar usuário.
3. Para User name, digite `StockTradeStreamUser`.
4. Em Access type (Tipo de acesso), escolha Programmatic access (Aceso programático) e, em seguida, escolha Next: Permissions (Próximo: permissões).
5. Escolha Anexar políticas existentes diretamente.
6. Pesquise por nome a política que você criou. Selecione a caixa à esquerda do nome da política e escolha Next: Review (Próximo: revisão).

7. Revise os detalhes e o resumo e, em seguida, escolha Create user (Criar usuário).
8. Copie o Access key ID (ID da chave de acesso) e salve-o de forma privada. Em Secret access key (Chave de acesso secreta), escolha Show (Mostrar) e salve a chave de forma privada também.
9. Cole as chaves de acesso e chaves secretas em um arquivo local em lugar seguro que somente você possa acessar. Para esse aplicativo, crie um arquivo denominado `~/.aws/credentials` (com permissões restritas). O arquivo deverá estar no seguinte formato:

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

Para anexar uma IAM política a um usuário

1. No IAM console, abra Políticas e escolha Ações de política.
2. Escolha `StockTradeStreamPolicy` e Attach (Anexar).
3. Escolha `StockTradeStreamUser` e Attach Policy (Anexar política).

Próximos Passos

Baixe e crie o código de implementação

Baixe e crie o código de implementação

O código esqueleto é fornecido para o the section called “Tutorial: Processe dados de estoque em tempo real usando KPL e KCL 1.x”. Ele contém uma implementação de stub para o consumo do streaming de negociações de ações (produtor) e para o processamento dos dados (consumidor). O procedimento a seguir mostra como concluir a implementação.

Para fazer download e compilação do código de implementação

1. Faça download do código-fonte no computador.
2. Crie um projeto em seu favorito IDE com o código-fonte, seguindo a estrutura de diretórios fornecida.
3. Adicione as seguintes bibliotecas ao projeto:

   - Biblioteca de cliente do Amazon Kinesis () KCL

- AWS SDK
- Apache HttpCore
- Apache HttpClient
- Apache Commons Lang
- Apache Commons Logging
- Guava (Google Core Libraries For Java)
- Jackson Annotations
- Jackson Core
- Jackson Databind
- Formato de dados Jackson: CBOR
- Joda Time

4. Dependendo do seuIDE, o projeto pode ser construído automaticamente. Caso contrário, crie o projeto usando as etapas apropriadas para o seuIDE.

Se concluiu essas etapas com êxito, você agora está pronto para ir para a próxima seção, the section called “Implemente o produtor”. Se a compilação gerar erros em qualquer estágio, investigue e os corrija antes de continuar.

Próximas etapas

Implemente o produtor

O aplicativo no Tutorial: Processe dados de estoque em tempo real usando KPL e KCL 1.x usa o cenário real de monitoramento de negociações em bolsa de valores. Os princípios a seguir explicam brevemente como este cenário é mapeado para o produtor e a estrutura de código de apoio.

Consulte o código-fonte e analise as informações a seguir.

StockTrade classe

Uma negociação de ação individual é representada por uma instância da classe `StockTrade`. Essa instância contém atributos como o símbolo ticker, o preço, o número de ações, o tipo da negociação (compra ou venda) e um ID que identifica a negociação com exclusividade. Essa classe é implementada para você.

Registro de stream

Um stream é uma sequência de registros. Um registro é uma serialização de uma `StockTrade` instância em JSON formato. Por exemplo:

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

StockTradeGenerator classe

`StockTradeGenerator` tem um método denominado `getRandomTrade()`, que retorna uma nova negociação de ações gerada aleatoriamente sempre que ela é invocada. Essa classe é implementada para você.

StockTradesWriter classe

O método `main` do produtor, `StockTradesWriter`, recupera continuamente uma negociação aleatória e a envia ao Kinesis Data Streams executando as seguintes tarefas:

1. Lê o nome do stream e o nome da região como entrada.
2. Cria um `AmazonKinesisClientBuilder`.
3. Usa o criador do cliente para definir região, credenciais e configuração do cliente.
4. Cria um cliente `AmazonKinesis` usando o criador do cliente.
5. Verifica se o stream existe e está ativo (se não, ele será encerrado com um erro).
6. Em um loop contínuo, chama o método `StockTradeGenerator.getRandomTrade()` e o método `sendStockTrade` para enviar a negociação ao stream a cada 100 milissegundos.

O método `sendStockTrade` da classe `StockTradesWriter` tem o seguinte 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 o desmembramento do código a seguir:

- O `PutRecord` API espera uma matriz de bytes e você deve converter `trade` para o JSON formato. Essa única linha de código executa a seguinte operação:

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

- Antes de enviar a negociação, você cria uma nova instância de `PutRecordRequest` (denominada `putRecord` neste caso):

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

  Cada chamada a `PutRecord` requer o nome do stream, uma chave de partição e um blob de dados. O código a seguir preenche esses campos no objeto `putRecord` usando seus métodos `setXxxx()`:

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

  O exemplo usa um tíquete de ações como uma chave de partição, que mapeia o registro para um determinado estilhaço. Na prática, você deve ter centenas ou milhares de chaves de partição por estilhaço, de forma que os registros sejam uniformemente disseminados no

seu stream. Para obter mais informações sobre como adicionar dados a um stream, consulte Adicionar dados a um stream.

Agora `putRecord` está pronto para enviar para o cliente (operação `put`):

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

- A verificação e o registro de erros são sempre inclusões úteis. Este código registra condições de erro:

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

Adicione o bloco try/catch ao redor da operação `put`:

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

Isso ocorre porque uma operação `put` do Kinesis Data Streams pode falhar devido a um erro de rede ou porque o fluxo de dados atinge o limite de throughput e tem sua utilização controlada. Recomendamos considerar cuidadosamente sua política de repetição de `put` operações para evitar perda de dados, como usar uma nova tentativa.

- O registro de status é útil mas opcional:

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

O produtor mostrado aqui usa a funcionalidade de registro único do Kinesis API Data Streams,. `PutRecord` Na prática, se um produtor individual gerar muitos registros, costuma ser mais eficiente usar a funcionalidade de vários registros de `PutRecords` e enviar lotes de registros por vez. Para obter mais informações, consulte Adicionar dados a um stream.

Para executar o produtor

1. Verifique se a chave de acesso e o par de chaves secretas recuperados anteriormente (ao criar o IAM usuário) estão salvos no arquivo`~/.aws/credentials`.
2. Execute a classe `StockTradeWriter` com os seguintes argumentos:

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

   Se você criou o stream em uma região diferente de `us-west-2`, precisará especificar essa região aqui.

Você deve ver saída semelhante a:

```
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
```

Seu fluxo de negociações de ações agora está sendo ingerido pelo Kinesis Data Streams.

Próximas etapas

Implemente o consumidor

Implemente o consumidor

O aplicativo consumidor no Tutorial: Processe dados de estoque em tempo real usando KPL e KCL 1.x processa continuamente o streaming de negociações de ações criado em . Em seguida, ele produz as ações mais populares compradas e vendidas a cada minuto. O aplicativo é construído com base na Kinesis Client Library (KCL), que faz grande parte do trabalho pesado comum aos aplicativos de consumo. Para obter mais informações, consulte Desenvolva KCL consumidores 1.x.

Consulte o código-fonte e analise as informações a seguir.

StockTradesProcessor classe

Principal classe do consumidor fornecida e que executa as seguintes tarefas:

- Lê o aplicativo, o streaming e os nomes de região passados como argumentos.
- Lê credenciais de `~/.aws/credentials`.
- Cria uma instância de `RecordProcessorFactory` que veicula instâncias de `RecordProcessor`, implementadas por uma instância de `StockTradeRecordProcessor`.
- Cria um KCL trabalhador com a `RecordProcessorFactory` instância e uma configuração padrão, incluindo o nome do stream, as credenciais e o nome do aplicativo.
- O operador cria um novo thread para cada fragmento (atribuído a essa instância de consumidor), que opera em loops contínuos para ler registros do Kinesis Data Streams. Em seguida, ele invoca a instância de `RecordProcessor` para processar cada lote de registros recebidos.

StockTradeRecordProcessor classe

Implementação da instância de `RecordProcessor`, que, por sua vez, implementa três métodos necessários: `initialize`, `processRecords` e `shutdown`.

Como os nomes sugerem, `initialize` e `shutdown` são usados pela Kinesis Client Library para permitir que o processador de registros saiba quando deve estar pronto para começar a receber registros e quando deve esperar parar de receber registros, respectivamente, para poder realizar tarefas de configuração e encerramento específicas da aplicação. O código disso é fornecido para você. O processamento principal ocorre no método `processRecords`, que, por sua vez, usa `processRecord` para cada registro. Esse último método é fornecido como um código esqueleto quase todo vazio, para você implementar na próxima etapa, onde é melhor explicado.

Observe também a implementação dos métodos de suporte de `processRecord`: `reportStats` e `resetStats`, que estão vazios no código-fonte original.

O método `processRecords`, implementado para você, executa as seguintes etapas:

- Para cada registro passado, chama `processRecord`.
- Se tiver decorrido pelo menos 1 minuto após o último relatório, chamará `reportStats()`, que imprime as estatísticas mais recentes e, em seguida, `resetStats()`, que limpa as estatísticas para que o próximo intervalo inclua apenas registros novos.
- Define o próximo horário para geração de relatórios.

- Se tiver decorrido pelo menos 1 minuto após o último ponto de verificação, chamará `checkpoint()`.
- Define o próximo horário do ponto de verificação.

Este método usa intervalos de 60 segundos como taxa de geração de relatórios e definição de pontos de verificação. Para obter mais informações sobre definição de pontos de verificação, consulte Informações adicionais sobre o consumidor.

StockStats classe

Essa classe fornece retenção de dados e rastreamento de estatísticas em relação às ações mais populares ao longo do tempo. Esse código, fornecido para você, contém os seguintes métodos:

- `addStockTrade(StockTrade)`: injeta o `StockTrade` conhecido nas estatísticas correntes.
- `toString()`: retorna as estatísticas em uma string formatada.

Essa classe rastreia as ações mais populares mantendo uma contagem contínua do número total de negociações de cada ação e a contagem máxima. Ela atualiza essas contagens sempre que chega uma negociação de ação.

Adicione código aos métodos da classe `StockTradeRecordProcessor`, como mostrado nas etapas a seguir.

Para implementar o consumidor

1. Implemente o método `processRecord` instanciando um objeto `StockTrade` de tamanho correto e adicionando a ele os dados do registro, registrando um aviso caso ocorra 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 um método `reportStats` simples. Sinta-se à vontade para modificar o formato de saída conforme suas preferências.

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

```
                    stockStats + "\n" +

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

3. Finalmente, implemente o método `resetStats`, que cria uma nova instância de `stockStats`.

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

Para executar o consumidor

1. Execute o produtor escrito em para injetar registros de negociações de ações no streaming.
2. Verifique se a chave de acesso e o par de chaves secretas recuperados anteriormente (ao criar o IAM usuário) estão salvos no arquivo`~/.aws/credentials`.
3. Execute a classe `StockTradesProcessor` com os seguintes argumentos:

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

   Observe que, se você criou o stream em uma região diferente de `us-west-2`, precisará especificar essa região aqui.

Depois de um minuto, deverá aparecer uma saída como a seguir, atualizada a cada minuto a partir de então:

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

Informações adicionais sobre o consumidor

Se você já conhecer as vantagens da Kinesis Client Library, abordada em Desenvolva KCL consumidores 1.x e em outros documentos, poderá estar se perguntando porque usá-la aqui. Embora você use apenas um único fluxo de fragmentos e uma única instância de consumidor para processá-lo, ainda é mais fácil implementar o consumidor usando o. KCL Compare as etapas de implementação do código na seção do produtor para o consumidor para ver a facilidade comparativa para implementar um consumidor. Isso se deve em grande parte aos serviços que eles KCL fornecem.

Nesse aplicativo, você se concentra na implementação de uma classe de processador de registros, capaz de processar registros individuais. Você não precisa se preocupar com a forma como os registros são obtidos no Kinesis Data Streams; KCL eles buscam os registros e invocam o processador de registros sempre que há novos registros disponíveis. Além disso, você não precisa se preocupar com a quantidade de estilhaços e de instâncias de consumidor. Se o streaming for escalonado, você não precisará reescrever o aplicativo para lidar com mais de um estilhaço ou com uma instância de consumidor.

O termo checkpoint significa registrar o ponto no fluxo até os registros de dados que foram consumidos e processados até o momento. Se o aplicativo falhar, o fluxo será lido a partir desse ponto e não do início do fluxo. O assunto da definição de pontos de verificação e os vários padrões de design e melhores práticas relativos estão fora do escopo deste capítulo. No entanto, é algo que você pode encontrar em ambientes de produção.

Como você aprendeu em, as `put` operações no Kinesis API Data Streams usam uma chave de partição como entrada. O Kinesis Data Streams usa uma chave de partição como um mecanismo para dividir registros em vários fragmentos (quando há mais de um fragmento no fluxo). A mesma chave de partição sempre roteia para o mesmo estilhaço. Isso permite que o consumidor que processa um determinado estilhaço seja projetado com a premissa de que os registros com a mesma chave de partição só sejam enviados a esse consumidor, e nenhum registro com a mesma chave de partição termine em qualquer outro consumidor. Portanto, o operador de um consumidor pode agregar todos os registros com a mesma chave de partição sem se preocupar com a ausência de dados necessários.

Nesse aplicativo, o processamento de registros pelo consumidor não é intensivo, então você pode usar um fragmento e fazer o processamento no mesmo encadeamento do KCL encadeamento. No entanto, na prática, considere primeiro escalar o número de estilhaços. Em alguns casos, talvez convenha mudar o processamento para outro thread ou usar um grupo de threads se for esperado que o processamento de registros seja intensivo. Dessa forma, eles KCL podem buscar novos registros mais rapidamente, enquanto os outros encadeamentos podem processar os registros em paralelo. O design multiencadeado não é trivial e deve ser abordado com técnicas avançadas, portanto, aumentar a contagem de fragmentos geralmente é a maneira mais eficaz de aumentar a escala.

Próximas etapas

[(Opcional) Estenda o consumidor]

(Opcional) Estenda o consumidor

O aplicativo no Tutorial: Processe dados de estoque em tempo real usando KPL e KCL 1.x já pode ser suficiente para os seus propósitos. Esta seção opcional mostra como estender o código de consumidor para um cenário um pouco mais elaborado.

Se quiser saber mais sobre os maiores pedidos de venda a cada minuto, você poderá modificar a classe `StockStats` em três locais para acomodar essa nova prioridade.

Para estender o consumidor

1. Adicione novas variáveis de instância:

```
// 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. Adicione o seguinte código a `addStockTrade`:

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

3. Modifique o método `toString` para imprimir as informações adicionais:

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

Se você executar o consumidor agora (lembre-se de executar o produtor também), deverá aparecer uma saída semelhante a esta:

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

Próximas etapas

Limpar os recursos

Limpar os recursos

Como você está pagando para usar o fluxo de dados do Kinesis, certifique-se de excluí-lo e de excluir a tabela do Amazon DynamoDB correspondente ao concluir. As cobranças nominais ocorrerão em um stream ativo mesmo quando você não estiver enviando e recebendo registros. Isso ocorre porque um stream ativo usa recursos por meio da "escuta" contínua de registros recebidos e solicitações para obter registros.

Para excluir o stream e tabela

1. Desligue os produtores e consumidores que possam estar em execução.
2. Abra o console do Kinesis em https://console.aws.amazon.com /kinesis.
3. Escolha o stream que você criou para este aplicativo (`StockTradeStream`).
4. Escolha Delete Stream (Excluir streaming).
5. Abra o console do DynamoDB em. https://console.aws.amazon.com/dynamodb/
6. Exclua a tabela `StockTradesProcessor`.

Resumo

O processamento de uma grande quantidade de dados quase em tempo real não exige escrever códigos complicados ou desenvolver uma grande infraestrutura. É tão básico quanto escrever uma lógica para processar uma pequena quantidade de dados (como escrever`processRecord(Record)`), mas usar o Kinesis Data Streams para escalar de forma que funcione para uma grande quantidade de dados de streaming. Você não precisa se preocupar com

a escalabilidade do processamento, porque o Kinesis Data Streams cuida de tudo. Você só precisa enviar seus registros de streaming ao Kinesis Data Streams e escrever a lógica para processar cada novo registro recebido.

Veja aqui alguns aprimoramentos potenciais para este aplicativo.

Agregar em todos os estilhaços

Atualmente, você obtém estatísticas resultantes da agregação de registros de dados recebidos por um único operador proveniente de um único estilhaço. (Um estilhaço não pode ser processado por mais de um operador em um aplicativo ao mesmo tempo). Naturalmente, quando escala e tem mais de um estilhaço, você pode agregar em todos os estilhaços. É possível fazer isso tendo uma arquitetura de pipeline em que a saída de cada operador é alimentada em outro fluxo com um único estilhaço, o qual é processado por um operador que agrega as saídas do primeiro estágio. Como os dados do primeiro estágio são limitados (um exemplo por minuto por estilhaço), eles podem ser facilmente tratados por um estilhaço.

Escalar o processamento

Quando o stream é expandido para ter muitos estilhaços (porque muitos produtores estão enviando dados), a maneira de escalar o processamento é adicionando mais operadores. Você pode executar os trabalhadores em EC2 instâncias da Amazon e usar grupos de Auto Scaling.

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

Como um fluxo é processado continuamente, sua saída pode ser enviada para outros destinos. AWS fornece conectores para integrar o Kinesis Data Streams com AWS outros serviços e ferramentas de terceiros.

Próximas etapas

- Para obter mais informações sobre o uso das operações do Kinesis API Data StreamsDesenvolva produtores usando o Amazon Kinesis API Data Streams com o AWS SDK for Java, Desenvolva consumidores personalizados com taxa de transferência compartilhada usando o AWS SDK for Java consulte, e. Crie e gerencie streams de dados do Kinesis
- Para obter mais informações sobre a Kinesis Client Library, consulte Desenvolva KCL consumidores 1.x.
- Para obter mais informações sobre como otimizar seu aplicativo, consulte Otimize os consumidores do Amazon Kinesis Data Streams.

Tutorial: Analise dados de estoque em tempo real usando o Amazon Managed Service para Apache Flink

O cenário deste tutorial envolve consumir negociações do mercado de ações em um fluxo de dados e criar uma aplicação simples do Amazon Managed Service for Apache Flink para realizar cálculos no fluxo. Você aprenderá a enviar um stream de registros para o Kinesis Data Streams e a implementar um aplicativo que consome e processa os registros quase em tempo real.

Com o Amazon Managed Service para Apache Flink, você pode usar Java ou Scala para processar e analisar dados de streaming. O serviço permite criar e executar código Java ou Scala em fontes de streaming para realizar análises de séries temporais, alimentar painéis em tempo real e criar métricas em tempo real.

Você pode criar aplicações Flink no Managed Service for Apache Flink usando bibliotecas de código aberto baseadas no Apache Flink. O Apache Flink é uma estrutura popular e um mecanismo para o processamento de fluxos de dados.

> **Important**
>
> Depois de criar dois fluxos de dados e um aplicativo, sua conta incorre em cobranças nominais pelo Kinesis Data Streams e pelo Managed Service for Apache Flink porque eles não estão qualificados para o nível gratuito. AWS Quando você terminar de usar esse aplicativo, exclua seus AWS recursos para parar de incorrer em cobranças.

O código não acessa os dados reais da bolsa de valores, ele simula o stream de negociações de ações. Isso é feito com o uso de um gerador de negociações de ações aleatórias. Se tiver acesso a um streaming de negociações de ações em tempo real, você poderá se interessar em derivar estatísticas úteis e em tempo hábil desse streaming. Por exemplo, talvez convenha executar uma análise de janela deslizante na qual você determina a ação mais popular que foi adquirida nos últimos 5 minutos. Ou talvez convenha uma notificação sempre que uma ordem de venda for muito grande (ou seja, tenha muitas quotas). Você pode estender o código nesta série para oferecer essa funcionalidade.

Os exemplos mostrados usam a região Oeste dos EUA (Oregon), mas funcionam em qualquer região da AWS que oferece suporte ao Managed Service for Apache Flink.

Tarefas

- Pré-requisitos para concluir os exercícios
- Configurar uma AWS conta e criar um usuário administrador
- Configure o AWS Command Line Interface (AWS CLI)
- Crie e execute um serviço gerenciado para o aplicativo Apache Flink

Pré-requisitos para concluir os exercícios

Para concluir as etapas neste guia, você deve ter o seguinte:

- Java Development Kit (JDK) versão 8. Defina a variável de `JAVA_HOME` ambiente para apontar para o local de JDK instalação.
- Recomendamos que você use um ambiente de desenvolvimento (como Eclipse Java Neon ou IntelliJ Idea) para desenvolver e compilar seu aplicativo.
- Cliente do Git. Instale o cliente do Git se você ainda não tiver feito isso.
- Apache Maven Compiler Plugin. Maven deve estar em seu caminho de trabalho. Para testar a instalação do Apache Maven, insira o seguinte:

```
$ mvn -version
```

Para começar a usar, vá até Configurar uma AWS conta e criar um usuário administrador.

Configurar uma AWS conta e criar um usuário administrador

Antes de usar o Amazon Managed Service para Apache Flink pela primeira vez, conclua as seguintes tarefas:

1. Inscreva-se para AWS
2. Criar um usuário do IAM

Inscreva-se para AWS

Quando você se inscreve no Amazon Web Services (AWS), sua AWS conta é automaticamente cadastrada em todos os serviços AWS, incluindo o Amazon Managed Service para Apache Flink. Você será cobrado apenas pelos serviços que usar.

Com o Managed Service for Apache Flink, você paga apenas pelos recursos que usa. Novos clientes da AWS podem começar a usar o Managed Service for Apache Flink gratuitamente. Para obter mais informações, consulte Nível gratuito da AWS.

Se você já tiver uma AWS conta, vá para a próxima tarefa. Se você não tiver uma conta da AWS , siga as etapas a seguir para criar uma.

Para criar uma AWS conta

1. Abra a https://portal.aws.amazon.com/billing/inscrição.
2. Siga as instruções online.

   Parte do procedimento de inscrição envolve receber uma chamada telefônica e inserir um código de verificação no teclado do telefone.

   Quando você se inscreve em um Conta da AWS, um Usuário raiz da conta da AWSé criado. O usuário raiz tem acesso a todos os Serviços da AWS e atributos na conta. Como prática recomendada de segurança, atribua o acesso administrativo a um usuário e use somente o usuário-raiz para executar tarefas que exigem acesso de usuário-raiz.

Anote o ID da sua AWS conta porque você precisará dele para a próxima tarefa.

Criar um usuário do IAM

Serviços em AWS, como o Amazon Managed Service para Apache Flink, exigem que você forneça credenciais ao acessá-los. Dessa maneira, o serviço pode determinar se você tem permissões para acessar os recursos próprios desse serviço. O AWS Management Console exige que você digite sua senha.

Você pode criar chaves de acesso para sua AWS conta para acessar o AWS Command Line Interface (AWS CLI) ouAPI. No entanto, não recomendamos que você acesse AWS usando as credenciais da sua AWS conta. Em vez disso, recomendamos que você use AWS Identity and Access Management (IAM). Crie um IAM usuário, adicione-o a um IAM grupo com permissões administrativas e, em seguida, conceda permissões administrativas ao IAM usuário que você criou. Em seguida, você pode acessar AWS usando um especial URL e as credenciais desse IAM usuário.

Se você se inscreveu AWS, mas não criou um IAM usuário para si mesmo, você pode criar um usando o IAM console.

Os exercícios de conceitos básicos deste guia pressupõem que você tenha um usuário (`adminuser`) com permissões de administrador. Siga o procedimento para criar `adminuser` na conta.

Para criar um grupo de administradores

1. Faça login no AWS Management Console e abra o IAM console em https://console.aws.amazon.com/iam/.
2. No painel de navegação, escolha Groups (Grupos) e Create New Group (Criar novo grupo).
3. Em Group Name (Nome do grupo), digite um nome para o grupo, como **`Administrators`** e escolha Next Step (Próxima etapa).
4. Na lista de políticas, marque a caixa de seleção ao lado da AdministratorAccesspolítica. Você pode usar o menu Filtro e a caixa Pesquisar para filtrar a lista de políticas.
5. Selecione Next Step (Próximo passo) e, em seguida, Create Group (Criar grupo).

O grupo novo é listado em Group Name (Nome do grupo).

Para criar um IAM usuário para você, adicione-o ao grupo Administradores e crie uma senha

1. No painel de navegação, escolha Usuários e depois Adicionar usuário.
2. Na caixa User name (Nome de usuário), insira um nome de usuário.
3. Escolha tanto Acesso programático como Acesso ao console de Gerenciamento da AWS .
4. Selecione Next: Permissions (Próximo: permissões).
5. Marque a caixa de seleção ao lado do grupo Administrators (Administradores). Então, escolha Próximo: Análise.
6. Selecione Criar usuário.

Para entrar como o novo IAM usuário

1. Saia do AWS Management Console.
2. Use o seguinte URL formato para entrar no console:

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

   A ferramenta *`aws_account_number`* é o ID AWS da sua conta sem hífens. Por exemplo, se o ID da sua AWS conta for 1234-5678-9012, substitua *`aws_account_number`*

com**123456789012**. Para obter informações sobre como encontrar o número da sua conta, consulte o ID AWS da sua conta e seu alias no Guia do IAM usuário.

3. Insira o nome de IAM usuário e a senha que você acabou de criar. Quando você está conectado, a barra de navegação é exibida *your_user_name* @ *your_aws_account_id*.

> **Note**
>
> Se você não quiser que sua página URL de login contenha o ID da sua AWS conta, você pode criar um alias de conta.

Para criar ou remover um alias de conta

1. Abra o IAM console em https://console.aws.amazon.com/iam/.
2. No painel de navegação, selecione Dashboard (Painel).
3. Encontre o link de login dos IAM usuários.
4. Para criar o alias, escolha Customize (Personalizar). Insira o nome que você deseja usar para o alias e escolha Yes, Create (Sim, criar).
5. Para remover o alias, selecione Personalizar e, em seguida, selecione Sim, excluir. O login é URL revertido para o uso do ID da sua AWS conta.

Para entrar depois de criar um alias de conta, use o seguinte: URL

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

Para verificar o link de login dos IAM usuários da sua conta, abra o IAM console e verifique o link de login IAM dos usuários no painel.

Para obter mais informações sobreIAM, consulte o seguinte:

- AWS Identity and Access Management (IAM)
- Conceitos básicos
- IAMGuia do usuário

Próxima etapa

Configure o AWS Command Line Interface (AWS CLI)

Configure o AWS Command Line Interface (AWS CLI)

Nesta etapa, você baixa e configura o AWS CLI para uso com o Amazon Managed Service para Apache Flink.

> **Note**
>
> Os exercícios de conceitos básicos neste guia pressupõem que você esteja usando credenciais de administrador (`adminuser`) em sua conta para executar as operações.

> **Note**
>
> Se você já tem o AWS CLI instalado, talvez seja necessário fazer o upgrade para obter a funcionalidade mais recente. Para obter mais informações, consulte Instalando a interface de linha de AWS comando no Guia AWS Command Line Interface do usuário. Para verificar a versão do AWS CLI, execute o seguinte comando:
>
> ```
> aws --version
> ```
>
> Os exercícios deste tutorial exigem a seguinte AWS CLI versão ou posterior:
>
> ```
> aws-cli/1.16.63
> ```

Para configurar o AWS CLI

1. Faça download e configure a AWS CLI. Para obter instruções, consulte os seguintes tópicos no Guia do usuário do AWS Command Line Interface :
   - Instalar a AWS Command Line Interface
   - Configurar a AWS CLI
2. Adicione um perfil nomeado para o usuário administrador no arquivo de AWS CLI configuração. Você usa esse perfil ao executar os AWS CLI comandos. Para obter mais informações sobre perfis nomeados, consulte Perfis nomeados no Guia do usuário da 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 obter uma lista das AWS regiões disponíveis, consulte AWS Regiões e endpoints no Referência geral da Amazon Web Services.

3. Verifique a configuração digitando o seguinte comando no prompt de comando:

```
aws help
```

Depois de configurar uma AWS conta e a AWS CLI, você pode tentar o próximo exercício, no qual você configura um aplicativo de amostra e testa a end-to-end configuração.

Próxima etapa

Crie e execute um serviço gerenciado para o aplicativo Apache Flink

Crie e execute um serviço gerenciado para o aplicativo Apache Flink

Neste exercício, você cria um aplicativo Managed Service for Apache Flink com fluxos de dados como origem e coletor.

Esta seção contém as seguintes etapas:

- Crie dois streams de dados do Amazon Kinesis
- Gravação de registros de amostra no fluxo de entrada
- Baixe e examine o código Java de streaming do Apache Flink
- Compilar o código do aplicativo
- Faça o upload do código Java de streaming do Apache Flink
- Crie e execute o aplicativo Managed Service for Apache Flink

Crie dois streams de dados do Amazon Kinesis

Antes de criar um Amazon Managed Service para Apache Flink para este exercício, crie dois streams de dados do Kinesis (e). `ExampleInputStream` `ExampleOutputStream` O aplicativo usa esses fluxos para os fluxos de origem e de destino do aplicativo.

Você pode criar esses fluxos usando o console do Amazon Kinesis ou o comando da AWS CLI a seguir. Para instruções do console, consulte Criar e atualizar streamings de dados.

Como criar os fluxos de dados (AWS CLI)

1. Para criar o primeiro stream (`ExampleInputStream`), use o seguinte comando do Amazon Kinesis `create-stream` AWS CLI .

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

2. Para criar o segundo fluxo que o aplicativo usa para gravar a saída, execute o mesmo comando, alterando o nome da transmissão para `ExampleOutputStream`.

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

Gravação de registros de amostra no fluxo de entrada

Nesta seção, você usa um script Python para gravar registros de amostra no fluxo para o aplicativo processar.

> **Note**
>
> Essa seção requer AWS SDK for Python (Boto).

1. Crie um arquivo denominado `stock.py` com o conteúdo a seguir:

```
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. Mais adiante neste tutorial, você executa o script `stock.py` para enviar dados para o aplicativo.

```
$ python stock.py
```

Baixe e examine o código Java de streaming do Apache Flink

O código do aplicativo Java para esses exemplos está disponível em GitHub. Para fazer download do código do aplicativo, faça o seguinte:

1. Duplique o repositório remoto com o seguinte comando:

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

2. Navegue até o diretório `GettingStarted`.

O código do aplicativo está localizado nos arquivos `CloudWatchLogSink.java` e `CustomSinkStreamingJob.java`. Observe o seguinte sobre o código do aplicativo:

- A aplicação usa uma origem do Kinesis para ler o fluxo de origem. O trecho a seguir cria o coletor do Kinesis:

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

Compilar o código do aplicativo

Nesta seção, você usa o compilador do Apache Maven para criar o código Java para o aplicativo. Para obter informações sobre a instalação do Apache Maven e do Java Development Kit (JDK), consulte. Pré-requisitos para concluir os exercícios

Seu aplicativo Java requer os seguintes componentes:

- Um arquivo Project Object Model (pom.xml). Esse arquivo contém informações sobre a configuração e as dependências do aplicativo, incluindo as bibliotecas do Amazon Managed Service para Apache Flink.
- Um método `main` que contém a lógica do aplicativo.

> **Note**
>
> Para usar o conector Kinesis no aplicativo a seguir, você deve baixar o código-fonte do conector e criá-lo conforme descrito na documentação do Apache Flink.

Como criar e compilar o código do aplicativo

1. Crie um aplicativo Java/Maven em seu ambiente de desenvolvimento. Para obter informações sobre como criar um aplicativo, consulte a documentação do seu ambiente de desenvolvimento:
   - Como criar seu primeiro projeto Java (Eclipse Java Neon) (em inglês)
   - Como criar, executar e empacotar seu primeiro aplicativo Java (IntelliJ Idea) (em inglês)
2. Use o código a seguir para um arquivo chamado `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");
    }
}
```

Observe o seguinte sobre o exemplo de código anterior:

- Este arquivo contém o método `main` que define a funcionalidade do aplicativo.
- Seu aplicativo cria conectores de origem e de destino para acessar recursos externos usando um objeto `StreamExecutionEnvironment`.
- O aplicativo cria conectores de origem e de destino usando propriedades estáticas. Para usar as propriedades do aplicativo dinâmico, use os métodos `createSinkFromApplicationProperties` e `createSourceFromApplicationProperties` para criar os conectores. Esses métodos leem as propriedades do aplicativo para configurar os conectores.

3. Para usar o código do aplicativo, você o compila e o empacota em um JAR arquivo. Há duas formas de compilar e empacotar o código:

   - Use a ferramenta de linha de comando do Maven. Crie seu JAR arquivo executando o seguinte comando no diretório que contém o `pom.xml` arquivo:

     ```
     mvn package
     ```

   - Use o ambiente de desenvolvimento. Consulte a documentação de seu ambiente de desenvolvimento para obter mais detalhes.

   Você pode carregar seu pacote como um JAR arquivo ou compactar seu pacote e carregá-lo como um ZIP arquivo. Se você criar seu aplicativo usando o AWS CLI, você especifica seu tipo de conteúdo de código (JARouZIP).

4. Se houver erros durante a compilação, verifique se sua variável de ambiente `JAVA_HOME` está definida corretamente.

Se o aplicativo for compilado com êxito, o arquivo a seguir é criado:

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

Faça o upload do código Java de streaming do Apache Flink

Nesta seção, você cria um bucket do Amazon Simple Storage Service (Amazon S3) e faz upload do código do aplicativo.

Para fazer upload do código do aplicativo

1. Abra o console do Amazon S3 em. https://console.aws.amazon.com/s3/
2. Selecione Criar bucket.
3. Insira **ka-app-code-*<username>*** no campo Nome do bucket. Adicione um sufixo para o nome do bucket, como o nome do usuário, para torná-lo globalmente exclusivo. Selecione Next (Próximo).
4. Na etapa Configurar opções, mantenha as configurações como estão e selecione Próximo.
5. Na etapa Definir permissões, mantenha as configurações como estão e selecione Próximo.
6. Selecione Criar bucket.
7. No console do Amazon S3, escolha o - ka-app-code***<username>***bucket e escolha Upload.
8. Na etapa Selecionar arquivos, selecione Adicionar arquivos. Navegue até o arquivo `java-getting-started-1.0.jar` que você criou na etapa anterior. Escolha Próximo.
9. Na etapa Definir permissões, mantenha as configurações como estão. Escolha Próximo.
10. Na etapa Definir propriedades, mantenha as configurações como estão. Escolha Carregar.

O código passa a ser armazenado em um bucket do Amazon S3 que pode ser acessado pela aplicação.

Crie e execute o aplicativo Managed Service for Apache Flink

Você pode criar e executar um aplicativo Managed Service for Apache Flink usando o console ou a AWS CLI.

> **Note**
>
> Quando você cria o aplicativo usando o console, seus recursos AWS Identity and Access Management (IAM) e do Amazon CloudWatch Logs são criados para você. Ao criar o aplicativo usando o AWS CLI, você cria esses recursos separadamente.

Tópicos

- Crie e execute o aplicativo (Console)
- Crie e execute o aplicativo (AWS CLI)

Crie e execute o aplicativo (Console)

Siga estas etapas para criar, configurar, atualizar e executar o aplicativo usando o console.

Criar o aplicativo

1. Abra o console do Kinesis em https://console.aws.amazon.com /kinesis.
2. No painel do Amazon Kinesis, escolha Criar aplicativo de análise.
3. Na página Kinesis Analytics - Create application (Kinesis Analytics – Criar aplicativo), forneça os detalhes do aplicativo da seguinte forma:
   - Em Nome do aplicativo, insira **`MyApplication`**.
   - Em Descrição, insira **`My java test app`**.
   - Em Runtime (Tempo de execução), escolha Apache Flink 1.6.
4. Para permissões de acesso, escolha Criar/atualizar IAM função **`kinesis-analytics-MyApplication-us-west-2`**.
5. Selecione Create application (Criar aplicativo).

> **Note**
>
> Ao criar um aplicativo Amazon Managed Service para Apache Flink usando o console, você tem a opção de criar uma IAM função e uma política para seu aplicativo. O aplicativo usa essa função e política para acessar os recursos dependentes. Esses IAM recursos são nomeados usando o nome do aplicativo e a região da seguinte forma:
>
> - Política: `kinesis-analytics-service-`*`MyApplication`*`-`*`us-west-2`*
> - Função: `kinesis-analytics-`*`MyApplication`*`-`*`us-west-2`*

Edite a IAM política

Edite a IAM política para adicionar permissões para acessar os fluxos de dados do Kinesis.

1. Abra o IAM console em https://console.aws.amazon.com/iam/.

2. Selecione Policies (Políticas). Selecione a política **`kinesis-analytics-service-MyApplication-us-west-2`** que o console criou para você na seção anterior.
3. Na página Summary (Resumo), selecione Edit policy (Editar política). Escolha a JSONguia.
4. Adicione a seção destacada do exemplo de política a seguir à política. Substitua a conta de amostra IDs (*012345678901*) com o ID da sua conta.

```
{
    "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 o aplicativo

1. Na MyApplicationpágina, escolha Configurar.
2. Na página Configurar aplicativo, forneça o Local do código:
   - Em Bucket do Amazon S3, insira **ka-app-code-*<username>***.
   - Em Caminho do objeto do Amazon S3, insira **java-getting-started-1.0.jar**.
3. Em Acesso aos recursos do aplicativo, para permissões de acesso, escolha Criar/atualizar IAM função **kinesis-analytics-MyApplication-us-west-2**.
4. Em Properties (Propriedades), Group ID (ID do grupo), insira **ProducerConfigProperties**.
5. Insira as seguintes propriedades e valores de aplicativo:

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

6. Em Monitoramento, confirme se Nível de monitoramento de métricas está definido como Aplicativo.
7. Para CloudWatch registrar, marque a caixa de seleção Ativar.
8. Selecione Atualizar.

> **Note**
>
> Quando você opta por ativar o CloudWatch registro, o Managed Service for Apache Flink cria um grupo de registros e um fluxo de registros para você. Os nomes desses recursos são os seguintes:
>
> - Grupo de logs: `/aws/kinesis-analytics/MyApplication`
> - Fluxo de logs: `kinesis-analytics-log-stream`

Execute o aplicativo

1. Na MyApplicationpágina, escolha Executar. Confirme a ação.
2. Quando o aplicativo estiver em execução, atualize a página. O console mostra o Gráfico do aplicativo.

Pare o aplicativo

Na MyApplicationpágina, escolha Parar. Confirme a ação.

Atualizar o aplicativo

Usando o console, você pode atualizar as configurações do aplicativo, como propriedades do aplicativo, configurações de monitoramento e o local ou nome do arquivo do aplicativoJAR. Você

também pode recarregar o aplicativo a JAR partir do bucket do Amazon S3 se precisar atualizar o código do aplicativo.

Na MyApplicationpágina, escolha Configurar. Atualize as configurações do aplicativo e selecione Update (Atualizar).

Crie e execute o aplicativo (AWS CLI)

Nesta seção, você usa o AWS CLI para criar e executar o aplicativo Managed Service for Apache Flink. O Managed Service for Apache Flink usa o `kinesisanalyticsv2` AWS CLI comando para criar e interagir com o Managed Service for Apache Flink aplicativos.

Criar uma política de permissões

Primeiro, crie uma política de permissões com duas instruções: uma que concede permissões para a ação `read` no fluxo de origem, e outra que concede permissões para ações `write` no fluxo de destino. Em seguida, você anexa a política a uma IAM função (que você cria na próxima seção). Assim, ao assumir o perfil, o serviço Managed Service for Apache Flink terá as permissões necessárias para ler o fluxo de origem e gravar no fluxo de coleta.

Use o código a seguir para criar a política de permissões `KAReadSourceStreamWriteSinkStream`. Substitua *`username`* pelo nome de usuário que você usou para criar o bucket do Amazon S3 e armazenar o código do aplicativo. Substitua o ID da conta nos nomes de recursos da Amazon (ARNs) (*`012345678901`*) pelo ID da sua conta.

```
{
    "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 obter instruções sobre como criar uma política de permissões, consulte o Tutorial: Criar e anexar sua primeira política gerenciada pelo cliente no Guia IAM do usuário.

> **Note**
>
> Para acessar outros AWS serviços, você pode usar AWS SDK for Java o. O Managed Service for Apache Flink define automaticamente as credenciais exigidas pelo com SDK as da IAM função de execução do serviço associada ao seu aplicativo. Não é necessária nenhuma etapa adicional.

Criar uma IAM função

Nesta seção, você cria uma IAM função que o Managed Service for Apache Flink pode assumir para ler um fluxo de origem e gravar no stream do coletor.

O Managed Service for Apache Flink não pode acessar seu fluxo sem permissões. Você concede essas permissões por meio de uma IAM função. Cada IAM função tem duas políticas anexadas. A política de confiança concede ao Managed Service for Apache Flink permissão para assumir o perfil, e a política de permissões determina o que o serviço pode fazer depois de assumir a função.

Você anexa a política de permissões que criou na seção anterior a essa função.

Para criar uma função do IAM

1. Abra o IAM console em https://console.aws.amazon.com/iam/.

2. No painel de navegação, selecione Roles (Funções) e Create Role (Criar função).
3. Em Selecionar tipo de identidade de confiança, selecione Serviço da AWS . Em Choose the service that will use this role (Selecionar o serviço que usará esta função), selecione Kinesis. Em Select your use case (Selecionar seu caso de uso), selecione Kinesis Analytics.

   Selecione Next: Permissions (Próximo: permissões).
4. Na página Attach permissions policies, selecione Next: Review. Você pode anexar políticas de permissões depois de criar a função.
5. Na página Create role (Criar função), insira **`KA-stream-rw-role`** para o Role name (Nome da função). Selecione Criar função.

   Agora você criou uma nova IAM função chamada`KA-stream-rw-role`. Em seguida, você atualiza as políticas de confiança e de permissões para a função.
6. Anexe a política de permissões à função.

   > **Note**
   >
   > Para este exercício, o Managed Service for Apache Flink assume esse perfil para ler dados de um fluxo de dados do Kinesis (origem) e gravar a saída em outro fluxo de dados do Kinesis. Depois, você anexa a política que criou na etapa anterior, the section called “Criar uma política de permissões”.

   a. Na página Summary (Resumo), selecione a guia Permissions (Permissões).
   b. Selecione Attach Policies.
   c. Na caixa de pesquisa, insira **`KAReadSourceStreamWriteSinkStream`** (a política que você criou na seção anterior).
   d. Escolha a KAReadInputStreamWriteOutputStreampolítica e escolha Anexar política.

Agora você criou a função de execução de serviço que seu aplicativo usa para acessar os recursos. Anote ARN a nova função.

Para step-by-step obter instruções sobre como criar uma função, consulte Criação de uma IAM função (console) no Guia IAM do usuário.

Criar o aplicativo do Managed Service for Apache Flink

1. Salve o JSON código a seguir em um arquivo chamado`create_request.json`. Substitua ARN a função de ARN exemplo pela função que você criou anteriormente. Substitua o ARN sufixo do bucket (*`username`*) pelo sufixo que você escolheu na seção anterior. Substitua o ID da conta de exemplo (*`012345678901`*) na função de execução do serviço pelo ID da conta.

```
{
    "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. Execute a ação `CreateApplication` com a solicitação anterior para criar o aplicativo:

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

O aplicativo agora é criado. Você inicia o aplicativo na próxima etapa.

Iniciar o aplicativo

Nesta seção, você usa a ação `StartApplication` para iniciar o aplicativo.

Para iniciar o aplicativo

1. Salve o JSON código a seguir em um arquivo chamado`start_request.json`.

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

2. Execute a ação `StartApplication` com a solicitação anterior para iniciar o aplicativo:

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

O aplicativo agora está em execução. Você pode verificar as métricas do Managed Service for Apache Flink no CloudWatch console da Amazon para verificar se o aplicativo está funcionando.

Interromper o aplicativo

Nesta seção, você usa a ação `StopApplication` para interromper o aplicativo.

Como interromper o aplicativo

1. Salve o JSON código a seguir em um arquivo chamado`stop_request.json`.

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

2. Execute a ação StopApplication com a seguinte solicitação para interromper o aplicativo:

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

O aplicativo agora está interrompido.

Tutorial: Use AWS Lambda com o Amazon Kinesis Data Streams

Neste tutorial, você cria uma função do Lambda para consumir eventos de um fluxo de dados do Kinesis. Neste cenário de exemplo, um aplicativo personalizado grava registros em um stream de dados do Kinesis. AWS Lambda em seguida, pesquisa esse fluxo de dados e, quando detecta novos registros de dados, invoca sua função Lambda. AWS Lambda em seguida, executa a função Lambda assumindo a função de execução que você especificou ao criar a função Lambda.

Para obter instruções detalhadas passo a passo, consulte Tutorial: Usando o AWS Lambda com o Amazon Kinesis.

> **Note**
>
> Este tutorial pressupõe que você tenha algum conhecimento das operações básicas do Lambda e AWS Lambda do console. Se ainda não o fez, siga as instruções em Introdução ao AWS Lambda para criar sua primeira função do Lambda.

Use a solução AWS de streaming de dados para o Amazon Kinesis

A solução AWS de dados de streaming para Amazon Kinesis configura automaticamente os AWS serviços necessários para capturar, armazenar, processar e entregar dados de streaming com facilidade. A solução oferece várias opções para resolver casos de uso de dados de streaming que usam vários AWS serviços, incluindo Kinesis Data AWS Lambda Streams, API Amazon Gateway e Amazon Managed Service para Apache Flink.

Cada solução inclui os seguintes componentes:

- Um AWS CloudFormation pacote para implantar o exemplo completo.
- Um CloudWatch painel para exibir as métricas do aplicativo.
- CloudWatch alarmes sobre as métricas de aplicação mais relevantes.
- Todas as IAM funções e políticas necessárias.

A solução pode ser encontrada em Solução de dados de transmissão para o Amazon Kinesis

Crie e gerencie streams de dados do Kinesis

O Amazon Kinesis Data Streams ingere uma grande quantidade de dados em tempo real, armazena os dados de forma durável e os torna disponíveis para consumo. Um registro de dados é a unidade de dados armazenada pelo Kinesis Data Streams. Um stream de dados representa um grupo de registros de dados. Os registros de dados em um stream de dados são distribuídos em estilhaços.

Um estilhaço tem uma sequência de registros de dados em um streaming. Ele serve como uma unidade base de throughput em um fluxo de dados do Kinesis. Um fragmento oferece suporte a 1 MB/s e 1.000 registros por segundo para gravações e 2 MB/s para leituras nos modos de capacidade sob demanda e provisionada. Os limites de fragmentos garantem prever a performance, facilitando o design e a operação de um fluxo de trabalho de streaming de dados altamente confiável.

Nesta seção, você aprende como definir o modo de capacidade do stream e como criar um stream usando o AWS Management Console ouAPIs. Depois, você pode realizar ações adicionais no stream.

Tópicos

Escolha o modo de capacidade do fluxo de dados

Os tópicos a seguir explicam como escolher o modo de capacidade adequado para seu aplicativo e como alternar entre os modos de capacidade, se necessário.

Tópicos

O que é um modo de capacidade de fluxo de dados?

O modo de capacidade determina como a capacidade de um fluxo de dados é gerenciada e como são geradas cobranças pelo seu uso. Atualmente, no Amazon Kinesis Data Streams, você pode escolher entre os modos sob demanda e provisionado para os fluxos de dados.

- Sob demanda: os fluxos de dados no modo sob demanda não exigem planejamento de capacidade e escalam automaticamente para lidar com gigabytes de throughput de gravação e leitura por minuto. No modo sob demanda, o Kinesis Data Streams gerencia automaticamente os fragmentos para fornecer a throughput necessária.
- Provisionado: no modo provisionado, você precisa especificar o número de fragmentos para o fluxo de dados. A capacidade total de um fluxo de dados é a soma das capacidades de seus fragmentos. Você pode aumentar ou diminuir o número de fragmentos em um fluxo de dados de acordo com a necessidade.

Você pode usar o Kinesis Data `PutRecords` APIs Streams e gravar `PutRecord` dados em seus streams de dados nos modos de capacidade sob demanda e provisionada. Para recuperar dados, os dois modos de capacidade oferecem suporte aos consumidores padrão que usam o `GetRecords` API e aos consumidores Enhanced Fan-Out (EFO) que usam o. `SubscribeToShard` API

Todos os recursos do Kinesis Data Streams, incluindo modo de retenção, criptografia, métricas de monitoramento e outros, são compatíveis com os modos sob demanda e provisionado. O Kinesis Data Streams fornece alta durabilidade e disponibilidade nos modos de capacidade sob demanda e provisionada.

Recursos e casos de uso do modo sob demanda

Os fluxos de dados no modo sob demanda não exigem planejamento de capacidade e escalam automaticamente para lidar com gigabytes de throughput de gravação e leitura por minuto. O modo sob demanda simplifica a ingestão e o armazenamento de grandes volumes de dados com baixa latência, pois elimina o provisionamento e o gerenciamento de servidores, armazenamento ou throughput. Você pode ingerir bilhões de registros por dia sem nenhuma sobrecarga operacional.

O modo sob demanda é ideal para atender às necessidades de tráfego de aplicações altamente variável e imprevisível. Você não precisa mais provisionar essas cargas de trabalho na capacidade máxima, o que pode resultar em custos mais altos devido à baixa utilização. O modo sob demanda é adequado para cargas de trabalho com padrões de tráfego imprevisíveis e altamente variáveis.

Com o modo de capacidade sob demanda, você paga por GB de dados gravados e lidos em seus fluxos de dados. Você não precisa especificar o throughput de leitura e gravação que espera que a aplicação execute. O Kinesis Data Streams ajusta-se instantaneamente ao crescimento e à redução das workloads. Para ter mais informações, consulte Definição de preço do Amazon Kinesis Data Streams.

Um fluxo de dados no modo sob demanda acomoda até o dobro do pico de throughput de gravação observado nos 30 dias anteriores. Quando a throughput de gravação do fluxo de dados atinge um novo pico, o Kinesis Data Streams escala automaticamente a capacidade do fluxo de dados. Por exemplo, se a throughput de gravação do fluxo de dados variar entre 10 MB/s e 40 MB/s, o Kinesis Data Streams garantirá que seja possível expandir facilmente até o dobro da throughput máxima anterior, ou seja, 80 MB/s. Se o mesmo fluxo de dados tiver um novo pico de throughput de 50 MB/s, o Kinesis Data Streams garantirá capacidade suficiente para ingerir 100 MB/s de throughput de gravação. No entanto, poderá ocorrer controle de utilização se o tráfego aumentar para mais que o dobro do pico anterior em um período de 15 minutos. Você precisa repetir as solicitações em controle de utilização.

A capacidade de leitura de agregados de um fluxo de dados no modo sob demanda aumenta proporcionalmente com a throughput de gravação. Isso ajuda a garantir que as aplicações de consumo sempre tenham uma throughput de leitura adequada para processar os dados recebidos em tempo real. Você obtém pelo menos o dobro da taxa de transferência de gravação em comparação com os dados de leitura usando o `GetRecords` API Recomendamos que você use um aplicativo consumidor com o `GetRecord`API, para que ele tenha espaço suficiente para se atualizar quando o aplicativo precisar se recuperar do tempo de inatividade. É recomendável usar o recurso de distribuição avançada do Kinesis Data Streams em cenários que exijam a adição de mais de uma aplicação de consumo. O Enhanced Fan-Out suporta a adição de até 20 aplicativos de consumo a um fluxo de dados usando o `SubscribeToShard`API, com cada aplicativo de consumidor tendo uma taxa de transferência dedicada.

Lidar com exceções de taxa de transferência de leitura e gravação

Com o modo de capacidade sob demanda (da mesma forma que com a capacidade provisionada), você precisa especificar uma chave de partição com cada registro para gravar dados no fluxo. O Kinesis Data Streams usa suas chaves de partição para distribuir dados entre fragmentos. O Kinesis

Data Streams monitora o tráfego de cada fragmento. Quando o tráfego de entrada excede 500 KB/s por fragmento, o serviço divide o fragmento em 15 minutos. Os valores da chave de hash do fragmento pai são redistribuídos uniformemente entre os fragmentos filho.

Se o tráfego de entrada exceder o dobro do pico anterior, poderão ocorrer exceções de leitura ou gravação por cerca de 15 minutos, mesmo quando os dados forem distribuídos uniformemente entre os fragmentos. Recomendamos que você repita todas essas solicitações para que todos os registros sejam armazenados adequadamente no Kinesis Data Streams.

As exceções de leitura e gravação podem ocorrer quando você usa uma chave de partição que causa uma distribuição desigual de dados, e os registros atribuídos a um fragmento específico excedem seus limites. Com o modo sob demanda, o fluxo de dados se adaptará automaticamente para lidar com padrões desiguais de distribuição de dados, a menos que uma única chave de partição exceda os limites de throughput de 1 MB/s e 1.000 registros por segundo por fragmento.

No modo sob demanda, o Kinesis Data Streams divide os fragmentos uniformemente quando detecta um aumento no tráfego. No entanto, ele não detecta nem isola as chaves de hash que estão direcionando uma parte maior do tráfego de entrada para um fragmento específico. Se você estiver usando chaves de partição altamente desiguais, as exceções de gravação poderão continuar ocorrendo. Para esses casos de uso, é recomendável usar o modo de capacidade provisionada que oferece suporte a divisões granulares de fragmentos.

Recursos e casos de uso do modo provisionado

Com o modo provisionado, depois de criar o fluxo de dados, você pode aumentar ou reduzir dinamicamente a capacidade do fragmento usando o ou o. AWS Management Console UpdateShardCountAPI Você pode fazer atualizações enquanto uma aplicação de produção ou de consumo do Kinesis Data Streams grava ou lê dados do fluxo.

O modo provisionado é adequado para tráfego com requisitos de capacidade fáceis de prever. Você pode usar o modo provisionado quando quiser ter um controle refinado da distribuição dos entre os fragmentos.

No modo provisionado, você precisa especificar o número de fragmentos para o fluxo de dados. Para determinar o tamanho inicial de um fluxo de dados, você precisa dos seguintes valores de entrada:

- O tamanho médio do registro de dados gravado no fluxo em kilobytes (KB), arredondado para o 1 KB mais próximo (`average_data_size_in_KB`).
- O número de registros de dados gravados e lidos no stream por segundo (`records_per_second`).

- O número de consumidores, que são as aplicações do Kinesis Data Streams que consomem dados de forma simultânea e independente do fluxo (`number_of_consumers`).
- A largura de banda de gravação de entrada em KB (`incoming_write_bandwidth_in_KB`), que é igual a `average_data_size_in_KB` multiplicado por `records_per_second`.
- A largura de banda de leitura de saída em KB (`outgoing_read_bandwidth_in_KB`), que é igual a `incoming_write_bandwidth_in_KB` multiplicado por `number_of_consumers`.

Você pode calcular o número dos fragmentos (`number_of_shards`) necessários para o fluxo usando os valores de entrada na seguinte fórmula:

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

Ainda será possível ocorrer exceções de throughput de leitura e gravação no modo provisionado se você não configurar o fluxo de dados para lidar com a throughput máxima. Nesse caso, será preciso escalar manualmente o fluxo para acomodar o tráfego de dados.

As exceções de leitura e gravação também podem ocorrer quando você usa uma chave de partição que causa uma distribuição desigual de dados, e os registros atribuídos a um fragmento excedem seus limites. Para resolver esse problema no modo provisionado, identifique esses fragmentos e divida-os manualmente para acomodar melhor o tráfego. Para obter mais informações, consulte Resharding a Stream.

Alternar entre os modos de capacidade

Você pode alternar o modo de capacidade do fluxo de dados de sob demanda para provisionado ou vice-versa. Para cada fluxo de dados na conta da AWS , você pode alternar entre os modos de capacidade sob demanda e provisionada duas vezes a cada 24 horas.

Alternar entre os modos de capacidade de um fluxo de dados não causa nenhuma interrupção nas aplicações usando o fluxo. Você pode continuar gravando e lendo o fluxo de dados. Durante a operação de mudar o modo de capacidade, de sob demanda para provisionado ou vice-versa, o status do fluxo é definido como Atualizando. Você precisa esperar que o status do fluxo de dados passe a Ativo antes de tentar modificar suas propriedades novamente.

Quando você muda do modo de capacidade provisionada para o modo de capacidade sob demanda, o fluxo de dados retém inicialmente a quantidade de fragmentos que tinha antes da transição. A

partir desse momento, o Kinesis Data Streams monitora o tráfego de dados e escala a contagem de fragmentos do fluxo de dados sob demanda de acordo com a throughput de gravação.

Quando você muda do modo de capacidade sob demanda para o modo de capacidade provisionada, o fluxo de dados também retém inicialmente a quantidade de fragmentos que tinha antes da transição. Mas, a partir desse momento, você é responsável por monitorar e ajustar a contagem de fragmentos do fluxo de dados para acomodar a throughput de gravação da forma apropriada.

Crie um stream usando o AWS Management Console

Você pode criar um stream usando o console do Kinesis Data Streams, o API Kinesis Data Streams ou o (). AWS Command Line Interface AWS CLI

Para criar um stream de dados usando o console

1. Faça login no AWS Management Console e abra o console do Kinesis em https://console.aws.amazon.com /kinesis.
2. Na barra de navegação, expanda o seletor de região e escolha uma região.
3. Selecione Criar fluxo de dados.
4. Na página Criar stream do Kinesis, insira um nome para o fluxo de dados e escolha o modo de capacidade Sob demanda ou Provisionado. O modo Sob demanda está selecionado por padrão. Para obter mais informações, consulte Escolha o modo de capacidade do fluxo de dados.

   No modo sob demanda, você pode, em seguida, escolher Create stream do Kinesis para criar o fluxo de dados. No modo provisionado, você precisa especificar o número de fragmentos necessários e, em seguida, escolher Criar stream do Kinesis.

   Na página Kinesis streams (Streamings do Kinesis), o Status do streaming é Creating (Criando) enquanto o streaming está sendo criado. Quando o streaming estiver pronto para ser usado, o Status mudará para Active (Ativo).
5. Escolha o nome do fluxo. A página Stream Details (Detalhes do streaming) exibe um resumo da configuração do streaming com informações de monitoramento.

Para criar um stream usando o Kinesis Data Streams API

- Para obter informações sobre como criar um stream usando o Kinesis API Data StreamsCrie um fluxo usando o APIs, consulte.

Para criar um stream usando o AWS CLI

- Para obter informações sobre como criar um stream usando o AWS CLI, consulte o comando create-stream.

Crie um fluxo usando o APIs

Use as etapas a seguir para criar o fluxo de dados do Kinesis.

Crie o cliente Kinesis Data Streams

Para trabalhar com fluxos de dados do Kinesis, você precisa criar um objeto de cliente. O seguinte código Java cria uma instância de um criador de cliente e a usa para definir a região, as credenciais e a configuração do cliente. Em seguida, ele cria um objeto do cliente.

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

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

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

Para obter mais informações, consulte Kinesis Data Streams Regions and Endpoints na Referência geral da AWS.

Crie o stream

Agora que você criou seu cliente Kinesis Data Streams, você pode criar um stream usando o console ou programaticamente. Para criar um fluxo programaticamente, instancie um `CreateStreamRequest` objeto e especifique um nome para o fluxo. Se você quiser usar o modo provisionado, especifique o número de fragmentos a serem usados pelo stream.

- Sob demanda:

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

- Provisionado:

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

O nome do stream identifica o stream. O nome tem como escopo a AWS conta usada pelo aplicativo. Ele também é delimitado por região. Ou seja, dois fluxos em duas AWS contas diferentes podem ter o mesmo nome, e dois fluxos na mesma AWS conta, mas em duas regiões diferentes, podem ter o mesmo nome, mas não dois fluxos na mesma conta e na mesma região.

A taxa de transferência do fluxo é uma função do número de fragmentos. Para uma maior taxa de transferência provisionada, você precisa de mais fragmentos. Mais fragmentos também aumentam o custo AWS cobrado pela transmissão. Para obter mais informações sobre como calcular um número apropriado de estilhaços para o aplicativo, consulte Escolha o modo de capacidade do fluxo de dados.

Depois de configurar o `createStreamRequest` objeto, crie um fluxo chamando o `createStream` método no cliente. Após chamar `createStream`, aguarde o stream alcançar o estado `ACTIVE` antes de executar qualquer operação nele. Para verificar o estado do stream, chame o método `describeStream`. Se o stream não existir, `describeStream` lançará uma exceção, portanto, coloque a chamada a `describeStream` em um bloco `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" );
}
```

Atualizar um stream

Você pode atualizar os detalhes de um stream usando o console do Kinesis Data Streams, o API Kinesis Data Streams ou o. AWS CLI

> **Note**
>
> Você pode ativar a criptografia no lado do servidor para streams existentes ou para os recém-criados.

Usar o console do

Para atualizar um stream de dados usando o console

1. Abra o console do Amazon Kinesis em. https://console.aws.amazon.com/kinesis/
2. Na barra de navegação, expanda o seletor de região e escolha uma região.
3. Escolha o nome do streaming na lista. A página Stream Details (Detalhes do streaming) exibe um resumo das informações de configuração e monitoramento do streaming.
4. Para alternar entre os modos de capacidade sob demanda e provisionada para um fluxo de dados, escolha Editar modo de capacidade na guia Configuração. Para obter mais informações, consulte Escolha o modo de capacidade do fluxo de dados.

> **Important**
>
> Para cada fluxo de dados em sua AWS conta, você pode alternar entre os modos sob demanda e provisionado duas vezes em 24 horas.

5. Para editar o número de fragmentos de um fluxo de dados no modo provisionado, escolha Editar fragmentos provisionados na guia Configuração e, em seguida, insira uma nova contagem de fragmentos.
6. Para ativar a criptografia de registros de dados no lado do servidor, selecione Edit (Editar) na seção Server-side encryption (Criptografia no lado do servidor). Escolha uma KMS chave para usar como chave mestra para criptografia ou use a chave mestra padrão, aws/kinesis, gerenciada pelo Kinesis. Se você habilitar a criptografia para um stream e usar sua própria chave AWS KMS mestra, certifique-se de que seus aplicativos de produtor e consumidor tenham acesso à chave AWS KMS mestra que você usou. Para atribuir permissões a um aplicativo para acessar uma chave do AWS KMS gerada pelo usuário, consulte the section called “Permissões para usar chaves geradas pelo usuário KMS”.
7. Para editar o período de retenção de dados, selecione Edit (Editar) na seção Data retention period (Período de retenção de dados) e insira um novo período de retenção de dados.
8. Se você tiver habilitado métricas personalizadas em sua conta, selecione Edit (Editar) na seção Shard level metrics (Métricas em nível de estilhaço) e, em seguida, especifique as métricas de seu streaming. Para obter mais informações, consulte the section called “Monitore o serviço Kinesis Data Streams com CloudWatch”.

Use o API

Para atualizar os detalhes do stream usando oAPI, consulte os seguintes métodos:

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

- StopStreamEncryption
- UpdateShardCount

Use o AWS CLI

Para obter informações sobre como atualizar um stream usando o AWS CLI, consulte a referência do Kinesis CLI.

Listar streams

Os streams têm como escopo a AWS conta associada às AWS credenciais usadas para instanciar o cliente Kinesis Data Streams e também a região especificada para o cliente. Uma conta da AWS pode ter vários fluxos ativos ao mesmo tempo. Você pode listar os fluxos no console do Kinesis Data Streams ou de forma programática. O código nesta seção mostra como listar todos os streams da sua AWS conta.

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

Este exemplo de código primeiro cria uma nova instância de `ListStreamsRequest` e chama seu método `setLimit` para especificar que um máximo de 20 streams devem ser retornados para cada chamada a `listStreams`. Se você não especificar um valor para `setLimit`, o Kinesis Data Streams retornará um número de fluxos menor ou igual ao número na conta. Em seguida, o código passa `listStreamsRequest` ao método `listStreams` do cliente. O valor de retorno `listStreams` é armazenado em um objeto `ListStreamsResult`. O código chama o método `getStreamNames` para esse objeto e armazena os nomes de stream retornados na lista `streamNames`. Observe que o Kinesis Data Streams pode retornar menos fluxos do que o limite especificado, mesmo quando houver um número maior de fluxos na conta e na região. Para garantir que você recupere todos os streams, use o método `getHasMoreStreams` como descrito no próximo exemplo de código.

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

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

Esse código chama o método `getHasMoreStreams` para `listStreamsRequest` a fim de verificar se há streams adicionais disponíveis além dos que foram retornados na chamada inicial a `listStreams`. Se houver, o código chamará o método `setExclusiveStartStreamName` com o nome do último stream que foi retornado na chamada anterior a `listStreams`. O método `setExclusiveStartStreamName` faz com que a próxima chamada a `listStreams` comece depois desse stream. Em seguida, o grupo de nomes de stream retornados pela chamada é adicionado à lista `streamNames`. Esse processo continua até que todos os nomes de stream tenham sido coletados na lista.

Os streams retornados por `listStreams` podem estar em um dos seguintes estados:

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

Você pode verificar o estado de um stream usando o método `describeStream`, como mostrado na seção anterior, Crie um fluxo usando o APIs.

Listar fragmentos

Um fluxo de dados pode ter um ou mais fragmentos. O método recomendado para listar ou recuperar os fragmentos de um fluxo de dados é usar o. ListShardsAPI O exemplo a seguir mostra como obter uma lista de fragmentos em um fluxo de dados. Para obter uma descrição completa da operação principal usada neste exemplo e de todos os parâmetros que você pode definir para a operação, 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 executar o exemplo de código anterior, você pode usar um POM arquivo como o seguinte.

```
<?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>
```

Com o `ListShards`API, você pode usar o ShardFilterparâmetro para filtrar a resposta doAPI. Só é possível especificar um filtro de cada vez.

Se você usar o `ShardFilter` parâmetro ao invocar o ListShardsAPI, a `Type` é a propriedade necessária e deve ser especificada. Se você especificar os tipos `AT_TRIM_HORIZON`, `FROM_TRIM_HORIZON` ou `AT_LATEST`, não precisará especificar as propriedades opcionais `ShardId` e `Timestamp`.

Se você especificar o tipo `AFTER_SHARD_ID`, também deverá fornecer o valor para a propriedade opcional `ShardId`. A `ShardId` propriedade é idêntica em termos de funcionalidade ao `ExclusiveStartShardId` parâmetro do ListShards API. Quando a propriedade `ShardId` é especificada, a resposta inclui os fragmentos a partir daquele cuja ID segue imediatamente a `ShardId` fornecida.

Se você especificar o tipo `AT_TIMESTAMP` ou `FROM_TIMESTAMP_ID`, também deverá fornecer o valor para a propriedade opcional `Timestamp`. Se você especificar o tipo `AT_TIMESTAMP`, todos os fragmentos abertos no timestamp fornecido serão retornados. Se você especificar o `FROM_TIMESTAMP` tipo, todos os fragmentos a partir do carimbo de data/hora fornecido serão retornadosTIP.

> **Important**
>
> `DescribeStreamSummarye` `ListShard` APIs forneça uma forma mais escalável de recuperar informações sobre seus fluxos de dados. Mais especificamente, as cotas do DescribeStream API podem causar limitação. Para obter mais informações, consulte Cotas e limites. Observe também que as `DescribeStream` cotas são compartilhadas em todos os aplicativos que interagem com todos os fluxos de dados em sua AWS conta. As cotas para o ListShards API, por outro lado, são específicas para um único fluxo de dados. Portanto,

você não apenas aumenta TPS com o ListShards API, mas a ação é melhor dimensionada à medida que você cria mais fluxos de dados.
Recomendamos que você migre todos os produtores e consumidores que ligam DescribeStream API para o para, em vez disso, invocar o DescribeStreamSummary e o. ListShard APIs Para identificar esses produtores e consumidores, recomendamos usar o Athena para analisar CloudTrail registros como agentes de usuário KPL e KCL capturados nas chamadas. 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
```

Também recomendamos que as integrações do AWS Lambda e do Amazon Firehose com o Kinesis Data Streams que invocam o sejam reconfiguradas para que, em vez disso, as integrações invoquem `DescribeStream` API e. `DescribeStreamSummary` `ListShards` Especificamente, para o AWS Lambda, você deve atualizar o mapeamento da fonte do evento. Para o Amazon Firehose, as IAM permissões correspondentes devem ser atualizadas para que incluam a `ListShards` IAM permissão.

Excluir um stream

Você pode excluir um fluxo no console do Kinesis Data Streams ou de forma programática. Para excluir um streaming de maneira programática, use `DeleteStreamRequest`, conforme mostrado no código a seguir.

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

Desative todos os aplicativos que estejam operando no streaming antes de excluí-lo. Se um aplicativo tentar operar em um stream excluído, ele receberá exceções `ResourceNotFound`. Além disso, se você criar um novo streaming com o mesmo nome do streaming anterior e os aplicativos que operavam nele ainda estiverem em execução, esses aplicativos poderão tentar interagir com o streaming novo como se fosse o anterior, com resultados imprevisíveis.

Refragmentar um stream

> **Important**
>
> Você pode refragmentar seu stream usando o. UpdateShardCountAPI Caso contrário, é possível continuar executando divisões e mesclagens, como explicado aqui.

O Amazon Kinesis Data Streams oferece suporte à refragmentação, o que permite ajustar o número de fragmentos no fluxo para se adaptar a alterações na taxa de dados no fluxo. O reestilhaçamento é considerado uma operação avançada. Se você estiver começando com o Kinesis Data Streams, volte a este tópico depois de se familiarizar com todos os outros aspectos do serviço.

Há dois tipos de operações de reestilhaçamento: divisão de estilhaço e mesclagem de estilhaço. Na divisão de estilhaço, um único estilhaço é dividido em dois. Na mesclagem de estilhaço, você combina dois estilhaços em um. O reestilhaçamento sempre ocorre em pares, ou seja, não é possível dividir em mais de dois estilhaços em uma única operação, e não é possível mesclar mais de dois estilhaços em uma única operação. O estilhaço (ou o par de estilhaços) que é objeto da operação de reestilhaçamento é chamado de estilhaço pai. O estilhaço (ou o par de estilhaços) resultante da operação de novo estilhaçamento é chamado de estilhaço filho.

A divisão aumenta o número de estilhaços no stream e, portanto, aumenta a capacidade de dados do stream. Como você é cobrado por estilhaço, divisão aumenta o custo do seu stream. Comparativamente, a mesclagem reduz o número de estilhaços no stream e, portanto, diminui a capacidade de dados e o custo do stream.

O reestilhaçamento costuma ser executado por um aplicativo administrativo, que é diferente dos aplicativos de produtor (put) e dos aplicativos de consumidor (get). Esse aplicativo administrativo monitora o desempenho geral do stream com base nas métricas fornecidas pela Amazon CloudWatch ou com base nas métricas coletadas dos produtores e consumidores. O aplicativo administrativo também precisa de um conjunto mais amplo de IAM permissões do que os consumidores ou produtores, porque os consumidores e produtores geralmente não precisam

acessar o APIs usado para refragmentação. Para obter mais informações sobre IAM permissões para o Kinesis Data StreamsControle do acesso aos recursos do Amazon Kinesis Data Streams usando IAM, consulte.

Para obter mais informações sobre refragmentação, consulte How do I change the number of open shards in Kinesis Data Streams?

Tópicos

Decida uma estratégia para refragmentar

A finalidade da refragmentação no Amazon Kinesis Data Streams é permitir que o fluxo se adapte a alterações na taxa do fluxo de dados. Você divide estilhaços para aumentar a capacidade (e o custo) do seu stream. Você mescla estilhaços para reduzir o custo (e a capacidade) do seu stream.

Uma abordagem de refragmentação pode ser dividir cada fragmento do fluxo, o que dobraria sua capacidade. No entanto, isso pode fornecer mais capacidade adicional do que você realmente precisa e, portanto, gerar um custo desnecessário.

Você também pode usar métricas para identificar os fragmentos quentes ou frios, ou seja, os fragmentos que estão recebendo muito mais ou muito menos dados do que o esperado. Em seguida, você pode seletivamente dividir os estilhaços quentes para aumentar a capacidade das chaves de hash que almejam esses estilhaços. Comparativamente, você pode mesclar os estilhaços frios para dar uma melhor serventia à capacidade não usada.

Você pode obter alguns dados de desempenho do seu stream a partir das CloudWatch métricas da Amazon que o Kinesis Data Streams publica. No entanto, você mesmo também pode coletar algumas métricas dos seus streams. Uma abordagem é registrar em log os valores de chave de hash gerados pelas chaves de partição dos seus registros de dados. Lembre-se de que você especifica a chave de partição no momento em que adiciona o registro ao stream.

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

O Kinesis Data MD5Streams usa para calcular a chave de hash a partir da chave de partição. Como você especifica a chave de partição para o registro, você pode MD5 usá-la para calcular o valor da chave de hash desse registro e registrá-lo.

Você também pode registrar os IDs fragmentos aos quais seus registros de dados estão atribuídos. O ID do estilhaço é obtido usando-se o método `getShardId` do objeto `putRecordResults` retornado pelo método `putRecords` e o objeto `putRecordResult` retornado pelo método `putRecord`.

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

Com os valores do fragmento IDs e da chave de hash, você pode determinar quais fragmentos e chaves de hash estão recebendo mais ou menos tráfego. Em seguida, você pode usar o reestilhaçamento para fornecer mais ou menos capacidade, conforme apropriado para essas chaves.

Divida um fragmento

Para dividir um fragmento no Amazon Kinesis Data Streams, você precisa especificar como os valores de chave de hash dos fragmentos pai devem ser redistribuídos para os fragmentos filho. Quando você adiciona um registro de dados a um stream, ele é atribuído a um estilhaço com base em um valor de chave de hash. O valor da chave de hash é o MD5hash da chave de partição que você especifica para o registro de dados no momento em que adiciona o registro de dados ao fluxo. Os registros de dados que têm a mesma chave de partição também têm o mesmo valor de chave de hash.

Os valores possíveis de chave de hash de um determinado estilhaço constituem um conjunto de números inteiros contíguos, não negativos e ordenados. Esse intervalo de possíveis valores de chave de hash é determinado pelo seguinte:

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

Quando divide o estilhaço, você especifica um valor neste intervalo. Esse valor de chave de hash e todos os valores de chave de hash maiores são distribuídos para um dos estilhaços filhos. Todos os valores de chave de hash menores são distribuídos para os outros estilhaços filhos.

O seguinte código demonstra uma operação de divisão de estilhaço que redistribui as chaves de hash uniformemente entre cada um dos estilhaços filhos, basicamente, dividindo o estilhaço pai

no meio. Essa é apenas uma das maneiras possíveis de dividir o estilhaço pai. Você pode, por exemplo, dividir o estilhaço de maneira que o terço inferior das chaves do pai vá para um estilhaço filho e os dois terços superiores das chaves vão para o outro estilhaço filho. No entanto, para muitos aplicativos, dividir os estilhaços no meio é uma abordagem eficiente.

O código pressupõe que `myStreamName` contém o nome do seu stream e a variável de objeto `shard` contém o estilhaço a dividir. Comece instanciando um novo objeto `splitShardRequest` e definindo o nome do stream e o ID do estilhaço.

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

Determine o valor da chave de hash que é meio caminho entre o maior valor e o menor valor no estilhaço. Trata-se do valor de chave de hash inicial para o estilhaço filho que conterá a metade superior das chaves de hash do estilhaço pai. Especifique esse valor no método `setNewStartingHashKey`. Você só precisa especificar esse valor. O Kinesis Data Streams distribui automaticamente as chaves de hash abaixo desse valor para os outros fragmentos filho criados pela divisão. A última etapa é chamar o método `splitShard` no cliente do 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);
```

A primeira etapa após este procedimento é mostrada em Aguarde até que um stream se torne ativo novamente.

Mesclar dois fragmentos

Uma operação de mesclagem de estilhaços usa dois estilhaços especificados e combina-os em um único estilhaço. Após a mesclagem, o único estilhaço filho recebe dados para todos os valores de chave de hash cobertos pelos dois estilhaços pais.

Adjacência de estilhaço

Para mesclar dois estilhaços, eles precisam estar adjacentes. Dois estilhaços são considerados adjacentes quando a união dos intervalos de chave de hash dos dois estilhaços forma um conjunto contíguo sem lacunas. Por exemplo, suponha que você tenha dois estilhaços, um com o intervalo de chaves de hash 276...381 e o outro com o intervalo de chaves de hash 382...454. Você pode mesclar esses dois estilhaços em um só, que teria um intervalo de chaves de hash 276...454.

Para usar outro exemplo, suponha que você tenha dois estilhaços, um com um intervalo de chaves de hash 276...381 e o outro com um intervalo de chaves de hash 455...560. Você não poderia mesclar esses dois estilhaços porque haveria um ou mais estilhaços entre eles que estariam no intervalo 382...454.

O conjunto de todos os `OPEN` fragmentos em um fluxo, como um grupo, sempre abrange toda a faixa de valores da chave de hash. MD5 Para obter mais informações sobre esses estados de estilhaço (como `CLOSED`), consulte Considere o roteamento de dados, a persistência de dados e o estado do fragmento após uma nova fragmentação.

Para identificar os estilhaços candidatos para mesclagem, você deve filtrar todos os estilhaços que estão no estado `CLOSED`. Os fragmentos `OPEN` (ou seja, não `CLOSED`) têm um número de sequência final `null`. Você pode testar o número sequencial de término de um estilhaço usando:

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

Após filtrar os estilhaços fechados, classifique os estilhaços restantes pelo valor de chave de hash mais alto aceito por cada estilhaço. Você pode recuperar esse valor usando:

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

Se dois estilhaços são adjacentes nessa lista filtrada e classificada, eles podem ser mesclados.

Código da operação de mesclagem

O código a seguir mescla dois estilhaços. O código pressupõe que `myStreamName` contém o nome do seu stream e as variáveis de objeto `shard1` e `shard2` contém os dois estilhaços adjacentes a mesclar.

Para a operação de mesclagem, comece instanciando um novo objeto `mergeShardsRequest`. Especifique o nome do stream com o método `setStreamName`. Em seguida, especifique os dois estilhaços a mesclar usando os métodos `setShardToMerge` e `setAdjacentShardToMerge`. Por fim, chame o método `mergeShards` no cliente do Kinesis Data Streams para executar a operação.

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

A primeira etapa após este procedimento é mostrada em Aguarde até que um stream se torne ativo novamente.

Conclua a ação de refragmentação

Após qualquer tipo de procedimento de refragmentação no Amazon Kinesis Data Streams e antes de retomar o processamento normal de registros, é necessário realizar outros procedimentos e fazer algumas considerações. As seções a seguir descrevem esses itens.

Tópicos

- Aguarde até que um stream se torne ativo novamente
- Considere o roteamento de dados, a persistência de dados e o estado do fragmento após uma nova fragmentação

Aguarde até que um stream se torne ativo novamente

Depois de chamar uma operação de refragmentaçãoMergeShards, você deve esperar que o stream fique ativo novamente. `splitShard` O código a ser usado é o mesmo de quando você espera que um streaming se torne ativo após a criação de um streaming. Esse código é o seguinte:

```
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" );
}
```

Considere o roteamento de dados, a persistência de dados e o estado do fragmento após uma nova fragmentação

O Kinesis Data Streams é um serviço de streaming de dados em tempo real. Seus aplicativos devem presumir que os dados estão fluindo continuamente pelos fragmentos do seu stream. Quando você reestilhaça, os registros de dados que estavam fluindo para os estilhaços pais são re-roteados para fluírem para os estilhaços filhos com base nos valores de chave de hash para as quais são mapeadas as chaves de partição de registro de dados. No entanto, os registros de dados que estavam nos estilhaços pais antes do reestilhaçamento permanecem nesses estilhaços. Os fragmentos originais não desaparecem quando a nova fragmentação ocorre. Eles persistem com os dados que continham antes do reestilhaçamento. Os registros de dados nos fragmentos principais podem ser acessados usando as `getRecords` operações `getShardIteratore` no Kinesis Data API Streams ou por meio da Kinesis Client Library.

> **Note**
>
> Os registros de dados podem ser acessados a partir do momento em que são adicionados ao stream até o período de retenção atual. Isso é verdadeiro independentemente de quaisquer alterações aos estilhaços no stream durante esse período. Para obter mais informações sobre o período de retenção de um stream, consulte Alterar o período de retenção de dados.

No processo de reestilhaçamento, um estilhaço pai passa de um estado `OPEN` para um estado `CLOSED` para um estado `EXPIRED`.

- OPEN: antes de uma operação de refragmentação, um fragmento principal está no `OPEN` estado, o que significa que os registros de dados podem ser adicionados ao fragmento e recuperados do fragmento.
- CLOSED: após uma operação de refragmentação, o fragmento principal passa para um estado. `CLOSED` Isso significa que os registros de dados não são mais adicionados ao estilhaço. Os registros de dados que foram adicionados a esse estilhaço agora são adicionados a um estilhaço filho. No entanto, os registros de dados ainda podem ser recuperados do estilhaço por tempo limitado.
- EXPIRED: após o término do período de retenção do stream, todos os registros de dados no fragmento principal expiram e não estão mais acessíveis. Neste momento, o próprio estilhaço muda para um estado `EXPIRED`. As chamadas a `getStreamDescription().getShards` para enumerar os estilhaços no stream não incluem os estilhaços `EXPIRED` na lista de estilhaços retornados. Para obter mais informações sobre o período de retenção de um stream, consulte Alterar o período de retenção de dados.

Depois que o reestilhaçamento ocorreu e o stream está novamente em um estado `ACTIVE`, você pode iniciar imediatamente a ler dados dos estilhaços filhos. No entanto, os fragmentos principais que permanecem após a refragmentação ainda podem conter dados que você ainda não leu e que foram adicionados ao stream antes da refragmentação. Se você ler dados de estilhaços filhos antes ler todos os dados dos estilhaços pais, poderá ler dados de uma determinada chave de hash fora da ordem determinada pelos números sequenciais dos registros de dados. Portanto, pressupondo que a ordem dos dados é importante, você deve, após um reestilhaçamento, sempre continuar lendo dados dos estilhaços pais até esgotá-los. Só depois você deverá começar a ler dados dos estilhaços filhos. Quando `getRecordsResult.getNextShardIterator` retorna `null`, indica que você leu todos

os dados no estilhaço pai. Se você estiver lendo dados usando a Biblioteca de cliente do Kinesis, talvez você não receba os dados em ordem após a ocorrência de uma nova fragmentação.

Alterar o período de retenção de dados

O Amazon Kinesis Data Streams oferece suporte a alterações do período de retenção do registro de dados no fluxo de dados. Um fluxo de dados do Kinesis é uma sequência ordenada de registros de dados projetada para gravação e leitura em tempo real. Os registros de dados são, portanto, armazenados em estilhaços no seu stream temporariamente. O período entre o momento de adição de um registro e o momento em que ele deixa de estar acessível é chamado de período de retenção. Por padrão, um fluxo de dados do Kinesis armazena registros de 24 horas até 8.760 horas (365 dias).

Você pode atualizar o período de retenção por meio do console do Kinesis Data Streams ou IncreaseStreamRetentionPeriodusando as DecreaseStreamRetentionPeriodoperações e. No console do Kinesis Data Streams, você pode editar em massa o período de retenção de mais de um fluxo de dados ao mesmo tempo. Você pode aumentar o período de retenção até um máximo de 8760 horas (365 dias) usando a IncreaseStreamRetentionPeriodoperação ou o console do Kinesis Data Streams. Você pode reduzir o período de retenção para um mínimo de 24 horas usando a DecreaseStreamRetentionPeriodoperação ou o console do Kinesis Data Streams. A sintaxe de solicitação das duas operações inclui o nome do stream e o período de retenção em horas. Por fim, você pode verificar o período de retenção atual de um stream chamando a DescribeStreamoperação.

Este é um exemplo de alteração do período de retenção que usa a AWS CLI:

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

O Kinesis Data Streams para de tornar inacessíveis os registros no período de retenção antigo vários minutos após o aumento do período. Por exemplo, alterar o período de retenção de 24 horas para 48 horas significa que os registros adicionados ao streaming 23 horas 55 minutos antes ainda estarão disponíveis depois de 24 horas.

O Kinesis Data Streams torna inacessíveis os registros mais antigos que o novo período de retenção quase imediatamente após a diminuição do período. Portanto, tome muito cuidado ao chamar a DecreaseStreamRetentionPeriodoperação.

Defina o período de retenção dos dados para garantir que os consumidores possam ler dados antes de expirar, se ocorrerem problemas. Você deve considerar cuidadosamente todas as possibilidades,

como um problema com a lógica de processamento de registro ou a inatividade de uma dependência de downstream por um longo período. Ideia do período de retenção como uma rede de segurança para dar mais tempo para os consumidores de dados se recuperarem. As API operações do período de retenção permitem que você configure isso de forma proativa ou responda aos eventos operacionais de forma reativa.

Encargos adicionais incidem sobre streams com período de retenção definido acima de 24 horas. Para obter mais informações, consulte Definição de preço do Amazon Data Kinesis Streams.

Marque seus streams no Amazon Kinesis Data Streams

Você pode atribuir os próprios metadados aos fluxos que cria no Amazon Kinesis Data Streams na forma de tags. Tag é um par de chave-valor que você define para um stream. Usar tags é uma maneira simples, porém poderosa, de gerenciar AWS recursos e organizar dados, incluindo dados de faturamento.

Conteúdo

Princípios básicos da tag de revisão

Você usa o AWS CLI console do Kinesis Data Streams ou o API Kinesis Data Streams para concluir as seguintes tarefas:

- Adicionar tags a um stream
- Listar as tags para os streams
- Remover tags de um stream

Você pode usar tags para categorizar seus streams. Por exemplo, você pode categorizar streams por finalidade, proprietário ou ambiente. Como você define a chave e o valor para cada marca,

você pode criar um conjunto de categorias personalizado para atender às suas necessidades específicas. Por exemplo, você pode definir um conjunto de tags que ajude a monitorar os streams por proprietário e aplicativo associado. Aqui estão alguns exemplos de marcas:

- Projeto: nome do projeto
- Proprietário: nome
- Objetivo: testes de carga
- Aplicação: nome da aplicação
- Ambiente: produção

Monitore os custos usando a marcação

Você pode usar tags para categorizar e monitorar seus AWS custos. Quando você aplica tags aos seus AWS recursos, incluindo fluxos, seu relatório de alocação de AWS custos inclui o uso e os custos agregados por tags. É possível aplicar tags que representem categorias de negócios (como centros de custos, nomes de aplicações ou proprietários) para organizar seus custos de vários serviços. Para obter mais informações, consulte Usar etiquetas de alocação de custos para relatórios de faturamento personalizados no Manual do usuário do AWS Billing .

Entenda as restrições de tags

As restrições a seguir se aplicam a marcas.

Restrições básicas

- O número máximo de tags por recurso (stream) é 50.
- As chaves e os valores de tags diferenciam maiúsculas de minúsculas.
- Você não pode alterar nem editar as tags de um stream excluído.

Restrições de chaves de marcas

- Cada chave de marca deve ser exclusiva. Se você adicionar uma marca com uma chave que já estiver em uso, sua nova marca existente substituirá o par de chave-valor.
- Você não pode iniciar uma chave de tag com `aws:` porque esse prefixo é reservado para uso por AWS. AWS cria tags que começam com esse prefixo em seu nome, mas você não pode editá-las nem excluí-las.

- As chaves de marca devem ter entre 1 e 128 caracteres Unicode.
- As chaves de marca devem conter os seguintes caracteres: letras Unicode, dígitos, espaço em branco e os seguintes caracteres especiais: `_ . / = + - @`.

Restrições de valor de marcas

- Os valores de marca devem ter entre 0 e 255 caracteres Unicode.
- Os valores de marca podem estar em branco. Caso contrário, elas devem conter os seguintes caracteres: letras Unicode, dígitos, espaço em branco e qualquer um dos seguintes caracteres especiais: `_ . / = + - @`.

Marque streams usando o console do Kinesis Data Streams

Você pode adicionar, listar e remover tags usando o console do Kinesis Data Streams.

Para visualizar as tags de um stream

1. Abra o console do Kinesis Data Streams. Na barra de navegação, expanda o seletor de região e selecione uma região.
2. Na página Stream List (Lista de streamings), selecione um streaming.
3. Na página Detalhes do stream, escolha a guia Tags.

Para adicionar uma tag a um stream

1. Abra o console do Kinesis Data Streams. Na barra de navegação, expanda o seletor de região e selecione uma região.
2. Na página Stream List (Lista de streamings), selecione um streaming.
3. Na página Detalhes do stream, escolha a guia Tags.
4. Especifique a chave de tag no campo Chave, opcionalmente especifique um valor de tag no campo Valor e escolha Adicionar tag.

   Se o botão Add Tag (Adicionar tag) não estiver habilitado, a chave ou o valor da tag que você especificou não atenderá as restrições de tag. Para ter mais informações, consulte Entenda as restrições de tags.
5. Para ver sua nova tag na lista na guia Tags, escolha o ícone de atualização.

Para remover uma tag de um stream

1. Abra o console do Kinesis Data Streams. Na barra de navegação, expanda o seletor de região e selecione uma região.
2. Na página Stream List, selecione um stream.
3. Na página Detalhes do stream, escolha a guia Tags e, em seguida, escolha o ícone Remover para a tag.
4. Na caixa de diálogo Excluir tag, escolha Sim, excluir.

Marque fluxos usando o AWS CLI

Você pode adicionar, listar e remover tags usando a AWS CLI. Para obter exemplos, consulte a seguinte documentação.

add-tags-to-stream

Adiciona ou atualiza as tags para o stream especificado.

list-tags-for-stream

Lista as tags para o stream especificado.

remove-tags-from-stream

Remove as tags do stream especificado.

Marque streams usando o Kinesis Data Streams API

Você pode adicionar, listar e remover tags usando o Kinesis API Data Streams. Para obter exemplos, consulte a seguinte documentação:

AddTagsToStream

Adiciona ou atualiza as tags para o stream especificado.

ListTagsForStream

Lista as tags para o stream especificado.

RemoveTagsFromStream

Remove as tags do stream especificado.

Grave dados no Amazon Kinesis Data Streams

Um produtor é uma aplicação que grava dados no Amazon Kinesis Data Streams. Você pode criar produtores para o Kinesis Data Streams AWS SDK for Java usando a e a Kinesis Producer Library (). KPL

Se você nunca usou o Kinesis Data Streams, comece familiarizando-se com os conceitos e a terminologia apresentados em O que é o Amazon Kinesis Data Streams? e Use o AWS CLI para realizar operações do Amazon Kinesis Data Streams.

> **Important**
>
> O Kinesis Data Streams oferece suporte a alterações do período de retenção do registro de dados no fluxo de dados. Para ter mais informações, consulte Alterar o período de retenção de dados.

Para colocar dados no stream, é necessário especificar o nome do stream, uma chave de partição e o blob de dados que serão adicionados ao stream. A chave de partição é usada para determinar em que estilhaço do stream o registro de dados será adicionado.

Todos os dados no estilhaço são enviados para o mesmo operador que está processando o estilhaço. A chave de partição que você usa depende da lógica do aplicativo. O número de chaves de partição normalmente deve ser muito maior que o número de estilhaços. Isso ocorre porque a chave de partição é usada para determinar como mapear um registro de dados para um determinado estilhaço. Se você tiver um número suficiente de chaves de partição, os dados podem ser distribuídos uniformemente pelos estilhaços de um stream.

Tópicos

Desenvolva produtores usando a Amazon Kinesis Producer Library () KPL

Um produtor do Amazon Kinesis Data Streams é uma aplicação que coloca registros de dados de usuários em um fluxo de dados do Kinesis (o que também é chamado de ingestão de dados). A Kinesis Producer Library (KPL) simplifica o desenvolvimento de aplicativos para produtores, permitindo que os desenvolvedores alcancem uma alta taxa de transferência de gravação em um stream de dados do Kinesis.

Você pode monitorá-los KPL com a Amazon CloudWatch. Para ter mais informações, consulte Monitore a Kinesis Producer Library com a Amazon CloudWatch.

Tópicos

> **Note**
>
> É recomendável que você atualize para a KPL versão mais recente. KPLé atualizado regularmente com versões mais recentes que incluem os patches de dependência e

segurança mais recentes, correções de bugs e novos recursos compatíveis com versões anteriores. Para obter mais informações, consulte https://github.com/awslabs/amazon-kinesis-producer/releases/.

Analise o papel do KPL

KPLÉ uma easy-to-use biblioteca altamente configurável que ajuda você a gravar em um stream de dados do Kinesis. Ele atua como intermediário entre o código do aplicativo produtor e as ações do Kinesis Data Streams. API O KPL executa as seguintes tarefas principais:

- Grava em um ou mais fluxos de dados do Kinesis com um mecanismo automático e configurável de novas tentativas
- Coleta registros e usa `PutRecords` para gravar vários registros em vários estilhaços por solicitação
- Agrega registros de usuário para aumentar o tamanho da carga útil e melhorar a throughput
- Integra-se perfeitamente à Kinesis Client Library (KCL) para desagregar registros em lote no consumidor
- Envia CloudWatch métricas da Amazon em seu nome para fornecer visibilidade sobre o desempenho do produtor

Observe que KPL é diferente do Kinesis API Data Streams que está disponível no. AWS SDKs O Kinesis API Data Streams ajuda você a gerenciar muitos aspectos do Kinesis Data Streams (incluindo criar streams, refragmentar e colocar e obter registros)KPL, enquanto fornece uma camada de abstração específica para a ingestão de dados. Para obter informações sobre o Kinesis API Data Streams, consulte a Referência do Amazon API Kinesis.

Perceba as vantagens de usar o KPL

A lista a seguir representa algumas das principais vantagens de usar o KPL para desenvolver produtores do Kinesis Data Streams.

O KPL pode ser usado em casos de uso síncronos ou assíncronos. Sugerimos usar o desempenho mais alto da interface assíncrona, a menos que haja um motivo específico para usar o comportamento síncrono. Para obter mais informações sobre esses dois casos de uso e o código de exemplo, consulte Grave em seu stream de dados do Kinesis usando o KPL.

Benefícios de desempenho

Eles KPL podem ajudar a criar produtores de alto desempenho. Considere uma situação em que suas EC2 instâncias da Amazon sirvam como proxy para coletar eventos de 100 bytes de centenas ou milhares de dispositivos de baixo consumo de energia e gravar registros em um stream de dados do Kinesis. Cada uma dessas EC2 instâncias deve gravar milhares de eventos por segundo em seu stream de dados. Para alcançar a throughput necessária, os produtores precisam implementar uma lógica complexa, como agrupamento em lotes ou multithreading, lógica de retentativa e desagregação de registros no lado do consumidor. O KPL executa todas essas tarefas para você.

Facilidade de uso do lado do consumidor

Para desenvolvedores do lado do consumidor que usam o KCL em Java, ele KPL se integra sem esforço adicional. Quando KCL recupera um registro agregado do Kinesis Data Streams que consiste em KPL vários registros de usuário, ele o invoca automaticamente para extrair os registros individuais KPL do usuário antes de devolvê-los ao usuário.

Para desenvolvedores do lado do consumidor que não usam a operação, KCL mas usam a API operação `GetRecords` diretamente, uma biblioteca KPL Java está disponível para extrair os registros individuais do usuário antes de devolvê-los ao usuário.

Monitoramento de produtor

Você pode coletar, monitorar e analisar seus produtores do Kinesis Data Streams usando a CloudWatch Amazon e o. KPL Ele KPL emite taxa de transferência, erro e outras métricas CloudWatch em seu nome e é configurável para monitoramento no nível de stream, fragmento ou produtor.

Arquitetura assíncrona

Como eles KPL podem armazenar registros em buffer antes de enviá-los para o Kinesis Data Streams, isso não força o aplicativo chamador a bloquear e esperar pela confirmação de que o registro chegou ao servidor antes de continuar a execução. Uma chamada para colocar um registro no KPL sempre retorna imediatamente e não espera que o registro seja enviado ou que uma resposta seja recebida do servidor. Em vez disso, é criado um objeto `Future` que posteriormente recebe o resultado do envio do registro ao Kinesis Data Streams. Esse é o mesmo comportamento dos clientes assíncronos no. AWS SDK

Entenda quando não usar o KPL

Eles KPL podem incorrer em um atraso adicional de processamento de até `RecordMaxBufferedTime` dentro da biblioteca (configurável pelo usuário). Valores maiores de `RecordMaxBufferedTime` geram maiores eficiências de empacotamento e melhor desempenho. Os aplicativos que não toleram esse atraso adicional podem precisar usá-los AWS SDK diretamente. Para obter mais informações sobre como usar o AWS SDK com o Kinesis Data StreamsDesenvolva produtores usando o Amazon Kinesis API Data Streams com o AWS SDK for Java, consulte. Para obter mais informações `RecordMaxBufferedTime` e outras propriedades configuráveis pelo usuário doKPL, consulte. Configurar a Kinesis Producer Library

Instalar a KPL

A Amazon fornece binários pré-criados da C++ Kinesis Producer Library (KPL) para macOS, Windows e distribuições Linux recentes (para obter detalhes da plataforma compatível, consulte a próxima seção). Esses binários, empacotados como parte dos arquivos .jar do Java, são invocados e usados automaticamente se você está usando o Maven para instalar o pacote. Para localizar as versões mais recentes do KPL eKCL, use os seguintes links de pesquisa do Maven:

- KPL
- KCL

Os binários do Linux foram compilados com o GNU Compiler Collection (GCC) e vinculados estaticamente ao libstdc++ no Linux. Espera-se que eles funcionem em qualquer distribuição do Linux de 64 bits que inclua um glibc versão 2.5 ou superior.

Os usuários de distribuições Linux mais antigas podem criar o KPL usando as instruções de compilação fornecidas junto com o código-fonte. GitHub Para fazer o download KPL do GitHub, consulte Kinesis Producer Library.

Transição para certificados Amazon Trust Services (ATS) para o KPL

Em 9 de fevereiro de 2018, às 9hPST, o Amazon Kinesis Data Streams instalou certificados. ATS Para continuar a gravar registros no Kinesis Data Streams usando a Kinesis Producer KPL Library (), você deve atualizar sua instalação KPL do para a versão 0.12.6 ou posterior. Essa mudança afeta todas as AWS regiões.

Para obter informações sobre a mudança paraATS, consulte Como se preparar para AWS a mudança para sua própria autoridade de certificação.

Se encontrar problemas e precisar de suporte técnico, abra um caso na central de suporte da AWS .

Plataformas com suporte do KPL

A Kinesis Producer Library (KPL) é escrita em C++ e é executada como um processo secundário do processo do usuário principal. Binários nativos pré-compilados de 64 bits são fornecidos com a versão do Java e gerenciados pelo wrapper Java.

O pacote Java é executado sem a necessidade de instalar qualquer biblioteca adicional nos seguintes sistemas operacionais:

- Distribuições do Linux com kernel 2.6.18 (setembro de 2006) e posterior
- Apple OS X 10.9 e posterior
- Windows Server 2008 e posterior

> **Important**
>
> O Windows Server 2008 e versões posteriores são compatíveis com todas as KPL versões até a versão 0.14.0.
> A plataforma Windows é NOT suportada a partir da KPL versão 0.14.0 ou superior.

Observe que KPL é somente de 64 bits.

Código-fonte

Se os binários fornecidos na KPL instalação não forem suficientes para seu ambiente, o núcleo do KPL será escrito como um módulo C++. O código-fonte do módulo C++ e da interface Java é lançado sob a Amazon Public License e está disponível GitHub na Kinesis Producer Library. Embora o KPL possa ser usado em qualquer plataforma para a qual um compilador C++ recente e compatível com os padrões esteja JRE disponível, a Amazon não oferece suporte oficial a nenhuma plataforma que não esteja na lista de plataformas suportadas.

Principais conceitos da KPL

As seções a seguir contêm conceitos e terminologia necessários para entender e se beneficiar da Kinesis Producer Library KPL ().

Tópicos

Registros

Neste guia, distinguimos entre registros de KPLusuários e registros do Kinesis Data Streams. Quando usamos o termo registro sem um qualificador, nos referimos a um registro de KPL usuário. Um registro do Kinesis Data Streams será explicitamente chamado de registro do Kinesis Data Streams.

Um registro de KPL usuário é uma bolha de dados que tem um significado particular para o usuário. Os exemplos incluem um JSON blob representando um evento de interface de usuário em um site ou uma entrada de registro de um servidor web.

Um registro do Kinesis Data Streams é uma instância `Record` da estrutura de dados definida pelo serviço Kinesis Data Streams. API Ele contém uma chave de partição, um número sequencial e um blob de dados.

Agrupamento em lotes

Envio em lotes refere-se à execução de uma única ação em vários itens, em vez de executar a ação repetidamente em cada item.

Nesse contexto, o “item” é um registro, e a ação é seu envio ao Kinesis Data Streams. Em uma situação sem lotes, você colocaria cada registro em um registro separado do Kinesis Data Streams e faria HTTP uma solicitação para enviá-lo ao Kinesis Data Streams. Com o agrupamento em lotes, cada HTTP solicitação pode conter vários registros em vez de apenas um.

O KPL suporta dois tipos de lotes:

- Agregação: armazenamento de vários registros em um único registro do Kinesis Data Streams.
- Coleta — Usando a API operação `PutRecords` para enviar vários registros do Kinesis Data Streams para um ou mais fragmentos em seu stream de dados do Kinesis.

Os dois tipos de KPL lotes foram projetados para coexistir e podem ser ativados ou desativados independentemente um do outro. Por padrão, ambos estão ativados.

Agregação

A agregação refere-se ao armazenamento de vários registros em um registro do Kinesis Data Streams. A agregação permite que os clientes aumentem o número de registros enviados por API chamada, o que aumenta efetivamente a produtividade do produtor.

Os fragmentos do Kinesis Data Streams oferecem suporte a até 1.000 registros do Kinesis Data Streams por segundo, ou uma throughput de 1 MB. O limite de registros por segundo do Kinesis Data Streams restringe clientes com registros menores que 1 KB. A agregação de registros permite aos clientes combinar vários registros em um único registro do Kinesis Data Streams. Isso permite que os clientes melhorem a própria throughput por estilhaço.

Considere o caso de um estilhaço na região us-east-1 que está atualmente em execução à taxa constante de 1.000 registros por segundo, com registros de 512 bytes cada. Com KPL a agregação, você pode empacotar 1.000 registros em apenas 10 registros do Kinesis Data Streams, RPS reduzindo-os para 10 (com 50 KB cada).

Coleta

A coleta se refere ao agrupamento em lote de vários registros do Kinesis Data Streams e ao envio deles em uma HTTP única solicitação com uma chamada para a API `PutRecords` operação, em vez de enviar cada registro do Kinesis Data Streams em sua própria solicitação. HTTP

Isso aumenta a produtividade em comparação com o uso de nenhuma coleta, pois reduz a sobrecarga de fazer muitas solicitações separadasHTTP. Na verdade, `PutRecords`, por si só, foi projetado especificamente para essa finalidade.

A coleta difere da agregação porque opera com grupos de registros do Kinesis Data Streams. Os registros do Kinesis Data Streams coletados ainda podem conter vários registros do usuário. O relacionamento pode ser visualizado da seguinte maneira:

```
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--|
```

Integre o KPL com o código do produtor

A Kinesis Producer Library (KPL) é executada em um processo separado e se comunica com seu processo de usuário principal usando. IPC Essa arquitetura, às vezes chamada de microsserviço, é escolhida por dois motivos principais:

1) Seu processo de usuário não falhará, mesmo que o processo KPL falhe

Seu processo pode ter tarefas não relacionadas ao Kinesis Data Streams e pode ser capaz de continuar operando mesmo se as falhas ocorrerem. KPL Também é possível que seu processo de usuário pai KPL reinicie o e se recupere para um estado totalmente funcional (essa funcionalidade está nos wrappers oficiais).

Um exemplo é um servidor Web que envia métricas ao Kinesis Data Streams. O servidor pode continuar entregando páginas mesmo que a parte do Kinesis Data Streams tenha parado de funcionar. Falhar todo o servidor devido a um bug no KPL causaria, portanto, uma interrupção desnecessária.

2) Clientes arbitrários podem ser aceitos

Há sempre clientes que usam linguagens diferentes das oficialmente aceitas. Esses clientes também devem ser capazes de usá-los KPL facilmente.

Matriz de uso recomendada

A matriz de uso a seguir enumera as configurações recomendadas para diferentes usuários e o aconselha sobre se e como você deve usar o. KPL Lembre-se de que, se a agregação estiver habilitada, será preciso usar a desagregação para extrair seus registros no lado do consumidor.

Linguagem do lado do produtor	Linguagem do lado do consumidor	KCLVersão	Lógica do ponto de verificação	Você consegue ver a KPL?	Advertências
Tudo menos Java	*	*	*	Não	N/D
Java	Java	Usa Java SDK diretamente	N/D	Sim	Se a agregação for usada, você precisará usar a biblioteca de desagrega ção fornecida após as chamadas a `GetRecord s .`
Java	Tudo menos Java	Usa SDK diretamente	N/D	Sim	É preciso desabilitar a agregação.
Java	Java	1.3.x	N/D	Sim	É preciso desabilitar a agregação.
Java	Java	1.4.x	Chama o ponto de verificaç ão sem argumento algum	Sim	Nenhum
Java	Java	1.4.x	Chama o ponto de verificaç	Sim	Desative a agregação ou altere o

Linguagem do lado do produtor	Linguagem do lado do consumidor	KCLVersão	Lógica do ponto de verificação	Você consegue ver a KPL?	Advertências
			ão com um número sequencial explícito		código para usar números sequenciais estendidos para definir pontos de verificação.
Java	Tudo menos Java	1.3.x + daemon de várias linguagens + wrapper específico de linguagem	N/D	Sim	É preciso desabilitar a agregação.

Grave em seu stream de dados do Kinesis usando o KPL

As seções a seguir mostram exemplos de código em uma progressão do produtor mais básico para o código totalmente assíncrono.

Código do produtor Barebones

Só é preciso o código a seguir para gravar um produtor de trabalho mínimo. Os registros do usuário da Kinesis Producer Library (KPL) são processados em 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 ...
```

Responda aos resultados de forma síncrona

No exemplo anterior, o código não verificou se os registros do KPL usuário foram bem-sucedidos. KPLEle executa todas as novas tentativas necessárias para contabilizar as falhas. No entanto, para verificar os resultados, você poderá examiná-los usando os objetos `Future` que são retornados de `addUserRecord`, como no exemplo a seguir (exemplo anterior mostrado para fins de 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
        }
    }
}
```

Responda aos resultados de forma assíncrona

O exemplo anterior está chamando `get()` para um objeto `Future`, o que bloqueia a execução. Se não quiser bloquear a execução, você poderá usar um retorno de chamada assíncrono, como mostrado no exemplo a seguir:

```
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);
}
```

Configurar a Kinesis Producer Library

Embora as configurações padrão devam funcionar bem para a maioria dos casos de uso, talvez convenha alterar algumas configurações padrão para ajustar o comportamento do `KinesisProducer` às suas necessidades. Uma instância da classe `KinesisProducerConfiguration` pode ser passada ao construtor `KinesisProducer` para esse propósito, por exemplo:

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

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

Você também pode carregar uma configuração de um arquivo de propriedades:

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

Você pode substituir qualquer caminho e nome de arquivo a que o processo de usuário tem acesso. Você também pode chamar métodos definidos para a instância de `KinesisProducerConfiguration` criada dessa forma para personalizar a configuração.

O arquivo de propriedades deve especificar parâmetros usando seus nomes em PascalCase. Os nomes correspondem aos usados nos métodos definidos na classe `KinesisProducerConfiguration`. Por exemplo: .

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

Para obter mais informações sobre regras de uso de parâmetros de configuração e limites de valor, consulte o arquivo de exemplo de propriedades de configuração em GitHub.

Observe que, depois que o `KinesisProducer` é inicializado, alterar a instância de `KinesisProducerConfiguration` que foi usada não tem mais efeito. No momento, o `KinesisProducer` não oferece suporte à reconfiguração dinâmica.

Implemente a desagregação de consumidores

A partir da versão 1.4.0, ele KCL suporta a desagregação automática dos registros do KPL usuário. O código do aplicativo do consumidor escrito com versões anteriores do KCL será compilado sem qualquer modificação após a atualização doKCL. No entanto, se a KPL agregação estiver sendo usada pelo produtor, há uma sutileza envolvendo a verificação: todos os sub-registros em um registro agregado têm o mesmo número de sequência, portanto, dados adicionais precisam ser armazenados no ponto de verificação se você precisar distinguir entre sub-registros. Esses dados adicionais são chamados de números de subsequência.

Opções

- Migre de versões anteriores do KCL
- Use KCL extensões para KPL desagregação
- Use GetRecords diretamente

Migre de versões anteriores do KCL

Não é necessário alterar as chamadas existentes para que definam pontos de verificação em conjunto com a agregação. A recuperação bem-sucedida de todos os registros armazenados no Kinesis Data Streams ainda é garantida. O KCL agora fornece duas novas operações de ponto de verificação para dar suporte a casos de uso específicos, descritos abaixo.

Caso seu código existente tenha sido escrito para o KPL suporte KCL anterior e sua operação de ponto de verificação seja chamada sem argumentos, isso equivale a verificar o número de sequência do último registro KPL do usuário no lote. Se a operação de ponto de verificação é chamada com uma string de número sequencial, isso equivale a definir o ponto de verificação do número sequencial do lote conhecido junto com o número 0 (zero) da subsequência implícita.

Chamar a nova operação de KCL ponto de verificação `checkpoint()` sem nenhum argumento é semanticamente equivalente a verificar o número de sequência da última `Record` chamada no lote, junto com o número de subsequência implícito 0 (zero).

Chamar a nova operação de KCL ponto de verificação `checkpoint(Record record)` é semanticamente equivalente a verificar o número de sequência fornecido `Record` junto com o número de subsequência implícito 0 (zero). Se a chamada a `Record` é na verdade um `UserRecord`, é definido o ponto de verificação do número sequencial `UserRecord` e do número subsequencial.

Chamar a nova operação KCL de ponto de verificação `checkpoint(String sequenceNumber, long subSequenceNumber)` verifica explicitamente o número de sequência fornecido junto com o número de subsequência fornecido.

Em qualquer um desses casos, depois que o ponto de verificação é armazenado na tabela de pontos de verificação do Amazon DynamoDBKCL, ele pode retomar corretamente a recuperação de registros mesmo quando o aplicativo falha e reinicia. Se houver mais registros contidos na sequência, a recuperação ocorrerá a partir do próximo registro de número subsequencial dentro do registro com o número sequencial cujo ponto de verificação foi definido mais recentemente. Se o ponto de verificação mais recente inclui o último número subsequencial do registro de número sequencial anterior, a recuperação ocorrerá a partir do registro com o próximo número sequencial.

A próxima seção discute detalhes de definição do ponto de verificação de sequência e subsequência para consumidores que precisam evitar a omissão e a duplicação de registros. Se a omissão (ou duplicação) de registros ao parar e reiniciar o processamento de registros do consumidor não é importante, você pode executar seu código existente sem modificação.

Use KCL extensões para KPL desagregação

KPLa desagregação pode envolver pontos de verificação subsequentes. Para facilitar o uso do ponto de verificação subsequente, uma `UserRecord` classe foi adicionada ao: 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 */
    }
}
```

Essa classe agora é usada em vez de `Record`. Ela não interrompe o código existente por ser uma subclasse de `Record`. A classe `UserRecord` representa os sub-registros reais e os registros padrão não agregados. Os registros não agregados podem ser considerados como registros agregadas com exatamente um sub-registro.

Além disso, duas operações novas são adicionadas a `IRecordProcessorCheckpointer`:

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

Para começar a usar a definição de ponto de verificação de número subsequencial, você pode executar a seguinte conversão. Altere o seguinte código de formulário:

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

Novo código de formulário:

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

Recomendamos usar o formulário `checkpoint(Record record)` para definição de ponto de verificação de subsequência. No entanto, se você já estiver armazenando `sequenceNumbers`

em strings para usar na definição de pontos de verificação, agora deverá também armazenar `subSequenceNumber`, como mostrado no exemplo a seguir:

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

O elenco de `Record` para `UserRecord` sempre é bem-sucedido porque a implementação sempre usa`UserRecord`. A menos que haja uma necessidade de executar aritmética nos números sequenciais, essa abordagem não é recomendada.

Ao processar os registros KPL do usuário, KCL ele grava o número subsequente no Amazon DynamoDB como um campo extra para cada linha. As versões anteriores do KCL costumavam buscar registros `AFTER_SEQUENCE_NUMBER` ao retomar os pontos de verificação. A corrente KCL com KPL suporte usa `AT_SEQUENCE_NUMBER` em vez disso. Quando é recuperado o registro no número sequencial para o qual foi definido o ponto de verificação, esse número é verificado e os sub-registros são descartados conforme apropriado (que podem ser todos eles, se o último sub-registro é aquele verificado). Novamente, os registros não agregados podem ser considerados como registros agregados com um único sub-registro. Portanto, o mesmo algoritmo funciona para os registros agregados e não agregados.

Use GetRecords diretamente

Você também pode optar por não usar a operaçãoKCL, mas sim invocar a API operação `GetRecords` diretamente para recuperar os registros do Kinesis Data Streams. Para descompactar esses registros recuperados em seus registros de KPL usuário originais, chame uma das seguintes operações estáticas em: `UserRecord.java`

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

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

A primeira operação usa o valor `0` (zero) padrão para `startingHashKey` e o valor `2^128 -1` padrão para `endingHashKey`.

Cada uma dessas operações desagrega uma determinada lista de registros do Kinesis Data Streams em uma lista de registros de usuários. KPL Todos os registros de KPL usuário cuja chave de

hash ou chave de partição explícita esteja fora do intervalo de `startingHashKey` (inclusive) e `endingHashKey` (inclusive) são descartados da lista de registros retornada.

Use o KPL com o Amazon Data Firehose

Se você usa a Kinesis Producer Library (KPL) para gravar dados em um stream de dados do Kinesis, você pode usar a agregação para combinar os registros que você grava nesse stream de dados do Kinesis. Se você usar esse stream de dados como fonte para o stream de entrega do Firehose, o Firehose desagregará os registros antes de entregá-los ao destino. Se você configurar seu stream de entrega para transformar os dados, o Firehose desagregará os registros antes de entregá-los. AWS Lambda Para obter mais informações, consulte Writing to Amazon Firehose Using Kinesis Data Streams.

Use o KPL com o Registro do AWS Glue Esquema

Você pode integrar seus streams de dados do Kinesis com o AWS Glue Schema Registry. O Registro de AWS Glue Esquemas permite que você descubra, controle e desenvolva esquemas de forma centralizada, ao mesmo tempo em que garante que os dados produzidos sejam continuamente validados por um esquema registrado. O esquema define a estrutura e o formato de um registro de dados. Um esquema é uma especificação versionada para publicação, consumo ou datastore confiáveis. O AWS Glue Schema Registry permite que você melhore a qualidade end-to-end dos dados e a governança de dados em seus aplicativos de streaming. Para obter mais informações, consulte Registro de esquemas do AWS Glue. Uma das formas de configurar essa integração é por meio das bibliotecas Kinesis Client Library () KPL e da Kinesis Client Library (KCL) em Java.

> **Important**
>
> Atualmente, a integração do Kinesis Data AWS Glue Streams e do registro de esquemas só é compatível com os KPL streams de dados do Kinesis que usam produtores implementados em Java. Suporte a várias linguagens não é fornecido.

Para obter instruções detalhadas sobre como configurar a integração do Kinesis Data Streams com o Schema Registry KPL usando o, consulte a seção “Interagindo com dados KPL usando as/ Bibliotecas” em Caso de uso: Integração KCL do Amazon Kinesis Data Streams com o Glue Schema Registry. AWS

Configurar a configuração do KPL proxy

Para aplicativos que não podem se conectar diretamente à Internet, todos os AWS SDK clientes oferecem suporte ao uso de proxies HTTP ou HTTPS proxies. Em um ambiente empresarial típico, todo o tráfego de saída da rede precisa passar por servidores proxy. Se seu aplicativo usa o Kinesis Producer Library (KPL) para coletar e enviar dados AWS em um ambiente que usa servidores proxy, seu aplicativo exigirá configuração de KPL proxy. KPLé uma biblioteca de alto nível construída sobre o AWS KinesisSDK. Ele é dividido em um processo nativo e um wrapper. O processo nativo executa todas as tarefas de processamento e envio de registros, enquanto o wrapper gerencia o processo nativo e se comunica com ele. Para obter mais informações, consulte Implementing Efficient and Reliable Producers with the Amazon Kinesis Producer Library.

O wrapper é escrito em Java e o processo nativo é escrito em C++ com o uso do Kinesis. SDK KPLA versão 0.14.7 e superior agora suporta a configuração de proxy no wrapper Java, que pode passar todas as configurações de proxy para o processo nativo. Para obter mais informações, consulte https://github.com/awslabs/amazon-kinesis-producer/releases/tag/v0.14.7.

Você pode usar o código a seguir para adicionar configurações de proxy aos seus KPL aplicativos.

```
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);
```

Desenvolva produtores usando o Amazon Kinesis API Data Streams com o AWS SDK for Java

Você pode desenvolver produtores usando o Amazon Kinesis API Data Streams AWS SDK com o for Java. Se você nunca usou o Kinesis Data Streams, comece familiarizando-se com os conceitos e a terminologia apresentados em O que é o Amazon Kinesis Data Streams? e Use o AWS CLI para realizar operações do Amazon Kinesis Data Streams.

Esses exemplos discutem o Kinesis API Data Streams e AWS SDKusam o for Java para adicionar (colocar) dados em um stream. No entanto, para a maioria dos casos de uso, você deve preferir a biblioteca Kinesis KPL Data Streams. Para obter mais informações, consulte Desenvolva produtores usando a Amazon Kinesis Producer Library () KPL.

O código de exemplo de Java neste capítulo demonstra como realizar operações básicas do Kinesis Data API Streams e é dividido logicamente por tipo de operação. Esses exemplos não representam um código pronto para produção, pois não verificam todas as exceções possíveis nem abrangem todas as considerações de segurança ou de performance possíveis. Além disso, você pode chamar o Kinesis API Data Streams usando outras linguagens de programação. Para obter mais informações sobre todas as opções disponíveis AWS SDKs, consulte Comece a desenvolver com a Amazon Web Services.

Cada tarefa tem pré-requisitos. Por exemplo, não é possível adicionar dados a um stream enquanto o stream não é criado, o que requer a criação de um cliente. Para obter mais informações, consulte Crie e gerencie streams de dados do Kinesis.

Tópicos

Adicionar dados a um stream

Quando um stream é criado, você pode adicionar dados a ele na forma de registros. Um registro é uma estrutura de dados que contém os dados a serem processados na forma de um blob de dados. Depois que você armazena os dados no registro, o Kinesis Data Streams não inspeciona, interpreta nem altera dados de forma alguma. Cada registro também tem um número sequencial e uma chave de partição associados.

Há duas operações diferentes no Kinesis Data Streams que adicionam API dados a um `PutRecords`stream, e. `PutRecord` A `PutRecords` operação envia vários registros para seu stream por HTTP solicitação, e a `PutRecord` operação singular envia registros para seu stream, um por vez (uma HTTP solicitação separada é necessária para cada registro). Talvez convenha usar `PutRecords` para a maioria dos aplicativos, pois ele atingirá uma throughput mais alta por produtor de dados. Para obter mais informações sobre cada uma dessas operações, consulte as subseções abaixo.

Tópicos

- Adicione vários registros com PutRecords
- Adicione um único registro com PutRecord

Lembre-se sempre de que, como seu aplicativo de origem está adicionando dados ao stream usando o Kinesis API Data Streams, provavelmente há um ou mais aplicativos consumidores que estão processando simultaneamente dados fora do stream. Para obter informações sobre como os consumidores obtêm dados usando o Kinesis API Data StreamsObter dados de um stream, consulte.

> **Important**
>
> Alterar o período de retenção de dados

Adicione vários registros com PutRecords

A operação `PutRecords` envia vários registros ao Kinesis Data Streams em uma única solicitação. Ao usar `PutRecords`, os produtores podem obter uma throughput mais alta ao enviar dados para o fluxo de dados do Kinesis. Cada solicitação `PutRecords` pode oferecer suporte a até 500 registros. Cada registro na solicitação pode ter no máximo 1 MB, até um limite de 5 MB para toda a solicitação, incluindo chaves de partição. Assim como a operação única `PutRecord` descrita abaixo, `PutRecords` usa números de sequência e chaves de partição. No entanto, o parâmetro `PutRecord` de `SequenceNumberForOrdering` não é incluído em uma chamada a `PutRecords`. A operação `PutRecords` tenta processar todos os registros na ordem natural da solicitação.

Cada registro de dados tem um número sequencial exclusivo. O número de sequência é atribuído pelo Kinesis Data Streams depois que `client.putRecords` é chamada para adicionar os registros de dados ao fluxo. Os números sequenciais da mesma chave de partição geralmente aumentam com o tempo: quanto maior o período entre as solicitações `PutRecords`, maiores ficam os números sequenciais.

> **Note**
>
> Os números de sequência não podem ser usados como índices para conjuntos de dados dentro do mesmo stream. Para separar logicamente conjuntos de dados, use chaves de partição ou crie um stream separado para cada conjunto de dados.

Uma solicitação `PutRecords` pode incluir registros com diferentes chaves de partição. O escopo da solicitação é um stream; cada solicitação pode incluir qualquer combinação de chaves de partição e registros, dentro dos limites da solicitação. As solicitações feitas com diferentes chaves de partição a streams com muitos estilhaços diferentes costumam ser mais rápidas do que as solicitações com um pequeno número de chaves de partição para um pequeno número de estilhaços. O número de chaves de partição deve ser muito maior do que o número de estilhaços para reduzir a latência e maximizar a throughput.

PutRecordsexemplo

O código a seguir cria 100 registros de dados com chaves de partição sequenciais e os coloca em um stream denominado `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);
```

A resposta a `PutRecords` inclui uma matriz de resposta `Records`. Cada registro na matriz de resposta se correlaciona diretamente com um registro na matriz de solicitação por ordenação natural,

do início ao fim da solicitação e da resposta. A matriz de resposta de `Records` sempre inclui o mesmo número de registros da matriz de solicitação.

Lidar com falhas ao usar PutRecords

Por padrão, a falha de registros individuais em uma solicitação não interrompe o processamento de registros subsequentes em uma solicitação `PutRecords`. Isso significa que uma matriz de resposta `Records` inclui os registros com processamento bem-sucedido e malsucedido. É preciso detectar os registros com processamento malsucedido e incluí-los em uma chamada subsequente.

Os registros bem-sucedidos incluem os valores `SequenceNumber` e `ShardID`, e os registros malsucedidos incluem os valores `ErrorCode` e `ErrorMessage`. O parâmetro `ErrorCode` reflete o tipo de erro e pode ter um dos seguintes valores: `ProvisionedThroughputExceededException` ou `InternalFailure`. `ErrorMessage` fornece informações mais detalhadas sobre a exceção `ProvisionedThroughputExceededException`, incluindo o ID da conta, o nome do streaming e o ID do estilhaço do registro que foi limitado. O exemplo abaixo tem três registros em uma solicitação `PutRecords`. O segundo registro falha e isso é refletido na resposta.

Example PutRecords Sintaxe da solicitação

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

Example PutRecords Sintaxe de resposta

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

Os registros com processamento malsucedido podem ser incluídos nas solicitações `PutRecords` subsequentes. Primeiro, verifique o parâmetro `FailedRecordCount` no `putRecordsResult` para confirmar se há registros com falha na solicitação. Assim sendo, cada `putRecordsEntry` com um `ErrorCode` que não seja `null` deve ser adicionado a uma solicitação subsequente. Para obter um exemplo desse tipo de handler, consulte o seguinte código.

Example PutRecords manipulador de falhas

```
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);
}
```

Adicione um único registro com PutRecord

Cada chamada para `PutRecord` opera em um único registro. Prefira a operação `PutRecords` descrita em Adicione vários registros com PutRecords, a menos que seu aplicativo precise especificamente enviar sempre registos únicos por solicitação ou algum outro motivo para o não uso de `PutRecords`.

Cada registro de dados tem um número sequencial exclusivo. O número de sequência é atribuído pelo Kinesis Data Streams depois que `client.putRecord` é chamada para adicionar o registro de dados ao fluxo. Os números sequenciais da mesma chave de partição geralmente aumentam com o tempo: quanto maior o período entre as solicitações `PutRecord`, maiores ficam os números sequenciais.

Quando ocorrem colocações em rápida sucessão, não há garantia de que os números de sequência retornados aumentem, porque as operações put aparentam ser essencialmente simultâneas para o Kinesis Data Streams. Para garantir estritamente o aumento de números sequenciais para a mesma chave de partição, use o parâmetro `SequenceNumberForOrdering`, como mostrado no código de exemplo em PutRecordexemplo.

Usando ou não `SequenceNumberForOrdering`, os registros que o Kinesis Data Streams recebe por meio de uma chamada a `GetRecords` são estritamente ordenados por número de sequência.

> **Note**
>
> Os números de sequência não podem ser usados como índices para conjuntos de dados dentro do mesmo stream. Para separar logicamente conjuntos de dados, use chaves de partição ou crie um stream separado para cada conjunto de dados.

Uma chave de partição é usada para agrupar os dados dentro de um stream. Um registro de dados é atribuído a um estilhaço dentro do stream com base em sua chave de partição. Especificamente, o Kinesis Data Streams usa a chave de partição como entrada para uma função de hash que mapeia essa chave (e os dados associados) a um determinado fragmento.

Como resultado desse mecanismo de hashing, todos os registros de dados com a mesma chave de partição são mapeados para o mesmo estilhaço no stream. No entanto, se o número de chaves de partição ultrapassar o número de estilhaços, alguns estilhaços conterão necessariamente registros com chaves de partição diferentes. Do ponto de vista do design, para garantir que todos os seus estilhaços sejam bem utilizados, o número de estilhaços (especificado pelo método `setShardCount` de `CreateStreamRequest`) deve ser substancialmente menor que o número de chaves de partição exclusivas, e o volume de dados que flui para uma única chave de partição deve ser substancialmente menor que a capacidade do estilhaço.

PutRecordexemplo

O código a seguir cria dez registros de dados, distribuídos entre duas chaves de partição, e os coloca em um stream denominado `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();
}
```

O código de exemplo anterior usa `setSequenceNumberForOrdering` para garantir estritamente o aumento da ordenação dentro de cada chave de partição. Para usar esse parâmetro de forma eficaz,

defina o `SequenceNumberForOrdering` do registro atual (registro n) como o número de sequência do registro anterior (registro n-1). Para obter o número sequencial de um registro que foi adicionado ao stream, chame `getSequenceNumber` para o resultado de `putRecord`.

O parâmetro `SequenceNumberForOrdering` garante estritamente o aumento de números de sequência para a mesma chave de partição. `SequenceNumberForOrdering` não fornece a ordenação de registros em várias chaves de partição.

Interaja com os dados usando o AWS Glue Schema Registry

Você pode integrar seus streams de dados do Kinesis com o AWS Glue Schema Registry. O Registro de AWS Glue Esquemas permite que você descubra, controle e desenvolva esquemas de forma centralizada, ao mesmo tempo em que garante que os dados produzidos sejam continuamente validados por um esquema registrado. O esquema define a estrutura e o formato de um registro de dados. Um esquema é uma especificação versionada para publicação, consumo ou datastore confiáveis. O AWS Glue Schema Registry permite que você melhore a qualidade end-to-end dos dados e a governança de dados em seus aplicativos de streaming. Para obter mais informações, consulte Registro de esquemas do AWS Glue. Uma das formas de configurar essa integração é por meio do `PutRecords` `PutRecord` Kinesis APIs Data Streams disponível AWS em Java. SDK

Para obter instruções detalhadas sobre como configurar a integração do Kinesis Data Streams com o registro do esquema usando o e o PutRecord Kinesis Data Streams, consulte a seção “Interagindo com dados PutRecords usando o APIs Kinesis Data Streams” em Caso de uso: Integração do Amazon Kinesis Data APIs Streams com o Glue Schema Registry. AWS

Grave no Amazon Kinesis Data Streams usando o Kinesis Agent

O Kinesis Agent é uma aplicação de software Java independente que oferece uma maneira fácil de coletar e enviar dados ao Kinesis Data Streams. O agente monitora continuamente um conjunto de arquivos e envia novos dados ao stream. Ele manipula o rodízio de arquivos, os pontos de verificação e as novas tentativas após falhas. Seus dados são entregues de maneira confiável, imediata e simples. Ele também emite CloudWatch métricas da Amazon para ajudar você a monitorar e solucionar melhor o processo de streaming.

Por padrão, os registros são analisados em cada arquivo com base no caractere de nova linha (`'\n'`). No entanto, o agente também pode ser configurado para analisar registros de várias linhas (consulte Especifique as configurações do agente).

Você pode instalar o agente em ambientes de servidor baseados no Linux, como servidores web, servidores de log e servidores de banco de dados. Após instalar o agente, configure-o especificando os arquivos a serem monitorados e o stream dos dados. Depois que o agente é configurado, ele coleta dados dos arquivos de forma durável e os envia confiavelmente ao stream.

Tópicos

Conclua os pré-requisitos do Kinesis Agent

- Seu sistema operacional deve ser o Amazon Linux AMI com a versão 2015.09 ou posterior ou o Red Hat Enterprise Linux versão 7 ou posterior.
- Se você estiver usando EC2 a Amazon para executar seu agente, inicie sua EC2 instância.
- Gerencie suas AWS credenciais usando um dos seguintes métodos:
  - Especifique uma IAM função ao iniciar sua EC2 instância.
  - Especifique AWS as credenciais ao configurar o agente (consulte awsAccessKeyID e awsSecretAccesschave).
  - Edite `/etc/sysconfig/aws-kinesis-agent` para especificar sua região e suas chaves de AWS acesso.
  - Se sua EC2 instância estiver em uma AWS conta diferente, crie uma IAM função para fornecer acesso ao serviço Kinesis Data Streams e especifique essa função ao configurar o agente assumeRoleARN(assumeRoleExternalconsulte e ID). Use um dos métodos anteriores para especificar AWS as credenciais de um usuário na outra conta que tenha permissão para assumir essa função.
- A IAM função ou AWS as credenciais que você especificar devem ter permissão para realizar a operação do Kinesis Data PutRecordsStreams para que o agente envie dados para seu stream. Se

você ativar o CloudWatch monitoramento para o agente, a permissão para realizar a CloudWatch PutMetricDataoperação também será necessária. Para obter mais informações, consulte Controle do acesso aos recursos do Amazon Kinesis Data Streams usando IAMMonitore a integridade do Kinesis Data Streams Agent com a Amazon CloudWatch, e Controle de CloudWatch acesso.

Baixe e instale o agente

Primeiro, conecte-se à instância. Para obter mais informações, consulte Connect to Your Instance no Guia EC2 do usuário da Amazon. Se você tiver problemas para se conectar, consulte Solução de problemas de conexão com sua instância no Guia EC2 do usuário da Amazon.

Para configurar o agente usando o Amazon Linux AMI

Use o comando a seguir para fazer download do agente e instalá-lo:

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

Para configurar o agente usando o Red Hat Enterprise Linux

Use o comando a seguir para fazer download do agente e instalá-lo:

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

Para configurar o agente usando GitHub

1. Baixe o agente em amazon-kinesis-agentawlabs/.
2. Instale o agente navegando até o diretório de download e executando o comando a seguir:

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

Para configurar o agente em um contêiner do Docker

O Kinesis Agent também pode ser executado em um contêiner por meio da base de contêineres amazonlinux. Use o Dockerfile a seguir e depois execute o `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"]
```

Configurar e iniciar o agente

Para configurar e iniciar o agente

1. Abra e edite o arquivo de configuração (como superusuário, se você estiver usando permissões padrão de acesso a arquivos): `/etc/aws-kinesis/agent.json`

   Nesse arquivo de configuração, especifique os arquivos (`"filePattern"`) nos quais o agente coleta dados e o nome do stream (`"kinesisStream"`) ao qual o agente envia dados. Observe que o nome do arquivo é um padrão, e o agente reconhece os rodízios de arquivos. Você só pode fazer o rodízio de arquivos ou criar novos arquivos uma vez por segundo, no máximo. O agente usa o carimbo de data e hora de criação de arquivo para determinar quais arquivos serão rastreados e colocados no final do stream; a criação de novos arquivos ou o rodízio de arquivos em uma frequência superior a uma vez por segundo não permite que o agente faça a distinção entre eles corretamente.

   ```
   {
      "flows": [
           {
               "filePattern": "/tmp/app.log*",
               "kinesisStream": "yourkinesisstream"
           }
      ]
   }
   ```

2. Inicie o agente manualmente:

   ```
   sudo service aws-kinesis-agent start
   ```

3. (Opcional) Configure o agente para ser iniciado durante o startup do sistema:

   ```
   sudo chkconfig aws-kinesis-agent on
   ```

Agora o agente está sendo executado como um serviço do sistema em segundo plano. Ele monitora continuamente os arquivos especificados e envia dados ao stream especificado. A atividade do agent é registrada em `/var/log/aws-kinesis-agent/aws-kinesis-agent.log`.

Especifique as configurações do agente

O agente oferece suporte a duas configurações obrigatórias, `filePattern` e `kinesisStream`, além das configurações opcionais de recursos adicionais. É possível especificar configurações obrigatórias e opcionais em `/etc/aws-kinesis/agent.json`.

Sempre que você alterar o arquivo de configuração, deverá interromper e iniciar o agente, usando os seguintes comandos:

```
sudo service aws-kinesis-agent stop
sudo service aws-kinesis-agent start
```

Se desejar, você pode usar o seguinte comando:

```
sudo service aws-kinesis-agent restart
```

Estas são as configurações gerais.

Definição da configuração	Descrição
`assumeRoleARN`	ARNA função a ser assumida pelo usuário. Para obter mais informações, consulte Delegar acesso entre AWS contas usando IAM funções no Guia do IAM usuário.
`assumeRoleExternalId`	Um identificador opcional que determina quem pode assumir a função. Para obter mais informações, consulte Como usar uma ID externa no Guia do IAM usuário.
`awsAccessKeyId`	AWS ID da chave de acesso que substitui as credenciais padrão. Essa configuração tem precedência sobre todos os outros provedores de credenciais.
`awsSecretAccessKey`	AWS chave secreta que substitui as credenciais padrão. Essa configuração tem precedência sobre todos os outros provedores de credenciais.

Definição da configuração	Descrição
`cloudwatch.emitMetrics`	Permite que o agente emita métricas para, CloudWatch se definidas (verdadeiras). Padrão: True
`cloudwatch.endpoint`	O endpoint regional para CloudWatch. Padrão: `monitoring.us-east-1.amazonaws.com`
`kinesis.endpoint`	O endpoint regional do Kinesis Data Streams. Padrão: `kinesis.us-east-1.amazonaws.com`

Estas são as configurações de fluxo.

Definição da configuração	Descrição
`dataProcessingOptions`	A lista das opções de processamento aplicadas a cada registro analisado antes que ele seja enviado ao stream. As opções de processamento são executadas na ordem especificada. Para ter mais informações, consulte Use o agente para pré-processar dados.
`kinesisStream`	[Obrigatório] O nome do stream.
`filePattern`	[Obrigatório] O diretório e o padrão de arquivo que devem ser combinados para serem coletados pelo agente. Para todos os arquivos correspondentes a esse padrão, deve ser concedida uma permissão de leitura a `aws-kinesis-agent-user` . Para o diretório que contém os arquivos, devem ser concedidas permissões de leitura e execução a `aws-kinesis-agent-user` .
`initialPosition`	A posição em que o arquivo começou a ser analisado. Os valores válidos são `START_OF_FILE` e `END_OF_FILE` .

Definição da configuração	Descrição
	Padrão: `END_OF_FILE`
`maxBufferAgeMillis`	O tempo máximo, em milissegundos, durante o qual o agente armazena os dados em buffer antes de enviá-los ao stream. Intervalo de valores: 1.000 a 900.000 (1 segundo a 15 minutos) Padrão: 60.000 (1 minuto)
`maxBufferSizeBytes`	O tamanho máximo, em bytes, durante o qual o agente armazena os dados em buffer antes de enviá-los ao stream. Intervalo de valores: 1 a 4.194.304 (4 MB) Padrão: 4.194.304 (4 MB)
`maxBufferSizeRecords`	O número máximo de registros para os quais o agente armazena os dados em buffer antes de enviá-los ao stream. Intervalo de valores: 1 a 500 Padrão: 500
`minTimeBetweenFilePollsMillis`	O intervalo de tempo, em milissegundos, em que o agente consulta e analisa os arquivos monitorados em busca de novos dados. Intervalo de valores: 1 ou mais Padrão: 100
`multiLineStartPattern`	O padrão de identificação do início de um registro. Um registro é composto por uma linha que corresponde ao padrão e pelas linhas subsequentes que não correspondem ao padrão. Os valores válidos são expressões regulares. Por padrão, cada nova linha nos arquivos de log é analisada como um único registro.

Definição da configuração	Descrição
`partitionKeyOption`	O método para gerar a chave de partição. Os valores válidos são `RANDOM` (inteiro gerado aleatoriamente) e `DETERMINISTIC` (um valor de hash calculado a partir dos dados). Padrão: `RANDOM`
`skipHeaderLines`	O número de linhas em que o agente ignorará a análise no início dos arquivos monitorados. Intervalo de valores: 0 ou mais Padrão: 0 (zero)
`truncatedRecordTerminator`	A string que o agente usa para truncar um registro analisado que excede o limite de tamanho de registro do Kinesis Data Streams. (1,000 KB) Padrão: `'\n'` (nova linha)

Monitore vários diretórios de arquivos e grave em vários fluxos

Ao especificar vários fluxos de configurações, você pode configurar o agente para monitorar vários diretórios de arquivos e enviar dados a vários streams. No exemplo de configuração a seguir, o agente monitora dois diretórios de arquivos e envia dados para um stream do Kinesis e um stream de entrega do Firehose, respectivamente. Observe que você pode especificar endpoints diferentes para o Kinesis Data Streams e o Firehose para que o stream do Kinesis e o stream de entrega do Firehose não precisem estar na mesma região.

```
{
    "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 obter informações mais detalhadas sobre o uso do agente com o Firehose, consulte Gravando no Amazon Data Firehose com o Kinesis Agent.

Use o agente para pré-processar dados

O agente pode pré-processar os registros analisados a partir dos arquivos monitorados antes de enviá-los ao stream. Você pode habilitar esse recurso adicionando a configuração `dataProcessingOptions` ao fluxo de arquivos. Um ou mais opções de processamento podem ser adicionadas e serão executadas na ordem especificada.

O agente oferece suporte às seguintes opções de processamento. Como o agente é de código aberto, você pode desenvolver e estender ainda mais suas opções de processamento. Você pode baixar o agente em Kinesis Agent.

Opções de processamento

`SINGLELINE`

Converte um registro de várias linhas em um registro de única linha removendo caracteres de nova linha, e espaços à esquerda e à direita.

```
{
    "optionName": "SINGLELINE"
}
```

`CSVTOJSON`

Converte um registro do formato separado por delimitador em formato. JSON

```
{
    "optionName": "CSVTOJSON",
    "customFieldNames": [ "field1", "field2", ... ],
    "delimiter": "yourdelimiter"
}
```

`customFieldNames`

[Obrigatório] Os nomes dos campos usados como chaves em cada par de JSON valores-chave. Por exemplo, se você especificar `["f1", "f2"]`, o registro "v1, v2" será convertido em `{"f1":"v1","f2":"v2"}`.

`delimiter`

A string usada como delimitador no registro. O padrão é uma vírgula (,).

`LOGTOJSON`

Converte um registro de um formato de log em JSON formato. Os formatos de log compatíveis são Apache Common Log, Apache Combined Log, Apache Error Log e RFC3164 Syslog.

```
{
    "optionName": "LOGTOJSON",
    "logFormat": "logformat",
    "matchPattern": "yourregexpattern",
    "customFieldNames": [ "field1", "field2", … ]
}
```

`logFormat`

[Obrigatório] O formato da entrada de log. Os valores possíveis são:

- `COMMONAPACHELOG`: o formato do Apache Common Log. Cada entrada de log tem o seguinte padrão: `"%{host} %{ident} %{authuser} [%{datetime}] \"%{request}\" %{response} %{bytes}"`.
- `COMBINEDAPACHELOG`: o formato do Apache Combined Log. Cada entrada de log tem o seguinte padrão: `"%{host} %{ident} %{authuser} [%{datetime}] \"%{request}\" %{response} %{bytes} %{referrer} %{agent}"`.
- `APACHEERRORLOG`: o formato do Apache Error Log. Cada entrada de log tem o seguinte padrão: `"[%{timestamp}] [%{module}:%{severity}] [pid %{processid}:tid %{threadid}] [client: %{client}] %{message}"`.
- `SYSLOG`— O formato RFC3164 Syslog. Cada entrada de log tem o seguinte padrão: `"%{timestamp} %{hostname} %{program}[%{processid}]: %{message}"`.

`matchPattern`

O padrão da expressão regular usada para extrair valores de entradas de log. Essa configuração é usada se a entrada de log não estiver em um dos formatos de log predefinidos. Se essa configuração for usado, você também terá que especificar `customFieldNames`.

`customFieldNames`

Os nomes dos campos personalizados usados como chaves em cada par de JSON valores-chave. Você pode usar essa configuração para definir nomes de campo para valores extraídos de `matchPattern` ou substituir os nomes de campo padrão de formatos de log predefinidos.

Example : LOGTOJSON Configuração

Aqui está um exemplo de uma `LOGTOJSON` configuração para uma entrada do Apache Common Log convertida em JSON formato:

```
{
    "optionName": "LOGTOJSON",
    "logFormat": "COMMONAPACHELOG"
}
```

Antes da conversão:

```
64.242.88.10 - - [07/Mar/2004:16:10:02 -0800] "GET /mailman/listinfo/hsdivision
 HTTP/1.1" 200 6291
```

Depois da conversão:

```
{"host":"64.242.88.10","ident":null,"authuser":null,"datetime":"07/
Mar/2004:16:10:02 -0800","request":"GET /mailman/listinfo/hsdivision
 HTTP/1.1","response":"200","bytes":"6291"}
```

Example : LOGTOJSON Configuração com campos personalizados

Aqui está outro exemplo de configuração `LOGTOJSON`:

```
{
```

```
    "optionName": "LOGTOJSON",
    "logFormat": "COMMONAPACHELOG",
    "customFieldNames": ["f1", "f2", "f3", "f4", "f5", "f6", "f7"]
}
```

Com essa configuração, a mesma entrada do Apache Common Log do exemplo anterior é convertida para o JSON formato da seguinte forma:

```
{"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 : Conversão da entrada Apache Common Log

A configuração de fluxo a seguir converte uma entrada do Apache Common Log em um registro de linha única no formato: JSON

```
{
    "flows": [
        {
            "filePattern": "/tmp/app.log*",
            "kinesisStream": "my-stream",
            "dataProcessingOptions": [
                {
                    "optionName": "LOGTOJSON",
                    "logFormat": "COMMONAPACHELOG"
                }
            ]
        }
    ]
}
```

Example : Conversão de registros de várias linhas

A configuração de fluxo a seguir analisa registros de várias linha cuja primeira linha começa com "`[SEQUENCE=`". Cada registro é convertido primeiro em um registro de única linha. Em seguida, os valores são extraídos do registro com base em um delimitador por tabulações. Os valores extraídos são mapeados para `customFieldNames` valores especificados para formar um registro de linha única no formato. JSON

```
{
    "flows": [
```

```
        {
            "filePattern": "/tmp/app.log*",
            "kinesisStream": "my-stream",
            "multiLineStartPattern": "\\[SEQUENCE=",
            "dataProcessingOptions": [
                {
                    "optionName": "SINGLELINE"
                },
                {
                    "optionName": "CSVTOJSON",
                    "customFieldNames": [ "field1", "field2", "field3" ],
                    "delimiter": "\\t"
                }
            ]
        }
    ]
}
```

Example : LOGTOJSON Configuração com padrão de correspondência

Aqui está um exemplo de uma LOGTOJSON configuração para uma entrada do Apache Common Log convertida em JSON formato, com o 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 da conversão:

```
123.45.67.89 - - [27/Oct/2000:09:27:09 -0400] "GET /java/javaResources.html HTTP/1.0"
 200
```

Depois da conversão:

```
{"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"}
```

Use CLI comandos do agente

Inicie automaticamente o agente durante o startup do sistema:

```
sudo chkconfig aws-kinesis-agent on
```

Verifique o status do agente:

```
sudo service aws-kinesis-agent status
```

Interrompa o agente:

```
sudo service aws-kinesis-agent stop
```

Leia o arquivo de log do agente a partir deste local:

```
/var/log/aws-kinesis-agent/aws-kinesis-agent.log
```

Desinstale o agente:

```
sudo yum remove aws-kinesis-agent
```

FAQ

Existe um Kinesis Agent para Windows?

O Kinesis Agent para Windows é um software diferente das plataformas do Kinesis Agent para Linux.

Por que o Kinesis Agent está ficando mais lento e/ou aumentando os **`RecordSendErrors`**?

Isso geralmente ocorre devido ao controle de utilização do Kinesis. Verifique a `WriteProvisionedThroughputExceeded` métrica do Kinesis Data Streams `ThrottledRecords` ou a métrica do Firehose Delivery Streams. Qualquer aumento de 0 nessas métricas indica que os limites do fluxo precisam ser aumentados. Para obter mais informações, consulte Kinesis Data Stream limits e Amazon Firehose Delivery Streams.

Depois de descartar o controle de utilização como causa, verifique se o Kinesis Agent está configurado para seguir um número grande de arquivos pequenos. Há um atraso quando o Kinesis Agent exibe os dados do final de um arquivo novo, portanto, o Kinesis Agent deveria estar exibindo os dados do final de um pequeno número de arquivos maiores. Tente consolidar os arquivos de log em arquivos maiores.

Por que estou recebendo exceções **`java.lang.OutOfMemoryError`** ?

O Kinesis Agent não tem memória suficiente para lidar com a workload atual. Tente aumentar `JAVA_START_HEAP` e `JAVA_MAX_HEAP` no `/usr/bin/start-aws-kinesis-agent` e reiniciar o agente.

Por que estou recebendo exceções **`IllegalStateException : connection pool shut down`**?

O Kinesis Agent não tem conexões suficientes para lidar com a workload atual. Tente aumentar `maxConnections` e `maxSendingThreads` nas configurações gerais do agente em `/etc/aws-kinesis/agent.json`. O valor padrão para esses campos é 12 vezes o número de processadores de runtime disponíveis. Consulte AgentConfiguration.java para saber mais sobre as configurações avançadas do agente.

Como posso depurar outro problema com o Kinesis Agent?

Os logs do nível `DEBUG` podem ser habilitados em `/etc/aws-kinesis/log4j.xml`.

Como devo configurar o Kinesis Agent?

Quanto menor o `maxBufferSizeBytes`, mais frequentemente o Kinesis Agent enviará dados. Isso pode ser bom, pois diminui o tempo de entrega dos registros, mas também aumenta as solicitações por segundo feitas ao Kinesis.

Por que o Kinesis Agent está enviando registros duplicados?

Isso ocorre devido a uma configuração incorreta da exibição dos dados do final dos arquivos. Certifique-se de que cada `fileFlow's filePattern` corresponda a apenas um arquivo. Isso também pode ocorrer se o modo `logrotate` que está sendo usado estiver no modo `copytruncate`. Tente mudar o modo para o modo padrão ou criar para evitar duplicações. Para obter mais informações sobre como lidar com registros duplicados, consulte Handling Duplicate Records.

Grave no Kinesis Data Streams usando outros serviços AWS

Os AWS serviços a seguir podem se integrar diretamente ao Amazon Kinesis Data Streams para gravar dados nos streams de dados do Kinesis. Revise as informações de cada serviço em que você está interessado e consulte as referências fornecidas.

Tópicos

Grave no Kinesis Data Streams usando AWS Amplify

Você pode usar o Amazon Kinesis Data Streams para transmitir dados de seus aplicativos móveis criados AWS com o Amplify para processamento em tempo real. Isso permite que você crie painéis em tempo real, capture exceções, gere alertas, estimule recomendações e tome outras decisões comerciais ou operacionais oportunas. Você também pode enviar dados para outros serviços, como Amazon Simple Storage Service, Amazon DynamoDB e Amazon Redshift.

Para obter mais informações, consulte Using Amazon Kinesis no AWS Amplify Developer Center.

Escreva para o Kinesis Data Streams usando o Amazon Aurora

Você pode usar o Amazon Kinesis Data Streams para monitorar atividades em clusters de banco de dados do Amazon Aurora. Usando o Database Activity Streams, seu cluster de banco de dados

do Aurora envia atividades para um fluxo de dados do Amazon Kinesis em tempo real. Em seguida, você pode criar aplicações de gerenciamento de conformidade para consumir essas atividades, auditá-las e gerar alertas. Você também pode usar o Amazon Firehose para armazenar dados.

Para obter mais informações, consulte Database Activity Streams no Guia do desenvolvedor do Amazon Aurora.

Escreva para o Kinesis Data Streams usando a Amazon CloudFront

Você pode usar o Amazon Kinesis Data CloudFront Streams com registros em tempo real e obter informações sobre solicitações feitas para uma distribuição em tempo real. Em seguida, você pode criar seu próprio consumidor de stream de dados do Kinesis ou usar o Amazon Data Firehose para enviar os dados de log para o Amazon S3, Amazon Redshift, Amazon Service ou um serviço de processamento de log de OpenSearch terceiros.

Para obter mais informações, consulte Registros em tempo real no Amazon CloudFront Developer Guide.

Grave no Kinesis Data Streams CloudWatch usando o Amazon Logs

Você pode usar CloudWatch assinaturas para ter acesso a um feed em tempo real de eventos de log do Amazon CloudWatch Logs e entregá-lo a um stream de dados do Kinesis para processamento, análise e carregamento em outros sistemas.

Para obter mais informações, consulte Processamento em tempo real de dados de log com assinaturas no Guia do usuário do Amazon CloudWatch Logs.

Grave no Kinesis Data Streams usando o Amazon Connect

Você pode usar o Kinesis Data Streams para exportar registros de contatos e eventos de agentes em tempo real da sua instância do Amazon Connect. Você também pode ativar o streaming de dados dos perfis de clientes do Amazon Connect para receber automaticamente atualizações em um stream de dados do Kinesis sobre a criação de novos perfis ou alterações nos existentes.

Em seguida, você pode criar aplicativos de consumo para processar e analisar os dados em tempo real. Por exemplo, usando registros de contato e dados do perfil do cliente, você pode manter os dados do sistema de origem, como CRMs ferramentas de automação de marketing, up-to-date com as informações mais recentes. Usando os dados de eventos do agente, você pode criar painéis que exibem informações e eventos, além de acionar notificações personalizadas de atividades específicas do agente.

Para obter mais informações, consulte Data streaming for your instance, Set up real-time export e Agent event streams no Guia do administrador do Amazon Connect.

Grave no Kinesis Data Streams usando AWS Database Migration Service

Você pode usar AWS Database Migration Service para migrar dados para um stream de dados do Kinesis. Em seguida, você pode criar aplicações de consumo para processar os registros de dados em tempo real. Você também pode enviar facilmente dados downstream para outros serviços, como Amazon Simple Storage Service, Amazon DynamoDB e Amazon Redshift

Para obter mais informações, consulte Using Kinesis Data Streams no Guia do usuário do AWS Database Migration Service .

Grave no Kinesis Data Streams usando o Amazon DynamoDB

É possível usar o Amazon Kinesis Data Streams para capturar alterações no Amazon DynamoDB. O Kinesis Data Streams capta alterações no nível de item em qualquer DynamoDB e as replica em um fluxo de dados do Kinesis. Suas aplicações de consumo podem acessar esse fluxo para visualizar as alterações nos itens em tempo real e entregá-las downstream ou realizar ações com base no seu conteúdo.

Para obter mais informações, consulte Como o Kinesis Data Streams funciona com o DynamoDB no Guia do desenvolvedor do Amazon DynamoDB.

Escreva para o Kinesis Data Streams usando a Amazon EventBridge

Usando o Kinesis Data Streams, você AWS API pode enviar EventBridge eventos de chamada para um stream, criar aplicativos de consumo e processar grandes quantidades de dados. Você também pode usar o Kinesis Data Streams como EventBridge destino no Pipes e entregar registros de um stream de uma das fontes disponíveis após filtragem e enriquecimento opcionais.

Para obter mais informações, consulte Enviar eventos para um stream do Amazon Kinesis e EventBridge Pipes no Guia EventBridge do usuário da Amazon.

Grave no Kinesis Data Streams usando AWS IoT Core

Você pode gravar dados em tempo real a partir de MQTT mensagens no AWS IoT Core usando ações de regras de AWS IoT. Em seguida, você pode criar aplicações para processar os dados,

analisar seu conteúdo, gerar alertas e entregar os dados a aplicações de análise ou outros serviços da AWS .

Para obter mais informações, consulte Kinesis Data Streams no Guia do desenvolvedor do AWS IoT Core.

Grave no Kinesis Data Streams usando o Amazon Relational Database Service (Amazon Relational Database Service)

Você pode usar o Amazon Kinesis Data Streams para monitorar atividades em suas RDS instâncias da Amazon. Usando o Database Activity Streams, a Amazon RDS envia atividades para um stream de dados do Kinesis em tempo real. Em seguida, você pode criar aplicações de gerenciamento de conformidade para consumir essas atividades, auditá-las e gerar alertas. Você também pode usar o Amazon Data Firehose para armazenar os dados.

Para obter mais informações, consulte Database Activity Streams no Amazon RDS Developer Guide.

Grave no Kinesis Data Streams Pinpoint usingAmazon

É possível configurar o Amazon Pinpoint para enviar dados de eventos ao Amazon Kinesis Data Streams. O Amazon Pinpoint pode enviar dados de eventos para campanhas, viagens e e-mails e mensagens transacionais. SMS Em seguida, você pode ingerir os dados em aplicações de análise ou criar as próprias aplicações de consumo para realizar ações com base no conteúdo dos eventos.

Para obter mais informações, consulte Streaming Events no Guia do desenvolvedor do Amazon Pinpoint.

Grave no Kinesis Data Streams usando o Amazon Quantum Ledger Database (Amazon) QLDB

Você pode criar um stream na Amazon QLDB que capture todas as revisões de documentos comprometidas com seu diário e entregue esses dados ao Amazon Kinesis Data Streams em tempo real. Um QLDB stream é um fluxo contínuo de dados do diário do seu livro contábil para um recurso de stream de dados do Kinesis. Em seguida, você pode usar a plataforma de streaming Kinesis ou a Kinesis Client Library para consumir o fluxo, processar os registros de dados e analisar o conteúdo dos dados. Um QLDB stream grava seus dados no Kinesis Data Streams em três tipos de `control` registros`block summary`:, e. `revision details`

Para obter mais informações, consulte Streams no Guia do QLDB desenvolvedor da Amazon.

Grave no Kinesis Data Streams usando integrações de terceiros

Você pode gravar dados no Kinesis Data Streams usando uma das seguintes opções de terceiros que se integram ao Kinesis Data Streams. Selecione a opção sobre a qual você deseja saber mais e encontre recursos e links para a documentação relevante.

Tópicos

Apache Flink

O Apache Flink é um framework de código aberto distribuído para computações com estado em fluxos de dados delimitados e não delimitados. Para obter mais informações sobre como gravar no Kinesis Data Streams do Apache Flink, consulte Amazon Kinesis Data Streams Connector.

Fluentd

O Fluentd é um coletor de dados de código aberto para uma camada de registro unificada. Para obter mais informações sobre como gravar no Kinesis Data Streams do Fluentd, consulte Stream processing with Kinesis.

Debezium

O Debezium é uma plataforma distribuída de código aberto para captura de mudanças nos dados. Para obter mais informações sobre como gravar no Kinesis Data Streams a partir do Debezium, consulte Streaming de minhas alterações de SQL dados para o Amazon Kinesis.

Oráculo GoldenGate

GoldenGate O Oracle é um produto de software que permite replicar, filtrar e transformar dados de um banco de dados para outro. Para obter mais informações sobre como gravar no Kinesis Data

Streams GoldenGate da Oracle, consulte Replicação de dados para o Kinesis Data Stream usando Oracle. GoldenGate

Kafka Connect

O Kafka Connect é uma ferramenta para transmitir dados de forma escalável e confiável entre o Apache Kafka e outros sistemas. Para obter mais informações sobre como gravar no Kinesis Data Streams do Apache Kafka, consulte Kinesis kafka connector.

Adobe Experience

A Adobe Experience Platform permite que as organizações centralizem e padronizem dados dos clientes em qualquer sistema. Em seguida, a plataforma aplica ciência de dados e machine learning para melhorar significativamente o design e a entrega de experiências ricas e personalizadas. Para obter mais informações sobre como gravar dados no Kinesis Data Streams da Adobe Experience Platform, aprenda como criar uma conexão com o Amazon Kinesis.

Striim

O Striim é uma plataforma completa em memória para coletar, filtrar, transformar, enriquecer, agregar, analisar e fornecer dados em tempo real. end-to-end Para obter mais informações sobre como gravar dados no Kinesis Data Streams do Striim, consulte Kinesis Writer.

Solucionar problemas dos produtores do Amazon Kinesis Data Streams

Os tópicos a seguir oferecem soluções para problemas comuns com os produtores do Amazon Kinesis Data Streams:

- Meu aplicativo produtor está gravando em um ritmo mais lento do que o esperado
- Eu recebo um erro de permissão de chave KMS mestra não autorizada
- Solucione outros problemas comuns para produtores

Meu aplicativo produtor está gravando em um ritmo mais lento do que o esperado

Os motivos mais comuns para a taxa de transferência de gravação ser mais lenta do que o esperado são:

- Limites de serviço excedidos
- Quero otimizar meu produtor

Limites de serviço excedidos

Para descobrir se os limites do serviço estão sendo excedidos, verifique se seu produtor está lançando exceções de taxa de transferência do serviço e valide quais API operações estão sendo limitadas. Lembre-se de que há limites diferentes de acordo com a chamada, consulte Cotas e limites. Por exemplo, além dos limites de nível de estilhaço para gravações e leituras que são mais comumente conhecidas, há limites de nível de stream a seguir:

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

As operações `CreateStream`, `DeleteStream`, `ListStreams`, `GetShardIterator` e `MergeShards` são limitadas a 5 chamadas por segundo. A operação `DescribeStream` é limitada a 10 chamadas por segundo. A operação `DescribeStreamSummary` é limitada a 20 chamadas por segundo.

Se essas chamadas não forem o problema, selecione uma chave de partição que permita distribuir operações put uniformemente em todos os estilhaços e não tenha uma determinada chave de partição que esteja colidindo com os limites de serviço quando as restantes não estão. Isso requer que você meça a throughput de pico e leve em conta o número de estilhaços no seu stream. Para obter mais informações sobre o gerenciamento de streams, consulte Crie e gerencie streams de dados do Kinesis.

> **Tip**
>
> Lembre-se de arredondar para o quilobyte mais próximo para cálculos de limitação da taxa de transferência ao usar a operação de registro único PutRecord, enquanto a operação de vários registros PutRecordsarredonda a soma cumulativa dos registros em cada chamada.

Por exemplo, uma solicitação `PutRecords` com 600 registros com tamanho de 1,1 KB não serão aceleradas.

Quero otimizar meu produtor

Antes de começar a otimizar seu produtor, conclua as seguintes tarefas principais. Primeiro, identifique sua throughput de pico desejada em termos de tamanho do registro e registros por segundo. Em seguida, descarte a capacidade de stream conforme o fator de limitação (Limites de serviço excedidos). Se você excluiu a capacidade de stream, use as seguintes dicas de solução de problemas e diretrizes de otimização para os dois tipos comuns de produtores.

Produtor grande

Um grande produtor geralmente está executando a partir de um servidor local ou de uma EC2 instância da Amazon. Os clientes que precisam de uma throughput mais alta de um grande produtor normalmente se preocupam com a latência por registro. As estratégias para lidar com a latência incluem o seguinte: Se o cliente puder armazenar registros em microlote/buffer, use a Kinesis Producer Library (que tem lógica de agregação avançada), a operação de vários registros ou agregar registros em um arquivo maior antes de usar a operação PutRecordsde registro único. PutRecord Se não for possível criar microlotes ou armazenar registros em buffer, use vários threads para gravar no serviço Kinesis Data Streams ao mesmo tempo. Os AWS SDK for Java e outros SDKs incluem clientes assíncronos que podem fazer isso com muito pouco código.

Produtor pequeno

Um pequeno produtor geralmente é um aplicativo móvel, dispositivo IoT ou cliente web. Se for um aplicativo móvel, recomendamos usar a `PutRecords` operação ou o Kinesis Recorder no celular. AWS SDKs Para obter mais informações, consulte AWS Mobile SDK for Android Guia de introdução e AWS Mobile SDK for iOS Guia de introdução. Aplicativos móveis devem lidar com conexões intermitentes inerentemente e precisam de algum tipo de alocação em lote, como `PutRecords`. Se você não for capaz de alocar em lote por algum motivo, consulte as informações sobre Grande produtor acima. Se o seu produtor é um navegador, a quantidade de dados que está sendo gerada geralmente é muito pequena. No entanto, você está colocando as operações put no caminho crítico do aplicativo, o que não recomendamos.

Eu recebo um erro de permissão de chave KMS mestra não autorizada

Esse erro ocorre quando um aplicativo produtor grava em um fluxo criptografado sem permissões na chave KMS mestra. Para atribuir permissões a um aplicativo para acessar uma KMS chave, consulte Usando políticas de chaves em AWS KMS e Usando IAM políticas com AWS KMS.

Solucione outros problemas comuns para produtores

- Why is my Kinesis data stream returning a 500 Internal Server Error?
- How do I troubleshoot timeout errors when writing from Flink to Kinesis Data Streams?
- How do I troubleshoot throttling errors in Kinesis Data Streams?
- Why is my Kinesis data stream throttling?
- Como posso colocar registros de dados em um stream de dados do Kinesis usando o? KPL

Otimize os produtores do Kinesis Data Streams

Você pode otimizar ainda mais seus produtores do Amazon Kinesis Data Streams, dependendo do comportamento específico que você vê. Analise os tópicos a seguir para identificar soluções.

Tópicos

- Personalize KPL novas tentativas e o comportamento do limite de taxa
- Aplique as melhores práticas à KPL agregação

Personalize KPL novas tentativas e o comportamento do limite de taxa

Quando você adiciona registros de usuário da Kinesis Producer Library (KPL) usando a KPL `addUserRecord()` operação, um registro recebe um registro de data e hora e é adicionado a um buffer com um prazo definido pelo parâmetro de configuração. `RecordMaxBufferedTime` Essa combinação time stamp/prazo define a prioridade do buffer. Os registros são descarregados do buffer com base nos seguintes critérios:

- Prioridade do buffer
- Configuração de agregação
- Configuração de coleta

Os parâmetros de configuração de coleta e agregação que afetam o comportamento do buffer são os seguintes:

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

Os registros liberados são então enviados para seu stream de dados do Kinesis como registros do Amazon Kinesis Data Streams usando uma chamada para a operação do Kinesis Data Streams. API `PutRecords` A operação `PutRecords` envia solicitações ao stream que ocasionalmente apresenta falhas parciais ou totais. Os registros que falham são automaticamente adicionados de volta ao KPL buffer. O novo prazo é definido com base no mínimo destes dois valores:

- Metade da configuração de `RecordMaxBufferedTime` atual
- O time-to-live valor do registro

Essa estratégia permite que registros repetidos de KPL usuários sejam incluídos em chamadas subsequentes do Kinesis Data API Streams, para melhorar a taxa de transferência e reduzir a complexidade, ao mesmo tempo em que reforça o valor do registro do Kinesis Data Streams. time-to-live Não há um algoritmo de recuo, tornando essa uma estratégia de tentativa relativamente agressiva. O spam devido ao excesso de tentativas é evitado pela limitação de taxas, discutida na próxima seção.

Limitação de taxa

KPLIsso inclui um recurso de limitação de taxa, que limita a taxa de transferência por fragmento enviada por um único produtor. A limitação de taxas é implementada usando um algoritmo de bucket de token com buckets separados para bytes e registros do Kinesis Data Streams. Cada gravação bem-sucedida em um fluxo de dados do Kinesis adiciona um token (ou vários tokens) a cada bucket, até um determinado limite. Esse limite é configurável, mas por padrão é definido como 50% maior que o limite de estilhaço real, para permitir a saturação de estilhaços a partir de um único produtor.

Você pode reduzir esse limite para diminuir o spam devido ao excesso de tentativas. No entanto, a melhor prática é que cada produtor tente novamente para uma throughput máxima de maneira agressiva e lide com qualquer limitação resultante determinada como excessiva por meio da

expansão da capacidade do stream e da implementação de uma estratégia de chave de partição apropriada.

Aplique as melhores práticas à KPL agregação

Embora o esquema de números de sequência dos registros resultantes do Amazon Kinesis Data Streams permaneça o mesmo, a agregação faz com que a indexação dos registros de usuário da Kinesis Producer KPL Library () contidos em um registro agregado do Kinesis Data Streams comece em 0 (zero); no entanto, desde que você não confie em números de sequência para KPL identificar exclusivamente seus registros de usuário, seu código pode ignorar isso, pois o agregação (de seus registros de usuário em um registro do Kinesis Data Streams) e posterior desagregação (de um KPL registro do Kinesis Data Streams em seu KPLregistros de usuário) cuida automaticamente disso para você. Isso se aplica se seu consumidor estiver usando AWS SDK o. KCL Para usar essa funcionalidade de agregação, você precisará inserir a parte Java do KPL em sua compilação se seu consumidor for escrito usando o API fornecido no. AWS SDK

Se você pretende usar números de sequência como identificadores exclusivos para seus registros de KPL usuário, recomendamos que você use o cumprimento do contrato e as `public boolean equals(Object obj)` operações fornecidas em `public int hashCode()` `Record` e `UserRecord` para permitir a comparação de seus registros de usuário. KPL Além disso, se quiser examinar o número subsequente do seu registro de KPL usuário, você pode convertê-lo em uma `UserRecord` instância e recuperar o número subsequente.

Para obter mais informações, consulte Implemente a desagregação de consumidores.

Leia dados do Amazon Kinesis Data Streams

Um consumidor é uma aplicação que processa todos os dados de um fluxo de dados do Kinesis. Quando um consumidor usa distribuição avançada, ele recebe sua própria alocação de throughput de leitura de 2 MB/s permitindo que vários consumidores leiam dados do mesmo streaming em paralelo, sem disputa com outros consumidores por throughput de leitura. Para usar o recurso de distribuição avançada de estilhaços, consulte Desenvolva consumidores personalizados com taxa de transferência dedicada (distribuição aprimorada).

Por padrão, os estilhaços em um streaming fornecem 2 MB/s de throughput de leitura por estilhaço. Essa throughput é compartilhada entre todos os consumidores que fazem a leitura a partir de um determinado estilhaço. Em outras palavras, o padrão de 2 MB/s de throughput por estilhaço é fixo, ainda que haja vários consumidores fazendo a leitura pelo estilhaço. Para usar essa throughput padrão de estilhaços, consulte Desenvolva consumidores personalizados com taxa de transferência compartilhada.

A tabela a seguir compara a throughput padrão para a distribuição avançada. O atraso na propagação da mensagem é definido como o tempo gasto em milissegundos para que uma carga útil enviada usando o despacho de carga útil (como e) chegue ao aplicativo consumidor por meio da carga útil consumidora APIs (como PutRecord e PutRecords). APIs GetRecords SubscribeToShard

Características	Consumidores não registrados sem distribuição avançada	Consumidores registrados com distribuição avançada
Throughput de leitura de estilhaço	Corrigida em um total de 2 MB/s por estilhaço. Se houver vários consumidores lendo a partir do mesmo estilhaço, todos eles compartilham essa throughput. A soma das taxas de transferência que eles recebem do estilhaço não excede 2 MB/s.	Dimensionada de acordo com o registro dos consumidores para usar a distribuição avançada. Cada consumidor registrado para usar a distribuição avançada recebe sua própria throughput de leitura por estilhaço, de até 2 MB/s, independentemente de outros consumidores.
Atraso de propagação da mensagem	Uma média de cerca de 200 ms se você tiver um consumidor lendo no stream. Essa média chega até	Normalmente, uma média de 70 ms se você tiver um ou cinco consumidores.

Características	Consumidores não registrados sem distribuição avançada	Consumidores registrados com distribuição avançada
	cerca de 1000 ms se você tiver cinco consumidores.	
Custo	N/D	Há um custo de recuperação de dados e um custo de hora de estilhaço por consumidor. Para obter mais informações, consulte Definição de preço do Amazon Kinesis Data Streams.
Registro de modelo de entrega	Pare de HTTP usar o modelo GetRecords.	O Kinesis Data Streams envia os registros para você HTTP mais de /2 usando o. SubscribeToShard

Tópicos

- Use o Visualizador de dados no console do Kinesis
- Consulte seus streams de dados no console do Kinesis
- Desenvolva consumidores usando AWS Lambda
- Desenvolva consumidores usando o Amazon Managed Service para Apache Flink
- Desenvolva consumidores usando o Amazon Data Firehose
- Use a biblioteca de cliente Kinesis
- Desenvolva consumidores personalizados com taxa de transferência compartilhada
- Desenvolva consumidores personalizados com taxa de transferência dedicada (distribuição aprimorada)
- Migre os consumidores da KCL versão 1.x para a 2.x KCL
- Leia dados do Kinesis Data Streams AWS usando outros serviços
- Leia o Kinesis Data Streams usando integrações de terceiros
- Solucionar problemas dos consumidores do Kinesis Data Streams
- Otimize os consumidores do Amazon Kinesis Data Streams

Use o Visualizador de dados no console do Kinesis

O Visualizador de dados no Kinesis Management Console permite que você visualize registros de dados dentro do fragmento especificado do seu stream de dados sem precisar desenvolver um aplicativo consumidor. Para usar o visualizador de dados, siga estas etapas:

1. Faça login AWS Management Console e abra o console do Kinesis em https://console.aws.amazon.com/kinesis.
2. Escolha o fluxo de dados ativo cujos registros você deseja visualizar e, em seguida, escolha a guia Visualizador de dados.
3. Na guia Visualizador de dados do fluxo de dados ativo selecionado, escolha o fragmento cujos registros você deseja visualizar, escolha a Posição inicial e clique em Obter registros. Você pode definir a posição inicial como um dos seguintes valores:

   - No número de sequência: mostra os registros da posição indicada pelo número de sequência especificado no campo de número de sequência.
   - Depois do número de sequência: mostra os registros imediatamente após a posição indicada pelo número de sequência especificado no campo de número de sequência.
   - No carimbo de data/hora: mostra os registros da posição indicada pelo timestamp especificado no campo de timestamp.
   - Horizonte de corte: mostra os registros no último registro não cortado no fragmento, que é o registro de dados mais antigo no fragmento.
   - Mais recentes: mostra os registros imediatamente após o registro mais recente no fragmento, para que você sempre leia os dados mais recentes no fragmento.

     Os registros de dados gerados que correspondem ao ID do fragmento e à posição inicial especificados são exibidos em uma tabela de registros no console. São exibidos no máximo 50 registros de cada vez. Para visualizar o próximo conjunto de registros, clique no botão Próximo.
4. Clique em qualquer registro individual para visualizar a carga útil do registro em dados brutos ou JSON formato em uma janela separada.

Observe que quando você clica em Obter registros ou nos botões Avançar no Visualizador de Dados, isso invoca o GetRecordsAPIe isso se aplica ao GetRecordsAPIlimite de 5 transações por segundo.

Consulte seus streams de dados no console do Kinesis

A guia Análise de dados no console do Kinesis Data Streams permite que você consulte seus streams de dados usando. SQL Para usar esse recurso, siga estas etapas:

1. Faça login AWS Management Console e abra o console do Kinesis em https://console.aws.amazon.com/kinesis.
2. Escolha o fluxo de dados ativo com o qual você deseja consultar SQL e, em seguida, escolha a guia Análise de dados.
3. Na guia Análise de dados, você pode realizar a inspeção e a visualização do stream com um notebook gerenciado do Apache Flink Studio. Você pode realizar SQL consultas ad-hoc para inspecionar seu fluxo de dados e visualizar os resultados em segundos usando o Apache Zeppelin. Na guia Análise de dados, escolha Eu concordo e, em seguida, escolha Criar caderno para criar um caderno.
4. Depois que o notebook for criado, escolha Abrir no Apache Zeppelin. Isso abrirá seu caderno em uma nova guia. Um notebook é uma interface interativa na qual você pode enviar suas SQL consultas. Escolha a nota que contém o nome do seu stream.
5. Você verá uma nota com um exemplo de `SELECT` consulta para gerar os dados no stream já em execução. Isso permite que você visualize o esquema do seu fluxo de dados.
6. Para testar outras consultas, como janelas giratórias ou deslizantes, escolha Exibir exemplos de consultas na guia Análise de dados. Copie a consulta, modifique-a de acordo com seu esquema de fluxo de dados e, em seguida, execute-a em um novo parágrafo em sua nota do Zeppelin.

Desenvolva consumidores usando AWS Lambda

Você pode usar uma AWS Lambda função para processar registros em um fluxo de dados. AWS Lambda é um serviço de computação que permite executar código sem provisionar ou gerenciar servidores. Ele executa seu código somente quando necessário e escala automaticamente, de algumas solicitações por dia a milhares por segundo. Você paga apenas pelo tempo de computação consumido. Não haverá cobranças quando seu código não estiver em execução. Com AWS Lambda, você pode executar código para praticamente qualquer tipo de aplicativo ou serviço de back-end, tudo sem nenhuma administração. Ele executa seu código em uma infraestrutura de computação de alta disponibilidade e executa toda a administração dos recursos computacionais, inclusive manutenção de servidor e sistema operacional, provisionamento de capacidade e escalabilidade

automática, monitoramento do código e registro em log. Para obter mais informações, consulte Como usar AWS Lambda com o Amazon Kinesis.

Para obter informações sobre solução de problemas, consulte Por que o gatilho do Kinesis Data Streams não consegue invocar minha função do Lambda?

Desenvolva consumidores usando o Amazon Managed Service para Apache Flink

Você pode usar um aplicativo Amazon Managed Service para Apache Flink para processar e analisar dados em um stream do Kinesis usando Java ou SQL Scala. As aplicações do Managed Service for Apache Flink podem enriquecer os dados usando fontes de referência, agregar dados ao longo do tempo ou usar machine learning para encontrar anomalias nos dados. Em seguida, você pode gravar os resultados da análise em outro stream do Kinesis, em um stream de entrega do Firehose ou em uma função Lambda. Para obter mais informações, consulte o Guia do desenvolvedor do Managed Service for Apache Flink para SQL aplicativos ou o Guia do desenvolvedor do Managed Service for Apache Flink Applications.

Desenvolva consumidores usando o Amazon Data Firehose

Você pode usar um Firehose para ler e processar registros de um stream do Kinesis. O Firehose é um serviço totalmente gerenciado para fornecer dados de streaming em tempo real para destinos como Amazon S3, Amazon Redshift, Amazon OpenSearch Service e Splunk. O Firehose também oferece suporte a qualquer HTTP endpoint ou HTTP endpoints personalizados de propriedade de provedores de serviços terceirizados compatíveis, incluindo Datadog, MongoDB e New Relic. Você também pode configurar o Firehose para transformar seus registros de dados e converter o formato do registro antes de entregar os dados ao destino. Para obter mais informações, consulte Gravando no Firehose usando o Kinesis Data Streams.

Use a biblioteca de cliente Kinesis

Um dos métodos de desenvolvimento de aplicativos de consumo personalizados que podem processar dados de fluxos de KDS dados é usar a Kinesis Client Library KCL ().

Tópicos

- O que é a Kinesis Client Library?

- KCLversões disponíveis
- Conceitos KCL
- Use uma tabela de leasing para rastrear os fragmentos processados pelo aplicativo consumidor KCL
- Processe vários fluxos de dados com o mesmo KCL 2.x para o aplicativo consumidor Java
- Use o KCL com o Registro do AWS Glue Esquema

> **Note**
>
> Para KCL 1.x e KCL 2.x, é recomendável que você atualize para a versão KCL 1.x ou KCL 2.x mais recente, dependendo do seu cenário de uso. Tanto o KCL 1.x quanto o KCL 2.x são atualizados regularmente com versões mais recentes que incluem os patches de dependência e segurança mais recentes, correções de bugs e novos recursos compatíveis com versões anteriores. Para obter mais informações, consulte https://github.com/awslabs/amazon-kinesis-client/releases.

O que é a Kinesis Client Library?

KCLajuda você a consumir e processar dados de um stream de dados do Kinesis ao cuidar de muitas das tarefas complexas associadas à computação distribuída. Isso inclui balanceamento de carga em várias instâncias de aplicações de consumo, resposta a falhas nas instâncias de aplicações de consumo, verificação de registros processados e reação à refragmentação. O KCL cuida de todas essas subtarefas para que você possa concentrar seus esforços na criação de sua lógica personalizada de processamento de registros.

Isso KCL é diferente dos Kinesis APIs Data Streams que estão disponíveis no. AWS SDKs O Kinesis APIs Data Streams ajuda você a gerenciar muitos aspectos do Kinesis Data Streams, incluindo a criação de streams, a refragmentação e a colocação e obtenção de registros. O KCL fornece uma camada de abstração em torno de todas essas subtarefas, especificamente para que você possa se concentrar na lógica de processamento de dados personalizada do seu aplicativo consumidor. Para obter informações sobre o Kinesis API Data Streams, consulte a Referência do Amazon API Kinesis.

> **Important**
>
> KCLÉ uma biblioteca Java. Support para linguagens diferentes de Java é fornecido usando uma interface multilíngue chamada de MultiLangDaemon. Esse daemon é baseado em Java

e é executado em segundo plano quando você está usando uma KCL linguagem diferente de Java. Por exemplo, se você instalar o KCL for Python e escrever seu aplicativo de consumidor inteiramente em Python, você ainda precisará do Java instalado em seu sistema por causa do. MultiLangDaemon Além disso, MultiLangDaemon tem algumas configurações padrão que talvez você precise personalizar para seu caso de uso, por exemplo, a AWS região à qual ele se conecta. Para obter mais informações sobre o MultiLangDaemon on GitHub, consulte KCL MultiLangDaemon projeto.

Ele KCL atua como intermediário entre sua lógica de processamento de registros e o Kinesis Data Streams. A KCL executa as seguintes tarefas:

- Conecta-se ao fluxo de dados
- Enumera os fragmentos no fluxo de dados
- Usa arrendamentos para coordenar associações de fragmentos com seus trabalhadores
- Cria uma instância de um processador de registro para cada estilhaço que gerencia
- Extrai registros de dados do fluxo de dados
- Envia os registros ao processador de registros correspondente
- Registros processados pelos pontos de verificação
- Equilibra as associações (concessões) entre fragmentos e operadores quando a contagem de instâncias de operador muda ou quando o fluxo de dados é refragmentado (os fragmentos são divididos ou mesclados)

KCLversões disponíveis

Atualmente, você pode usar qualquer uma das seguintes versões compatíveis do KCL para criar seus aplicativos de consumo personalizados:

- KCL1.x

  Para ter mais informações, consulte Desenvolva KCL consumidores 1.x.

- KCL2. x

  Para ter mais informações, consulte Desenvolva KCL consumidores 2.x.

Você pode usar KCL 1.x ou KCL 2.x para criar aplicativos de consumo que usam taxa de transferência compartilhada. Para ter mais informações, consulte Desenvolva consumidores personalizados com taxa de transferência compartilhada usando KCL.

Para criar aplicativos de consumo que usem taxa de transferência dedicada (consumidores de fan-out aprimorados), você só pode usar 2.x. KCL Para ter mais informações, consulte Desenvolva consumidores personalizados com taxa de transferência dedicada (distribuição aprimorada).

Para obter informações sobre as diferenças entre KCL 1.x e KCL 2.x e instruções sobre como migrar de KCL 1.x para KCL 2.x, consulte. Migre os consumidores da KCL versão 1.x para a 2.x KCL

Conceitos KCL

- KCLaplicativo de consumidor — um aplicativo que é personalizado usando KCL e projetado para ler e processar registros de fluxos de dados.
- Instância do aplicativo KCL consumidor - os aplicativos do consumidor são normalmente distribuídos, com uma ou mais instâncias do aplicativo sendo executadas simultaneamente para coordenar as falhas e equilibrar dinamicamente a carga do processamento do registro de dados.
- Worker — uma classe de alto nível que uma instância KCL do aplicativo consumidor usa para começar a processar dados.

> **Important**
>
> Cada instância KCL do aplicativo consumidor tem um trabalhador.

O operador inicializa e supervisiona várias tarefas, incluindo a sincronização de informações de fragmentos e concessões, o monitoramento de atribuições de fragmentos e o processamento dos dados dos fragmentos. Um trabalhador KCL fornece as informações de configuração do aplicativo consumidor, como o nome do fluxo de dados cujos registros de dados esse aplicativo KCL consumidor processará e AWS as credenciais necessárias para acessar esse fluxo de dados. O trabalhador também inicia essa instância específica KCL do aplicativo do consumidor para entregar registros de dados do fluxo de dados aos processadores de registros.

> **Important**
>
> Em KCL 1.x, essa classe é chamada de Worker. Para obter mais informações (esses são os KCL repositórios Java), consulte https://github.com/awslabs/amazon-kinesis-client/blob/

v1.x/src/main/java/com/amazonaws/services/kinesis/clientlibrary/lib/worker/worker.java. Na KCL versão 2.x, essa classe é chamada de Scheduler. O propósito do Scheduler em KCL 2.x é idêntico ao propósito do Worker em KCL 1.x. Para obter mais informações sobre a classe Scheduler na KCL versão 2.x, consulte https://github.com/awslabs/amazon-kinesis-client/blob/master/ amazon-kinesis-client /src/main/java/software/amazon/kinesis/coordinator/Scheduler.java.

- Concessão: dado que define a ligação entre um operador e um fragmento. Aplicativos de KCL consumo distribuídos usam concessões para particionar o processamento de registros de dados em uma frota de trabalhadores. A qualquer momento, cada fragmento de registros de dados é vinculado a um determinado trabalhador por uma locação identificada pela leaseKeyvariável.

  Por padrão, um trabalhador pode manter um ou mais arrendamentos (sujeito ao valor da variável maxLeasesForWorker) ao mesmo tempo.

  > **Important**
  >
  > Os operadores competem para manter todas as concessões disponíveis para todos os fragmentos disponíveis em um fluxo de dados. Mas apenas um operador consegue manter uma concessão de cada vez.

  Por exemplo, se você tiver uma instância da aplicação de consumo A com o operador A que está processando um fluxo de dados com quatro fragmentos, o operador A poderá reter as concessões aos fragmentos 1, 2, 3 e 4 ao mesmo tempo. Mas se você tiver duas instâncias de aplicações de consumo A e B com os operadores A e B, e essas instâncias estiverem processando um fluxo de dados com quatro fragmentos, o operador A e o operador B não poderão reter a concessão ao fragmento 1 ao mesmo tempo. Um operador retém a concessão a um fragmento específico até estar pronto para parar de processar os registros de dados do fragmento ou até que uma falha ocorra. Quando um operador libera a concessão, outro operador a assume e a retém.

  Para obter mais informações (esses são os KCL repositórios Java), consulte https://github.com/awslabs/amazon-kinesis-client/blob/v1.x/src/main/java/com/amazonaws/services/kinesis/leases/impl/lease.java para 1.x e /blob/master/ /src/main/java/software/amazon/kinesis/leases/Lease.java para 2.x. KCL https://github.com/awslabs/ amazon-kinesis-client amazon-kinesis-client KCL

- Tabela de lease — uma tabela exclusiva do Amazon DynamoDB usada para acompanhar os fragmentos KDS em um stream de dados que estão sendo alugados e processados pelos

trabalhadores do aplicativo consumidor. KCL A tabela de leasing deve permanecer sincronizada (dentro de um trabalhador e entre todos os trabalhadores) com as informações mais recentes do fragmento do fluxo de dados enquanto o aplicativo do KCL consumidor está em execução. Para ter mais informações, consulte Use uma tabela de leasing para rastrear os fragmentos processados pelo aplicativo consumidor KCL.

- Processador de registros — a lógica que define como seu aplicativo KCL consumidor processa os dados obtidos dos fluxos de dados. Em tempo de execução, uma instância KCL do aplicativo consumidor instancia um trabalhador, e esse trabalhador instancia um processador de registros para cada fragmento ao qual ele tem uma concessão.

Use uma tabela de leasing para rastrear os fragmentos processados pelo aplicativo consumidor KCL

Tópicos

O que é uma tabela de locação

Para cada KCL aplicativo do Amazon Kinesis Data Streams, usa uma tabela de lease exclusiva (armazenada em uma tabela do Amazon DynamoDB) para acompanhar os fragmentos KDS em um stream de dados que estão sendo alugados e processados pelos trabalhadores do aplicativo consumidor. KCL

> **Important**
>
> KCLusa o nome do aplicativo consumidor para criar o nome da tabela de concessão que esse aplicativo consumidor usa, portanto, cada nome de aplicativo consumidor deve ser exclusivo.

Você pode visualizar a tabela usando o console do Amazon DynamoDB enquanto a aplicação de consumo está em execução.

Se a tabela de concessão do seu aplicativo KCL consumidor não existir quando o aplicativo for inicializado, um dos trabalhadores criará a tabela de concessão para esse aplicativo.

> **Important**
>
> Sua conta é cobrada pelos custos associados à tabela do DynamoDB, além dos custos associados ao próprio Kinesis Data Streams.

Cada linha na tabela de concessões representa um fragmento que está sendo processado pelos operadores da aplicação de consumo. Se seu aplicativo KCL consumidor processa somente um fluxo de dados, `leaseKey` qual é a chave de hash da tabela de leasing é o ID do fragmento. Se você éProcesse vários fluxos de dados com o mesmo KCL 2.x para o aplicativo consumidor Java, então a estrutura do leaseKey se parece com esta:`account-id:StreamName:streamCreationTimestamp:ShardId`. Por exemplo, `111111111:multiStreamTest-1:12345:shardId-000000000336`.

Além do ID do estilhaço, cada linha também inclui os seguintes dados:

- checkpoint: número de sequência do ponto de verificação mais recente do estilhaço. Esse valor é exclusivo entre todos os fragmentos no fluxo de dados.
- checkpointSubSequenceNúmero: ao usar o recurso de agregação da Kinesis Producer Library, essa é uma extensão do ponto de verificação que rastreia registros individuais de usuários dentro do registro do Kinesis.
- leaseCounter: Usado para o controle de versão da locação para que os trabalhadores possam detectar que a locação foi contratada por outro trabalhador.
- leaseKey: Um identificador exclusivo para uma locação. Cada concessão é específica a um fragmento no fluxo de dados e é retida por um operador por vez.
- leaseOwner: O trabalhador que está detendo esse contrato.
- ownerSwitchesSincePonto de verificação: Quantas vezes esse contrato de arrendamento mudou de trabalhador desde a última vez que um posto de controle foi escrito.
- parentShardId: usado para garantir que o fragmento principal seja totalmente processado antes do início do processamento nos fragmentos secundários. Isso garante que os registros sejam processados na mesma ordem em que foram colocados no stream.

- hashrange: usado pelo `PeriodicShardSyncManager` para executar sincronizações periódicas a fim de encontrar fragmentos ausentes na tabela de concessões e criar concessões para eles, se necessário.

  > **Note**
  >
  > Esses dados estão presentes na tabela de leasing para cada fragmento começando com KCL 1.14 e 2.3. KCL Para obter mais informações sobre `PeriodicShardSyncManager` e a sincronização periódica entre concessões e fragmentos, consulte Como uma tabela de leasing é sincronizada com os fragmentos em um stream de dados do Kinesis.

- childshards: usado por `LeaseCleanupManager` para revisar o status de processamento do fragmento filho e decidir se o fragmento pai pode ser excluído da tabela de concessões.

  > **Note**
  >
  > Esses dados estão presentes na tabela de leasing para cada fragmento começando com KCL 1.14 e 2.3. KCL

- shardID: o ID do fragmento.

  > **Note**
  >
  > Esse dado só estará presente na tabela de concessões se você estiver Processe vários fluxos de dados com o mesmo KCL 2.x para o aplicativo consumidor Java. Isso só é suportado no KCL 2.x para Java, começando com KCL 2.3 para Java e versões posteriores.

- stream name o identificador do fluxo de dados no formato `account-id:StreamName:streamCreationTimestamp`.

  > **Note**
  >
  > Esse dado só estará presente na tabela de concessões se você estiver Processe vários fluxos de dados com o mesmo KCL 2.x para o aplicativo consumidor Java. Isso só é suportado no KCL 2.x para Java, começando com KCL 2.3 para Java e versões posteriores.

Throughput

Se sua aplicação do Amazon Kinesis Data Streams receber exceções de throughput provisionada, você deverá aumentar a throughput provisionada para a tabela do DynamoDB. Ele KCL cria a tabela com uma taxa de transferência provisionada de 10 leituras por segundo e 10 gravações por segundo, mas isso pode não ser suficiente para seu aplicativo. Por exemplo, se sua aplicação do Amazon Kinesis Data Streams definir pontos de verificação ou usar operadores com frequência em um fluxo de dados composto por vários fragmentos, você poderá precisar de uma throughput maior.

Para obter informações sobre a throughput provisionada no DynamoDB, consulte Modo de capacidade de leitura/gravação e Trabalhar com tabelas e dados no DynamoDB no Guia do desenvolvedor do Amazon DynamoDB.

Como uma tabela de leasing é sincronizada com os fragmentos em um stream de dados do Kinesis

Os trabalhadores em aplicativos de KCL consumo usam concessões para processar fragmentos de um determinado fluxo de dados. As informações sobre qual operador usa a concessão a um fragmento em um momento determinado são armazenadas em uma tabela de concessões. A tabela de leasing deve permanecer sincronizada com as informações mais recentes do fragmento do stream de dados enquanto o aplicativo KCL consumidor estiver em execução. KCLsincroniza a tabela de leasing com as informações de fragmentos adquiridas do serviço Kinesis Data Streams durante a inicialização do aplicativo consumidor (quando o aplicativo consumidor é inicializado ou reiniciado) e também sempre que um fragmento que está sendo processado chega ao fim (refragmentação). Em outras palavras, os trabalhadores ou um aplicativo KCL consumidor são sincronizados com o fluxo de dados que estão processando durante a inicialização inicial do aplicativo consumidor e sempre que o aplicativo consumidor encontra um evento de refragmentação do fluxo de dados.

Tópicos

- Sincronização em KCL 1.0 - 1.13 e KCL 2.0 - 2.2
- Sincronização em KCL 2.x, começando com KCL 2.3 e versões posteriores
- Sincronização em KCL 1.x, começando com KCL 1.14 e versões posteriores

Sincronização em KCL 1.0 - 1.13 e KCL 2.0 - 2.2

Nas versões KCL 1.0 - 1.13 e KCL 2.0 - 2.2, durante a inicialização do aplicativo consumidor e também durante cada evento de refragmentação do stream de dados, KCL sincroniza a tabela de

leasing com as informações de fragmentos adquiridas do serviço Kinesis Data Streams invocando a ou a descoberta. `ListShards` `DescribeStream` APIs Em todas as KCL versões listadas acima, cada trabalhador de um aplicativo KCL consumidor conclui as seguintes etapas para realizar o processo de sincronização de lease/shard durante a inicialização do aplicativo consumidor e em cada evento de refragmentação de stream:

- Busca todos os fragmentos de dados do fluxo sendo processado
- Busca todas as concessões do fragmento da tabela de concessões
- Filtra cada fragmento aberto sem uma concessão na tabela de concessões
- Itera em todos os fragmentos abertos encontrados e, para cada fragmento aberto sem pai aberto:
  - Percorre a árvore hierárquica no caminho dos ancestrais para determinar se o fragmento é um descendente. Um fragmento será considerado descendente se um fragmento ancestral estiver sendo processado (a entrada de concessão do fragmento ancestral existe na tabela de concessões) ou se houver um fragmento ancestral que deve ser processado (por exemplo, a posição inicial é `TRIM_HORIZON` ou `AT_TIMESTAMP`).
  - Se o fragmento aberto no contexto for descendente, KCL verifique o fragmento com base na posição inicial e cria concessões para seus pais, se necessário

Sincronização em KCL 2.x, começando com KCL 2.3 e versões posteriores

Começando com as versões mais recentes suportadas da KCL 2.x (KCL2.3) e posteriores, a biblioteca agora oferece suporte às seguintes alterações no processo de sincronização. Essas mudanças na sincronização de lease/fragmento reduzem significativamente o número de API chamadas feitas por aplicativos de KCL consumo para o serviço Kinesis Data Streams e otimizam o gerenciamento de leasing em seu aplicativo de consumidor. KCL

- Durante a inicialização do aplicativo, se a tabela de concessão estiver vazia, KCL utiliza a opção de filtragem (o parâmetro `ListShard` API de solicitação `ShardFilter` opcional) para recuperar e criar concessões somente para um instantâneo dos fragmentos abertos no momento especificado pelo parâmetro. `ShardFilter` O `ShardFilter` parâmetro permite filtrar a resposta do `ListShards`API. A única propriedade obrigatória do parâmetro `ShardFilter` é `Type`. KCLusa a propriedade de `Type` filtro e os seguintes valores válidos para identificar e retornar um instantâneo dos fragmentos abertos que podem exigir novas concessões:
  - `AT_TRIM_HORIZON`: a resposta inclui todos os fragmentos abertos em`TRIM_HORIZON`.
  - `AT_LATEST`: a resposta inclui somente os fragmentos do fluxo de dados abertos no momento.

- `AT_TIMESTAMP`: a resposta inclui todos os fragmentos com timestamp inicial menor ou igual ao timestamp fornecido e timestamp final maior ou igual ao timestamp fornecido ou ainda abertos.

`ShardFilter` é usado ao criar uma concessão em uma tabela de concessões vazia para inicializar concessões para um instantâneo dos fragmentos especificados em `RetrievalConfig#initialPositionInStreamExtended`.

Para obter mais informações sobre o `ShardFilter`, consulte https://docs.aws.amazon.com/kinesis/latest/APIReference/API_ShardFilter.html.

- Em vez de ter todos os operadores realizando a sincronização de concessão/fragmento para manter a tabela de concessões atualizada com os fragmentos mais recentes no fluxo de dados, um único operador é eleito como líder para executar a sincronização.
- KCL2.3 usa o parâmetro de `ChildShards` retorno do `GetRecords` e do `SubscribeToShard` APIs para realizar a sincronização de arrendamento/fragmento que acontece em `SHARD_END` para fragmentos fechados, permitindo que um KCL trabalhador crie concessões somente para os fragmentos secundários do fragmento que ele concluiu o processamento. Para aplicativos compartilhados em todos os consumidores, essa otimização da sincronização de arrendamento/fragmento usa o parâmetro do. `ChildShards` `GetRecords` API Para aplicativos de consumo de taxa de transferência dedicada (fan-out aprimorado), essa otimização da sincronização de arrendamento/fragmento usa o parâmetro do. `ChildShards` `SubscribeToShard` API Para obter mais informações GetRecords, consulte SubscribeToShards, ChildSharde.
- Com as mudanças acima, o comportamento de KCL está passando do modelo de todos os trabalhadores aprendendo sobre todos os fragmentos existentes para o modelo de trabalhadores aprendendo apenas sobre os fragmentos dos filhos que cada trabalhador possui. Portanto, além da sincronização que ocorre durante a inicialização do aplicativo do consumidor e os eventos de refragmentação, KCL agora também realiza varreduras periódicas adicionais de fragmentos/concessões para identificar possíveis lacunas na tabela de leasing (em outras palavras, para aprender sobre todos os novos fragmentos) para garantir que o intervalo de hash completo do fluxo de dados esteja sendo processado e crie concessões para eles, se necessário. `PeriodicShardSyncManager`é o componente responsável pela execução periódica de varreduras de arrendamento/fragmento.

  Para obter mais informações sobre `PeriodicShardSyncManager` a versão KCL 2.3, consulte https://github.com/awslabs/amazon-kinesis-client/blob/master/ /src/main/java/software/amazon/kinesis/leases/ amazon-kinesis-client LeaseManagementConfig .java #L201 -L213.

Na KCL versão 2.3, novas opções de configuração estão disponíveis para configuração `PeriodicShardSyncManager` em`LeaseManagementConfig`:

Nome	Valor padrão	Descrição
leasesRecoveryAuditorExecutionFrequencyMillis	120.000 (2 minutos)	Frequência (em milissegundos) do trabalho do auditor para verificar concessões parciais na tabela de concessões. Se detectar alguma falha nas concessões de um fluxo, o auditor acionará a sincronização de fragmentos com base em `leasesRecoveryAuditorInconsistencyConfidenceThreshold` .

Nome	Valor padrão	Descrição
leasesRecoveryAuditorInconsistencyConfidenceThreshold	3	Limite de confiança no trabalho periódico do auditor para determinar se as concessões de um fluxo de dados na tabela de concessões são inconsistentes. Se encontrar consecutivamente o mesmo conjunto de inconsistências em um fluxo de dados pelo número de vezes definido, o auditor acionará uma sincronização de fragmentos.

Agora, novas CloudWatch métricas também são emitidas para monitorar a integridade do`PeriodicShardSyncManager`. Para ter mais informações, consulte PeriodicShardSyncManager.

- Inclui uma otimização de `HierarchicalShardSyncer` para criar apenas concessões em uma camada de fragmentos.

Sincronização em KCL 1.x, começando com KCL 1.14 e versões posteriores

Começando com as versões mais recentes suportadas da KCL 1.x (KCL1.14) e posteriores, a biblioteca agora oferece suporte às seguintes alterações no processo de sincronização. Essas mudanças na sincronização de lease/fragmento reduzem significativamente o número de API chamadas feitas por aplicativos de KCL consumo para o serviço Kinesis Data Streams e otimizam o gerenciamento de leasing em seu aplicativo de consumidor. KCL

- Durante a inicialização do aplicativo, se a tabela de concessão estiver vazia, KCL utiliza a opção de filtragem (o parâmetro `ListShard` API de solicitação `ShardFilter` opcional) para recuperar e criar concessões somente para um instantâneo dos fragmentos abertos no momento especificado pelo parâmetro. `ShardFilter` O `ShardFilter` parâmetro permite filtrar a resposta do `ListShards`API. A única propriedade obrigatória do parâmetro `ShardFilter` é `Type`. KCLusa a propriedade de `Type` filtro e os seguintes valores válidos para identificar e retornar um instantâneo dos fragmentos abertos que podem exigir novas concessões:
  - `AT_TRIM_HORIZON`: a resposta inclui todos os fragmentos abertos em`TRIM_HORIZON`.
  - `AT_LATEST`: a resposta inclui somente os fragmentos do fluxo de dados abertos no momento.
  - `AT_TIMESTAMP`: a resposta inclui todos os fragmentos com timestamp inicial menor ou igual ao timestamp fornecido e timestamp final maior ou igual ao timestamp fornecido ou ainda abertos.

  `ShardFilter` é usado ao criar uma concessão em uma tabela de concessões vazia para inicializar concessões para um instantâneo dos fragmentos especificados em `KinesisClientLibConfiguration#initialPositionInStreamExtended`.

  Para obter mais informações sobre o `ShardFilter`, consulte https://docs.aws.amazon.com/kinesis/latest/APIReference/API_ShardFilter.html.
- Em vez de ter todos os operadores realizando a sincronização de concessão/fragmento para manter a tabela de concessões atualizada com os fragmentos mais recentes no fluxo de dados, um único operador é eleito como líder para executar a sincronização.
- KCL1.14 usa o parâmetro de `ChildShards` retorno do `GetRecords` e do `SubscribeToShard` APIs para realizar a sincronização de arrendamento/fragmento que acontece em `SHARD_END` para fragmentos fechados, permitindo que um KCL trabalhador crie concessões somente para os fragmentos secundários do fragmento que ele concluiu o processamento. Para obter mais informações, consulte GetRecordsChildSharde.
- Com as mudanças acima, o comportamento de KCL está passando do modelo de todos os trabalhadores aprendendo sobre todos os fragmentos existentes para o modelo de trabalhadores aprendendo apenas sobre os fragmentos dos filhos que cada trabalhador possui. Portanto,

além da sincronização que ocorre durante a inicialização do aplicativo do consumidor e os eventos de refragmentação, KCL agora também realiza varreduras periódicas adicionais de fragmentos/concessões para identificar possíveis lacunas na tabela de leasing (em outras palavras, para aprender sobre todos os novos fragmentos) para garantir que o intervalo de hash completo do fluxo de dados esteja sendo processado e crie concessões para eles, se necessário. `PeriodicShardSyncManager`é o componente responsável pela execução periódica de varreduras de arrendamento/fragmento.

Quando `KinesisClientLibConfiguration#shardSyncStrategyType` é definido como `ShardSyncStrategyType.SHARD_END`, `PeriodicShardSync` `leasesRecoveryAuditorInconsistencyConfidenceThreshold` é usado para determinar o limite do número de varreduras consecutivas contendo lacunas na tabela de concessões após o qual é necessário impor uma sincronização de fragmentos. Quando `KinesisClientLibConfiguration#shardSyncStrategyType` é definido como `ShardSyncStrategyType.PERIODIC`, `leasesRecoveryAuditorInconsistencyConfidenceThreshold` é ignorado.

Para obter mais informações sobre a KCL versão 1.14, consulte `PeriodicShardSyncManager` https://github.com/awslabs/amazon-kinesis-client/blob/v1.x/src/main/java/com/amazonaws/services/kinesis/clientlibrary/lib/worker/ .java #L987 -L999 KinesisClientLibConfiguration.

Na KCL versão 1.14, uma nova opção de configuração está disponível para configuração `PeriodicShardSyncManager` em: `LeaseManagementConfig`

Nome	Valor padrão	Descrição
leasesRecoveryAuditorInconsistencyConfidenceThreshold	3	Limite de confiança no trabalho periódico do auditor para determinar se as concessões de um fluxo de dados na tabela de concessões são inconsistentes. Se encontrar consecutivamente o mesmo conjunto de inconsistências em um fluxo de dados pelo número de vezes definido, o auditor acionará uma sincronização de fragmentos.

Agora, novas CloudWatch métricas também são emitidas para monitorar a integridade do`PeriodicShardSyncManager`. Para ter mais informações, consulte PeriodicShardSyncManager.

- KCLO 1.14 agora também oferece suporte à limpeza de arrendamento diferido. As concessões são excluídas de forma assíncrona por `LeaseCleanupManager` ao chegar ao `SHARD_END`

quando um fragmento ultrapassar o período de retenção do fluxo de dados ou quando for fechado por uma operação de refragmentação.

Novas opções disponíveis para configuração de `LeaseCleanupManager`:

Nome	Valor padrão	Descrição
leaseCleanupIntervalMillis	1 minuto	Intervalo de execução do thread de limpeza de concessões.
completedLeaseCleanupIntervalMillis	5 minutos	Intervalo de verificação de conclusão da concessão.
garbageLeaseCleanupIntervalMillis	30 minutos	Intervalo de verificação do estado de lixo de uma concessão (ou seja, reduzida após o período de retenção do fluxo de dados).

- Inclui uma otimização de `KinesisShardSyncer` para criar apenas concessões em uma camada de fragmentos.

Processe vários fluxos de dados com o mesmo KCL 2.x para o aplicativo consumidor Java

Esta seção descreve as seguintes alterações na KCL versão 2.x para Java que permitem criar aplicativos de KCL consumo que podem processar mais de um fluxo de dados ao mesmo tempo.

> **Important**
>
> O processamento multistream só é suportado no KCL 2.x para Java, começando com KCL 2.3 para Java e versões posteriores.
> O processamento multistream é NOT compatível com qualquer outra linguagem na qual o KCL 2.x possa ser implementado.
> O processamento multistream é NOT suportado em qualquer versão da KCL 1.x.

- MultistreamTracker interface

  Para criar um aplicativo de consumidor que possa processar vários fluxos ao mesmo tempo, você deve implementar uma nova interface chamada MultistreamTracker. Essa interface inclui o `streamConfigList` método que retorna a lista de fluxos de dados e suas configurações a serem processadas pelo aplicativo KCL consumidor. Observe que os fluxos de dados que estão sendo processados podem ser alterados durante o tempo de execução do aplicativo consumidor. `streamConfigListé` chamado periodicamente pelo KCL para saber mais sobre as mudanças nos fluxos de dados a serem processados.

  O `streamConfigList` método preenche a 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;
}
```

  Observe que os campos `StreamIdentifier` e `InitialPositionInStreamExtended` são obrigatórios, enquanto `consumerArn` é opcional. Você deve fornecer o `consumerArn` somente se estiver usando KCL 2.x para implementar um aplicativo de consumidor fan-out aprimorado.

Para obter mais informaçõesStreamIdentifier, consulte https://github.com/awslabs/amazon-kinesis-client/blob/v2.5.8/ amazon-kinesis-client /src/main/java/software/amazon/kinesis/common/ .java #L129. StreamIdentifier Para criar umaStreamIdentifier, recomendamos que você crie uma instância multistream a partir do streamArn e do streamCreationEpoch que esteja disponível na v2.5.0 e versões posteriores. Nas KCL v2.3 e v2.4, que não oferecem suportestreamArm, crie uma instância multistream usando o formato. account-id:StreamName:streamCreationTimestamp Esse formato será descontinuado e não será mais suportado a partir da próxima versão principal.

MultistreamTracker também inclui uma estratégia para excluir concessões de fluxos antigos na tabela de concessões (formerStreamsLeasesDeletionStrategy). Observe que a estratégia é CANNOT alterada durante o tempo de execução do aplicativo do consumidor. Para obter mais informações, consulte https://github.com/awslabs/amazon-kinesis-client/blob/0c5042dadf794fe988438436252a5a8fe70b6b0b/ /src/main/java/software/amazon/kinesis/processor/ amazon-kinesis-client .java FormerStreamsLeasesDeletionStrategy

- ConfigsBuilderé uma classe de todo o aplicativo que você pode usar para especificar todas as configurações KCL 2.x a serem usadas ao criar seu aplicativo consumidor. KCL ConfigsBuildera classe agora tem suporte para a MultistreamTracker interface. Você pode inicializar ConfigsBuilder com o nome do único fluxo de dados do qual consumir 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;
    }
```

Ou você pode inicializar ConfigsBuilder com `MultiStreamTracker` se quiser implementar um aplicativo de KCL consumidor que processe vários fluxos ao mesmo tempo.

```
* 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;
    }
```

- Com o suporte multistream implementado para seu aplicativo KCL consumidor, cada linha da tabela de leasing do aplicativo agora contém o ID do fragmento e o nome do stream dos vários fluxos de dados que esse aplicativo processa.

- Quando o suporte multistream para seu aplicativo de KCL consumidor é implementado, ele leaseKey assume a seguinte estrutura:`account-id:StreamName:streamCreationTimestamp:ShardId`. Por exemplo, `111111111:multiStreamTest-1:12345:shardId-000000000336`.

> **Important**
>
> Quando seu aplicativo de KCL consumidor existente está configurado para processar somente um fluxo de dados, o leaseKey (que é a chave de hash da tabela de concessão) é o ID do fragmento. Se você reconfigurar este aplicativo de KCL consumidor existente para processar vários fluxos de dados, ele quebrará sua tabela de leasing, porque com suporte multistream, a leaseKey estrutura deve ser a seguinte:. `account-id:StreamName:StreamCreationTimestamp:ShardId`

Use o KCL com o Registro do AWS Glue Esquema

Você pode integrar seus streams de dados do Kinesis com o AWS Glue Schema Registry. O Registro de AWS Glue Esquemas permite que você descubra, controle e desenvolva esquemas de forma centralizada, ao mesmo tempo em que garante que os dados produzidos sejam continuamente validados por um esquema registrado. O esquema define a estrutura e o formato de um registro de dados. Um esquema é uma especificação versionada para publicação, consumo ou datastore confiáveis. O AWS Glue Schema Registry permite que você melhore a qualidade end-to-end dos dados e a governança de dados em seus aplicativos de streaming. Para obter mais informações, consulte Registro de esquemas do AWS Glue. Uma das formas de configurar essa integração é por meio do KCL em Java.

> **Important**
>
> Atualmente, a integração do Kinesis Data AWS Glue Streams e do Schema Registry só é compatível com os streams de dados do Kinesis que usam consumidores 2.3 implementados em JavaKCL. Suporte a várias linguagens não é fornecido. KCLOs consumidores 1.0 não são suportados. KCLNão há suporte para consumidores 2.x anteriores à KCL 2.3.

Para obter instruções detalhadas sobre como configurar a integração do Kinesis Data Streams com o Schema Registry KCL usando o, consulte a seção “Interagindo com dados KPL usando as/

Bibliotecas” em Caso de uso: Integração KCL do Amazon Kinesis Data Streams com o Glue Schema Registry. AWS

Desenvolva consumidores personalizados com taxa de transferência compartilhada

Se não precisar de throughput específica ao receber dados do Kinesis Data Streams, nem de atrasos de propagação de leitura de até 200 ms, você poderá criar aplicações de consumo seguindo as etapas descritas nos tópicos a seguir. Você pode usar a Kinesis Client Library (KCL) ou a. AWS SDK for Java

Tópicos

Para obter informações sobre a criação de consumidores que podem receber registros de fluxos de dados do Kinesis com throughput dedicada, consulte Desenvolva consumidores personalizados com taxa de transferência dedicada (distribuição aprimorada).

Desenvolva consumidores personalizados com taxa de transferência compartilhada usando KCL

Um dos métodos de desenvolvimento de um aplicativo de consumidor personalizado com taxa de transferência compartilhada é usar a Kinesis Client Library KCL ().

Escolha um dos tópicos a seguir para a KCL versão que você está usando.

Tópicos

Desenvolva KCL consumidores 1.x

Você pode desenvolver um aplicativo de consumidor para o Amazon Kinesis Data Streams usando a Kinesis Client Library (). KCL

Para obter mais informações sobreKCL, consulteO que é a Kinesis Client Library? .

Escolha entre os tópicos a seguir, dependendo da opção que você deseja usar.

Conteúdo

Desenvolva um consumidor da Kinesis Client Library em Java

Você pode usar a Kinesis Client Library (KCL) para criar aplicativos que processam dados dos seus streams de dados do Kinesis. A Kinesis Client Library está disponível em várias linguagens. Este tópico discute Java. Para ver a referência de Javadoc, consulte o tópico AWS Javadoc para Classe. AmazonKinesisClient

Para baixar o Java KCL de GitHub, acesse a Kinesis Client Library (Java). Para localizar o Java KCL no Apache Maven, acesse a página de resultados da KCL pesquisa. Para baixar o código de amostra para um aplicativo KCL consumidor Java em GitHub, acesse a página do projeto de amostra KCL para Java em GitHub.

O aplicativo de exemplo usa Apache Commons Logging. Você pode alterar a configuração do registro em log no método `configure` estático definido no arquivo `AmazonKinesisApplicationSample.java`. Para obter mais informações sobre como usar o Apache Commons Logging com aplicativos Log4j e AWS Java, consulte Logging with Log4j no Guia do desenvolvedor.AWS SDK for Java

Você deve concluir as seguintes tarefas ao implementar um aplicativo KCL consumidor em Java:

Tarefas

- Implemente os IRecordProcessor métodos
- Implemente uma fábrica de classes para a IRecordProcessor interface
- Crie um trabalhador
- Modifique as propriedades de configuração

- Migrar para a versão 2 da interface do processador de registros

Implemente os IRecordProcessor métodos

KCLAtualmente, o suporta duas versões da `IRecordProcessor` interface: a interface original está disponível com a primeira versão doKCL, e a versão 2 está disponível a partir da versão 1.5.0. KCL As duas interfaces têm suporte total. A escolha depende dos requisitos de cenário específicos. Consulte os Javadocs criados localmente ou o código-fonte para ver todas as diferenças. As seções a seguir descrevem a implementação mínima para os conceitos básicos.

IRecordProcessorVersões

- Interface original (versão 1)
- Interface atualizada (versão 2)

Interface original (versão 1)

A interface `IRecordProcessor` original (`package com.amazonaws.services.kinesis.clientlibrary.interfaces`) expõe os seguintes métodos de processador de registros que o consumidor precisa implementar. O exemplo fornece implementações que você pode usar como ponto 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)
```

inicializar

Ele KCL chama o `initialize` método quando o processador de registro é instanciado, passando um ID de fragmento específico como parâmetro. Esse processador de registros processa apenas esse estilhaço e, normalmente, o inverso também é verdadeiro (esse estilhaço é processado somente por esse processador de registro). No entanto, o consumidor deve considerar a possibilidade de que um registro de dados pode ser processado mais de uma vez. A semântica do Kinesis Data Streams é do tipo pelo menos uma vez, o que significa que cada registro de dados de um fragmento é processado pelo menos uma vez por um operador no consumidor. Para obter mais informações sobre casos em que um estilhaço específico pode ser processado por mais de um

operador, consulte Use refragmentação, escalonamento e processamento paralelo para alterar o número de fragmentos.

```
public void initialize(String shardId)
```

processRecords

O KCL chama o `processRecords` método, passando uma lista de registros de dados do fragmento especificado pelo `initialize(shardId)` método. O processador de registros processa os dados nesses registros de acordo com a semântica do consumidor. Por exemplo, o operador pode executar uma transformação nos dados e, em seguida, armazenar o resultado em um bucket do Amazon Simple Storage Service (Amazon S3).

```
public void processRecords(List<Record> records, IRecordProcessorCheckpointer
 checkpointer)
```

Além dos dados em si, o registro também contém um número de sequência e uma chave de partição. O operador pode usar esses valores ao processar os dados. Por exemplo, o operador pode escolher o bucket do S3 no qual armazenar os dados com base no valor da chave de partição. A classe `Record` expõe os seguintes métodos que oferecem acesso aos dados do registro, número de sequência e chave de partição.

```
record.getData()
record.getSequenceNumber()
record.getPartitionKey()
```

No exemplo, o método privado `processRecordsWithRetries` tem código que mostra como um operador pode acessar os dados do registro, o número de sequência e a chave de partição.

O Kinesis Data Streams requer que o processador de registros rastreie os registros que já foram processados em um fragmento. O KCL cuida desse rastreamento para você passando um checkpointer (`IRecordProcessorCheckpointer`) para. `processRecords` O processador de registros chama o `checkpoint` método nessa interface para informar o KCL quanto ele progrediu no processamento dos registros no fragmento. Se o trabalhador falhar, ele KCL usa essas informações para reiniciar o processamento do fragmento no último registro processado conhecido.

Para uma operação de divisão ou mesclagem, eles KCL não começarão a processar os novos fragmentos até que os processadores dos fragmentos originais liguem `checkpoint` para sinalizar que todo o processamento nos fragmentos originais foi concluído.

Se você não passar um parâmetro, KCL presume que a chamada para `checkpoint` significa que todos os registros foram processados, até o último registro que foi passado para o processador de registros. Portanto, o processador de registros deve chamar `checkpoint` somente após ter processado todos os registros na lista que foi passada a ele. Os processadores de registros não precisam chamar `checkpoint` em cada chamada para `processRecords`. Um processador pode, por exemplo, chamar `checkpoint` a cada terceira chamada para `processRecords`. Você pode, opcionalmente, especificar o número de sequência exato de um registro como um parâmetro para `checkpoint`. Nesse caso, KCL pressupõe que todos os registros tenham sido processados até esse registro somente.

No exemplo, o método privado `checkpoint` mostra como chamar `IRecordProcessorCheckpointer.checkpoint` usando a lógica de novas tentativas e o tratamento de exceções apropriados.

Ele KCL depende de `processRecords` lidar com quaisquer exceções decorrentes do processamento dos registros de dados. Se uma exceção for lançada`processRecords`, ela KCL ignorará os registros de dados que foram passados antes da exceção. Ou seja, esses registros não serão reenviados para o processador de registros que lançou a exceção ou para qualquer outro processador de registros no consumidor.

shutdown

Ele KCL chama o `shutdown` método quando o processamento termina (o motivo do desligamento é`TERMINATE`) ou o trabalhador não está mais respondendo (o motivo do desligamento é). `ZOMBIE`

```
public void shutdown(IRecordProcessorCheckpointer checkpointer, ShutdownReason reason)
```

O processamento termina quando o processador de registros não recebe mais registros do estilhaço porque ele foi dividido ou intercalado, ou o stream foi excluído.

O KCL também passa uma `IRecordProcessorCheckpointer` interface para`shutdown`. Se o motivo do desligamento é `TERMINATE`, o processador de registros deve terminar o processamento de todos os registros de dados e, em seguida, chamar o método `checkpoint` nesta interface.

Interface atualizada (versão 2)

A interface `IRecordProcessor` atualizada (`package com.amazonaws.services.kinesis.clientlibrary.interfaces.v2`) expõe os seguintes métodos de processador de registros que o consumidor precisa implementar:

```
void initialize(InitializationInput initializationInput)
void processRecords(ProcessRecordsInput processRecordsInput)
void shutdown(ShutdownInput shutdownInput)
```

Todos os argumentos da versão original da interface podem ser acessados por meio de métodos get nos objetos de contêiner. Por exemplo, para recuperar a lista de registros em `processRecords()`, você pode usar `processRecordsInput.getRecords()`.

A partir da versão 2 dessa interface (KCL1.5.0 e posterior), as seguintes novas entradas estão disponíveis, além das entradas fornecidas pela interface original:

número de sequência inicial

No objeto `InitializationInput` passado para a operação `initialize()`, o número de sequência inicial a partir do qual os registros seriam fornecidos à instância do processador de registros. Esse é o número de sequência que foi verificado pela última vez pela instância do processador de registros que processou anteriormente o mesmo estilhaço. Isso será fornecido no caso de o aplicativo precisar de informações.

número de sequência do ponto de verificação pendente

No objeto `InitializationInput` passado para a operação `initialize()`, o número de sequência de verificação pendente (se houver) que não pôde ser confirmado antes que a instância do processador de registros anterior parasse.

Implemente uma fábrica de classes para a IRecordProcessor interface

Você também precisará implementar uma fábrica para a classe que implementa os métodos do processador de registros. Quando o consumidor instancia o operador, ele passa uma referência a essa fábrica.

O exemplo implementa a classe de fábrica no arquivo `AmazonKinesisApplicationSampleRecordProcessorFactory.java` usando a interface de processador de registros original. Se você deseja que a fábrica da classe crie a versão 2 dos processadores de registros, use o nome do pacote `com.amazonaws.services.kinesis.clientlibrary.interfaces.v2`.

```
public class SampleRecordProcessorFactory implements IRecordProcessorFactory {
    /**
```

```
        * Constructor.
        */
        public SampleRecordProcessorFactory() {
            super();
        }
        /**
        * {@inheritDoc}
        */
        @Override
        public IRecordProcessor createProcessor() {
            return new SampleRecordProcessor();
        }
    }
```

Crie um trabalhador

Conforme discutido emImplemente os IRecordProcessor métodos, há duas versões da interface do processador de KCL registros para escolher, o que afeta a forma como você criaria um trabalhador. A interface do processador de registros original usa a seguinte estrutura de código para criar um operador:

```
final KinesisClientLibConfiguration config = new KinesisClientLibConfiguration(...)
final IRecordProcessorFactory recordProcessorFactory = new RecordProcessorFactory();
final Worker worker = new Worker(recordProcessorFactory, config);
```

Com a versão 2 da interface do processador de registros, você pode usar `Worker.Builder` para criar um operador sem a necessidade de se preocupar com qual construtor usar e a ordem dos argumentos. A interface do processador de registros atualizada usa a seguinte estrutura de código para criar um operador:

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

Modifique as propriedades de configuração

O exemplo fornece valores padrão para propriedades de configuração. Esses dados de configuração para o operador são então consolidados em um objeto `KinesisClientLibConfiguration`.

Esse objeto e uma referência à fábrica de classe para `IRecordProcessor` são passados na chamada que instancia o operador. Você pode substituir qualquer uma dessas propriedades por seus próprios valores usando um arquivo de propriedades do Java (consulte `AmazonKinesisApplicationSample.java`).

Nome da aplicação

Isso KCL requer um nome de aplicativo que seja exclusivo em todos os seus aplicativos e nas tabelas do Amazon DynamoDB na mesma região. Ela usa o valor de configuração de nome de aplicativo das seguintes formas:

- Presume-se que todos os operadores associados com esse nome de aplicativo estejam trabalhando juntos no mesmo stream. Esses operadores podem ser distribuídos em várias instâncias. Se você executar uma instância adicional do mesmo código de aplicativo, mas com um nome de aplicativo diferente, KCL tratará a segunda instância como um aplicativo totalmente separado que também está operando no mesmo stream.
- O KCL cria uma tabela do DynamoDB com o nome do aplicativo e usa a tabela para manter as informações de estado (como pontos de verificação e mapeamento de fragmentos de trabalho) do aplicativo. Cada aplicação tem sua própria tabela do DynamoDB. Para obter mais informações, consulte Use uma tabela de leasing para rastrear os fragmentos processados pelo aplicativo consumidor KCL.

Configurar credenciais

Você deve disponibilizar suas AWS credenciais para um dos provedores de credenciais na cadeia de provedores de credenciais padrão. Por exemplo, se você estiver executando seu consumidor em uma EC2 instância, recomendamos que você execute a instância com uma IAM função. AWS as credenciais que refletem as permissões associadas a essa IAM função são disponibilizadas aos aplicativos na instância por meio dos metadados da instância. Essa é a maneira mais segura de gerenciar credenciais para um consumidor em execução em uma EC2 instância.

O aplicativo de amostra primeiro tenta recuperar as IAM credenciais dos metadados da instância:

```
credentialsProvider = new InstanceProfileCredentialsProvider();
```

Se o aplicativo de exemplo não consegue obter credenciais dos metadados da instância, ele tenta recuperar as credenciais de um arquivo de propriedades:

```
credentialsProvider = new ClasspathPropertiesFileCredentialsProvider();
```

Para obter mais informações sobre os metadados da instância, consulte Metadados da instância no Guia EC2do usuário da Amazon.

Use o ID do trabalhador para várias instâncias

O código de inicialização de exemplo cria um ID para o operador, `workerId`, usando o nome do computador local e anexando um identificador exclusivo globalmente, conforme mostrado no seguinte trecho de código. Essa abordagem é compatível com o cenário de várias instâncias do aplicativo de consumidor em execução em um único computador.

```
String workerId = InetAddress.getLocalHost().getCanonicalHostName() + ":" +
 UUID.randomUUID();
```

Migrar para a versão 2 da interface do processador de registros

Se você quiser migrar o código que usa a interface original, além das etapas descritas anteriormente, as seguintes etapas serão necessárias:

1. Altere a classe do processador de registros para importar a versão 2 da interface do processador de registros:

   ```
   import com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor;
   ```

2. Altere as referências para as entradas para usar métodos `get` nos objetos de contêiner. Por exemplo, na operação `shutdown()`, altere "`checkpointer`" para "`shutdownInput.getCheckpointer()`".
3. Altere a classe da fábrica do processador de registros para importar a versão 2 da interface da fábrica do processador de registros:

   ```
   import
    com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessorFactory;
   ```

4. Altere a construção do operador para usar `Worker.Builder`. Por exemplo:

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

```
    .build();
```

Desenvolva um consumidor da Kinesis Client Library em Node.js

Você pode usar a Kinesis Client Library (KCL) para criar aplicativos que processam dados dos seus streams de dados do Kinesis. A Kinesis Client Library está disponível em várias linguagens. Este tópico discute Node.js.

KCLÉ uma biblioteca Java; o suporte para linguagens diferentes de Java é fornecido usando uma interface multilíngue chamada de MultiLangDaemon. Esse daemon é baseado em Java e é executado em segundo plano quando você está usando uma KCL linguagem diferente de Java. Portanto, se você instalar o KCL for Node.js e escrever seu aplicativo de consumidor inteiramente em Node.js, ainda precisará do Java instalado em seu sistema por causa do MultiLangDaemon. Além disso, MultiLangDaemon tem algumas configurações padrão que você pode precisar personalizar para seu caso de uso, por exemplo, a AWS região à qual ele se conecta. Para obter mais informações sobre o MultiLangDaemon on GitHub, acesse a página do KCL MultiLangDaemon projeto.

Para baixar o Node.js KCL de GitHub, acesse a Biblioteca de Cliente Kinesis (Node.js).

Downloads de códigos de exemplo

Há dois exemplos de código disponíveis KCL em Node.js:

- basic-sample

  Usado nas seções a seguir para ilustrar os fundamentos da criação de um aplicativo de KCL consumidor no Node.js.
- click-stream-sample

  Levemente mais avançado e usa um cenário real, para depois que você se familiarizar com o código de exemplo básico. Esse exemplo não é discutido aqui, mas tem um README arquivo com mais informações.

Você deve concluir as seguintes tarefas ao implementar um aplicativo KCL consumidor no Node.js:

Tarefas

- Implemente o processador de registros

- Modifique as propriedades de configuração

Implemente o processador de registros

O consumidor mais simples possível que usa o KCL for Node.js deve implementar uma `recordProcessor` função que, por sua vez`initialize`, contém as funções`processRecords`, `shutdown` e. O exemplo fornece uma implementação que você pode usar como ponto de partida (consulte `sample_kcl_app.js`).

```
function recordProcessor() {
  // return an object that implements initialize, processRecords and shutdown
 functions.}
```

inicializar

O KCL chama a `initialize` função quando o processador de gravação é iniciado. Esse processador de registros processa apenas o ID do estilhaço passado como `initializeInput.shardId` e, normalmente, o inverso também é verdadeiro (esse estilhaço é processado somente por esse processador de registro). No entanto, o consumidor deve considerar a possibilidade de que um registro de dados pode ser processado mais de uma vez. Isso acontece porque a semântica do Kinesis Data Streams é do tipo pelo menos uma vez, o que significa que cada registro de dados de um fragmento é processado pelo menos uma vez por um operador no consumidor. Para obter mais informações sobre casos em que um estilhaço específico pode ser processado por mais de um operador, consulte Use refragmentação, escalonamento e processamento paralelo para alterar o número de fragmentos.

```
initialize: function(initializeInput, completeCallback)
```

processRecords

O KCL chama essa função com uma entrada que contém uma lista de registros de dados do fragmento especificado para a `initialize` função. O processador de registros que você implementa processa os dados nesses registros de acordo com a semântica do consumidor. Por exemplo, o operador pode executar uma transformação nos dados e, em seguida, armazenar o resultado em um bucket do Amazon Simple Storage Service (Amazon S3).

```
processRecords: function(processRecordsInput, completeCallback)
```

Além dos dados em si, o registro também contém um número de sequência e uma chave de partição, que o operador pode usar ao processar os dados. Por exemplo, o operador pode escolher o bucket do S3 no qual armazenar os dados com base no valor da chave de partição. O dicionário `record` expõe os seguintes pares de chave/valor para acessar os dados do registro, o número de sequência e a chave de partição:

```
record.data
record.sequenceNumber
record.partitionKey
```

Observe que os dados são codificados em Base64.

No exemplo básico, a função `processRecords` tem código que mostra como um operador pode acessar os dados do registro, o número de sequência e a chave de partição.

O Kinesis Data Streams requer que o processador de registros rastreie os registros que já foram processados em um fragmento. O KCL cuida desse rastreamento com um `checkpointer` objeto passado como`processRecordsInput.checkpointer`. Seu processador de registros chama a `checkpointer.checkpoint` função para informar o KCL quanto ela progrediu no processamento dos registros no fragmento. Caso o trabalhador falhe, ele KCL usa essas informações quando você reinicia o processamento do fragmento para que ele continue a partir do último registro processado conhecido.

Para uma operação de divisão ou mesclagem, o processamento dos novos fragmentos KCL não é iniciado até que os processadores dos fragmentos originais sejam chamados `checkpoint` para sinalizar que todo o processamento nos fragmentos originais foi concluído.

Se você não passar o número de sequência para a `checkpoint` função, KCL presume que a chamada para `checkpoint` significa que todos os registros foram processados, até o último registro passado para o processador de registros. Portanto, o processador de registros deve chamar `checkpoint` somente após ter processado todos os registros na lista que foi passada para ele. Os processadores de registros não precisam chamar `checkpoint` em cada chamada para `processRecords`. Um processador pode, por exemplo, chamar `checkpoint` a cada terceira chamada, ou algum evento externo para o processador de registros, como um serviço de validação/verificação personalizado que você tiver implementado.

Você pode, opcionalmente, especificar o número de sequência exato de um registro como um parâmetro para `checkpoint`. Nesse caso, KCL pressupõe que todos os registros tenham sido processados até esse registro somente.

O aplicativo de exemplo básico mostra a chamada mais simples possível para a função `checkpointer.checkpoint`. Você pode adicionar outra lógica de verificação que precisar para o consumidor neste ponto da função.

shutdown

Ele KCL chama a `shutdown` função quando o processamento termina (`shutdownInput.reason`é`TERMINATE`) ou o trabalhador não está mais respondendo (`shutdownInput.reason`está`ZOMBIE`).

```
shutdown: function(shutdownInput, completeCallback)
```

O processamento termina quando o processador de registros não recebe mais registros do estilhaço porque ele foi dividido ou intercalado, ou o stream foi excluído.

O KCL também passa um `shutdownInput.checkpointer` objeto para`shutdown`. Se o motivo do desligamento for `TERMINATE`, você deverá verificar se o processador de registros terminou o processamento de todos os registros de dados e, em seguida, chamar a função `checkpoint` nessa interface.

Modifique as propriedades de configuração

O exemplo fornece valores padrão para as propriedades de configuração. Você pode substituir qualquer uma dessas propriedades por seus próprios valores (consulte `sample.properties` no exemplo básico).

Nome da aplicação

Isso KCL exige que um aplicativo seja exclusivo entre seus aplicativos e entre as tabelas do Amazon DynamoDB na mesma região. Ela usa o valor de configuração de nome de aplicativo das seguintes formas:

- Presume-se que todos os operadores associados com esse nome de aplicativo estejam trabalhando juntos no mesmo stream. Esses operadores podem ser distribuídos em várias instâncias. Se você executar uma instância adicional do mesmo código de aplicativo, mas com um nome de aplicativo diferente, KCL tratará a segunda instância como um aplicativo totalmente separado que também está operando no mesmo stream.
- O KCL cria uma tabela do DynamoDB com o nome do aplicativo e usa a tabela para manter as informações de estado (como pontos de verificação e mapeamento de fragmentos de trabalho) do

aplicativo. Cada aplicação tem sua própria tabela do DynamoDB. Para obter mais informações, consulte Use uma tabela de leasing para rastrear os fragmentos processados pelo aplicativo consumidor KCL.

Configurar credenciais

Você deve disponibilizar suas AWS credenciais para um dos provedores de credenciais na cadeia de provedores de credenciais padrão. Você pode usar a propriedade `AWSCredentialsProvider` para definir um provedor de credenciais. O arquivo `sample.properties` precisa disponibilizar as credenciais para um dos provedores de credenciais na cadeia de provedores de credenciais padrão. Se você estiver executando seu consumidor em uma EC2 instância da Amazon, recomendamos que você configure a instância com uma IAM função. AWS as credenciais que refletem as permissões associadas a essa IAM função são disponibilizadas aos aplicativos na instância por meio dos metadados da instância. Essa é a maneira mais segura de gerenciar as credenciais de um aplicativo de consumidor em execução em uma EC2 instância.

O exemplo a seguir é configurado KCL para processar um stream de dados do Kinesis `kclnodejssample` chamado usando o processador de registros fornecido em: `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
```

Desenvolva um consumidor da Kinesis Client Library em. NET

Você pode usar a Kinesis Client Library (KCL) para criar aplicativos que processam dados dos seus streams de dados do Kinesis. A Kinesis Client Library está disponível em várias linguagens. Este tópico discute. NET.

KCLÉ uma biblioteca Java; o suporte para linguagens diferentes de Java é fornecido usando uma interface multilíngue chamada de MultiLangDaemon. Esse daemon é baseado em Java e é

executado em segundo plano quando você está usando uma KCL linguagem diferente de Java. Portanto, se você instalar o KCL for. NETe escreva seu aplicativo de consumidor inteiramente em. NET, você ainda precisa do Java instalado em seu sistema por causa do MultiLangDaemon. Além disso, MultiLangDaemon tem algumas configurações padrão que você pode precisar personalizar para seu caso de uso, por exemplo, a AWS região à qual ele se conecta. Para obter mais informações sobre o MultiLangDaemon on GitHub, acesse a página do KCL MultiLangDaemon projeto.

Para baixar o. NETKCLde GitHub, acesse a Kinesis Client Library (. NET). Para baixar o código de amostra para um. NETKCLaplicativo para consumidores, acesse KCLo formulário. NETamostra da página do projeto do consumidor em GitHub.

Você deve concluir as tarefas a seguir ao implementar um aplicativo KCL consumidor no. NET:

Tarefas

- Implemente os métodos IRecordProcessor de classe
- Modifique as propriedades de configuração

Implemente os métodos IRecordProcessor de classe

O consumidor precisa implementar os seguintes métodos para `IRecordProcessor`. O consumidor de exemplo fornece implementações que você pode usar como ponto de partida (consulte a classe `SampleRecordProcessor` em `SampleConsumer/AmazonKinesisSampleConsumer.cs`).

```
public void Initialize(InitializationInput input)
public void ProcessRecords(ProcessRecordsInput input)
public void Shutdown(ShutdownInput input)
```

Inicializar

Ele KCL chama esse método quando o processador de registros é instanciado, passando um ID de fragmento específico no `input` parâmetro (). `input.ShardId` Esse processador de registros processa apenas esse estilhaço e, normalmente, o inverso também é verdadeiro (esse estilhaço é processado somente por esse processador de registro). No entanto, o consumidor deve considerar a possibilidade de que um registro de dados pode ser processado mais de uma vez. Isso acontece porque a semântica do Kinesis Data Streams é do tipo pelo menos uma vez, o que significa que cada registro de dados de um fragmento é processado pelo menos uma vez por um operador no consumidor. Para obter mais informações sobre casos em que um estilhaço específico

pode ser processado por mais de um operador, consulte Use refragmentação, escalonamento e processamento paralelo para alterar o número de fragmentos.

```
public void Initialize(InitializationInput input)
```

ProcessRecords

Ele KCL chama esse método, passando uma lista de registros de dados no `input` parâmetro (`input.Records`) do fragmento especificado pelo `Initialize` método. O processador de registros que você implementa processa os dados nesses registros de acordo com a semântica do consumidor. Por exemplo, o operador pode executar uma transformação nos dados e, em seguida, armazenar o resultado em um bucket do Amazon Simple Storage Service (Amazon S3).

```
public void ProcessRecords(ProcessRecordsInput input)
```

Além dos dados em si, o registro também contém um número de sequência e uma chave de partição. O operador pode usar esses valores ao processar os dados. Por exemplo, o operador pode escolher o bucket do S3 no qual armazenar os dados com base no valor da chave de partição. A classe `Record` expõe os seguintes itens para acessar os dados do registro, o número de sequência e a chave de partição:

```
byte[] Record.Data
string Record.SequenceNumber
string Record.PartitionKey
```

No exemplo, o método `ProcessRecordsWithRetries` tem código que mostra como um operador pode acessar os dados do registro, o número de sequência e a chave de partição.

O Kinesis Data Streams requer que o processador de registros rastreie os registros que já foram processados em um fragmento. O KCL cuida desse rastreamento para você passando um `Checkpointer` objeto para `ProcessRecords` (`input.Checkpointer`). O processador de registros chama o `Checkpointer.Checkpoint` método para informar o KCL quanto ele progrediu no processamento dos registros no fragmento. Se o trabalhador falhar, ele KCL usa essas informações para reiniciar o processamento do fragmento no último registro processado conhecido.

Para uma operação de divisão ou mesclagem, o processamento dos novos fragmentos KCL não é iniciado até que os processadores dos fragmentos originais sejam chamados `Checkpointer.Checkpoint` para sinalizar que todo o processamento nos fragmentos originais foi concluído.

Se você não passar um parâmetro, KCL presume que a chamada para `Checkpointer.Checkpoint` significa que todos os registros foram processados, até o último registro que foi passado para o processador de registros. Portanto, o processador de registros deve chamar `Checkpointer.Checkpoint` somente após ter processado todos os registros na lista que foi passada a ele. Os processadores de registros não precisam chamar `Checkpointer.Checkpoint` em cada chamada para `ProcessRecords`. Um processador pode, por exemplo, chamar `Checkpointer.Checkpoint` a cada terceira ou quarta chamada. Você pode, opcionalmente, especificar o número de sequência exato de um registro como um parâmetro para `Checkpointer.Checkpoint`. Nesse caso, KCL pressupõe que os registros tenham sido processados somente até esse registro.

No exemplo, o método privado `Checkpoint(Checkpointer checkpointer)` mostra como chamar o método `Checkpointer.Checkpoint` usando a lógica de novas tentativas e o tratamento de exceções apropriados.

O KCL para. NETtrata exceções de forma diferente de outras bibliotecas de KCL idiomas, pois não trata nenhuma exceção decorrente do processamento dos registros de dados. As exceções não detectadas do código do usuário causam uma falha no programa.

Desligamento

Ele KCL chama o `Shutdown` método quando o processamento termina (o motivo do desligamento é`TERMINATE`) ou o trabalhador não está mais respondendo (o `input.Reason` valor do desligamento é). `ZOMBIE`

```
public void Shutdown(ShutdownInput input)
```

O processamento termina quando o processador de registros não recebe mais registros do estilhaço porque ele foi dividido ou intercalado, ou o stream foi excluído.

O KCL também passa um `Checkpointer` objeto para`shutdown`. Se o motivo do desligamento é `TERMINATE`, o processador de registros deve terminar o processamento de todos os registros de dados e, em seguida, chamar o método `checkpoint` nesta interface.

Modifique as propriedades de configuração

O consumidor de exemplo fornece valores padrão para as propriedades de configuração. Você pode substituir qualquer uma dessas propriedades por seus próprios valores (consulte `SampleConsumer/kcl.properties`).

Nome da aplicação

Isso KCL exige que um aplicativo seja exclusivo entre seus aplicativos e entre as tabelas do Amazon DynamoDB na mesma região. Ela usa o valor de configuração de nome de aplicativo das seguintes formas:

- Presume-se que todos os operadores associados com esse nome de aplicativo estejam trabalhando juntos no mesmo stream. Esses operadores podem ser distribuídos em várias instâncias. Se você executar uma instância adicional do mesmo código de aplicativo, mas com um nome de aplicativo diferente, KCL tratará a segunda instância como um aplicativo totalmente separado que também está operando no mesmo stream.
- O KCL cria uma tabela do DynamoDB com o nome do aplicativo e usa a tabela para manter as informações de estado (como pontos de verificação e mapeamento de fragmentos de trabalho) do aplicativo. Cada aplicação tem sua própria tabela do DynamoDB. Para obter mais informações, consulte Use uma tabela de leasing para rastrear os fragmentos processados pelo aplicativo consumidor KCL.

Configurar credenciais

Você deve disponibilizar suas AWS credenciais para um dos provedores de credenciais na cadeia de provedores de credenciais padrão. Você pode usar a propriedade `AWSCredentialsProvider` para definir um provedor de credenciais. As sample.properties precisam disponibilizar as credenciais para um dos provedores de credenciais na cadeia de provedores de credenciais padrão. Se você estiver executando seu aplicativo consumidor em uma EC2 instância, recomendamos que você configure a instância com uma IAM função. AWS as credenciais que refletem as permissões associadas a essa IAM função são disponibilizadas aos aplicativos na instância por meio dos metadados da instância. Essa é a maneira mais segura de gerenciar credenciais para um consumidor em execução em uma EC2 instância.

O arquivo de propriedades da amostra é configurado KCL para processar um stream de dados do Kinesis chamado “palavras” usando o processador de registro fornecido em. `AmazonKinesisSampleConsumer.cs`

Desenvolva um consumidor da Kinesis Client Library em Python

Você pode usar a Kinesis Client Library (KCL) para criar aplicativos que processam dados dos seus streams de dados do Kinesis. A Kinesis Client Library está disponível em várias linguagens. Este tópico discute Python.

KCLÉ uma biblioteca Java; o suporte para linguagens diferentes de Java é fornecido usando uma interface multilíngue chamada de MultiLangDaemon. Esse daemon é baseado em Java e é executado em segundo plano quando você está usando uma KCL linguagem diferente de Java. Portanto, se você instalar o KCL for Python e escrever seu aplicativo de consumidor inteiramente em Python, ainda precisará do Java instalado em seu sistema por causa do. MultiLangDaemon Além disso, MultiLangDaemon tem algumas configurações padrão que você pode precisar personalizar para seu caso de uso, por exemplo, a AWS região à qual ele se conecta. Para obter mais informações sobre o MultiLangDaemon on GitHub, acesse a página do KCL MultiLangDaemon projeto.

Para baixar o Python em GitHub, acesse a Biblioteca KCL de Cliente Kinesis (Python). Para baixar o código de amostra para um aplicativo KCL consumidor em Python, acesse a página do projeto de amostra para KCL Python em. GitHub

Você deve concluir as seguintes tarefas ao implementar um aplicativo KCL consumidor em Python:

Tarefas

- Implemente os métodos RecordProcessor de classe
- Modifique as propriedades de configuração

Implemente os métodos RecordProcessor de classe

A classe `RecordProcess` precisa estender o `RecordProcessorBase` para implementar os métodos a seguir. O exemplo fornece implementações que você pode usar como ponto de partida (consulte `sample_kclpy_app.py`).

```
def initialize(self, shard_id)
def process_records(self, records, checkpointer)
def shutdown(self, checkpointer, reason)
```

inicializar

Ele KCL chama o `initialize` método quando o processador de registro é instanciado, passando um ID de fragmento específico como parâmetro. Esse processador de registros processa apenas esse estilhaço e, normalmente, o inverso também é verdadeiro (esse estilhaço é processado somente por esse processador de registro). No entanto, o consumidor deve considerar a possibilidade de que um registro de dados pode ser processado mais de uma vez. Isso acontece porque a semântica do Kinesis Data Streams é do tipo pelo menos uma vez, o que significa que

cada registro de dados de um fragmento é processado pelo menos uma vez por um operador no consumidor. Para obter mais informações sobre casos em que um estilhaço específico pode ser processado por mais de um operador, consulte Use refragmentação, escalonamento e processamento paralelo para alterar o número de fragmentos.

```
def initialize(self, shard_id)
```

process_records

Ele KCL chama esse método, passando uma lista de registros de dados do fragmento especificado pelo `initialize` método. O processador de registros que você implementa processa os dados nesses registros de acordo com a semântica do consumidor. Por exemplo, o operador pode executar uma transformação nos dados e, em seguida, armazenar o resultado em um bucket do Amazon Simple Storage Service (Amazon S3).

```
def process_records(self, records, checkpointer)
```

Além dos dados em si, o registro também contém um número de sequência e uma chave de partição. O operador pode usar esses valores ao processar os dados. Por exemplo, o operador pode escolher o bucket do S3 no qual armazenar os dados com base no valor da chave de partição. O dicionário `record` expõe os seguintes pares de chave/valor para acessar os dados do registro, o número de sequência e a chave de partição:

```
record.get('data')
record.get('sequenceNumber')
record.get('partitionKey')
```

Observe que os dados são codificados em Base64.

No exemplo, o método `process_records` tem código que mostra como um operador pode acessar os dados do registro, o número de sequência e a chave de partição.

O Kinesis Data Streams requer que o processador de registros rastreie os registros que já foram processados em um fragmento. O KCL cuida desse rastreamento para você, passando um `Checkpointer` objeto para `process_records` o. O processador de registros chama o `checkpoint` método nesse objeto para informar o KCL quanto ele progrediu no processamento dos registros no fragmento. Se o trabalhador falhar, ele KCL usa essas informações para reiniciar o processamento do fragmento no último registro processado conhecido.

Para uma operação de divisão ou mesclagem, o processamento dos novos fragmentos KCL não é iniciado até que os processadores dos fragmentos originais sejam chamados `checkpoint` para sinalizar que todo o processamento nos fragmentos originais foi concluído.

Se você não passar um parâmetro, KCL presume que a chamada para `checkpoint` significa que todos os registros foram processados, até o último registro que foi passado para o processador de registros. Portanto, o processador de registros deve chamar `checkpoint` somente após ter processado todos os registros na lista que foi passada a ele. Os processadores de registros não precisam chamar `checkpoint` em cada chamada para `process_records`. Um processador pode, por exemplo, chamar `checkpoint` a cada terceira chamada. Você pode, opcionalmente, especificar o número de sequência exato de um registro como um parâmetro para `checkpoint`. Nesse caso, KCL pressupõe que todos os registros tenham sido processados até esse registro somente.

No exemplo, o método privado `checkpoint` mostra como chamar o método `Checkpointer.checkpoint` usando a lógica de novas tentativas e o tratamento de exceções apropriados.

Ele KCL depende de `process_records` lidar com quaisquer exceções decorrentes do processamento dos registros de dados. Se uma exceção for lançada`process_records`, ela KCL ignorará os registros de dados que foram passados `process_records` antes da exceção. Ou seja, esses registros não serão reenviados para o processador de registros que lançou a exceção ou para qualquer outro processador de registros no consumidor.

shutdown

Ele KCL chama o `shutdown` método quando o processamento termina (o motivo do desligamento é`TERMINATE`) ou o trabalhador não está mais respondendo (o desligamento `reason` é). `ZOMBIE`

```
def shutdown(self, checkpointer, reason)
```

O processamento termina quando o processador de registros não recebe mais registros do estilhaço porque ele foi dividido ou intercalado, ou o stream foi excluído.

O KCL também passa um `Checkpointer` objeto para`shutdown`. Se o `reason` do desligamento é `TERMINATE`, o processador de registros deve terminar o processamento de todos os registros de dados e, em seguida, chamar o método `checkpoint` nesta interface.

Modifique as propriedades de configuração

O exemplo fornece valores padrão para as propriedades de configuração. Você pode substituir qualquer uma dessas propriedades por seus próprios valores (consulte `sample.properties`).

Nome da aplicação

Isso KCL requer um nome de aplicativo que seja exclusivo entre seus aplicativos e entre as tabelas do Amazon DynamoDB na mesma região. Ela usa o valor de configuração de nome de aplicativo das seguintes formas:

- Presume-se que todos os operadores associados a esse nome de aplicativo estejam trabalhando juntos no mesmo streaming. Esses operadores podem ser distribuídos em várias instâncias. Se você executar uma instância adicional do mesmo código de aplicativo, mas com um nome de aplicativo diferente, KCL tratará a segunda instância como um aplicativo totalmente separado que também está operando no mesmo stream.
- O KCL cria uma tabela do DynamoDB com o nome do aplicativo e usa a tabela para manter as informações de estado (como pontos de verificação e mapeamento de fragmentos de trabalho) do aplicativo. Cada aplicação tem sua própria tabela do DynamoDB. Para obter mais informações, consulte Use uma tabela de leasing para rastrear os fragmentos processados pelo aplicativo consumidor KCL.

Configurar credenciais

Você deve disponibilizar suas AWS credenciais para um dos provedores de credenciais na cadeia de provedores de credenciais padrão. Você pode usar a propriedade `AWSCredentialsProvider` para definir um provedor de credenciais. As sample.properties precisam disponibilizar as credenciais para um dos provedores de credenciais na cadeia de provedores de credenciais padrão. Se você estiver executando seu aplicativo de consumidor em uma EC2 instância da Amazon, recomendamos que você configure a instância com uma IAM função. AWS as credenciais que refletem as permissões associadas a essa IAM função são disponibilizadas aos aplicativos na instância por meio dos metadados da instância. Essa é a maneira mais segura de gerenciar as credenciais de um aplicativo de consumidor em execução em uma EC2 instância.

O arquivo de propriedades da amostra é configurado KCL para processar um stream de dados do Kinesis chamado “palavras” usando o processador de registro fornecido em. `sample_kclpy_app.py`

Desenvolva um consumidor da Kinesis Client Library em Ruby

Você pode usar a Kinesis Client Library (KCL) para criar aplicativos que processam dados dos seus streams de dados do Kinesis. A Kinesis Client Library está disponível em várias linguagens. Este tópico discute Ruby.

KCLÉ uma biblioteca Java; o suporte para linguagens diferentes de Java é fornecido usando uma interface multilíngue chamada de MultiLangDaemon. Esse daemon é baseado em Java e é executado em segundo plano quando você está usando uma KCL linguagem diferente de Java. Portanto, se você instalar o KCL for Ruby e escrever seu aplicativo de consumidor inteiramente em Ruby, ainda precisará do Java instalado em seu sistema por causa do. MultiLangDaemon Além disso, MultiLangDaemon tem algumas configurações padrão que você pode precisar personalizar para seu caso de uso, por exemplo, a AWS região à qual ele se conecta. Para obter mais informações sobre o MultiLangDaemon on GitHub, acesse a página do KCL MultiLangDaemon projeto.

Para baixar o Ruby KCL de GitHub, acesse a Kinesis Client Library (Ruby). Para baixar o código de amostra para um aplicativo de KCL consumidor Ruby, acesse a página do projeto de amostra KCL para Ruby em. GitHub

Para obter mais informações sobre a biblioteca de suporte do KCL Ruby, consulte a documentação do KCLRuby Gems.

Desenvolva KCL consumidores 2.x

Este tópico mostra como usar a versão 2.0 da Biblioteca de Cliente Kinesis ()KCL.

Para obter mais informações sobre oKCL, consulte a visão geral fornecida em Desenvolvendo consumidores usando a Kinesis Client Library 1.x.

Escolha entre os tópicos a seguir, dependendo da opção que você deseja usar.

Tópicos

- Desenvolva um consumidor da Kinesis Client Library em Java
- Desenvolva um consumidor da Kinesis Client Library em Python

Desenvolva um consumidor da Kinesis Client Library em Java

O código a seguir mostra uma implementação de exemplo em Java de `ProcessorFactory` e `RecordProcessor`. Se você quiser aproveitar o recurso de distribuição avançada, consulte Usar consumidores com distribuição avançada.

```
/*
 *  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);
            }
        }
    }

}
```

Desenvolva um consumidor da Kinesis Client Library em Python

Você pode usar a Kinesis Client Library (KCL) para criar aplicativos que processam dados dos seus streams de dados do Kinesis. A Kinesis Client Library está disponível em várias linguagens. Este tópico discute Python.

KCLÉ uma biblioteca Java; o suporte para linguagens diferentes de Java é fornecido usando uma interface multilíngue chamada de MultiLangDaemon. Esse daemon é baseado em Java e é executado em segundo plano quando você está usando uma KCL linguagem diferente de Java. Portanto, se você instalar o KCL for Python e escrever seu aplicativo de consumidor inteiramente em Python, ainda precisará do Java instalado em seu sistema por causa do. MultiLangDaemon Além disso, MultiLangDaemon tem algumas configurações padrão que você pode precisar personalizar para seu caso de uso, por exemplo, a AWS região à qual ele se conecta. Para obter mais informações sobre o MultiLangDaemon on GitHub, acesse a página do KCL MultiLangDaemon projeto.

Para baixar o Python em GitHub, acesse a Biblioteca KCL de Cliente Kinesis (Python). Para baixar o código de amostra para um aplicativo KCL consumidor em Python, acesse a página do projeto de amostra para KCL Python em. GitHub

Você deve concluir as seguintes tarefas ao implementar um aplicativo KCL consumidor em Python:

Tarefas

- Implemente os métodos RecordProcessor de classe
- Modifique as propriedades de configuração

Implemente os métodos RecordProcessor de classe

A classe `RecordProcess` precisa estender a classe `RecordProcessorBase` para implementar os seguintes métodos:

```
initialize
```

```
process_records
shutdown_requested
```

Este exemplo fornece implementações que você pode usar como ponto 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()
```

Modifique as propriedades de configuração

O exemplo fornece valores padrão para as propriedades de configuração, conforme mostrado no script a seguir. Você pode substituir qualquer uma dessas propriedades por seus próprios 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
```

Nome da aplicação

O KCL requer um nome de aplicativo que seja exclusivo entre seus aplicativos e entre as tabelas do Amazon DynamoDB na mesma região. Ela usa o valor de configuração de nome de aplicativo das seguintes formas:

- Presume-se que todos os operadores associados a esse nome de aplicativo estejam trabalhando juntos no mesmo streaming. Esses operadores podem ser distribuídos entre várias instâncias. Se você executar uma instância adicional do mesmo código de aplicativo, mas com um nome de

aplicativo diferente, KCL tratará a segunda instância como um aplicativo totalmente separado que também está operando no mesmo stream.

- O KCL cria uma tabela do DynamoDB com o nome do aplicativo e usa a tabela para manter as informações de estado (como pontos de verificação e mapeamento de fragmentos de trabalho) do aplicativo. Cada aplicação tem sua própria tabela do DynamoDB. Para obter mais informações, consulte Use uma tabela de leasing para rastrear os fragmentos processados pelo aplicativo consumidor KCL.

Credenciais

Você deve disponibilizar suas AWS credenciais para um dos provedores de credenciais na cadeia de provedores de credenciais padrão. Você pode usar a propriedade `AWSCredentialsProvider` para definir um provedor de credenciais. Se você executar seu aplicativo de consumidor em uma EC2 instância da Amazon, recomendamos que você configure a instância com uma IAM função. AWS as credenciais que refletem as permissões associadas a essa IAM função são disponibilizadas aos aplicativos na instância por meio dos metadados da instância. Essa é a maneira mais segura de gerenciar as credenciais de um aplicativo de consumidor em execução em uma EC2 instância.

Desenvolva consumidores personalizados com taxa de transferência compartilhada usando o AWS SDK for Java

Um dos métodos para desenvolver consumidores personalizados do Kinesis Data Streams com compartilhamento total é usar o Amazon Kinesis Data Streams. APIs Esta seção descreve o uso do Kinesis APIs Data Streams AWS SDK com o for Java. O código de amostra Java nesta seção demonstra como realizar KDS API operações básicas e é dividido logicamente por tipo de operação.

Esses exemplos não representam um código pronto para produção. Eles não verificam todas as exceções possíveis nem levam em conta todas as considerações de segurança ou desempenho possíveis.

Você pode chamar o Kinesis APIs Data Streams usando outras linguagens de programação diferentes. Para obter mais informações sobre todas as opções disponíveis AWS SDKs, consulte Comece a desenvolver com a Amazon Web Services.

> **Important**
>
> O método recomendado para desenvolver consumidores personalizados do Kinesis Data Streams com conteúdo compartilhado é usar a Kinesis Client Library (). KCL KCLajuda

você a consumir e processar dados de um stream de dados do Kinesis ao cuidar de muitas das tarefas complexas associadas à computação distribuída. Para obter mais informações, consulte Desenvolvimento de consumidores personalizados com o uso KCL de taxa de transferência compartilhada.

Tópicos

- Obter dados de um stream
- Use iteradores de fragmentos
- Use GetRecords
- Adapte-se a uma nova fragmentação
- Interaja com os dados usando o AWS Glue Schema Registry

Obter dados de um stream

Os Kinesis APIs Data Streams `getShardIterator` incluem `getRecords` os métodos e que você pode invocar para recuperar registros de um stream de dados. Esse é o modelo de pull, em que o código extrai registros de dados diretamente dos fragmentos do fluxo de dados.

Important

Recomendamos que você use o suporte do processador de registros fornecido pela KCL para recuperar registros de seus fluxos de dados. Esse é o modelo push, em que você implementa o código que processa os dados. O KCL recupera registros de dados do fluxo de dados e os entrega ao código do seu aplicativo. Além disso, KCL fornece funcionalidade de failover, recuperação e balanceamento de carga. Para obter mais informações, consulte Desenvolvimento de consumidores personalizados com o uso KCL de taxa de transferência compartilhada.

No entanto, em alguns casos, talvez você prefira usar o Kinesis APIs Data Streams. Um exemplo disso é a implementação de ferramentas personalizadas de monitoramento ou depuração dos fluxos de dados.

> **Important**
>
> O Kinesis Data Streams oferece suporte a alterações do período de retenção do registro de dados no fluxo de dados. Para obter mais informações, consulte Alterar o período de retenção de dados.

Use iteradores de fragmentos

Você recupera registros do stream por estilhaço. Para cada estilhaço e para cada lote de registros que recupera desse estilhaço, você precisa obter um iterador de estilhaços. O iterador de estilhaços é usado no objeto `getRecordsRequest` para especificar o estilhaço a partir do qual os registros devem ser recuperados. O tipo associado ao iterador de estilhaços determina o ponto no estilhaço a partir do qual os registros devem ser recuperados (veja mais à frente nesta seção para obter mais detalhes). Antes de trabalhar com o iterador de fragmento, você deve recuperar o fragmento. Para obter mais informações, consulte Listar fragmentos.

Obtenha o iterador de estilhaços inicial usando o método `getShardIterator`. Obtenha iteradores de estilhaços para obter mais lotes de registros usando o método `getNextShardIterator` do objeto`getRecordsResult` retornado pelo método `getRecords`. Um iterador de estilhaços é válido por 5 minutos. Se usar um iterador de estilhaços enquanto ele for válido, você receberá um novo. Cada iterador de estilhaços permanecerá válido por 5 minutos, mesmo depois de ser usado.

Para obter o iterador de estilhaços inicial, instancie `GetShardIteratorRequest` e passe-o ao método `getShardIterator`. Para configurar a solicitação, especifique o stream e o ID do estilhaço. Para obter informações sobre como obter os streams em sua AWS conta, consulteListar streams. Para obter informações sobre como obter os estilhaços em um stream, consulte Listar fragmentos.

```
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();
```

Esse código de exemplo especifica `TRIM_HORIZON` como o tipo de iterador ao obter o iterador de estilhaços inicial. Esse tipo de iterador significa que o retorno dos registros deve começar a partir do primeiro registro adicionado ao fragmento, em vez de começar pelo registro adicionado mais recentemente, também conhecido como extremidade. Estes são tipos de iterador possíveis:

- `AT_SEQUENCE_NUMBER`
- `AFTER_SEQUENCE_NUMBER`
- `AT_TIMESTAMP`
- `TRIM_HORIZON`
- `LATEST`

Para obter mais informações, consulte ShardIteratorType.

Alguns tipos de iterador exigem que você especifique um número de sequência, além do tipo. Por exemplo:

```
getShardIteratorRequest.setShardIteratorType("AT_SEQUENCE_NUMBER");
getShardIteratorRequest.setStartingSequenceNumber(specialSequenceNumber);
```

Depois de obter um registro usando `getRecords`, você pode conseguir o número de sequência do registro chamando o método `getSequenceNumber` do registro.

```
record.getSequenceNumber()
```

Além disso, o código que adiciona registros ao stream de dados pode obter o número de sequência de um registro adicional chamando `getSequenceNumber` no resultado de `putRecord`.

```
lastSequenceNumber = putRecordResult.getSequenceNumber();
```

Você pode usar números de sequência para garantir estritamente o ordenamento crescente do registros. Para obter mais informações, consulte o exemplo de código em PutRecordexemplo.

Use GetRecords

Depois que você obtiver o iterador de estilhaços, instancie um objeto `GetRecordsRequest`. Especifique o iterador para a solicitação usando o método `setShardIterator`.

Opcionalmente, você também pode definir o número de registros a recuperar usando o método `setLimit`. O número de registros retornados por `getRecords` sempre é igual ou menor que esse limite. Se você não especificar esse limite, `getRecords` retornará 10 MB de registros recuperados. O código de exemplo a seguir define esse limite para 25 registros.

Se nenhum registro for retornado, isso significa que não há registros de dados disponíveis atualmente nesse estilhaço no número de sequência referenciado pelo iterador de estilhaços. Nessa situação, a aplicação deve aguardar o tempo adequado de acordo com as fontes de dados do fluxo. Em seguida, tente obter os dados do estilhaço novamente usando o iterador de estilhaços retornado pela chamada anterior a `getRecords`.

Passe o `getRecordsRequest` para o método `getRecords` e capture o valor retornado como um objeto `getRecordsResult`. Para obter os registros de dados, chame o método `getRecords` no objeto `getRecordsResult`.

```
GetRecordsRequest getRecordsRequest = new GetRecordsRequest();
getRecordsRequest.setShardIterator(shardIterator);
getRecordsRequest.setLimit(25);

GetRecordsResult getRecordsResult = client.getRecords(getRecordsRequest);
List<Record> records = getRecordsResult.getRecords();
```

Para preparar-se para outra chamada para `getRecords`, obtenha o próximo iterador de estilhaços de `getRecordsResult`.

```
shardIterator = getRecordsResult.getNextShardIterator();
```

Para obter os melhores resultados, aguarde pelo menos 1 segundo (1.000 milissegundos) entre as chamadas para `getRecords` a fim de evitar exceder o limite na frequência de `getRecords`.

```
try {
  Thread.sleep(1000);
}
catch (InterruptedException e) {}
```

Normalmente, você deve chamar `getRecords` em um loop, mesmo se estiver recuperando um único registro em um cenário de teste. Uma única chamada para `getRecords` pode retornar uma lista de registros vazia, mesmo quando o estilhaço contém mais registros em números de sequência subsequentes. Quando isso ocorre, o `NextShardIterator` retornado com a lista de registros vazia faz referência a um número de sequência subsequente no estilhaço, e as chamadas posteriores a `getRecords` acabam retornando os registros. O exemplo a seguir demonstra o uso de um loop.

Exemplo: getRecords

O código de exemplo a seguir reflete as dicas de `getRecords` nesta seção, inclusive chamadas em um loop.

```
// Continuously read data records from a shard
List<Record> records;

while (true) {

  // Create a new getRecordsRequest with an existing shardIterator
  // Set the maximum records to return to 25

  GetRecordsRequest getRecordsRequest = new GetRecordsRequest();
  getRecordsRequest.setShardIterator(shardIterator);
  getRecordsRequest.setLimit(25);

  GetRecordsResult result = client.getRecords(getRecordsRequest);

  // Put the result into record list. The result can be empty.
  records = result.getRecords();

  try {
    Thread.sleep(1000);
  }
  catch (InterruptedException exception) {
    throw new RuntimeException(exception);
  }

  shardIterator = result.getNextShardIterator();
}
```

Se você estiver usando a Kinesis Client Library, ela poderá fazer várias chamadas antes de retornar dados. Esse comportamento é intencional e não indica um problema com os dados KCL ou com seus dados.

Adapte-se a uma nova fragmentação

`getRecordsResult.getNextShardIterator` retorna `null` para indicar que o fragmento passou por uma divisão ou uma mesclagem. O fragmento agora está em estado de `CLOSED`, e você leu todos os registros de dados disponíveis nele.

Nesse cenário, você pode usar `getRecordsResult.childShards` para conhecer os fragmentos filho que foram criados pela divisão ou mesclagem do fragmento sendo processado. Para obter mais informações, consulte ChildShard.

No caso de uma divisão, os dois novos estilhaços têm `parentShardId` igual ao ID do estilhaço que você estava processando anteriormente. O valor de `adjacentParentShardId` dos dois estilhaços é `null`.

No caso de uma fusão, o único estilhaço novo criado tem `parentShardId` igual ao ID de um dos estilhaços pai e `adjacentParentShardId` igual ao ID do outro estilhaço. Seu aplicativo já leu todos os dados de um desses estilhaços. Esse é o estilhaço para o qual `getRecordsResult.getNextShardIterator` retornou `null`. Se a ordem dos dados for importante para o aplicativo, você deverá garantir que ele também leia todos os dados de outro estilhaço pai antes de ler qualquer dado novo de estilhaço filho criado pela fusão.

Se estiver usando vários processadores para recuperar dados do streaming (digamos, um processador por estilhaço) e ocorrer uma divisão ou fusão de estilhaços, você deverá aumentar ou diminuir o número de processadores para se adaptar à alteração no número de estilhaços.

Para obter mais informações sobre a refragmentação, incluindo uma discussão sobre estados de estilhaços (como `CLOSED`), consulte Refragmentar um stream.

Interaja com os dados usando o AWS Glue Schema Registry

Você pode integrar seus streams de dados do Kinesis com o AWS Glue Schema Registry. O Registro de AWS Glue Esquemas permite que você descubra, controle e desenvolva esquemas de forma centralizada, ao mesmo tempo em que garante que os dados produzidos sejam continuamente validados por um esquema registrado. O esquema define a estrutura e o formato de um registro de dados. Um esquema é uma especificação versionada para publicação, consumo

ou datastore confiáveis. O AWS Glue Schema Registry permite que você melhore a qualidade end-to-end dos dados e a governança de dados em seus aplicativos de streaming. Para obter mais informações, consulte Registro de esquemas do AWS Glue. Uma das formas de configurar essa integração é por meio do `GetRecords` Kinesis API Data Streams disponível AWS em Java. SDK

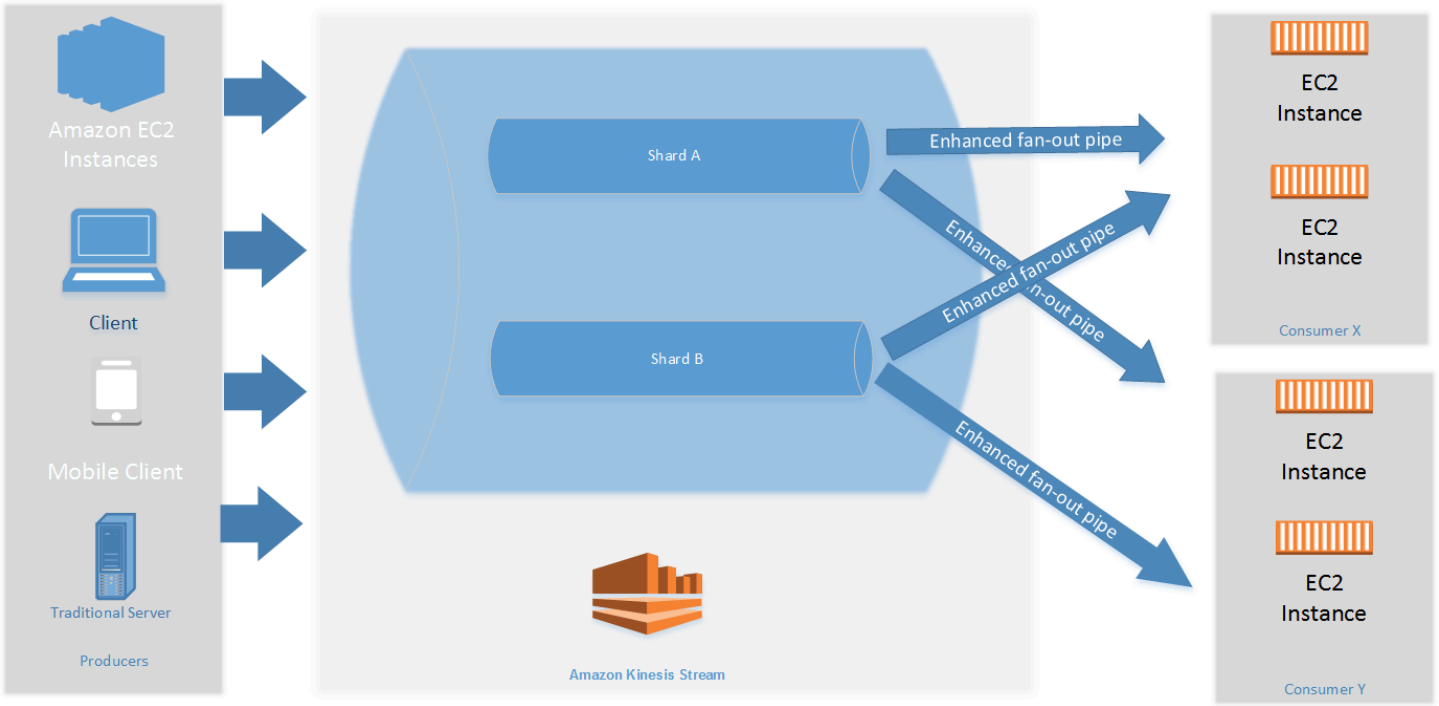
Para obter instruções detalhadas sobre como configurar a integração do Kinesis Data Streams com o Schema Registry `GetRecords` usando o Kinesis Data Streams, consulte a seção “Interagindo com dados usando o APIs Kinesis Data Streams” em Caso de uso: Integração do Amazon Kinesis Data APIs Streams com o Glue Schema Registry. AWS

Desenvolva consumidores personalizados com taxa de transferência dedicada (distribuição aprimorada)

No Amazon Kinesis Data Streams, você pode criar consumidores que usem um recurso chamado distribuição avançada. Esse recurso permite que consumidores recebam registros de um streaming com throughput de até 2 MB de dados por segundo por estilhaço. Essa throughput é dedicada, o que significa que consumidores que usam a divisão avançada não precisam lidar com outros consumidores que estejam recebendo dados do fluxo. O Kinesis Data Streams envia registros de dados do fluxo para consumidores que usam a distribuição avançada. Portanto, esses consumidores não precisam sondar dados.

> **Important**
>
> Você pode registrar até vinte consumidores por streaming para usar distribuição avançada.

O diagrama a seguir mostra a arquitetura de distribuição avançada. Se você usa a versão 2.0 ou posterior da Amazon Kinesis Client Library (KCL) para criar um consumidor, ela KCL configura o consumidor para usar o fan-out aprimorado para receber dados de todos os fragmentos do stream. Se você usar o API para criar um consumidor que usa fan-out aprimorado, poderá assinar fragmentos individuais.



O diagrama mostra o seguinte:

- Um streaming com dois estilhaços.
- Dois consumidores que estão usando a distribuição avançada para receber dados do streaming: consumidores X e Y. Os dois consumidores estão inscritos em todos os estilhaços e em todos os registros do streaming. Se você usar a versão 2.0 ou posterior do KCL para criar um consumidor, ele KCL inscreverá automaticamente esse consumidor em todos os fragmentos do stream. Por outro lado, se você usar o API para criar um consumidor, poderá assinar fragmentos individuais.
- Setas que representam as distribuições avançadas usadas pelos consumidores para receber dados do streaming. Uma distribuição avançada oferece até 2 MB/s de dados por estilhaço, independentemente de qualquer outro dado ou do número total de consumidores.

Tópicos

- Desenvolva consumidores expandidos aprimorados com 2.x KCL
- Desenvolva consumidores avançados com o Kinesis Data Streams API
- Gerencie consumidores de distribuição aprimorados com o AWS Management Console

Desenvolva consumidores expandidos aprimorados com 2.x KCL

Os consumidores que usam a distribuição avançada no Amazon Kinesis Data Streams podem receber registros de um fluxo de dados com throughput dedicada de até 2 MB de dados por segundo por fragmento. Esse tipo de consumidor não precisa lidar com outros consumidores que estejam recebendo dados do streaming. Para ter mais informações, consulte Desenvolva consumidores personalizados com taxa de transferência dedicada (distribuição aprimorada).

Você pode usar a versão 2.0 ou posterior da Kinesis Client Library (KCL) para desenvolver aplicativos que usam fan-out aprimorado para receber dados de streams. O inscreve KCL automaticamente seu aplicativo em todos os fragmentos de um stream e garante que seu aplicativo consumidor possa ler com um valor de taxa de transferência de 2 MB/seg por fragmento. Se você quiser usar o KCL sem ativar o fan-out aprimorado, consulte Desenvolvendo consumidores usando a Kinesis Client Library 2.0.

Tópicos

- Desenvolva consumidores avançados usando KCL 2.x em Java

Desenvolva consumidores avançados usando KCL 2.x em Java

Você pode usar a versão 2.0 ou posterior da Kinesis Client Library (KCL) para desenvolver aplicativos no Amazon Kinesis Data Streams para receber dados de streams usando fan-out aprimorado. O código a seguir mostra uma implementação de exemplo em Java de `ProcessorFactory` e `RecordProcessor`.

Recomendamos que você use `KinesisClientUtil` para criar `KinesisAsyncClient` e configurar `maxConcurrency` no `KinesisAsyncClient`.

> **Important**
>
> O Amazon Kinesis Client pode ter latência significativamente maior, a menos que você configure `KinesisAsyncClient` para ter um `maxConcurrency` alto o suficiente para permitir todas as concessões e usos adicionais do `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);
            }
        }
    }

}
```

Desenvolva consumidores avançados com o Kinesis Data Streams API

A distribuição avançada é um recurso do Amazon Kinesis Data Streams que permite que os consumidores recebam registros de um fluxo de dados com throughput dedicada de até 2 MB de dados por segundo por fragmento. Um consumidor que usa distribuição avançada não precisa lidar com outros consumidores que estejam recebendo dados do streaming. Para ter mais informações, consulte Desenvolva consumidores personalizados com taxa de transferência dedicada (distribuição aprimorada).

Você pode usar API as operações para criar um consumidor que usa o fan-out aprimorado no Kinesis Data Streams.

Para registrar um consumidor com fan-out aprimorado usando o Kinesis Data Streams API

1. Ligue RegisterStreamConsumerpara registrar seu aplicativo como um consumidor que usa fan-out aprimorado. O Kinesis Data Streams gera um Amazon Resource ARN Name () para o consumidor e o retorna na resposta.
2. Para começar a ouvir um fragmento específico, encaminhe ao consumidor ARN uma ligação para SubscribeToShard. Em seguida, o Kinesis Data Streams começa a enviar os registros desse fragmento para você, na forma de eventos SubscribeToShardEventdo tipo em uma conexão /2. HTTP A conexão permanece aberta por até 5 minutos. Ligue SubscribeToShardnovamente se quiser continuar recebendo registros do fragmento após o `future` retorno da chamada para ser SubscribeToShardconcluído normalmente ou excepcionalmente.

   > **Note**
   >
   > `SubscribeToShard`APItambém retorna a lista dos fragmentos secundários do fragmento atual quando o final do fragmento atual é atingido.

3. Para cancelar o registro de um consumidor que está usando o fan-out aprimorado, ligue. DeregisterStreamConsumer

O de código a seguir é um exemplo de como você pode inscrever o consumidor em um estilhaço, renovar a assinatura periodicamente e manipular os 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);
        }
    }
```

`event.ContinuationSequenceNumber` retorna `null` para indicar que o fragmento passou por uma divisão ou uma mesclagem. O fragmento agora está em estado de `CLOSED`, e você leu todos os registros de dados disponíveis nele. Nesse cenário, como mostrado no exemplo acima, você pode usar `event.childShards` para conhecer os fragmentos filho que foram criados pela divisão ou mesclagem do fragmento sendo processado. Para obter mais informações, consulte ChildShard.

Interaja com os dados usando o AWS Glue Schema Registry

Você pode integrar seus streams de dados do Kinesis com o AWS Glue Schema Registry. O Registro de AWS Glue Esquemas permite que você descubra, controle e desenvolva esquemas de forma centralizada, ao mesmo tempo em que garante que os dados produzidos sejam continuamente validados por um esquema registrado. O esquema define a estrutura e o formato de um registro de dados. Um esquema é uma especificação versionada para publicação, consumo ou datastore confiáveis. O AWS Glue Schema Registry permite que você melhore a qualidade end-to-end dos dados e a governança de dados em seus aplicativos de streaming. Para obter mais informações, consulte Registro de esquemas do AWS Glue. Uma das formas de configurar essa integração é por meio do `GetRecords` Kinesis API Data Streams disponível AWS em Java. SDK

Para obter instruções detalhadas sobre como configurar a integração do Kinesis Data Streams com o Schema Registry `GetRecords` usando o Kinesis Data Streams, consulte a seção “Interagindo com dados usando o APIs Kinesis Data Streams” em Caso de uso: Integração do Amazon Kinesis Data APIs Streams com o Glue Schema Registry. AWS

Gerencie consumidores de distribuição aprimorados com o AWS Management Console

Os consumidores que usam a distribuição avançada no Amazon Kinesis Data Streams podem receber registros de um fluxo de dados com throughput dedicada de até 2 MB de dados por segundo por fragmento. Para ter mais informações, consulte Desenvolva consumidores personalizados com taxa de transferência dedicada (distribuição aprimorada).

Você pode usar o AWS Management Console para ver uma lista de todos os consumidores registrados para usar o fan-out aprimorado com um stream específico. Para cada um desses consumidores, você pode ver detalhes como statusARN, data de criação e métricas de monitoramento.

Para visualizar consumidores registrados para usar distribuição avançada, o status, a data de criação e as métricas no console

1. Faça login AWS Management Console e abra o console do Kinesis em https://console.aws.amazon.com/kinesis.
2. Selecione Data Streams (Fluxos de dados) no painel de navegação.
3. Escolha um fluxo de dados do Kinesis para ver seus detalhes.
4. Na página de detalhes do streaming, escolha a guia Enhanced fan-out (Distribuição avançada).
5. Escolha um consumidor para visualizar seu nome, status e data de registro.

Para cancelar o registro de um consumidor

1. Abra o console do Kinesis em https://console.aws.amazon.com/kinesis.
2. Selecione Data Streams (Fluxos de dados) no painel de navegação.
3. Escolha um fluxo de dados do Kinesis para ver seus detalhes.
4. Na página de detalhes do streaming, escolha a guia Enhanced fan-out (Distribuição avançada).
5. Marque a caixa de seleção à esquerda do nome de cada consumidor cujo registro você deseja cancelar.
6. Escolha Deregister consumer (Cancelar registro de consumidor).

Migre os consumidores da KCL versão 1.x para a 2.x KCL

Este tópico explica as diferenças entre as versões 1.x e 2.x da Kinesis Client Library (). KCL Também mostra como migrar seu consumidor da versão 1.x para a versão 2.x do. KCL Depois que você migrar o cliente, ele iniciará o processamento de registros a partir do local verificado pela última vez.

A versão 2.0 do KCL apresenta as seguintes alterações na interface:

KCLMudanças na interface

KCLInterface 1.x	KCLInterface 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`	Compactada em `software.amazon.kinesis.processor.ShardRecordProcessor`

Tópicos

Migre o processador de registros

O exemplo a seguir mostra um processador de registros implementado para KCL 1.x:

```
package com.amazonaws.kcl;

import com.amazonaws.services.kinesis.clientlibrary.exceptions.InvalidStateException;
import com.amazonaws.services.kinesis.clientlibrary.exceptions.ShutdownException;
import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.IRecordProcessorCheckpointer;
import com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor;
import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IShutdownNotificationAware;
```

```
import com.amazonaws.services.kinesis.clientlibrary.lib.worker.ShutdownReason;
import com.amazonaws.services.kinesis.clientlibrary.types.InitializationInput;
import com.amazonaws.services.kinesis.clientlibrary.types.ProcessRecordsInput;
import com.amazonaws.services.kinesis.clientlibrary.types.ShutdownInput;

public class TestRecordProcessor implements IRecordProcessor,
 IShutdownNotificationAware {
    @Override
    public void initialize(InitializationInput initializationInput) {
        //
        // Setup record processor
        //
    }

    @Override
    public void processRecords(ProcessRecordsInput processRecordsInput) {
        //
        // Process records, and possibly checkpoint
        //
    }

    @Override
    public void shutdown(ShutdownInput shutdownInput) {
        if (shutdownInput.getShutdownReason() == ShutdownReason.TERMINATE) {
            try {
                shutdownInput.getCheckpointer().checkpoint();
            } catch (ShutdownException | InvalidStateException e) {
                throw new RuntimeException(e);
            }
        }
    }

    @Override
    public void shutdownRequested(IRecordProcessorCheckpointer checkpointer) {
        try {
            checkpointer.checkpoint();
        } catch (ShutdownException | InvalidStateException e) {
            //
            // Swallow exception
            //
            e.printStackTrace();
        }
    }
```

```
}
```

Como migrar a classe de processador de registro

1. Altere as interfaces de `com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor` e `com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IShutdownNotificat` para `software.amazon.kinesis.processor.ShardRecordProcessor`, da seguinte forma:

```
// 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. Atualize as instruções `import` para os métodos `initialize` e `processRecords`.

```
// import com.amazonaws.services.kinesis.clientlibrary.types.InitializationInput;
import software.amazon.kinesis.lifecycle.events.InitializationInput;

//import com.amazonaws.services.kinesis.clientlibrary.types.ProcessRecordsInput;
import software.amazon.kinesis.lifecycle.events.ProcessRecordsInput;
```

3. Substitua o método `shutdown` pelos novos métodos a seguir: `leaseLost`, `shardEnded`, e `shutdownRequested`.

```
//    @Override
//    public void shutdownRequested(IRecordProcessorCheckpointer checkpointer) {
//        //
//        // This is moved to shardEnded(...)
//        //
//        try {
//            checkpointer.checkpoint();
//        } catch (ShutdownException | InvalidStateException e) {
//            //
```

```
//            // Swallow exception
//            //
//            e.printStackTrace();
//        }
//    }

    @Override
    public void leaseLost(LeaseLostInput leaseLostInput) {

    }

    @Override
    public void shardEnded(ShardEndedInput shardEndedInput) {
        try {
            shardEndedInput.checkpointer().checkpoint();
        } catch (ShutdownException | InvalidStateException e) {
            //
            // Swallow the exception
            //
            e.printStackTrace();
        }
    }

//    @Override
//    public void shutdownRequested(IRecordProcessorCheckpointer checkpointer) {
//        //
//        // This is moved to shutdownRequested(ShutdownReauestedInput)
//        //
//        try {
//            checkpointer.checkpoint();
//        } catch (ShutdownException | InvalidStateException e) {
//            //
//            // Swallow exception
//            //
//            e.printStackTrace();
//        }
//    }

    @Override
    public void shutdownRequested(ShutdownRequestedInput shutdownRequestedInput) {
        try {
            shutdownRequestedInput.checkpointer().checkpoint();
        } catch (ShutdownException | InvalidStateException e) {
            //
```

```
            // Swallow the exception
            //
            e.printStackTrace();
        }
    }
```

Veja a seguir a versão atualizada da classe de processador de registro.

```
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();
        }
    }
}
```

Migre a fábrica de processadores de discos

A fábrica do processador de registros é responsável por criar processadores de registro quando uma concessão é realizada. Veja a seguir um exemplo de uma fábrica KCL 1.x.

```
package com.amazonaws.kcl;

import com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor;
import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessorFactory;

public class TestRecordProcessorFactory implements IRecordProcessorFactory {
    @Override
    public IRecordProcessor createProcessor() {
        return new TestRecordProcessor();
    }
}
```

Migrar a fábrica do processador de registros

1. Altere a interface implementada de `com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessorFa` para `software.amazon.kinesis.processor.ShardRecordProcessorFactory`, da seguinte forma:

```
// 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. Alterar a assinatura de retorno para `createProcessor`.

```
// public IRecordProcessor createProcessor() {
public ShardRecordProcessor shardRecordProcessor() {
```

Veja a seguir um exemplo da fábrica do processador de registros em 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();
    }
}
```

Migre o trabalhador

Na versão 2.0 doKCL, uma nova classe, chamada`Scheduler`, substitui a `Worker` classe. Veja a seguir um exemplo de um trabalhador KCL 1.x.

```
final KinesisClientLibConfiguration config = new KinesisClientLibConfiguration(...)
final IRecordProcessorFactory recordProcessorFactory = new RecordProcessorFactory();
final Worker worker = new Worker.Builder()
    .recordProcessorFactory(recordProcessorFactory)
    .config(config)
```

```
    .build();
```

Para migrar o operador

1. Altere a instrução `import` para a classe `Worker` para as instruções de importação para as classes `Scheduler` e `ConfigsBuilder`.

```
// import com.amazonaws.services.kinesis.clientlibrary.lib.worker.Worker;
import software.amazon.kinesis.coordinator.Scheduler;
import software.amazon.kinesis.common.ConfigsBuilder;
```

2. Crie o `ConfigsBuilder` e um `Scheduler` conforme mostrado no exemplo a seguir.

   Recomendamos que você use `KinesisClientUtil` para criar `KinesisAsyncClient` e configurar `maxConcurrency` no `KinesisAsyncClient`.

   > **Important**
   >
   > O Amazon Kinesis Client pode ter latência significativamente maior, a menos que você configure `KinesisAsyncClient` para ter um `maxConcurrency` alto o suficiente para permitir todas as concessões e usos adicionais do `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()
    );
```

Configurar o cliente Amazon Kinesis

Com a versão 2.0 da Kinesis Client Library, a configuração do cliente passou de uma única classe de configuração (`KinesisClientLibConfiguration`) para seis classes. A tabela a seguir descreve a migração.

Campos de Configuração e suas Novas Classes

Campo Original	Nova classe de configuração	Descrição
`applicationName`	`ConfigsBuilder`	O nome para isso é o KCL aplicativo. Usado como padrão para o `tableName` e o `consumerName` .
`tableName`	`ConfigsBuilder`	Permite substituir o nome usado para a tabela de concessão do Amazon DynamoDB.
`streamName`	`ConfigsBuilder`	O nome do stream a partir do qual esse aplicativo processa registros.
`kinesisEndpoint`	`ConfigsBuilder`	Essa opção não existe mais. Consulte remoções de configuração de cliente.

Campo Original	Nova classe de configuração	Descrição
`dynamoDBEndpoint`	`ConfigsBuilder`	Essa opção não existe mais. Consulte remoções de configuração de cliente.
`initialPositionInStreamExtended`	`RetrievalConfig`	O local no fragmento a partir do qual o KCL começa a buscar registros, começando com a execução inicial do aplicativo.
`kinesisCredentialsProvider`	`ConfigsBuilder`	Essa opção não existe mais. Consulte remoções de configuração de cliente.
`dynamoDBCredentialsProvider`	`ConfigsBuilder`	Essa opção não existe mais. Consulte remoções de configuração de cliente.
`cloudWatchCredentialsProvider`	`ConfigsBuilder`	Essa opção não existe mais. Consulte remoções de configuração de cliente.
`failoverTimeMillis`	LeaseManagementConfig	O número de milissegundos que devem passar antes que se considere uma falha do proprietário da concessão.
`workerIdentifier`	`ConfigsBuilder`	Um identificador exclusivo que representa a instanciação do processador do aplicativo. Isso deve ser exclusivo.
`shardSyncIntervalMillis`	LeaseManagementConfig	O tempo entre as chamadas de sincronização de estilhaços.
`maxRecords`	`PollingConfig`	Permite definir o número máximo de registros que o Kinesis retorna.

Campo Original	Nova classe de configuração	Descrição
`idleTimeBetweenReadsInMillis`	`CoordinatorConfig`	Essa opção não existe mais. Consulte a remoção do tempo ocioso.
`callProcessRecordsEvenForEmptyRecordList`	`ProcessorConfig`	Quando definido, o processador de registros é chamado mesmo quando o Kinesis não fornece nenhum registro.
`parentShardPollIntervalMillis`	`CoordinatorConfig`	Com que frequência um processador de registros deve sondar a conclusão de estilhaços pai.
`cleanupLeasesUponShardCompletion`	`LeaseManagementConfig`	Quando definidas, as concessões são removidas assim que as concessões filho iniciam o processamento.
`ignoreUnexpectedChildShards`	`LeaseManagementConfig`	Quando definidos, estilhaços filho que possuem um estilhaço aberto são ignorados. Essa configuração destina-se principalmente a fluxos do DynamoDB.
`kinesisClientConfig`	`ConfigsBuilder`	Essa opção não existe mais. Consulte remoções de configuração de cliente.
`dynamoDBClientConfig`	`ConfigsBuilder`	Essa opção não existe mais. Consulte remoções de configuração de cliente.
`cloudWatchClientConfig`	`ConfigsBuilder`	Essa opção não existe mais. Consulte remoções de configuração de cliente.
`taskBackoffTimeMillis`	`LifecycleConfig`	O tempo de espera para repetir tarefas com falha.

Campo Original	Nova classe de configuração	Descrição
`metricsBufferTimeMillis`	`MetricsConfig`	Controla a publicação de CloudWatch métricas.
`metricsMaxQueueSize`	`MetricsConfig`	Controla a publicação de CloudWatch métricas.
`metricsLevel`	`MetricsConfig`	Controla a publicação de CloudWatch métricas.
`metricsEnabledDimensions`	`MetricsConfig`	Controla a publicação de CloudWatch métricas.
`validateSequenceNumberBeforeCheckpointing`	`CheckpointConfig`	Essa opção não existe mais. Consulte a validação do número de sequência do ponto de verificação.
`regionName`	`ConfigsBuilder`	Essa opção não existe mais. Consulte remoção de configuração de cliente.
`maxLeasesForWorker`	`LeaseManagementConfig`	O número máximo de concessões que uma única instância do aplicativo deve aceitar.
`maxLeasesToStealAtOneTime`	`LeaseManagementConfig`	O número máximo de concessões que um aplicativo deve tentar roubar de uma só vez.
`initialLeaseTableReadCapacity`	`LeaseManagementConfig`	A IOPs leitura do DynamoDB que é usada se a bibliotec a cliente do Kinesis precisar criar uma nova tabela de lease do DynamoDB.

Campo Original	Nova classe de configuração	Descrição
`initialLeaseTableWriteCapacity`	`LeaseManagementConfig`	A IOPs leitura do DynamoDB que é usada se a bibliotec a cliente do Kinesis precisar criar uma nova tabela de lease do DynamoDB.
`initialPositionInStreamExtended`	LeaseManagementConfig	A posição inicial do aplicativo no stream. Isso é usado somente durante a criação da concessão inicial.
`skipShardSyncAtWorkerInitializationIfLeasesExist`	`CoordinatorConfig`	Desative a sincronização de dados de estilhaço se a tabela de concessão contiver concessões existentes. TODO: KinesisEco -438
`shardPrioritization`	`CoordinatorConfig`	A priorização de estilhaços a ser usada.
`shutdownGraceMillis`	N/D	Essa opção não existe mais. Consulte MultiLang Remoções.
`timeoutInSeconds`	N/D	Essa opção não existe mais. Consulte MultiLang Remoções.
`retryGetRecordsInSeconds`	`PollingConfig`	Configura o atraso entre as GetRecords tentativas de falhas.
`maxGetRecordsThreadPool`	`PollingConfig`	O tamanho do pool de fios usado para GetRecords.
`maxLeaseRenewalThreads`	`LeaseManagementConfig`	Controla o tamanho do grupo de threads de renovação de concessão. Quanto mais concessões seu aplicativo aceitar, maior esse grupo deve ser.

Campo Original	Nova classe de configuração	Descrição
`recordsFetcherFactory`	`PollingConfig`	Permite substituir a fábrica usada para criar extratores que recuperam dados dos streams.
`logWarningForTaskAfterMillis`	`LifecycleConfig`	Quanto tempo esperar antes de um aviso ser registrado caso uma tarefa não seja concluída.
`listShardsBackoffTimeInMillis`	`RetrievalConfig`	O número de milissegundos de espera entre as chamadas para `ListShards` em caso de falha.
`maxListShardsRetryAttempts`	`RetrievalConfig`	O número máximo de novas tentativas de `ListShards` antes de desistir.

Remoção do tempo de inatividade

Na versão 1.x doKCL, o `idleTimeBetweenReadsInMillis` correspondia a duas quantidades:

- A quantidade de tempo entre as verificações de envio de tarefas.
  Agora você pode configurar esse tempo entre tarefas, definindo `CoordinatorConfig#shardConsumerDispatchPollIntervalMillis`.
- A quantidade de tempo inativo quando nenhum registro é retornado do Kinesis Data Streams.
  Na versão 2.0, em distribuição avançada registros são enviados a partir de sua respectiva recuperação. Atividade no do consumidor estilhaço só ocorre quando uma solicitação é enviada.

Remoções da configuração do cliente

Na versão 2.0, o KCL não cria mais clientes. Ela depende do fornecimento de um cliente válido pelo usuário. Com essa alteração, todos os parâmetros de configuração que controlavam a criação do cliente foram removidos. Se precisar desses parâmetros, você pode configurá-los nos clientes antes de fornecê-los ao `ConfigsBuilder`.

Campo removido	Configuração equivalente
`kinesisEndpoint`	Configure o SDK `KinesisAsyncClient` com o endpoint preferido:`KinesisAsyncClient.builder().endpointOverride(URI.create("https://<kinesis endpoint>")).build()` .
`dynamoDBEndpoint`	Configure o SDK `DynamoDbAsyncClient` com o endpoint preferido:`DynamoDbAsyncClient.builder().endpointOverride(URI.create("https://<dynamodb endpoint>")).build()` .
`kinesisClientConfig`	Configure o SDK `KinesisAsyncClient` com a configuração necessária:`KinesisAsyncClient.builder().overrideConfiguration(<your configuration>).build()` .
`dynamoDBClientConfig`	Configure o SDK `DynamoDbAsyncClient` com a configuração necessária:`DynamoDbAsyncClient.builder().overrideConfiguration(<your configuration>).build()` .
`cloudWatchClientConfig`	Configure o SDK `CloudWatchAsyncClient` com a configuração necessária:`CloudWatchAsyncClient.builder().overrideConfiguration(<your configuration>).build()` .
`regionName`	Configure o SDK com a região preferida. Isso é o mesmo para todos os SDK clientes. Por exemplo, `KinesisAsyncClient.builder().region(Region.US_WEST_2).build()` .

Leia dados do Kinesis Data Streams AWS usando outros serviços

Os AWS serviços a seguir podem se integrar diretamente ao Amazon Kinesis Data Streams para ler dados dos fluxos de dados do Kinesis. Revise as informações de cada serviço em que você está interessado e consulte as referências fornecidas.

Tópicos

- Leia dados do Kinesis Data Streams usando a Amazon EMR
- Leia dados do Kinesis Data Streams EventBridge usando o Amazon Pipes
- Leia dados do Kinesis Data Streams usando AWS Glue

- Leia dados do Kinesis Data Streams usando o Amazon Redshift

Leia dados do Kinesis Data Streams usando a Amazon EMR

EMROs clusters da Amazon podem ler e processar streams do Kinesis diretamente, usando ferramentas conhecidas no ecossistema Hadoop, como Hive, Pig MapReduce, Hadoop Streaming e Cascading. API Você também pode unir dados em tempo real do Kinesis Data Streams com dados existentes no Amazon S3, no Amazon DynamoDB e em um cluster em execução. HDFS Você pode carregar diretamente os dados da Amazon EMR para o Amazon S3 ou o DynamoDB para atividades de pós-processamento.

Para obter mais informações, consulte Amazon Kinesis no Guia de EMRlançamento da Amazon.

Leia dados do Kinesis Data Streams EventBridge usando o Amazon Pipes

O Amazon EventBridge Pipes oferece suporte aos streams de dados do Kinesis como fonte. O Amazon EventBridge Pipes ajuda você a criar point-to-point integrações entre produtores e consumidores de eventos com etapas opcionais de transformação, filtragem e enriquecimento. Você pode usar o EventBridge Pipes para receber registros em um stream de dados do Kinesis e, opcionalmente, filtrar ou aprimorar esses registros antes de enviá-los para um dos destinos disponíveis para processamento, incluindo o Kinesis Data Streams.

Para obter mais informações, consulte Amazon Kinesis Stream como fonte no Amazon EventBridge Release Guide.

Leia dados do Kinesis Data Streams usando AWS Glue

Usando AWS Glue streamingETL, você pode criar trabalhos de streaming, extrair, transformar e carregar (ETL) que são executados continuamente e consomem dados do Amazon Kinesis Data Streams. Os trabalhos limpam e transformam os dados e, em seguida, carregam os resultados em data lakes JDBC ou datastores do Amazon S3.

Para obter mais informações, consulte ETLTrabalhos de streaming AWS Glue no Guia AWS Glue de lançamento.

Leia dados do Kinesis Data Streams usando o Amazon Redshift

A ingestão de streaming do Amazon Redshift oferece suporte ao Amazon Kinesis Data Streams. O recurso de ingestão de streaming do Amazon Redshift fornece ingestão de dados de baixa latência e

alta velocidade do Amazon Kinesis Data Streams em uma visão materializada do Amazon Redshift. A ingestão de streaming do Amazon Redshift elimina a necessidade de armazenar dados no Amazon S3 antes de ingeri-los no Amazon Redshift.

Para obter mais informações, consulte Ingestão de streaming no Guia de lançamento do Amazon Redshift.

Leia o Kinesis Data Streams usando integrações de terceiros

Você pode ler dados dos streams de dados do Amazon Kinesis Data Streams usando uma das seguintes opções de terceiros que se integram ao Kinesis Data Streams. Selecione a opção sobre a qual você deseja saber mais e encontre recursos e links para a documentação relevante.

Tópicos

- Apache Flink
- Adobe Experience Platform
- Apache Druid
- Apache Spark
- Databricks
- Kafka Confluent Platform
- Kinesumer
- Talend

Apache Flink

O Apache Flink é um framework de código aberto distribuído para computações com estado em fluxos de dados delimitados e não delimitados. Para obter mais informações sobre o consumo do Kinesis Data Streams usando o Apache Flink, consulte Amazon Kinesis Data Streams Connector.

Adobe Experience Platform

A Adobe Experience Platform permite que as organizações centralizem e padronizem dados dos clientes em qualquer sistema. Em seguida, a plataforma aplica ciência de dados e machine learning para melhorar significativamente o design e a entrega de experiências ricas e personalizadas.

Para obter mais informações sobre o consumo de streams de dados do Kinesis usando a Adobe Experience Platform, consulte Conector do Amazon Kinesis.

Apache Druid

O Druid é um banco de dados analítico em tempo real de alta performance que entrega consultas em menos de um segundo feitas em dados de streaming e lote em escala e sob carga. Para obter mais informações sobre a ingestão de streams de dados do Kinesis usando o Apache Druid, consulte Ingestão do Amazon Kinesis.

Apache Spark

O Apache Spark é um mecanismo de análise unificado para processamento de dados em grande escala. Ele fornece alto nível APIs em Java, Scala, Python e R e um mecanismo otimizado que suporta gráficos de execução geral. Você pode usar o Apache Spark para criar aplicativos de processamento de streams que consumam os dados em seus streams de dados do Kinesis.

Para consumir streams de dados do Kinesis usando o Apache Spark Structured Streaming, use o conector Amazon Kinesis Data Streams. Esse conector suporta o consumo com o Enhanced Fan-Out, que fornece ao seu aplicativo uma taxa de transferência de leitura dedicada de até 2 MB de dados por segundo por fragmento. Para obter mais informações, consulte Desenvolvimento de consumidores personalizados com taxa de transferência dedicada (fan-out aprimorado).

Para consumir streams de dados do Kinesis usando o Spark Streaming, consulte Spark Streaming + Kinesis Integration.

Databricks

O Databricks é uma plataforma baseada em nuvem que fornece um ambiente colaborativo para engenharia de dados, ciência de dados e machine learning. Para obter mais informações sobre o consumo de streams de dados do Kinesis usando o Databricks, consulte Connect to Amazon Kinesis.

Kafka Confluent Platform

A Confluent Platform, construída com base no Kafka, fornece recursos e funcionalidades adicionais que ajudam as empresas a criar e gerenciar pipelines de dados e aplicações de streaming em tempo real. Para obter mais informações sobre o consumo de streams de dados do Kinesis usando a plataforma Confluent, consulte Amazon Kinesis Source Connector for Confluent Platform.

Kinesumer

O Kinesumer é um cliente Go que implementa um cliente de grupo de consumidores distribuído do lado do cliente para fluxos de dados do Kinesis. Para obter mais informações, consulte o repositório do Kinesumer no GitHub.

Talend

O Talend é um software de integração e gerenciamento de dados que permite que os usuários coletem, transformem e conectem dados de várias fontes de maneira escalável e eficiente. Para obter mais informações sobre o consumo de streams de dados do Kinesis usando o Talend, consulte Connect talend a um stream do Amazon Kinesis.

Solucionar problemas dos consumidores do Kinesis Data Streams

Os tópicos a seguir oferecem soluções para problemas comuns com consumidores do Amazon Kinesis Data Streams:

- Alguns registros do Kinesis Data Streams são ignorados ao usar a Biblioteca de cliente do Kinesis
- Registros pertencentes ao mesmo fragmento são processados por processadores de registros diferentes ao mesmo tempo
- O aplicativo do consumidor está lendo em um ritmo mais lento do que o esperado
- GetRecords retorna uma matriz de registros vazia mesmo quando há dados no fluxo
- O uterador fragmentado expira inesperadamente
- O processamento de registros de consumidores está ficando para trás
- Erro de permissão de chave KMS mestra não autorizada
- Solucionar outros problemas comuns dos consumidores

Alguns registros do Kinesis Data Streams são ignorados ao usar a Biblioteca de cliente do Kinesis

A causa mais comum de registros ignorados é uma exceção não processada lançada de `processRecords`. A Kinesis Client Library (KCL) depende do seu `processRecords` código para lidar com quaisquer exceções decorrentes do processamento dos registros de dados. Qualquer exceção lançada `processRecords` é absorvida peloKCL. Para evitar repetições infinitas em

caso de falha recorrente, o KCL não reenvia o lote de registros processados no momento da exceção. KCLEm seguida, `processRecords` solicita o próximo lote de registros de dados sem reiniciar o processador de registros. Isso resulta efetivamente em aplicativos consumidores observando registros ignorados. Para impedir registros ignorados, processe todas as exceções em `processRecords` da forma apropriada.

Registros pertencentes ao mesmo fragmento são processados por processadores de registros diferentes ao mesmo tempo

Para qualquer aplicativo Kinesis Client Library (KCL) em execução, um fragmento tem apenas um proprietário. No entanto, vários processadores de registro pode temporariamente processar o mesmo estilhaço. No caso de uma instância de trabalho que perde a conectividade de rede, KCL pressupõe que o trabalhador inacessível não está mais processando registros após o término do tempo de failover e orienta outras instâncias de trabalho a assumirem o controle. Por um breve período, novos processadores de registros e processadores de registros provenientes do operador inacessível podem processar dados do mesmo estilhaço.

Você deve definir um tempo de failover que seja apropriado para seu aplicativo. Para aplicativos de baixa latência, o padrão de 10 segundos pode representar o tempo máximo que você deseja esperar. No entanto, nos casos em que você espera problemas de conectividade, como fazer chamadas em áreas geográficas em que a conectividade pode ser perdida com mais frequência, esse número pode ser muito baixo.

O aplicativo deve prever e lidar com esse cenário, especialmente porque a conectividade de rede normalmente é restaurada para o operador anteriormente inacessível. Se um processador de registros tem seus estilhaços executados por outro processador de registros, ele deve lidar com os dois casos seguintes para executar o encerramento normal:

1. Depois que a chamada atual `processRecords` for concluída, ele KCL invoca o método de desligamento no processador de gravação com o motivo de desligamento ". ZOMBIE Espera-se que seus processadores de registros limpem quaisquer recursos conforme apropriado e, em seguida, saiam.
2. Quando você tenta fazer o checkpoint de um trabalhador “zumbi”, ele joga. KCL `ShutdownException` Depois de receber essa exceção, seu código deve sair do método atual de forma limpa.

Para ter mais informações, consulte Lidar com registros duplicados.

O aplicativo do consumidor está lendo em um ritmo mais lento do que o esperado

Os motivos mais comuns para a throughput de leitura ser mais lenta do que o esperado são os seguintes:

1. Vários aplicativos consumidores com um total de leituras que excedem os limites por estilhaço. Para ter mais informações, consulte Cotas e limites. Nesse caso, aumente o número de fragmentos no fluxo de dados do Kinesis.
2. O limite que especifica o número máximo de GetRecords por chamada pode ter sido configurado com um valor baixo. Se você estiver usando oKCL, talvez tenha configurado o trabalhador com um valor baixo para a `maxRecords` propriedade. Em geral, recomendamos o uso dos padrões do sistema para essa propriedade.
3. A lógica de sua `processRecords` chamada pode estar demorando mais do que o esperado por vários motivos possíveis; a lógica pode ser CPU intensa, bloquear a E/S ou ter um gargalo na sincronização. Para testar se isso é verdadeiro, teste a execução de processadores de registros vazios e comparar a throughput de leitura. Para obter informações sobre como acompanhar os dados de entrada, consulte Use refragmentação, escalonamento e processamento paralelo para alterar o número de fragmentos.

Se você tem apenas um aplicativo consumidor, sempre é possível ler pelo menos duas vezes mais rápido do que a taxa de colocação. É possível gravar até 1.000 registros por segundo, até uma taxa máxima total de gravação de dados de 1 MB por segundo (incluindo chaves de partição). Cada fragmento aberto oferece suporte a até cinco transações por segundo para leituras, até uma taxa máxima total de leitura de dados de 2 MB por segundo. Observe que cada leitura (chamada a GetRecords) obtém um lote de registros. O tamanho dos dados retornados pelo GetRecords varia de acordo com a utilização do estilhaço. O tamanho máximo de dados que GetRecords pode retornar é 10 MB. Se uma chamada retorna esse limite, as chamadas subsequentes feitas nos próximos 5 segundos lançam `ProvisionedThroughputExceededException`.

GetRecords retorna uma matriz de registros vazia mesmo quando há dados no fluxo

O consumo, ou a obtenção, de registros é um modelo de envio. Espera-se que os desenvolvedores GetRecordsliguem em um loop contínuo, sem atrasos. Cada chamada a GetRecords também retorna um valor de `ShardIterator` que deve ser usado na próxima iteração do loop.

A operação GetRecords não bloqueia. Em vez disso, ela retorna imediatamente; com registros de dados relevantes ou com um elemento `Records` vazio. Um elemento `Records` vazio é retornado em duas condições:

1. Atualmente, não há mais dados no estilhaço.
2. Não há dados perto da parte do estilhaço indicada pelo `ShardIterator`.

A última condição é sutil, mas é uma característica de compensação necessária para evitar tempo de busca ilimitado (latência) ao recuperar registros. Desse modo, o aplicativo de consumo de streaming deve fazer um loop e chamar GetRecords tratando os registros vazios como de costume.

Em um cenário de produção, o único momento em que o loop contínuo deve ser encerrado é quando o valor de `NextShardIterator` é `NULL`. Quando `NextShardIterator` é `NULL`, significa que o estilhaço foi fechado e o valor de `ShardIterator` apontaria para além do último registro. Se o aplicativo consumidor nunca chamar SplitShard ou MergeShards, o estilhaço permanecerá aberto e as chamadas a GetRecords nunca retornarão um valor de `NextShardIterator` que seja `NULL`.

Se você usa a Kinesis Client Library (KCL), o padrão de consumo acima é resumido para você. Isso inclui a manipulação automática de um conjunto de estilhaços que mudam dinamicamente. Com oKCL, o desenvolvedor fornece apenas a lógica para processar os registros recebidos. Isso é possível porque a biblioteca faz chamadas contínuas a GetRecords para você.

O uterador fragmentado expira inesperadamente

Um novo iterador de estilhaço é retornado por toda solicitação a GetRecords (como `NextShardIterator`), que você usa na próxima solicitação GetRecords (como `ShardIterator`). Normalmente, esse iterador do estilhaço não expira antes de você usá-lo. No entanto, você pode descobrir que iteradores de estilhaço expiram porque você não chamou GetRecords por mais de cinco minutos, ou porque você executou uma reinicialização do seu aplicativo consumidor.

Se o iterador do fragmento expirar imediatamente, antes que você possa usá-lo, isso poderá indicar que a tabela do DynamoDB usada pelo Kinesis não tem capacidade suficiente para armazenar os dados de concessões. Essa situação tem maior probabilidade de ocorrer se você tiver um grande número de estilhaços. Para solucionar esse problema, aumente a capacidade de gravação atribuída à tabela do estilhaço. Para ter mais informações, consulte Use uma tabela de leasing para rastrear os fragmentos processados pelo aplicativo consumidor KCL.

O processamento de registros de consumidores está ficando para trás

Para a maioria dos casos de uso, os aplicativos consumidores estão lendo os últimos dados do stream. Em determinadas circunstâncias, as leituras dos consumidores podem ficar atrasadas, o que pode não ser desejado. Depois de identificar a dimensão do atraso da leitura dos consumidores, veja os motivos mais comuns disso ocorrer.

Comece com a métrica `GetRecords.IteratorAgeMilliseconds`, que rastreia a posição de leitura em todos os estilhaços e consumidores no stream. Observe que, se a idade de um iterador passar de 50% do período de retenção (24 horas por padrão, configurável até 365 dias), haverá risco de perda de dados devido à expiração de registro. Uma solução provisória rápida é aumentar o período de retenção. Isso interrompe a perda de dados importantes enquanto você soluciona o problema. Para ter mais informações, consulte Monitore o serviço Amazon Kinesis Data Streams com a Amazon CloudWatch. Em seguida, identifique o quão atrasado seu aplicativo consumidor está lendo cada fragmento usando uma CloudWatch métrica personalizada emitida pela Kinesis Client Library (),. KCL `MillisBehindLatest` Para ter mais informações, consulte Monitore a biblioteca de cliente do Kinesis com a Amazon CloudWatch.

Veja os motivos mais comuns para consumidores ficarem atrasados:

- Grandes aumentos repentinos `GetRecords.IteratorAgeMilliseconds` ou `MillisBehindLatest` geralmente indicam um problema transitório, como falhas de API operação em um aplicativo posterior. Você deve investigar esses aumentos repentinos se uma das métricas exibir esse comportamento constantemente.
- Um aumento gradual nessas métricas indica que um consumidor não está acompanhando o stream porque não está processando registros com a rapidez necessária. As causas raiz mais comuns para esse comportamento são recursos físicos insuficientes ou lógica de processamento de registros que não escalou com o aumento na throughput do stream. Você pode verificar esse comportamento observando as outras CloudWatch métricas personalizadas que ele KCL emite associadas à `processTask` operação, incluindo `RecordProcessor.processRecords.TimeSuccess`, e. `RecordsProcessed`
  - Caso veja um aumento na métrica `processRecords.Time` correlacionada ao aumento na throughput, você deve analisar a lógica do processamento de registros para identificar por que ela não está se dimensionando de acordo com a maior throughput.
  - Caso veja um aumento nos valores de `processRecords.Time` que não esteja correlacionado com aumento na throughput, verifique se você está fazendo chamadas de bloqueio no caminho crítico, que muitas vezes são a causa de lentidão no processamento de registros. Uma

abordagem alternativa é aumentar o paralelismo, aumentando o número de estilhaços. Por fim, confirme se você tem uma quantidade adequada de recursos físicos (memória, CPU utilização etc.) nos nós de processamento subjacentes durante o pico de demanda.

Erro de permissão de chave KMS mestra não autorizada

Esse erro ocorre quando um aplicativo consumidor lê um fluxo criptografado sem permissões na chave KMS mestra. Para atribuir permissões a um aplicativo para acessar uma KMS chave, consulte Usando políticas de chaves em AWS KMS e Usando IAM políticas com AWS KMS.

Solucionar outros problemas comuns dos consumidores

- Por que o gatilho do Kinesis Data Streams não consegue invocar minha função do Lambda?
- Como detecto e soluciono problemas de ReadProvisionedThroughputExceeded exceções no Kinesis Data Streams?
- Why am I experiencing high latency issues with Kinesis Data Streams?
- Why is my Kinesis data stream returning a 500 Internal Server Error?
- Como soluciono problemas de um KCL aplicativo bloqueado ou bloqueado no Kinesis Data Streams?
- Can I use different Amazon Kinesis Client Library applications with the same Amazon DynamoDB table?

Otimize os consumidores do Amazon Kinesis Data Streams

Você pode otimizar ainda mais seu consumidor do Amazon Kinesis Data Streams com base no comportamento específico que você vê.

Analise os tópicos a seguir para identificar soluções.

Tópicos

- Melhore o processamento de baixa latência
- Processe dados serializados usando AWS Lambda a Kinesis Producer Library
- Use refragmentação, escalonamento e processamento paralelo para alterar o número de fragmentos
- Lidar com registros duplicados

- Gerencie a inicialização, o desligamento e a aceleração

Melhore o processamento de baixa latência

O atraso de propagação é definido como a end-to-end latência do momento em que um registro é gravado no stream até ser lido por um aplicativo consumidor. Esse atraso varia dependendo de uma série de fatores, mas é afetado principalmente pelo intervalo de sondagem de aplicativos de consumidor.

Para a maioria dos aplicativos, recomendamos a sondagem de cada estilhaço uma vez por segundo por aplicativo. Isso permite que você tenha várias aplicações de consumo processando um fluxo simultaneamente sem atingir os limites do Amazon Kinesis Data Streams de cinco chamadas de `GetRecords` por segundo. Além disso, o processamento de lotes de dados maiores tende a ser mais eficiente na redução da latência de rede e outras latências de downstream no seu aplicativo.

Os KCL padrões são definidos para seguir as melhores práticas de pesquisa a cada 1 segundo. Esse padrão gera atrasos de propagação médios que costumam ser menores que 1 segundo.

Os registros do Kinesis Data Streams ficam disponíveis para serem lidos imediatamente após serem gravados. Há alguns casos de uso que precisam aproveitar isso e exigir o consumo de dados do stream assim que ele estiver disponível. Você pode reduzir significativamente o atraso de propagação substituindo as configurações KCL padrão para pesquisar com mais frequência, conforme mostrado nos exemplos a seguir.

Código de configuração Java da KCL:

```
kinesisClientLibConfiguration = new
        KinesisClientLibConfiguration(applicationName,
        streamName,
        credentialsProvider,

 workerId).withInitialPositionInStream(initialPositionInStream).withIdleTimeBetweenReadsInMilli
```

Configuração do arquivo de propriedades para Python e Ruby: KCL

```
idleTimeBetweenReadsInMillis = 250
```

> **Note**
>
> Como o Kinesis Data Streams tem um limite de cinco chamadas de `GetRecords` por segundo por fragmento, configurar a propriedade `idleTimeBetweenReadsInMillis` abaixo de 200 ms pode fazer com que a aplicação receba a exceção `ProvisionedThroughputExceededException`. Muitas dessas exceções podem gerar recuos exponenciais significativos e, portanto, causar latências inesperadas significativas no processamento. Se você definir essa propriedade em 200 ms ou acima e tiver mais de um aplicativo de processamento, ocorrerá limitação semelhante.

Processe dados serializados usando AWS Lambda a Kinesis Producer Library

A Kinesis Producer Library (KPL) agrega pequenos registros formatados pelo usuário em registros maiores de até 1 MB para fazer melhor uso da taxa de transferência do Amazon Kinesis Data Streams. Embora o KCL for Java ofereça suporte à desagregação desses registros, você precisa usar um módulo especial para desagregar registros ao usá-lo AWS Lambda como consumidor de seus fluxos. Você pode obter o código e as instruções do projeto necessários nos Módulos de GitHub desagregação da Kinesis Producer Library para Lambda. AWS Os componentes deste projeto oferecem a capacidade de processar dados KPL serializados em Java AWS Lambda, Node.js e Python. Esses componentes também podem ser usados como parte de um aplicativo multilíngue KCL.

Use refragmentação, escalonamento e processamento paralelo para alterar o número de fragmentos

O reestilhaçamento permite aumentar ou diminuir o número de estilhaços em um stream para se adaptar às alterações na taxa de dados que fluem pelo streaming. O reestilhaçamento costuma ser realizado por um aplicativo administrativo que monitora as métricas de tratamento de dados de estilhaço. Embora o KCL próprio não inicie operações de refragmentação, ele foi projetado para se adaptar às mudanças no número de fragmentos resultantes da refragmentação.

Conforme observado emUse uma tabela de leasing para rastrear os fragmentos processados pelo aplicativo consumidor KCL, ele KCL rastreia os fragmentos no stream usando uma tabela do Amazon DynamoDB. Quando novos fragmentos são criados como resultado da refragmentação, ele KCL descobre os novos fragmentos e preenche novas linhas na tabela. Os operadores

descobrem automaticamente os novos estilhaços e criam processadores para tratar os dados provenientes deles. Ele KCL também distribui os fragmentos no fluxo entre todos os trabalhadores e processadores de registros disponíveis.

KCLIsso garante que todos os dados que existiam nos fragmentos antes da refragmentação sejam processados primeiro. Depois do processamento dos dados, os dados dos novos estilhaços são enviados para processadores de registros. Dessa forma, KCL preserva a ordem na qual os registros de dados foram adicionados ao fluxo para uma chave de partição específica.

Exemplo: refragmentação, escalabilidade e processamento paralelo

O exemplo a seguir ilustra como isso KCL ajuda você a lidar com o dimensionamento e a refragmentação:

- Por exemplo, se seu aplicativo estiver sendo executado em uma EC2 instância e estiver processando um stream de dados do Kinesis com quatro fragmentos. Essa instância tem um KCL trabalhador e quatro processadores de registros (um processador de registros para cada fragmento). Esses quatro processadores de registros são executados em paralelo no mesmo processo.
- Em seguida, se você escalar o aplicativo para usar outra instância, terá duas instâncias processando um único stream que tem quatro estilhaços. Quando o KCL trabalhador é inicializado na segunda instância, ele balanceia a carga com a primeira instância, de forma que cada instância agora processe dois fragmentos.
- Se você decidir dividir os quatro estilhaços em cinco, O KCL novamente coordena o processamento entre instâncias: uma instância processa três fragmentos e a outra processa dois fragmentos. Uma coordenação semelhante ocorre quando você mescla estilhaços.

Normalmente, ao usar oKCL, você deve garantir que o número de instâncias não exceda o número de fragmentos (exceto para fins de espera de falhas). Cada fragmento é processado por exatamente um KCL trabalhador e tem exatamente um processador de registros correspondente, portanto, você nunca precisará de várias instâncias para processar um fragmento. Mas como um operador pode processar qualquer número de estilhaços, tudo bem se o número de estilhaços ultrapassar o número de instâncias.

Para expandir o processamento do seu aplicativo, você deve testar uma combinação destas abordagens:

- Aumentar o tamanho da instância (porque todos os processadores de registros são executados em paralelo em um processo)
- Aumentar o número de instâncias até o número máximo de estilhaços abertos (porque os estilhaços podem ser processados de forma independente)
- Aumentar o número de estilhaços (o que aumenta o nível de paralelismo)

Observe que você pode usar o ajuste de escala automático para escalar automaticamente as instâncias com base em métricas apropriadas. Para obter mais informações, consulte o Guia do usuário do Amazon EC2 Auto Scaling.

Quando a refragmentação aumenta o número de fragmentos no stream, o aumento correspondente no número de processadores de registro aumenta a carga nas EC2 instâncias que os hospedam. Se as instâncias fazem parte de um grupo de Auto Scaling e a carga aumenta suficientemente, o grupo de Auto Scaling adiciona mais instâncias para lidar com o aumento de carga. Você deve configurar as instâncias para iniciar a aplicação do Amazon Kinesis Data Streams na inicialização, para que os operadores e processadores de registros adicionais se tornem imediatamente ativos na nova instância.

Para obter mais informações sobre a refragmentação, consulte Refragmentar um stream.

Lidar com registros duplicados

Há dois principais motivos pelos quais os registros podem ser entregues mais de uma vez à aplicação do Amazon Kinesis Data Streams: novas tentativas de produtor e novas tentativas de consumidor. Seu aplicativo precisa prever e tratar devidamente o processamento de registros individuais várias vezes.

Produtor tenta novamente

Considere um produtor que tem um tempo limite esgotado de rede depois de fazer uma chamada a `PutRecord`, mas antes de poder receber uma confirmação do Amazon Kinesis Data Streams. O produtor não tem certeza de que o registro foi entregue ao Kinesis Data Streams. Supondo que cada registro é importante para o aplicativo, o produtor teria sido concebido de modo a tentar novamente a chamada com os mesmos dados. Se as duas chamadas a `PutRecord` para os mesmos dados foram confirmadas com sucesso no Kinesis Data Streams, haverá dois registros do Kinesis Data Streams. Embora os dois registros tenham dados idênticos, eles também têm números sequenciais exclusivos. Os aplicativos que precisam de rigorosas garantias devem incorporar uma chave

primária ao registro para remover duplicatas mais tarde ao processar. Observe que o número de duplicatas resultante das retentativas de produtor costuma ser baixo em comparação com o número de duplicatas resultante das retentativas de consumidor.

> **Note**
>
> Se você usa o AWS SDK`PutRecord`, saiba mais sobre o comportamento de SDK repetição no guia do usuário AWS SDKs e Ferramentas.

Tentativas do consumidor

As retentativas de consumidor (aplicativo de processamento de dados) acontecem quando os processadores de registros são reiniciados. Os processadores de registros do mesmo estilhaço reiniciam nos seguintes casos:

1. Um operador é encerrado inesperadamente
2. Instâncias de operador são adicionadas ou removidas
3. Os estilhaços são mesclados ou divididos
4. O aplicativo é implantado

Em todos esses casos, o mapeamento shards-to-worker-to -record-processor é atualizado continuamente para balancear a carga do processamento. Processadores de estilhaços que foram migrados para outras instâncias reiniciam o processamento de registros a partir do último ponto de verificação. Isso acarreta processamento de registros duplicado, conforme mostrado no exemplo abaixo. Para obter mais informações sobre balanceamento de carga, consulte Use refragmentação, escalonamento e processamento paralelo para alterar o número de fragmentos.

Exemplo: novas tentativas do consumidor resultam em registros reenviados

Neste exemplo, você tem uma aplicação que lê continuamente registros de um fluxo, agrega registros em um arquivo local e faz upload do arquivo para o Amazon S3. Para simplificar, suponha que haja apenas 1 estilhaço e 1 operador processando o estilhaço. Considere a sequência de eventos de exemplo a seguir, supondo que o último ponto de verificação ocorreu no registro de número 10.000:

1. Um operador lê o próximo lote de registros a partir do estilhaço, registros de 10.001 a 20.000.

2. O operador passa o lote de registros para o processador de registros associado.
3. O processador de registros agrega os dados, cria um arquivo do Amazon S3 e faz upload do arquivo para o Amazon S3 com êxito.
4. O operador é encerrado inesperadamente antes que ocorra um novo ponto de verificação.
5. O aplicativo, o operador e o processador de registros são reiniciados.
6. O operador agora começa a ler a partir do último ponto de verificação bem-sucedido, que neste caso é 10.001.

Desse modo, os registros de 10.001 a 20.000 são consumidos mais de uma vez.

Ser resiliente às novas tentativas do consumidor

Mesmo que os registros possam ser processados mais de uma vez, seu aplicativo pode apresentar efeitos colaterais como se os registros tivessem sido processados apenas uma vez (processamento idempotente). As soluções para esse problema variam em termos de complexidade e precisão. Se o destino dos dados finais puder tratar bem das duplicatas, recomendamos confiar no destino final para obter o processamento idempotente. Por exemplo, com o Opensearch, você pode usar uma combinação de controle de versão e exclusividade IDs para evitar o processamento duplicado.

A aplicação de exemplo da seção anterior lê continuamente os registros de um fluxo, agrega os registros em um arquivo local e faz upload do arquivo para o Amazon S3. Conforme ilustrado, os registros de 10.001 a 20.000 são consumidos mais de uma vez, resultando em vários arquivos do Amazon S3 com os mesmos dados. Uma forma de reduzir as duplicatas desse exemplo é garantir que a etapa 3 use o seguinte esquema:

1. O processador de registros usa um número fixo de registros por arquivo do Amazon S3, como 5.000.
2. O nome do arquivo usa este esquema: prefixo do Amazon S3, shard-id e `First-Sequence-Num`. Neste caso, pode ser algo como `sample-shard000001-10001`.
3. Depois de fazer upload do arquivo do Amazon S3, defina o ponto de verificação especificando `Last-Sequence-Num`. Neste caso, o ponto de verificação seria definido no registro de número 15.000.

Com esse esquema, mesmo que os registros sejam processados mais de uma vez, o arquivo do Amazon S3 resultante terá o mesmo nome e os mesmos dados. As retentativas geram apenas a gravação dos mesmos dados no mesmo arquivo mais de uma vez.

No caso de uma operação de reestilhaçamento, o número de registros deixados no estilhaço pode ser menor que o número fixo necessário. Nesse caso, o método `shutdown()` precisa descarregar o arquivo no Amazon S3 e definir o ponto de verificação no último número de sequência. O esquema acima também é compatível com as operações de reestilhaçamento.

Gerencie a inicialização, o desligamento e a aceleração

Leia aqui algumas considerações adicionais para incorporar ao projeto da aplicação do Amazon Kinesis Data Streams.

Tópicos

Inicie produtores e consumidores de dados

Por padrão, o KCL começa a ler os registros da ponta do stream, que é o registro adicionado mais recentemente. Nessa configuração, se um aplicativo de produção de dados adicionar registros ao stream antes da execução de qualquer processador de registros de recebimento, os registros não serão lidos pelos processadores de registros após a inicialização.

Para alterar o comportamento dos processadores de registros para que eles sempre leiam dados a partir do início do fluxo, defina o seguinte valor no arquivo de propriedades da aplicação do Amazon Kinesis Data Streams:

```
initialPositionInStream = TRIM_HORIZON
```

Por padrão, o Amazon Kinesis Data Streams armazena todos os dados por 24 horas. Ele também oferece suporte à retenção prolongada de até 7 dias e a retenção de longo prazo de até 365 dias. Esse período é chamado de período de retenção. Definir a posição de início como `TRIM_HORIZON` inicia o processador de registros com os dados mais antigos no stream, conforme definido pelo período de retenção. Mesmo com a definição de `TRIM_HORIZON`, se um processador de registros iniciar após decorrido um tempo maior que o período de retenção, alguns dos registros no stream não estarão mais disponíveis. Por esse motivo, você deve sempre ter aplicativos de consumo lendo o stream e usando a CloudWatch métrica `GetRecords.IteratorAgeMilliseconds` para monitorar se os aplicativos estão acompanhando os dados recebidos.

Em alguns cenários, pode ser adequado aos processadores de registros perder os primeiros registros no stream. Por exemplo, você pode executar alguns registros iniciais no stream para testar se o stream está funcionando end-to-end conforme o esperado. Depois de fazer essa verificação inicial, você inicia seus operadores e começa a colocar os dados de produção no stream.

Para obter mais informações sobre a configuração de `TRIM_HORIZON`, consulte Use iteradores de fragmentos.

Encerrar um aplicativo Amazon Kinesis Data Streams

Quando seu aplicativo Amazon Kinesis Data Streams tiver concluído a tarefa pretendida, você deverá desligá-lo EC2 encerrando as instâncias nas quais ele está sendo executado. É possível encerrar as instâncias usando o AWS Management Console ou a AWS CLI.

Depois de encerrar seu aplicativo Amazon Kinesis Data Streams, você deve excluir a tabela do Amazon DynamoDB usada para rastrear o estado do aplicativo. KCL

Limitação de leitura

A throughput de um stream é provisionada no nível do estilhaço. Cada fragmento tem um throughput de leitura de até cinco transações por segundo para leituras, até uma taxa máxima total de leitura de dados de 2 MB por segundo. Se uma aplicação (ou grupo de aplicações operando no mesmo fluxo) tentar obter dados de um fragmento a uma taxa mais rápida, o Kinesis Data Streams limitará as operações Get correspondentes.

Em uma aplicação do Amazon Kinesis Data Streams, se um processador de registros processar dados mais rapidamente do que o limite, como no caso de um failover, o controle de utilização ocorrerá. Como KCL gerencia as interações entre o aplicativo e o Kinesis Data Streams, as exceções de limitação ocorrem KCL no código e não no código do aplicativo. No entanto, como eles KCL registram essas exceções, você as vê nos registros.

Se você achar que o seu aplicativo fica limitado de forma consistente, considere aumentar o número de estilhaços do stream.

Monitore o Kinesis Data Streams

Você pode monitorar os fluxos de dados no Amazon Kinesis Data Streams usando os seguintes recursos:

- CloudWatch métricas — O Kinesis Data Streams CloudWatch envia métricas personalizadas à Amazon com monitoramento detalhado para cada stream.
- Kinesis Agent — O Kinesis Agent publica CloudWatch métricas personalizadas para ajudar a avaliar se o agente está funcionando conforme o esperado.
- APIregistro — O Kinesis Data AWS CloudTrail Streams usa API para registrar chamadas e armazenar os dados em um bucket do Amazon S3.
- Biblioteca de cliente Kinesis — A biblioteca de cliente Kinesis (KCL) fornece métricas por fragmento, trabalhador e aplicativo. KCL
- Kinesis Producer Library — A Kinesis Producer Library (KPL) fornece métricas por fragmento, trabalhador e aplicativo. KPL

Para obter mais informações sobre problemas comuns de monitoramento, perguntas e solução de problemas, consulte o seguinte:

- Which metrics should I use to monitor and troubleshoot Kinesis Data Streams issues?
- Por que o IteratorAgeMilliseconds valor do Kinesis Data Streams continua aumentando?

Monitore o serviço Amazon Kinesis Data Streams com a Amazon CloudWatch

O Amazon Kinesis Data Streams CloudWatch e o Amazon estão integrados para que você possa coletar CloudWatch, visualizar e analisar métricas para seus streams de dados do Kinesis. Por exemplo, para rastrear o uso de estilhaços, você pode monitorar as `IncomingBytes` e as métricas de `OutgoingBytes` e compará-las com o número de estilhaços no stream.

As métricas de stream e as métricas em nível de fragmento que você configura são coletadas automaticamente e enviadas a CloudWatch cada minuto. As métricas são arquivadas por duas semanas. Depois desse período, os dados serão descartados.

A tabela a seguir descreve o monitoramento básico no nível do fluxo e o monitoramento avançado no nível do fragmento em fluxos de dados do Kinesis.

Tipo	Descrição
Básico (nível de stream)	Dados de nível de stream são enviados automaticamente a cada minuto, sem custo adicional.
Avançado (nível de estilhaço)	Dados de nível de estilhaço são enviados a cada minuto por um custo adicional. Para obter esse nível de dados, você deve habilitá-lo especificamente para o stream usando a EnableEnhancedMonitoringoperação. Para obter informações sobre preços, consulte a página CloudWatch do produto Amazon.

Dimensões e métricas do Amazon Kinesis Data Streams

O Kinesis Data Streams envia CloudWatch métricas para dois níveis: o nível do stream e, opcionalmente, o nível do fragmento. As métricas no nível do stream se destinam aos casos de uso de monitoramento mais comuns em condições normais. As métricas de nível fragmentado são para tarefas de monitoramento específicas, geralmente relacionadas à solução de problemas, e são habilitadas usando a EnableEnhancedMonitoringoperação.

Para obter uma explicação das estatísticas coletadas a partir das CloudWatch métricas, consulte CloudWatch Estatísticas no Guia CloudWatch do Usuário da Amazon.

Tópicos

- Métricas básicas em nível de stream
- Métricas aprimoradas em nível de fragmento
- Dimensões das métricas do Amazon Kinesis Data Streams
- Métricas recomendadas do Amazon Kinesis Data Streams

Métricas básicas em nível de stream

O namespace `AWS/Kinesis` inclui métricas de nível do fluxo a seguir.

O Kinesis Data Streams envia essas métricas em nível de stream CloudWatch a cada minuto. Essas métricas estão sempre disponíveis.

Métrica	Descrição
`GetRecords.Bytes`	O número de bytes recuperados do fluxo do Kinesis, medido no período especificado. As estatísticas mínima, máxima e média representam os bytes em uma única operação `GetRecords` para o fluxo no período especificado. Nome da métrica do nível de estilhaço: `OutgoingBytes` Dimensões: StreamName Estatísticas válidas: mínima, máxima, média, soma, amostras Unidade: bytes
`GetRecords.IteratorAge`	Essa métrica foi substituída. Usar `GetRecords.IteratorAgeMilliseconds` .
`GetRecords.IteratorAgeMilliseconds`	A idade do último registro em todas as chamadas `GetRecords` feitas com relação a um fluxo do Kinesis, medida no período especificado. Idade é a diferença entre a hora atual e quando o último registro da chamada `GetRecords` foi gravado no stream. As estatísticas de mínimo e máximo podem ser usadas para acompanhar o progresso das aplicações de consumo do Kinesis. Um valor zero indica que os registros lidos estão em completa sincronização com o stream. Nome da métrica do nível de estilhaço: `IteratorAgeMilliseconds`

Métrica	Descrição
	Dimensões: StreamName Estatísticas válidas: mínima, máxima, média, amostras Unidade: milissegundos
`GetRecords.Latency`	O tempo gasto por operação `GetRecords` , medido ao longo do período de tempo especificado. Dimensões: StreamName Estatísticas: mínimo, máximo, média Unidade: milissegundos
`GetRecords.Records`	O número de registros recuperados do estilhaço, medido ao longo do período de tempo especificado. As estatísticas mínima, máxima e média representam os registros em uma única operação `GetRecords` para o fluxo no período especificado. Nome da métrica do nível de estilhaço: `OutgoingRecords` Dimensões: StreamName Estatísticas válidas: mínima, máxima, média, soma, amostras Unidades: contagem

Métrica	Descrição
`GetRecords.Success`	O número de operações `GetRecords` bem-sucedidas por stream, medido ao longo do período de tempo especificado. Dimensões: StreamName Estatísticas válidas: média, soma, amostras Unidades: contagem
`IncomingBytes`	O número de bytes colocados com êxito no fluxo do Kinesis no período especificado. Essa métrica inclui bytes das operações `PutRecord` e `PutRecords`. As estatísticas mínima, máxima e média representam os bytes em uma única operação put para o fluxo no período especificado. Nome da métrica do nível de estilhaço: `IncomingBytes` Dimensões: StreamName Estatísticas válidas: mínima, máxima, média, soma, amostras Unidade: bytes

Métrica	Descrição
IncomingRecords	O número de registros colocados com êxito no fluxo do Kinesis no período especificado. Essa métrica inclui registros das operações PutRecord e PutRecords . As estatísticas mínima, máxima e média representam os registros em uma única operação put para o fluxo no período especificado. Nome da métrica do nível de estilhaço: IncomingRecords Dimensões: StreamName Estatísticas válidas: mínima, máxima, média, soma, amostras Unidades: contagem
PutRecord.Bytes	O número de bytes colocados no fluxo do Kinesis usando a operação PutRecord no período especificado. Dimensões: StreamName Estatísticas válidas: mínima, máxima, média, soma, amostras Unidade: bytes
PutRecord.Latency	O tempo gasto por operação PutRecord , medido ao longo do período de tempo especificado. Dimensões: StreamName Estatísticas: mínimo, máximo, média Unidade: milissegundos

Métrica	Descrição
PutRecord.Success	O número de operações PutRecord bem-sucedidas por fluxo do Kinesis, medido no período especificado. A média reflete a porcentagem de gravações bem-suced idas em um stream. Dimensões: StreamName Estatísticas válidas: média, soma, amostras Unidades: contagem
PutRecords.Bytes	O número de bytes colocados no fluxo do Kinesis usando a operação PutRecords no período especificado. Dimensões: StreamName Estatísticas válidas: mínima, máxima, média, soma, amostras Unidade: bytes
PutRecords.Latency	O tempo gasto por operação PutRecords , medido ao longo do período de tempo especificado. Dimensões: StreamName Estatísticas: mínimo, máximo, média Unidade: milissegundos
PutRecords.Records	Essa métrica foi substituída. Usar PutRecord s.SuccessfulRecords . Dimensões: StreamName Estatísticas válidas: mínima, máxima, média, soma, amostras Unidades: contagem

Métrica	Descrição
PutRecords.Success	O número de operações PutRecords com pelo menos um registro bem-sucedido por fluxo do Kinesis, medido no período especificado. Dimensões: StreamName Estatísticas válidas: média, soma, amostras Unidades: contagem
PutRecords.TotalRecords	O número total de registros enviados em uma operação PutRecords por fluxo de dados do Kinesis, medido no período especificado. Dimensões: StreamName Estatísticas válidas: mínima, máxima, média, soma, amostras Unidades: contagem
PutRecords.SuccessfulRecords	O número de registros bem-sucedidos em uma operação PutRecords por fluxo de dados do Kinesis, medido no período especificado. Dimensões: StreamName Estatísticas válidas: mínima, máxima, média, soma, amostras Unidades: contagem

Métrica	Descrição
PutRecords.FailedRecords	O número de registros rejeitados devido a falhas internas em uma operação PutRecords por fluxo de dados do Kinesis, medido no período especificado. Falhas internas ocasionais são esperadas e a operação deve ser repetida. Dimensões: StreamName Estatísticas válidas: mínima, máxima, média, soma, amostras Unidades: contagem
PutRecords.ThrottledRecords	O número de registros rejeitados devido ao controle de utilização em uma operação PutRecords por fluxo de dados do Kinesis, medido no período especificado. Dimensões: StreamName Estatísticas válidas: mínima, máxima, média, soma, amostras Unidades: contagem

Métrica	Descrição
`ReadProvisionedThroughputExceeded`	O número de chamadas `GetRecords` limitadas para o fluxo ao longo do período especificado. A estatística mais usada para essa métrica é Média. Quando a estatística mínima tem um valor de 1, todos os registros foram limitados ao fluxo durante o período especificado. Quando a estatística máxima tem um valor de 0 (zero), nenhum registro foi limitado ao fluxo durante o período especificado. Nome da métrica do nível de estilhaço: `ReadProvisionedThroughputExceeded` Dimensões: StreamName Estatísticas válidas: mínima, máxima, média, soma, amostras Unidades: contagem
`SubscribeToShard.RateExceeded`	Essa métrica é emitida quando uma tentativa de nova assinatura apresenta falha porque já existe uma assinatura ativa com o mesmo consumidor ou se você exceder o número de chamadas por segundo permitido para essa operação. Dimensões: StreamName, ConsumerName
`SubscribeToShard.Success`	Essa métrica registra se a SubscribeToShard assinatura foi estabelecida com sucesso. A assinatura se mantém ativa por no máximo 5 minutos. Portanto, essa métrica é emitida pelo menos uma vez a cada 5 minutos. Dimensões: StreamName, ConsumerName

Métrica	Descrição
`SubscribeToShardEvent.Bytes`	O número de bytes recebidos do estilhaço, medidos no período especificado. As estatísticas mínima, máxima e média representam os bytes publicados em um único evento no período especificado. Nome da métrica do nível de estilhaço: `OutgoingBytes` Dimensões: StreamName, ConsumerName Estatísticas válidas: mínima, máxima, média, soma, amostras Unidade: bytes
`SubscribeToShardEvent.MillisBehindLatest`	O número de milissegundos em que os registros lidos estão na ponta do stream, indicando o quão atrasado o consumidor está no horário atual. Dimensões: StreamName, ConsumerName Estatísticas válidas: mínima, máxima, média, amostras Unidade: milissegundos

Métrica	Descrição
`SubscribeToShardEvent.Records`	O número de registros recebidos do estilhaço, medidos no período especificado. As estatísticas mínima, máxima e média representam os registros em um único evento no período especificado. Nome da métrica do nível de estilhaço: `OutgoingRecords` Dimensões: StreamName, ConsumerName Estatísticas válidas: mínima, máxima, média, soma, amostras Unidades: contagem
`SubscribeToShardEvent.Success`	Essa métrica é emitida sempre que um evento é publicado com êxito. Ela será emitida somente quando houver uma assinatura ativa. Dimensões: StreamName, ConsumerName Estatísticas válidas: mínima, máxima, média, soma, amostras Unidades: contagem

Métrica	Descrição
`WriteProvisionedThroughputExceeded`	O número de registros rejeitados por causa da limitação para o fluxo ao longo do período especificado. Essa métrica inclui a limitação das operações `PutRecord` e `PutRecords` . A estatística mais usada para essa métrica é Média. Quando a estatística mínima tem um valor diferente de zero, nenhum registro é limitado ao fluxo durante o período especificado. Quando a estatística máxima tem um valor de 0 (zero), nenhum registro foi limitado ao fluxo durante o período especificado. Nome da métrica do nível de estilhaço: `WriteProvisionedThroughputExceeded` Dimensões: StreamName Estatísticas válidas: mínima, máxima, média, soma, amostras Unidades: contagem

Métricas aprimoradas em nível de fragmento

O namespace `AWS/Kinesis` inclui métricas de nível do estilhaço a seguir.

O Kinesis envia as seguintes métricas em nível de fragmento a cada minuto. CloudWatch Cada dimensão métrica cria 1 CloudWatch métrica e faz aproximadamente 43.200 `PutMetricData` API chamadas por mês. Essas métricas não são permitidas por padrão. Há uma cobrança para as métricas aprimoradas emitidas pelo Kinesis. Para obter mais informações, consulte Amazon CloudWatch Pricing sob o título Amazon CloudWatch Custom Metrics. As cobranças são indicadas por estilhaço pela métrica por mês.

Métrica	Descrição
IncomingBytes	O número de bytes colocados com sucesso no estilhaço ao longo do período especificado. Essa métrica inclui bytes das operações PutRecord e PutRecords . As estatísticas mínima, máxima e média representam os bytes em uma única operação put para o estilhaço no período especificado. Nome da métrica no nível do fluxo: IncomingBytes Dimensões: StreamName, ShardId Estatísticas válidas: mínima, máxima, média, soma, amostras Unidade: bytes
IncomingRecords	O número de registros colocados com sucesso no estilhaço ao longo do período especificado. Essa métrica inclui registros das operações PutRecord e PutRecords . As estatísticas mínima, máxima e média representam os registros em uma única operação put para o estilhaço no período especificado. Nome da métrica no nível do fluxo: IncomingRecords Dimensões: StreamName, ShardId Estatísticas válidas: mínima, máxima, média, soma, amostras Unidades: contagem
IteratorAgeMilliseconds	A idade do último registro em todas as chamadas GetRecords feitas com relação a um estilhaço, medida ao longo do período de tempo especificado. Idade é a diferença entre a hora atual e quando o último registro da chamada GetRecords foi gravado no stream. As

Métrica	Descrição
	estatísticas de mínimo e máximo podem ser usadas para acompanhar o progresso das aplicações de consumo do Kinesis. Um valor de 0 (zero) indica que os registros lidos estão em completa sincronização com o fluxo. Nome da métrica no nível do fluxo: `GetRecords.IteratorAgeMilliseconds` Dimensões: StreamName, ShardId Estatísticas válidas: mínima, máxima, média, amostras Unidade: milissegundos
`OutgoingBytes`	O número de bytes recuperados do estilhaço, medido ao longo do período de tempo especificado. As estatísticas mínima, máxima e média representam os bytes retornados em uma única operação `GetRecords` ou publicados em um único evento `SubscribeToShard` para o estilhaço no período especificado. Nome da métrica no nível do fluxo: `GetRecords.Bytes` Dimensões: StreamName, ShardId Estatísticas válidas: mínima, máxima, média, soma, amostras Unidade: bytes

Métrica	Descrição
`OutgoingRecords`	O número de registros recuperados do estilhaço, medido ao longo do período de tempo especificado. As estatísticas mínima, máxima e média representam os registros retornados em uma única operação `GetRecords` ou publicados em um único evento `SubscribeToShard` para o estilhaço no período especificado. Nome da métrica no nível do fluxo: `GetRecords.Records` Dimensões: StreamName, ShardId Estatísticas válidas: mínima, máxima, média, soma, amostras Unidades: contagem

Métrica	Descrição
`ReadProvisionedThroughputExceeded`	O número de chamadas `GetRecords` limitadas para o estilhaço ao longo do período especificado. Esta contagem de exceções abrange todas as dimensões dos seguintes limites: 5 leituras por estilhaço por segundo ou 2 MB por segundo por estilhaço. A estatística mais usada para essa métrica é Média. Quando a estatística mínima tem um valor de 1, todos os registros foram limitados ao estilhaço durante o período especificado. Quando a estatística máxima tem um valor de 0 (zero), nenhum registro foi limitado ao estilhaço durante o período especificado. Nome da métrica no nível do fluxo: `ReadProvisionedThroughputExceeded` Dimensões: StreamName, ShardId Estatísticas válidas: mínima, máxima, média, soma, amostras Unidades: contagem

Métrica	Descrição
`WriteProvisionedThroughputExceeded`	O número de registros rejeitados por causa da limitação para o estilhaço ao longo do período especificado. Esta métrica inclui limitação das operações `PutRecord` e `PutRecords` e abrange todas as dimensões os seguintes limites: 1.000 registros por segundo por estilhaço ou 1 MB por segundo por estilhaço. A estatística mais usada para essa métrica é Média. Quando a estatística mínima tem um valor diferente de zero, nenhum registro é limitado ao estilhaço durante o período especificado. Quando a estatística máxima tem um valor de 0 (zero), nenhum registro foi limitado ao estilhaço durante o período especificado. Nome da métrica no nível do fluxo: `WriteProvisionedThroughputExceeded` Dimensões: StreamName, ShardId Estatísticas válidas: mínima, máxima, média, soma, amostras Unidades: contagem

Dimensões das métricas do Amazon Kinesis Data Streams

Dimensão	Descrição
`StreamName`	O nome do fluxo do Kinesis. Todas as estatísticas disponíveis são filtradas por `StreamName` .

Métricas recomendadas do Amazon Kinesis Data Streams

Várias métricas do Amazon Kinesis Data Streams podem ser particularmente interessantes para os clientes do Kinesis Data Streams. A lista a seguir oferece métricas recomendadas e suas utilizações.

Métrica	Observações sobre o uso
`GetRecords.IteratorAgeMilliseconds`	Rastreia a posição de leitura em todos os estilhaços e consumidores no stream. Se a idade de um iterador passa de 50% do período de retenção (por padrão 24 horas, configurável até 7 dias), há risco de perda de dados devido à expiração de registro. Recomendamos que você use CloudWatch alarmes na estatística Máximo para alertá-lo antes que essa perda seja um risco. Para ver um exemplo de cenário que usa essa métrica, consulte O processamento de registros de consumidores está ficando para trás.
`ReadProvisionedThroughputExceeded`	Quando o processamento do registro do lado do consumidor está ficando para trás, às vezes é difícil saber onde está o gargalo. Use essa métrica para determinar se as leituras estão sendo limitadas por terem ultrapassado os limites de throughput de leitura. A estatística mais usada para essa métrica é Média.
`WriteProvisionedThroughputExceeded`	Ela tem a mesma finalidade da métrica `ReadProvisionedThroughputExceeded` , mas para o lado do produtor (put) do stream. A estatística mais usada para essa métrica é Média.
`PutRecord.Success` , `PutRecords.Success`	Recomendamos o uso de CloudWatch alarmes na estatística média para indicar quando os registros estão falhando na transmissão. Escolha um ou ambos os tipos put, dependendo do que o produtor usa. Se estiver usando a Kinesis Producer Library (KPL), use. `PutRecords.Success`
`GetRecords.Success`	Recomendamos o uso de CloudWatch alarmes na estatística média para indicar quando os registros estão falhando na transmissão.

Acesse CloudWatch as métricas da Amazon para Kinesis Data Streams

Você pode monitorar as métricas do Kinesis Data Streams CloudWatch usando o console, a linha de comando ou o. CloudWatch API Os procedimentos a seguir mostram como acessar as métricas usando os seguintes métodos:

Para acessar métricas usando o CloudWatch console

1. Abra o CloudWatch console em https://console.aws.amazon.com/cloudwatch/.
2. Na barra de navegação, escolha uma Região.
3. No painel de navegação, selecione Métricas.
4. No painel CloudWatch Métricas por categoria, escolha Kinesis Metrics.
5. Clique na linha relevante para ver as estatísticas do MetricNameStreamNamee.

   Observação: a maioria dos nomes de estatísticas do console corresponde aos nomes de CloudWatch métricas correspondentes listados acima, exceto para taxa de transferência de leitura e taxa de transferência de gravação. Essas estatísticas são calculadas em intervalos de 5 minutos: a taxa de transferência de gravação monitora a `IncomingBytes` CloudWatch métrica e a taxa de transferência de leitura monitora. `GetRecords.Bytes`
6. (Opcional) No painel gráfico, selecione uma estatística e um período de tempo e, em seguida, crie um CloudWatch alarme usando essas configurações.

Para acessar métricas usando o AWS CLI

Use as métricas e get-metric-statisticsos comandos da lista.

Para acessar métricas usando o CloudWatch CLI

Use mon-list-metricsos mon-get-statscomandos e.

Para acessar métricas usando o CloudWatch API

Use ListMetricsas GetMetricStatisticsoperações e.

Monitore a integridade do Kinesis Data Streams Agent com a Amazon CloudWatch

O agente publica CloudWatch métricas personalizadas com um namespace de. AWS KinesisAgent Essas métricas ajudam você a avaliar se o agente está enviando dados para o Kinesis Data Streams conforme especificado e se eles estão íntegros e consumindo a quantidade adequada de recursos de memória CPU do produtor de dados. As métricas, como número de registros e bytes enviados, são úteis para compreender a taxa em que o agente está enviando dados ao stream. Quando essas métricas ficarem abaixo dos limites esperados em alguns percentuais ou caírem para zero, isso poderá indicar problemas de configuração, erros de rede ou problemas de integridade do agente. Métricas como consumo de memória CPU e no host e contadores de erros do agente indicam o uso dos recursos do produtor de dados e fornecem informações sobre possíveis erros de configuração ou do host. Por fim, o agente também registra exceções de serviço para ajudar a investigar problemas do agente. Essas métricas são reportadas na Região especificada na configuração de agente `cloudwatch.endpoint`. As métricas do Cloudwatch publicadas por vários agentes do Kinesis são agregadas ou combinadas. Para obter mais informações sobre a configuração do atendente, consulte Especifique as configurações do agente.

Monitor com CloudWatch

O agente do Kinesis Data Streams envia as seguintes métricas para o. CloudWatch

Métrica	Descrição
`BytesSent`	O número de bytes enviados para o Kinesis Data Streams no período especificado. Unidade: bytes
`RecordSendAttempts`	O número de tentativas de registro (primeira vez ou como nova tentativa) em uma chamada para `PutRecords` no período especificado. Unidades: contagem
`RecordSendErrors`	O número de registros que retornaram status de falha em uma chamada para `PutRecords`, incluindo novas tentativas, no período especificado. Unidades: contagem

Métrica	Descrição
`ServiceErrors`	O número de chamadas para `PutRecords` que resultaram em erro de serviço (diferente de um erro de controle de utilização) no período especificado. Unidades: contagem

Registre chamadas do Amazon Kinesis API Data Streams com AWS CloudTrail

O Amazon Kinesis Data Streams está AWS CloudTrail integrado com, um serviço que fornece um registro das ações realizadas por um usuário, função AWS ou serviço no Kinesis Data Streams. CloudTrail captura todas as API chamadas para o Kinesis Data Streams como eventos. As chamadas capturadas incluem chamadas do console do Kinesis Data Streams e chamadas de código para as operações do Kinesis Data API Streams. Se você criar uma trilha, poderá habilitar a entrega contínua de CloudTrail eventos para um bucket do Amazon S3, incluindo eventos para o Kinesis Data Streams. Se você não configurar uma trilha, ainda poderá ver os eventos mais recentes no CloudTrail console no Histórico de eventos. Usando as informações coletadas por CloudTrail, você pode determinar a solicitação que foi feita ao Kinesis Data Streams, o endereço IP a partir do qual a solicitação foi feita, quem fez a solicitação, quando ela foi feita e detalhes adicionais.

Para saber mais CloudTrail, inclusive como configurá-lo e ativá-lo, consulte o Guia AWS CloudTrail do usuário.

Informações do Kinesis Data Streams em CloudTrail

CloudTrail é ativado em sua AWS conta quando você cria a conta. Quando uma atividade de evento suportada ocorre no Kinesis Data Streams, essa atividade é registrada CloudTrail em um evento junto AWS com outros eventos de serviço no histórico de eventos. Você pode visualizar, pesquisar e baixar eventos recentes em sua AWS conta. Para obter mais informações, consulte Visualização de eventos com histórico de CloudTrail eventos.

Para um registro contínuo dos eventos em sua AWS conta, incluindo eventos do Kinesis Data Streams, crie uma trilha. Uma trilha permite CloudTrail entregar arquivos de log para um bucket do Amazon S3. Por padrão, quando você cria uma trilha no console, a trilha se aplica a todas as AWS regiões. A trilha registra eventos de todas as regiões na AWS partição e entrega os arquivos de

log ao bucket do Amazon S3 que você especificar. Além disso, você pode configurar outros AWS serviços para analisar e agir com base nos dados de eventos coletados nos CloudTrail registros. Para obter mais informações, consulte as informações a seguir.

- Visão Geral para Criar uma Trilha
- CloudTrail Serviços e integrações compatíveis
- Configurando as SNS notificações da Amazon para CloudTrail
- Recebendo arquivos de CloudTrail log de várias regiões e recebendo arquivos de CloudTrail log de várias contas

O Kinesis Data Streams oferece suporte ao registro das seguintes ações como CloudTrail eventos em arquivos de log:

- AddTagsToStream
- CreateStream
- DecreaseStreamRetentionPeriod
- DeleteStream
- DeregisterStreamConsumer
- DescribeStream
- DescribeStreamConsumer
- DisableEnhancedMonitoring
- EnableEnhancedMonitoring
- GetRecords
- GetShardIterator
- IncreaseStreamRetentionPeriod
- ListStreamConsumers
- ListStreams
- ListTagsForStream
- MergeShards
- PutRecord
- PutRecords
- RegisterStreamConsumer

- RemoveTagsFromStream
- SplitShard
- StartStreamEncryption
- StopStreamEncryption
- SubscribeToShard
- UpdateShardCount
- UpdateStreamMode

Cada entrada de log ou evento contém informações sobre quem gerou a solicitação. As informações de identidade ajudam a determinar:

- Se a solicitação foi feita com credenciais de usuário root ou AWS Identity and Access Management (IAM).
- Se a solicitação foi feita com credenciais de segurança temporárias de um perfil ou de um usuário federado.
- Se a solicitação foi feita por outro AWS serviço.

Para obter mais informações, consulte o CloudTrail userIdentity Elemento.

Exemplo: entradas do arquivo de log do Kinesis Data Streams

Uma trilha é uma configuração que permite a entrega de eventos como arquivos de log para um bucket do Amazon S3 que você especificar. CloudTrail os arquivos de log contêm uma ou mais entradas de log. Um evento representa uma única solicitação de qualquer fonte, e inclui informações sobre a ação solicitada, data e hora da ação, parâmetros de solicitação e assim por diante. CloudTrail os arquivos de log não são um rastreamento de pilha ordenado das API chamadas públicas, portanto, eles não aparecem em nenhuma ordem específica.

O exemplo a seguir mostra uma entrada de CloudTrail registro que demonstra as `MergeShards` ações `CreateStream` `DescribeStreamListStreams`,`DeleteStream`,`SplitShard`,, e.

```
{
    "Records": [
        {
            "eventVersion": "1.01",
            "userIdentity": {
                "type": "IAMUser",
```

```
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::012345678910:user/Alice",
                "accountId": "012345678910",
                "accessKeyId": "EXAMPLE_KEY_ID",
                "userName": "Alice"
            },
            "eventTime": "2014-04-19T00:16:31Z",
            "eventSource": "kinesis.amazonaws.com",
            "eventName": "CreateStream",
            "awsRegion": "us-east-1",
            "sourceIPAddress": "127.0.0.1",
            "userAgent": "aws-sdk-java/unknown-version Linux/x.xx",
            "requestParameters": {
                "shardCount": 1,
                "streamName": "GoodStream"
            },
            "responseElements": null,
            "requestID": "db6c59f8-c757-11e3-bc3b-57923b443c1c",
            "eventID": "b7acfcd0-6ca9-4ee1-a3d7-c4e8d420d99b"
        },
        {
            "eventVersion": "1.01",
            "userIdentity": {
                "type": "IAMUser",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::012345678910:user/Alice",
                "accountId": "012345678910",
                "accessKeyId": "EXAMPLE_KEY_ID",
                "userName": "Alice"
            },
            "eventTime": "2014-04-19T00:17:06Z",
            "eventSource": "kinesis.amazonaws.com",
            "eventName": "DescribeStream",
            "awsRegion": "us-east-1",
            "sourceIPAddress": "127.0.0.1",
            "userAgent": "aws-sdk-java/unknown-version Linux/x.xx",
            "requestParameters": {
                "streamName": "GoodStream"
            },
            "responseElements": null,
            "requestID": "f0944d86-c757-11e3-b4ae-25654b1d3136",
            "eventID": "0b2f1396-88af-4561-b16f-398f8eaea596"
        },
        {
```

```
            "eventVersion": "1.01",
            "userIdentity": {
                "type": "IAMUser",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::012345678910:user/Alice",
                "accountId": "012345678910",
                "accessKeyId": "EXAMPLE_KEY_ID",
                "userName": "Alice"
            },
            "eventTime": "2014-04-19T00:15:02Z",
            "eventSource": "kinesis.amazonaws.com",
            "eventName": "ListStreams",
            "awsRegion": "us-east-1",
            "sourceIPAddress": "127.0.0.1",
            "userAgent": "aws-sdk-java/unknown-version Linux/x.xx",
            "requestParameters": {
                "limit": 10
            },
            "responseElements": null,
            "requestID": "a68541ca-c757-11e3-901b-cbcfe5b3677a",
            "eventID": "22a5fb8f-4e61-4bee-a8ad-3b72046b4c4d"
        },
        {
            "eventVersion": "1.01",
            "userIdentity": {
                "type": "IAMUser",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::012345678910:user/Alice",
                "accountId": "012345678910",
                "accessKeyId": "EXAMPLE_KEY_ID",
                "userName": "Alice"
            },
            "eventTime": "2014-04-19T00:17:07Z",
            "eventSource": "kinesis.amazonaws.com",
            "eventName": "DeleteStream",
            "awsRegion": "us-east-1",
            "sourceIPAddress": "127.0.0.1",
            "userAgent": "aws-sdk-java/unknown-version Linux/x.xx",
            "requestParameters": {
                "streamName": "GoodStream"
            },
            "responseElements": null,
            "requestID": "f10cd97c-c757-11e3-901b-cbcfe5b3677a",
            "eventID": "607e7217-311a-4a08-a904-ec02944596dd"
```

```
        },
        {
            "eventVersion": "1.01",
            "userIdentity": {
                "type": "IAMUser",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::012345678910:user/Alice",
                "accountId": "012345678910",
                "accessKeyId": "EXAMPLE_KEY_ID",
                "userName": "Alice"
            },
            "eventTime": "2014-04-19T00:15:03Z",
            "eventSource": "kinesis.amazonaws.com",
            "eventName": "SplitShard",
            "awsRegion": "us-east-1",
            "sourceIPAddress": "127.0.0.1",
            "userAgent": "aws-sdk-java/unknown-version Linux/x.xx",
            "requestParameters": {
                "shardToSplit": "shardId-000000000000",
                "streamName": "GoodStream",
                "newStartingHashKey": "11111111"
            },
            "responseElements": null,
            "requestID": "a6e6e9cd-c757-11e3-901b-cbcfe5b3677a",
            "eventID": "dcd2126f-c8d2-4186-b32a-192dd48d7e33"
        },
        {
            "eventVersion": "1.01",
            "userIdentity": {
                "type": "IAMUser",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::012345678910:user/Alice",
                "accountId": "012345678910",
                "accessKeyId": "EXAMPLE_KEY_ID",
                "userName": "Alice"
            },
            "eventTime": "2014-04-19T00:16:56Z",
            "eventSource": "kinesis.amazonaws.com",
            "eventName": "MergeShards",
            "awsRegion": "us-east-1",
            "sourceIPAddress": "127.0.0.1",
            "userAgent": "aws-sdk-java/unknown-version Linux/x.xx",
            "requestParameters": {
                "streamName": "GoodStream",
```

```
                "adjacentShardToMerge": "shardId-000000000002",
                "shardToMerge": "shardId-000000000001"
            },
            "responseElements": null,
            "requestID": "e9f9c8eb-c757-11e3-bf1d-6948db3cd570",
            "eventID": "77cf0d06-ce90-42da-9576-71986fec411f"
        }
    ]
}
```

Monitore a biblioteca de cliente do Kinesis com a Amazon CloudWatch

A Kinesis Client Library (KCL) para Amazon Kinesis Data Streams publica métricas personalizadas da CloudWatch Amazon em seu nome, usando o nome do seu aplicativo como namespace. KCL Você pode visualizar essas métricas navegando até o CloudWatch console e escolhendo Métricas personalizadas. Para obter mais informações sobre métricas personalizadas, consulte Publicar métricas personalizadas no Guia CloudWatch do usuário da Amazon.

Há uma cobrança nominal pelas métricas enviadas CloudWatch peloKCL; especificamente, as cobranças do Amazon CloudWatch Custom Metrics e do Amazon CloudWatch API Requests se aplicam. Para obter mais informações, consulte Amazon CloudWatch Pricing.

Tópicos

- Métricas e namespace
- Níveis e dimensões métricas
- Configuração métrica
- Lista de métricas

Métricas e namespace

O namespace usado para carregar métricas é o nome do aplicativo que você especifica ao iniciar o. KCL

Níveis e dimensões métricas

Há duas opções para controlar quais métricas são enviadas para CloudWatch:

níveis de métrica

Cada métrica é atribuída a um nível individual. Quando você define um nível de relatório de métricas, métricas com um nível individual abaixo do nível de relatório não são enviadas para CloudWatch. Os níveis são: `NONE`, `SUMMARY` e `DETAILED`. A configuração padrão é`DETAILED`; ou seja, todas as métricas são enviadas para CloudWatch. Um nível de relatório `NONE` significa que nenhuma métrica é enviada. Para obter informações sobre quais níveis são atribuídos a quais métricas, consulte Lista de métricas.

dimensões habilitadas

Cada KCL métrica tem dimensões associadas que também são enviadas para CloudWatch. Na KCL versão 2.x, se KCL estiver configurada para processar um único fluxo de dados, todas as dimensões métricas (`Operation`,`ShardId`, e`WorkerIdentifier`) são ativadas por padrão. Além disso, na KCL versão 2.x, se KCL estiver configurada para processar um único fluxo de dados, a `Operation` dimensão não poderá ser desativada. Na KCL versão 2.x, se KCL estiver configurada para processar vários fluxos de dados, todas as dimensões métricas (`Operation`, `ShardIdStreamId`, e`WorkerIdentifier`) são ativadas por padrão. Além disso, no KCL 2.x, se KCL estiver configurado para processar vários fluxos de dados, as `StreamId` dimensões `Operation` e as não podem ser desativadas. `StreamId`a dimensão está disponível somente para as métricas por fragmento.

Na KCL versão 1.x, somente as `ShardId` dimensões `Operation` e as são ativadas por padrão, e a `WorkerIdentifier` dimensão está desativada. Na KCL versão 1.x, a `Operation` dimensão não pode ser desativada.

Para obter mais informações sobre dimensões CloudWatch métricas, consulte a seção Dimensões no tópico Amazon CloudWatch Concepts, no Guia CloudWatch do usuário da Amazon.

Quando a `WorkerIdentifier` dimensão é ativada, se um valor diferente for usado para a propriedade de ID do trabalhador toda vez que um determinado KCL trabalhador for reiniciado, novos conjuntos de métricas com novos valores de `WorkerIdentifier` dimensão serão enviados para CloudWatch. Se você precisar que o valor da `WorkerIdentifier` dimensão seja o mesmo em todas as reinicializações específicas do KCL trabalhador, você deverá especificar explicitamente o mesmo valor da ID do trabalhador durante a inicialização de cada trabalhador. Observe que o valor da ID do trabalhador para cada KCL trabalhador ativo deve ser exclusivo para todos os KCL trabalhadores.

Configuração métrica

Os níveis métricos e as dimensões habilitadas podem ser configurados usando a KinesisClientLibConfiguration instância, que é passada para o Worker ao iniciar o KCL aplicativo. MultiLangDaemon Nesse caso, as `metricsEnabledDimensions` propriedades `metricsLevel` e podem ser especificadas no arquivo.properties usado para iniciar o MultiLangDaemon KCL aplicativo.

Os níveis métricos podem ser atribuídos a um dos três valores: NONESUMMARY, ouDETAILED. Os valores das dimensões ativadas devem ser sequências de caracteres separadas por vírgula com a lista de dimensões permitidas para as métricas. CloudWatch As dimensões usadas pelo KCL aplicativo são `OperationShardId`, `WorkerIdentifier` e.

Lista de métricas

As tabelas a seguir listam as KCL métricas, agrupadas por escopo e operação.

Tópicos

- Métricas KCL por aplicativo
- Métricas por trabalhador
- Métricas por fragmento

Métricas KCL por aplicativo

Essas métricas são agregadas em todos os KCL trabalhadores dentro do escopo do aplicativo, conforme definido pelo CloudWatch namespace da Amazon.

Tópicos

- InitializeTask
- ShutdownTask
- ShardSyncTask
- BlockOnParentTask
- PeriodicShardSyncManager
- MultistreamTracker

InitializeTask

A `InitializeTask` operação é responsável por inicializar o processador de registros do KCL aplicativo. A lógica dessa operação inclui a obtenção de um iterador de fragmentos do Kinesis Data Streams e a inicialização do processador de registros.

Métrica	Descrição
KinesisDataFetcher.getIterator.Sucesso	Número de `GetShardIterator` operações bem-sucedidas por KCL aplicativo. Nível de métrica: detalhado Unidades: contagem
KinesisDataFetcher.getIterator.Hora	Tempo gasto por `GetShardIterator` operação para o KCL aplicativo em questão. Nível de métrica: detalhado Unidade: milissegundos
RecordProcessor.Initialize.time	Tempo percorrido pelo método de inicialização do processador de registros. Nível de métrica: resumo Unidade: milissegundos
Bem-sucedida	Número de inicializações bem-sucedidas do processador de registros. Nível de métrica: resumo Unidades: contagem
Tempo	Tempo gasto pelo KCL trabalhador para a inicialização do processador de registros. Nível de métrica: resumo Unidade: milissegundos

ShutdownTask

A operação `ShutdownTask` inicia a sequência de desligamento para o processamento de estilhaço. Isso pode ocorrer porque um estilhaço é dividido ou mesclado, ou quando a concessão do estilhaço é perdida no operador. Em ambos os casos, a função `shutdown()` do processador de registros é chamada. Novos estilhaços também são descobertos no caso em que um estilhaço é dividido ou mesclado, resultando na criação de um ou dois novos estilhaços.

Métrica	Descrição
CreateLease.Sucesso	Número de vezes que novos fragmentos secundários são adicionados com sucesso à tabela do KCL aplicativo DynamoDB após o desligamento do fragmento principal. Nível de métrica: detalhado Unidades: contagem
CreateLease.Hora	Tempo gasto para adicionar novas informações de fragmentos secundários na tabela do KCL aplicativo DynamoDB. Nível de métrica: detalhado Unidade: milissegundos
UpdateLease.Sucesso	Número de pontos de verificação finais bem-sucedidos durante o desligamento do processador de registros. Nível de métrica: detalhado Unidades: contagem
UpdateLease.Hora	Tempo necessário para a operação de pontos de verificação durante o desligamento do processador de registros. Nível de métrica: detalhado Unidade: milissegundos
RecordProcessor.Horário de desligamento	Tempo percorrido pelo método de desligamento do processador de registros.

Métrica	Descrição
	Nível de métrica: resumo Unidade: milissegundos
Bem-sucedida	Número de tarefas de desligamento bem-sucedidas. Nível de métrica: resumo Unidades: contagem
Tempo	Tempo gasto pelo KCL trabalhador para a tarefa de desligamento. Nível de métrica: resumo Unidade: milissegundos

ShardSyncTask

A `ShardSyncTask` operação descobre alterações nas informações de fragmentos do stream de dados do Kinesis, para que novos fragmentos possam ser processados pelo aplicativo. KCL

Métrica	Descrição
CreateLease.Sucesso	Número de tentativas bem-sucedidas de adicionar novas informações de fragmentos à tabela do KCL aplicativo DynamoDB. Nível de métrica: detalhado Unidades: contagem
CreateLease.Hora	Tempo gasto para adicionar novas informações de fragmentos na tabela do KCL aplicativo DynamoDB. Nível de métrica: detalhado Unidade: milissegundos
Bem-sucedida	Número de operações bem-sucedidas de sincronização de estilhaços.

Métrica	Descrição
	Nível de métrica: resumo Unidades: contagem
Tempo	Tempo percorrido para a operação de sincronização de estilhaços. Nível de métrica: resumo Unidade: milissegundos

BlockOnParentTask

Se o estilhaço é dividido ou mesclado com outros, novos estilhaços filhos são criados. A `BlockOnParentTask` operação garante que o processamento de registros dos novos fragmentos não comece até que os fragmentos principais sejam completamente processados pelo. KCL

Métrica	Descrição
Bem-sucedida	Número de verificações bem-sucedidas para a conclusão de estilhaços pai. Nível de métrica: resumo Unidades: contagem
Tempo	Tempo percorrido para a conclusão de estilhaços pai. Nível de métrica: resumo Unidade: milissegundos

PeriodicShardSyncManager

Ele `PeriodicShardSyncManager` é responsável por examinar os fluxos de dados que estão sendo processados pelo aplicativo KCL consumidor, identificando fluxos de dados com concessões parciais e entregando-os para sincronização.

As métricas a seguir estão disponíveis quando KCL configuradas para processar um único fluxo de dados (em seguida, o valor de NumStreamsToSync e NumStreamsWithPartialLeases são definidas como 1) e também quando KCL estão configuradas para processar vários fluxos de dados.

Métrica	Descrição
NumStreamsToSync	O número de fluxos de dados (por AWS conta) processados pelo aplicativo consumidor que contêm concessões parciais e que devem ser entregues para sincronização. Nível de métrica: resumo Unidades: contagem
NumStreamsWithPartialLeases	O número de fluxos de dados (por AWS conta) que o aplicativo do consumidor está processando e que contêm concessões parciais. Nível de métrica: resumo Unidades: contagem
Bem-sucedida	O número de vezes que `PeriodicShardSyncManager` conseguiu identificar com êxito as concessões parciais nos fluxos de dados que a aplicação de consumo está processando. Nível de métrica: resumo Unidades: contagem
Tempo	O tempo (em milissegundos) que `PeriodicShardSyncManager` leva para examinar os fluxos de dados sendo processados pela aplicação de consumo a fim de determinar quais dos fluxos exigem sincronização de fragmentos. Nível de métrica: resumo Unidade: milissegundos

MultistreamTracker

A `MultistreamTracker` interface permite que você crie aplicativos de KCL consumo que podem processar vários fluxos de dados ao mesmo tempo.

Métrica	Descrição
DeletedStreams.Contagem	O número de fluxos de dados excluídos no período. Nível de métrica: resumo Unidades: contagem
ActiveStreams.Contagem	O número de fluxos de dados ativos sendo processados. Nível de métrica: resumo Unidades: contagem
StreamsPendingDeletion.Contagem	O número de fluxos de dados com exclusão pendente com base em `FormerStreamsLeasesDeletionStrategy` . Nível de métrica: resumo Unidades: contagem

Métricas por trabalhador

Essas métricas são agregadas em todos os processadores de registros que consomem dados de um stream de dados do Kinesis, como uma instância da AmazonEC2.

Tópicos

- RenewAllLeases
- TakeLeases

RenewAllLeases

A operação `RenewAllLeases` renova periodicamente concessões de estilhaços de propriedade de uma determinada instância de operador.

Métrica	Descrição
RenewLease.Sucesso	Número de renovações bem-sucedidas de concessões por parte do operador. Nível de métrica: detalhado Unidades: contagem
RenewLease.Hora	O tempo necessário para a operação de renovação de concessões. Nível de métrica: detalhado Unidade: milissegundos
CurrentLeases	Número de concessões de estilhaços pertencentes ao operador depois que todas as concessões foram renovadas. Nível de métrica: resumo Unidades: contagem
LostLeases	Número de concessões de estilhaços perdidas após uma tentativa de renovar todas as concessões pertencentes ao operador. Nível de métrica: resumo Unidades: contagem
Bem-sucedida	Número de vezes que a operação de renovação da concessão foi bem-sucedida para o operador. Nível de métrica: resumo Unidades: contagem
Tempo	Tempo percorrido para renovar todas as concessões para o operador. Nível de métrica: resumo Unidade: milissegundos

TakeLeases

A `TakeLeases` operação equilibra o processamento de registros entre todos os KCL trabalhadores. Se o KCL trabalhador atual tiver menos concessões de fragmentos do que o necessário, ele receberá arrendamentos de fragmentos de outro trabalhador que esteja sobrecarregado.

Métrica	Descrição
ListLeases.Sucesso	Número de vezes que todas as concessões de fragmentos foram recuperadas com sucesso da tabela do aplicativo KCL DynamoDB. Nível de métrica: detalhado Unidades: contagem
ListLeases.Hora	Tempo gasto para recuperar todas as concessões de fragmentos da tabela do aplicativo KCL DynamoDB. Nível de métrica: detalhado Unidade: milissegundos
TakeLease.Sucesso	Número de vezes que o trabalhador obteve com sucesso contratos de arrendamento fragmentados de outros KCL trabalhadores. Nível de métrica: detalhado Unidades: contagem
TakeLease.Hora	Tempo percorrido para atualizar a tabela de concessão com concessões executadas pelo operador. Nível de métrica: detalhado Unidade: milissegundos
NumWorkers	Número total de operadores, conforme identificado por um operador específico. Nível de métrica: resumo

Métrica	Descrição
	Unidades: contagem
NeededLeases	Número de concessões de estilhaços que o operador atual precisa para uma carga de processamento de estilhaços equilibrada. Nível de métrica: detalhado Unidades: contagem
LeasesToTake	O número de concessões que o operador tentará executar. Nível de métrica: detalhado Unidades: contagem
TakenLeases	Número de concessões realizadas com sucesso pelo operador. Nível de métrica: resumo Unidades: contagem
TotalLeases	Número total de fragmentos que o KCL aplicativo está processando. Nível de métrica: detalhado Unidades: contagem
ExpiredLeases	Número total de estilhaços que não estão sendo processados por nenhum operador, conforme identificado pelo operador específico. Nível de métrica: resumo Unidades: contagem
Bem-sucedida	Número de vezes que a operação `TakeLeases` foi concluída com sucesso. Nível de métrica: resumo Unidades: contagem

Métrica	Descrição
Tempo	Tempo percorrido pela operação `TakeLeases` para um operador. Nível de métrica: resumo Unidade: milissegundos

Métricas por fragmento

Essas métricas são agregadas em um único processador de registros.

ProcessTask

A `ProcessTask` operação chama GetRecordscom a posição atual do iterador para recuperar registros do fluxo e invoca a função do processador de registros. `processRecords`

Métrica	Descrição
KinesisDataFetcher. getRecords.Sucesso	Número de operações `GetRecords` bem-sucedidas por fragmento do fluxo de dados do Kinesis. Nível de métrica: detalhado Unidades: contagem
KinesisDataFetcher. getRecords.Hora	Tempo decorrido por operação `GetRecords` para o fragmento do fluxo de dados do Kinesis. Nível de métrica: detalhado Unidade: milissegundos
UpdateLease.Sucess o	Número de pontos de verificação bem-sucedidos feitos pelo processador de registros para o determinado estilhaço. Nível de métrica: detalhado Unidades: contagem

Métrica	Descrição
UpdateLease.Hora	Tempo percorrido para cada operação de ponto de verificação para o determinado estilhaço. Nível de métrica: detalhado Unidade: milissegundos
DataBytesProcessed	Tamanho total de registros processados em bytes em cada chamada de `ProcessTask` . Nível de métrica: resumo Unidades: byte
RecordsProcessed	Número de registros processados em cada chamada de `ProcessTask` . Nível de métrica: resumo Unidades: contagem
ExpiredIterator	Número de ExpiredIteratorException recebidos ao ligar`GetRecords` . Nível de métrica: resumo Unidades: contagem
MillisBehindLatest	Tempo em que o iterador atual está atrás do registro mais recente (ponta) no estilhaço. Esse valor é menor ou igual à diferença da hora entre o registro mais recente em uma resposta e a hora atual. Esse é um reflexo mais preciso da distância em que um estilhaço está da ponta do que a comparação de time stamps no último registro de resposta. Esse valor se aplica ao último lote de registros, não a uma média de todos os time stamps em cada registro. Nível de métrica: resumo Unidade: milissegundos

Métrica	Descrição
RecordProcessor. processRecords.Hora	Tempo percorrido pelo método `processRecords` do processador de registros. Nível de métrica: resumo Unidade: milissegundos
Bem-sucedida	Número de operações bem-sucedidas de tarefas do processo. Nível de métrica: resumo Unidades: contagem
Tempo	Tempo percorrido para a operação de tarefas do processo. Nível de métrica: resumo Unidade: milissegundos

Monitore a Kinesis Producer Library com a Amazon CloudWatch

A Kinesis Producer Library (KPL) para Amazon Kinesis Data Streams publica métricas personalizadas da Amazon em seu nome. CloudWatch Você pode visualizar essas métricas navegando até o CloudWatch console e escolhendo Métricas personalizadas. Para obter mais informações sobre métricas personalizadas, consulte Publicar métricas personalizadas no Guia CloudWatch do usuário da Amazon.

Há uma cobrança nominal pelas métricas enviadas CloudWatch peloKPL; especificamente, as cobranças do Amazon CloudWatch Custom Metrics e do Amazon CloudWatch API Requests se aplicam. Para obter mais informações, consulte Amazon CloudWatch Pricing. A coleta de métricas locais não incorre em CloudWatch cobranças.

Tópicos

- Métricas, dimensões e namespaces
- Nível métrico e granularidade
- Acesso local e CloudWatch upload da Amazon

- Lista de métricas

Métricas, dimensões e namespaces

Você pode especificar um nome de aplicativo ao iniciar oKPL, que é então usado como parte do namespace ao fazer o upload das métricas. Isso é opcional; KPL fornece um valor padrão se o nome do aplicativo não estiver definido.

Você também pode configurar o KPL para adicionar dimensões adicionais arbitrárias às métricas. Isso é útil se você quiser dados mais detalhados em suas métricas. CloudWatch Por exemplo, você pode adicionar o nome de host como uma dimensão, o que permite que você identifique distribuições de carga irregulares na frota. Todas as KPL configurações são imutáveis, então você não pode alterar essas dimensões adicionais após a inicialização da KPL instância.

Nível métrico e granularidade

Há duas opções para controlar o número de métricas enviadas para CloudWatch:

nível de métrica

Este é um indicador aproximado da importância de uma métrica. Cada métrica é atribuída a um nível. Quando você define um nível, as métricas com níveis abaixo desses não são enviadas para CloudWatch. Os níveis são `NONE`, `SUMMARY` e `DETAILED`. A configuração padrão é `DETAILED`. Ou seja, todas as métricas. `NONE` significa que não há métricas, de modo que nenhuma métrica é atribuída a esse nível.

granularidade

Controla se a mesma métrica é emitida em níveis adicionais de granularidade. Os níveis são `GLOBAL`, `STREAM` e `SHARD`. A configuração padrão é `SHARD`, que contém as métricas mais granulares.

Quando `SHARD` é escolhida, as métricas são emitidas com o nome do stream e o ID do estilhaço como dimensões. Além disso, a mesma métrica também é emitida com somente a dimensão do nome do stream, e a métrica sem o nome do stream. Isso significa que, para uma métrica específica, dois fluxos com dois fragmentos cada produzirão sete CloudWatch métricas: uma para cada fragmento, uma para cada fluxo e uma no geral; todas descrevendo as mesmas estatísticas, mas em diferentes níveis de granularidade. Para ter um esclarecimento, consulte o diagrama abaixo.

Os diferentes níveis de granularidade formam uma hierarquia, e todas as métricas no sistema formam árvores, enraizadas nos nomes da métrica:

```
MetricName (GLOBAL):           Metric X                    Metric Y
                                  |                           |
                              -----------------           ------------
                              |               |           |          |
StreamName (STREAM):      Stream A        Stream B     Stream A   Stream B
                              |               |
                             --------        ---------
                             |      |        |       |
ShardID (SHARD):         Shard 0 Shard 1  Shard 0 Shard 1
```

Nem todas as métricas estão disponíveis no nível de estilhaço; algumas são de nível de stream ou globais por natureza. Elas não são produzidas no nível de estilhaço, mesmo se você tiver habilitado as métricas nesse nível (`Metric Y` no diagrama acima).

Quando você especifica uma dimensão adicional, precisa fornecer valores para `tuple:<DimensionName, DimensionValue, Granularity>`. A granularidade é usada para determinar onde a dimensão personalizada é inserida na hierarquia: `GLOBAL` significa que a dimensão adicional é inserida após o nome da métrica, `STREAM` significa que ela é inserida após o nome do stream e `SHARD` significa que ela é inserida depois do ID de estilhaço. Se várias dimensões adicionais são fornecidas por nível de granularidade, elas são inseridas na ordem determinada.

Acesso local e CloudWatch upload da Amazon

As métricas da KPL instância atual estão disponíveis localmente em tempo real; você pode consultá-las a qualquer KPL momento para obtê-las. O computa KPL localmente a soma, média, mínimo, máximo e contagem de cada métrica, como em CloudWatch.

Você pode obter estatísticas que são cumulativas desde o início do programa até o presente momento ou usar uma janela contínua ao longo dos últimos N segundos, em que N é um número inteiro entre 1 e 60.

Todas as métricas estão disponíveis para upload em CloudWatch. Isso é especialmente útil para agregar dados em vários hosts, monitoramentos e alarmes. Essa funcionalidade não está disponível localmente.

Conforme descrito anteriormente, é possível selecionar de quais métricas fazer upload com as configurações de nível de métrica e granularidade. As métricas que não são carregadas estão disponíveis localmente.

Carregar pontos de dados individualmente é insustentável, porque isso pode produzir milhões de carregamentos por segundo se o tráfego for alto. Por esse motivo, CloudWatch ele KPL agrega métricas localmente em intervalos de 1 minuto e carrega um objeto de estatísticas uma vez por minuto, por métrica ativada.

Lista de métricas

Métrica	Descrição
`UserRecordsReceived`	Contagem de quantos registros lógicos do usuário foram recebidos pelo KPL núcleo para operações de colocação. Não disponível no nível de estilhaço. Nível de métrica: detalhado Unidade: contagem
`UserRecordsPending`	Exemplo periódico de quantos registros de usuário estão pendentes atualmente. Um registro está pendente se está atualmente em buffer e esperando para ser enviado ou se foi enviado e está em trânsito para o serviço de back-end. Não disponível no nível de estilhaço. O KPL fornece um método dedicado para recuperar essa métrica em nível global para que os clientes gerenciem sua taxa de venda. Nível de métrica: detalhado Unidade: contagem
`UserRecordsPut`	Contagem de quantos registros de usuário lógicos foram colocados com êxito. O KPL não conta registros com falha para essa métrica. Isso permite que a média ofereça a taxa de sucesso, que a contagem ofereça o total

Métrica	Descrição
	de tentativas e que a diferença entre a contagem e a soma ofereça o número de falhas. Nível de métrica: resumo Unidade: contagem
`UserRecordsDataPut`	Bytes nos registros de usuário lógicos colocados com êxito. Nível de métrica: detalhado Unidade: bytes
`KinesisRecordsPut`	Contagem de registros do Kinesis Data Streams colocados com êxito (cada registro do Kinesis Data Streams pode conter vários registros de usuário). A KPL saída é zero para registros com falha. Isso permite que a média ofereça a taxa de sucesso, que a contagem ofereça o total de tentativas e que a diferença entre a contagem e a soma ofereça o número de falhas. Nível de métrica: resumo Unidade: contagem
`KinesisRecordsDataPut`	Bytes em registros do Kinesis Data Streams. Nível de métrica: detalhado Unidade: bytes

Métrica	Descrição
ErrorsByCode	Contagem de cada tipo de código de erro. Apresenta uma dimensão adicional de ErrorCode , além de dimensões normais, como StreamName e ShardId. Nem todo erro pode ser rastreado até um estilhaço. Os erros que não podem ser rastreados são apenas emitidos nos níveis globais ou de stream. Essa métrica captura informações sobre fatores como limitações, alterações no mapa de estilhaços, falhas internas, serviço indisponível, limites de tempo e assim por diante. Os erros do Kinesis API Data Streams são contados uma vez por registro do Kinesis Data Streams. Vários registros de usuário em um registro do Kinesis Data Streams não geram várias contagens. Nível de métrica: resumo Unidade: contagem
AllErrors	Isso é acionado pelos mesmos erros de Erros por código, mas não faz distinção entre tipos. Isso é útil como um monitor geral da taxa de erros sem exigir uma soma manual das contagens de todos os tipos diferentes de erros. Nível de métrica: resumo Unidade: contagem
RetriesPerRecord	Número de tentativas realizadas por registro de usuário. Zero é emitido para registros que são bem-sucedidos em uma tentativa. Os dados são emitidos no momento em que um usuário termina (quando o registro é bem-sucedido ou não pode mais ser repetido). Se o registro time-to-live for um valor grande, essa métrica poderá ser significa tivamente atrasada. Nível de métrica: detalhado Unidade: contagem

Métrica	Descrição
BufferingTime	O tempo entre a chegada KPL e saída de um registro do usuário para o back-end. Essas informações são transmitidas de volta ao usuário por registro, mas também estão disponíveis como estatística agregada. Nível de métrica: resumo Unidade: milissegundos
Request Time	O tempo necessário para executar PutRecordsRequests . Nível de métrica: detalhado Unidade: milissegundos
User Records per Kinesis Record	O número de registros de usuário lógicos agregados em um único registro do Kinesis Data Streams. Nível de métrica: detalhado Unidade: contagem
Amazon Kinesis Records per PutRecord sRequest	O número de registros do Kinesis Data Streams agregados em um único PutRecordsRequest . Não disponível no nível de estilhaço. Nível de métrica: detalhado Unidade: contagem
User Records per PutRecord sRequest	O número total de registros de usuário contidos em um PutRecord sRequest . Isso é equivalente aproximadamente ao produto das últimas duas métricas. Não disponível no nível de estilhaço. Nível de métrica: detalhado Unidade: contagem

Segurança no Amazon Kinesis Data Streams

A segurança na nuvem AWS é a maior prioridade. Como AWS cliente, você se beneficiará de uma arquitetura de data center e rede criada para atender aos requisitos das organizações mais sensíveis à segurança.

A segurança é uma responsabilidade compartilhada entre você AWS e você. O modelo de responsabilidade compartilhada descreve isto como segurança da nuvem e segurança na nuvem:

- Segurança da nuvem — AWS é responsável por proteger a infraestrutura que executa AWS os serviços na AWS nuvem. AWS também fornece serviços que você pode usar com segurança. A eficácia da nossa segurança é regularmente testada e verificada por auditores de terceiros como parte dos Programas de conformidade da AWS. Para saber mais sobre os programas de conformidade que se aplicam ao Kinesis Data Streams, consulte Serviços da AWS no escopo por programa de conformidade.
- Segurança na nuvem — Sua responsabilidade é determinada pelo AWS serviço que você usa. Você também é responsável por outros fatores, como a confidencialidade de seus dados, os requisitos da sua organização, leis e regulamentos aplicáveis.

Esta documentação ajuda você a entender como aplicar o modelo de responsabilidade compartilhada ao usar o Kinesis Data Streams. Os tópicos a seguir mostram como configurar o Kinesis Data Streams para atender aos seus objetivos de segurança e conformidade. Você também aprenderá a usar outros AWS serviços que podem ajudá-lo a monitorar e proteger seus recursos do Kinesis Data Streams.

Tópicos

Proteção de dados no Amazon Kinesis Data Streams

A criptografia do lado do servidor usando chaves AWS Key Management Service (AWS KMS) facilita o cumprimento de requisitos rígidos de gerenciamento de dados, criptografando seus dados em repouso no Amazon Kinesis Data Streams.

> **Note**
>
> Se você precisar de FIPS 140-2 módulos criptográficos validados ao acessar AWS por meio de uma interface de linha de comando ou umaAPI, use um endpoint. FIPS Para obter mais informações sobre os FIPS endpoints disponíveis, consulte Federal Information Processing Standard (FIPS) 140-2.

Tópicos

O que é criptografia do lado do servidor para o Kinesis Data Streams?

A criptografia do lado do servidor é um recurso do Amazon Kinesis Data Streams que criptografa automaticamente os dados antes que estejam em repouso AWS KMS usando uma chave mestra de cliente () especificada por você. CMK Os dados são criptografados antes de serem gravados na camada de armazenamento do fluxo do Kinesis e descriptografados depois de recuperados do armazenamento. Como resultado, os dados são criptografados em repouso no serviço Kinesis Data Streams. Isso permite que você atenda a requisitos normativos rígidos e aprimore a segurança de seus dados.

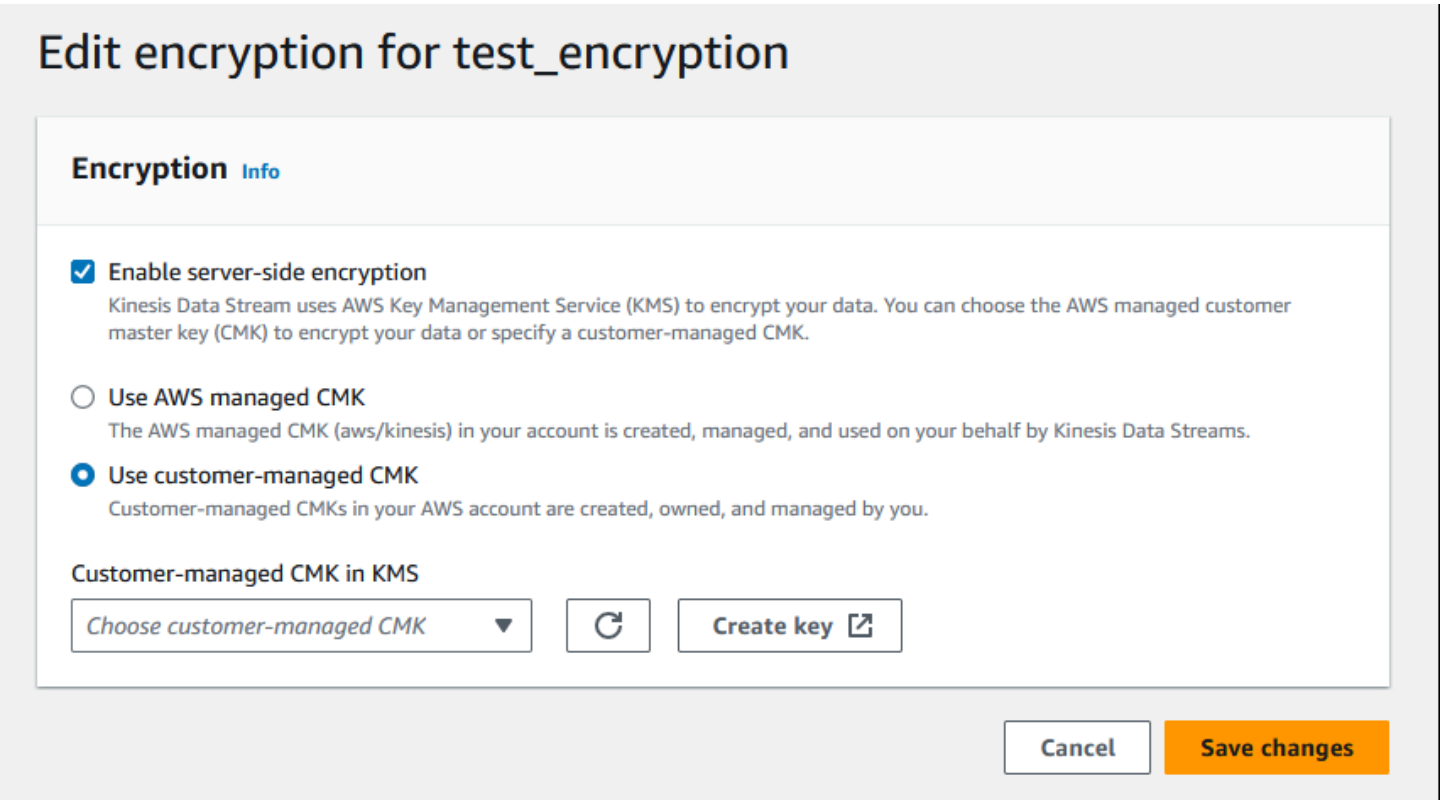
Com a criptografia no lado do servidor, seus produtores e consumidores de fluxos do Kinesis não precisam gerenciar chaves mestras ou operações de criptografia. Seus dados são criptografados

automaticamente quando entram e saem do serviço Kinesis Data Streams, então seus dados em repouso são criptografados. AWS KMS fornece todas as chaves mestras usadas pelo recurso de criptografia do lado do servidor. AWS KMS facilita o uso de um CMK for Kinesis gerenciado por AWS uma chave mestra especificada pelo usuário AWS KMS CMK ou importada para o serviço. AWS KMS

> **Note**
>
> Criptografia no lado do servidor criptografa os dados de entrada somente após a criptografia ser ativada. Os dados preexistentes em um stream não criptografado não são criptografados após a ativação da criptografia no lado do servidor.

Ao criptografar seus fluxos de dados e compartilhar o acesso a outros diretores, você deve conceder permissão na política de chaves da AWS KMS chave e nas IAM políticas da conta externa. Para obter mais informações, consulte Permitir que usuários de outras contas usem uma KMS chave.

Se você habilitou a criptografia do lado do servidor para um fluxo de dados com KMS chave AWS gerenciada e deseja compartilhar o acesso por meio de uma política de recursos, deverá passar a usar a chave gerenciada pelo cliente (CMK), conforme mostrado a seguir:



Além disso, você deve permitir que suas entidades principais de compartilhamento tenham acesso às suasCMK, usando recursos de compartilhamento KMS entre contas. Certifique-se também de fazer a alteração nas IAM políticas das entidades principais de compartilhamento. Para obter mais informações, consulte Permitir que usuários de outras contas usem uma KMS chave.

Custos, regiões e considerações de desempenho

Ao aplicar a criptografia do lado do servidor, você está sujeito ao AWS KMS API uso e aos custos principais. Ao contrário das chaves KMS mestras personalizadas, a chave mestra do `(Default) aws/kinesis` cliente (CMK) é oferecida gratuitamente. No entanto, você ainda deve pagar pelos custos de API uso incorridos pelo Amazon Kinesis Data Streams em seu nome.

APIos custos de uso se aplicam a todosCMK, inclusive os personalizados. O Kinesis Data Streams chama o AWS KMS aproximadamente a cada cinco minutos quando está mudando a chave de dados. Em um mês de 30 dias, o custo total das AWS KMS API chamadas iniciadas por um stream do Kinesis deve ser inferior a alguns dólares. Esse custo aumenta com o número de credenciais de usuário que você usa em seus produtores e consumidores de dados, pois cada credencial de usuário exige uma API chamada exclusiva para. AWS KMS Quando você usa uma IAM função para autenticação, cada chamada de função assume resulta em credenciais de usuário exclusivas. Para economizar KMS custos, talvez você queira armazenar em cache as credenciais do usuário que são retornadas pela chamada de função assume.

Veja a seguir a descrição dos custos por recurso:

Chaves

- O CMK for Kinesis gerenciado por AWS (alias =`aws/kinesis`) é gratuito.
- KMSAs chaves geradas pelo usuário estão sujeitas aos KMS principais custos. Para obter mais informações, consulte AWS Key Management Service Pricing.

APIos custos de uso se aplicam a todosCMK, inclusive os personalizados. O Kinesis Data KMS Streams liga aproximadamente a cada 5 minutos quando está girando a chave de dados. Em um mês de 30 dias, o custo total das KMS API chamadas iniciadas por um stream de dados do Kinesis deve ser inferior a alguns dólares. Observe que esse custo varia de acordo com o número de credenciais de usuário que você usa em seus produtores e consumidores de dados, pois cada credencial de usuário exige uma API chamada exclusiva para. AWS KMS Quando você usa a IAM função para autenticação, cada uma assume-role-call resultará em credenciais de usuário exclusivas

e talvez você queira armazenar em cache as credenciais do usuário retornadas pela assume-role-call para economizar KMS custos.

KMSAPIuso

Para cada stream criptografado, ao ler TIP e usar uma única chave de acesso de IAM conta/usuário entre leitores e gravadores, o serviço Kinesis liga para o AWS KMS serviço aproximadamente 12 vezes a cada 5 minutos. Não ler de TIP pode levar a um aumento no número de chamadas para o AWS KMS serviço. APIas solicitações para gerar novas chaves de criptografia de dados estão sujeitas aos custos AWS KMS de uso. Para obter mais informações, consulte AWS Key Management Service Pricing: Usage.

Disponibilidade de criptografia do lado do servidor por região

Atualmente, a criptografia do lado do servidor dos Kinesis Streams está disponível em todas as regiões compatíveis com o Kinesis Data Streams, incluindo (Oeste dos EUA) e as regiões da China. AWS GovCloud Para obter mais informações sobre regiões compatíveis com o Kinesis Data https://docs.aws.amazon.com/general/Streams, consulte latest/gr/ak.html.

Considerações sobre desempenho

Devido à sobrecarga de serviço da aplicação de criptografia, a aplicação de criptografia do lado do servidor aumenta a latência típica de `PutRecord`, `PutRecords` e `GetRecords` em menos de 100 µs.

Como faço para começar a usar a criptografia do lado do servidor?

A maneira mais fácil de começar a usar a criptografia do lado do servidor é usar a AWS Management Console e a chave de serviço do Amazon KMS Kinesis,. `aws/kinesis`

O procedimento a seguir demonstra como habilitar a criptografia no lado do servidor para um fluxo do Kinesis.

Habilitar a criptografia no lado do servidor para um fluxo do Kinesis

1. Faça login no AWS Management Console e abra o console do Amazon Kinesis Data Streams.
2. Crie ou selecione um fluxo do Kinesis no AWS Management Console.
3. Selecione a guia Detalhes.

4. Em Server-side encryption (Criptografia no lado do servidor), selecione Edit (Editar).
5. A menos que você queira usar uma chave KMS mestra gerada pelo usuário, certifique-se de que a chave mestra aws/kinesis KMS (padrão) esteja selecionada. Essa é a chave KMS mestra gerada pelo serviço Kinesis. Selecione Enabled (Habilitado) e, em seguida, selecione Save (Salvar).

   > **Note**
   >
   > A chave mestra do serviço Kinesis padrão é gratuita, no entanto, as API chamadas feitas pelo Kinesis para o AWS KMS serviço estão sujeitas aos custos de uso. KMS

6. O fluxo passa por um estado pending (pendente). Depois que o stream retorna ao estado ativo com a criptografia ativada, todos os dados recebidos gravados no stream são criptografados usando a chave KMS mestra que você selecionou.
7. Para desativar a criptografia do lado do servidor, escolha Desativado na criptografia do lado do servidor no e, em seguida, escolha Salvar AWS Management Console.

Criação e uso de chaves geradas pelo usuário KMS

Esta seção descreve como criar e usar suas próprias KMS chaves, em vez de usar a chave mestra administrada pelo Amazon Kinesis.

Criação de chaves geradas pelo usuário KMS

Para obter instruções sobre como criar suas próprias chaves, consulte Criação de chaves no Guia do AWS Key Management Service desenvolvedor. Depois de criar as chaves para sua conta, o serviço Kinesis Data Streams retorna essas chaves KMSna lista de chaves mestras.

Usando chaves geradas pelo usuário KMS

Depois que as permissões corretas forem aplicadas aos seus consumidores, produtores e administradores, você poderá usar KMS chaves personalizadas em sua própria AWS conta ou em outra AWS conta. Todas as chaves KMS mestras da sua conta aparecem na lista de chaves KMS mestras no AWS Management Console.

Para usar chaves KMS mestras personalizadas localizadas em outra conta, você precisa de permissões para usar essas chaves. Você também deve especificar ARN a chave KMS mestra na caixa ARN de entrada do AWS Management Console.

Permissões para usar chaves geradas pelo usuário KMS

Antes de usar a criptografia do lado do servidor com uma KMS chave gerada pelo usuário, você deve configurar políticas de AWS KMS chaves para permitir a criptografia de fluxos e a criptografia e descriptografia de registros de fluxo. Para obter exemplos e mais informações sobre AWS KMS permissões, consulte AWS KMSAPIPermissões: Referência de ações e recursos.

> **Note**
>
> O uso da chave de serviço padrão para criptografia não exige a aplicação de IAM permissões personalizadas.

Antes de usar chaves KMS mestras geradas pelo usuário, certifique-se de que seus produtores e consumidores de stream do Kinesis IAM (principais) sejam usuários na política de chaves KMS mestras. Caso contrário, as gravações e as leituras de um streaming falharão, o que pode resultar, em última análise, em perda de dados, processamento atrasado, ou travamento de aplicativos. Você pode gerenciar permissões para KMS chaves usando IAM políticas. Para obter mais informações, consulte Usando IAM políticas com AWS KMS.

Exemplo de permissões de produtor

Seus produtores de fluxos do Kinesis devem ter a permissão `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"
    }
  ]
}
```

Exemplo de permissões do consumidor

Seus consumidores de fluxos do Kinesis devem ter a permissão `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 e AWS Lambda use funções para consumir streams do Kinesis. Adicione a permissão `kms:Decrypt` às funções que esses consumidores usam.

Permissões de administrador de stream

Os administradores de fluxos do Kinesis precisam ter autorização para chamar `kms:List*` e `kms:DescribeKey*`.

Verificar e solucionar problemas de permissões de KMS chave

Depois de ativar a criptografia em um stream do Kinesis, recomendamos que você monitore o sucesso de suas `getRecords` chamadas`putRecord`,`putRecords`, e usando as seguintes métricas da Amazon CloudWatch :

- `PutRecord.Success`
- `PutRecords.Success`
- `GetRecords.Success`

Para ter mais informações, consulte Monitore o Kinesis Data Streams

Use o Amazon Kinesis Data Streams VPC com endpoints de interface

Você pode usar um VPC endpoint de interface para impedir que o tráfego entre o Amazon VPC e o Kinesis Data Streams saia da rede Amazon. VPCOs endpoints de interface não exigem um gateway de internet, NAT dispositivo, VPN conexão ou AWS Direct Connect conexão. Os VPC endpoints de interface são alimentados por AWS PrivateLink, uma AWS tecnologia que permite a comunicação privada entre AWS serviços usando uma interface de rede elástica com privacidade IPs em sua AmazonVPC. Para obter mais informações, consulte Amazon Virtual Private Cloud and Interface VPC Endpoints (AWS PrivateLink).

Tópicos

- Use VPC endpoints de interface para Kinesis Data Streams
- Controle o acesso aos VPC endpoints para o Kinesis Data Streams
- Disponibilidade de políticas de VPC endpoint para o Kinesis Data Streams

Use VPC endpoints de interface para Kinesis Data Streams

Para começar, você não precisa alterar as configurações para os fluxos, produtores ou consumidores. Basta criar um VPC endpoint de interface para que o tráfego do Kinesis Data Streams de e para seus recursos da VPC Amazon comece a fluir pelo endpoint da interface. VPC Para obter mais informações, consulte Criação de um endpoint de interface.

A Kinesis Producer Library (KPL) e a Kinesis Consumer Library (KCL) chamam serviços AWS como Amazon e Amazon CloudWatch DynamoDB usando endpoints públicos ou endpoints de interface

privada, qualquer que esteja em uso. VPC Por exemplo, se seu KCL aplicativo estiver sendo executado em uma interface do DynamoDB VPC VPC com endpoints habilitados, as chamadas entre o DynamoDB e KCL seu aplicativo fluirão pelo endpoint da interface. VPC

Controle o acesso aos VPC endpoints para o Kinesis Data Streams

VPCas políticas de endpoint permitem que você controle o acesso anexando uma política a um VPC endpoint ou usando campos adicionais em uma política anexada a um IAM usuário, grupo ou função para restringir o acesso a ocorrer somente por meio do endpoint especificado. VPC Essas políticas podem ser usadas para restringir o acesso a streams específicos em um VPC endpoint específico quando usadas em conjunto com IAM as políticas para conceder acesso somente às ações de stream de dados do Kinesis por meio do endpoint especificado. VPC

Veja a seguir exemplos de políticas de endpoint para acessar fluxos de dados do Kinesis.

- VPCexemplo de política: acesso somente para leitura - esse exemplo de política pode ser anexado a um VPC endpoint. (Para obter mais informações, consulte Controlando o acesso aos VPC recursos da Amazon). Ele restringe as ações a apenas listar e descrever um stream de dados do Kinesis por meio VPC do endpoint ao qual ele está conectado.

```
{
  "Statement": [
    {
      "Sid": "ReadOnly",
      "Principal": "*",
      "Action": [
        "kinesis:List*",
        "kinesis:Describe*"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

- VPCexemplo de política: restringir o acesso a um stream de dados específico do Kinesis — esse exemplo de política pode ser anexado a um VPC endpoint. Ele restringe o acesso a um fluxo de dados específico por meio do VPC endpoint ao qual está conectado.

```
{
  "Statement": [
    {
      "Sid": "AccessToSpecificDataStream",
      "Principal": "*",
      "Action": "kinesis:*",
      "Effect": "Allow",
      "Resource": "arn:aws:kinesis:us-east-1:123456789012:stream/MyStream"
    }
  ]
}
```

- IAMexemplo de política: restrinja o acesso a um Stream específico somente de um VPC endpoint específico - esse exemplo de política pode ser anexado a um IAM usuário, função ou grupo. Ela restringe o acesso a um stream de dados específico do Kinesis para que ocorra somente a partir de um VPC endpoint específico.

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

Disponibilidade de políticas de VPC endpoint para o Kinesis Data Streams

Os endpoints da interface do Kinesis Data VPC Streams com políticas são compatíveis com as seguintes regiões:

- Europa (Paris)
- Europa (Irlanda)
- Leste dos EUA (Norte da Virgínia)

- Europa (Estocolmo)
- Leste dos EUA (Ohio)
- Europa (Frankfurt)
- América do Sul (São Paulo)
- Europa (Londres)
- Ásia-Pacífico (Tóquio)
- Oeste dos EUA (N. da Califórnia)
- Ásia-Pacífico (Singapura)
- Ásia-Pacífico (Sydney)
- China (Pequim)
- China (Ningxia)
- Ásia-Pacífico (Hong Kong)
- Oriente Médio (Barém)
- Oriente Médio (UAE)
- Europa (Milão)
- África (Cidade do Cabo)
- Ásia-Pacífico (Mumbai)
- Ásia-Pacífico (Seul)
- Canadá (Central)
- Oeste dos EUA (Oregon), exceto usw2-az4
- AWS GovCloud (Leste dos EUA)
- AWS GovCloud (Oeste dos EUA)
- Ásia-Pacífico (Osaka)
- Europa (Zurique)
- Ásia-Pacífico (Hyderabad)

Controle do acesso aos recursos do Amazon Kinesis Data Streams usando IAM

AWS Identity and Access Management (IAM) permite que você faça o seguinte:

- Crie usuários e grupos em sua AWS conta
- Atribua credenciais de segurança exclusivas a cada usuário em sua conta AWS
- Controle as permissões de cada usuário para realizar tarefas usando AWS recursos
- Permita que os usuários de outra AWS conta compartilhem seus AWS recursos
- Crie funções para sua AWS conta e defina os usuários ou serviços que podem assumi-las
- Use identidades existentes para sua empresa para conceder permissões para realizar tarefas usando recursos AWS

Ao usar IAM com o Kinesis Data Streams, você pode controlar se os usuários da sua organização podem realizar uma tarefa usando ações específicas do API Kinesis Data Streams e se podem usar recursos específicos. AWS

Se você estiver desenvolvendo um aplicativo usando a Kinesis Client Library (KCL), sua política deve incluir permissões para o Amazon DynamoDB e a Amazon CloudWatch; eles KCL usam o DynamoDB para rastrear informações de estado do aplicativo e enviar métricas em seu nome. CloudWatch KCL CloudWatch Para obter mais informações sobre oKCL, consulteDesenvolva KCL consumidores 1.x.

Para obter mais informações sobreIAM, consulte o seguinte:

- AWS Identity and Access Management (IAM)
- Conceitos básicos
- IAMGuia do usuário

Para obter mais informações sobre o IAM Amazon DynamoDB, consulte Como IAM usar para controlar o acesso aos recursos do Amazon DynamoDB no Amazon DynamoDB Developer Guide.

Para obter mais informações sobre IAM a Amazon CloudWatch, consulte Controlando o acesso do usuário à sua AWS conta no Guia CloudWatch do usuário da Amazon.

Conteúdo

- Sintaxe da política
- Ações para o Kinesis Data Streams
- Nomes de recursos da Amazon (ARNs) para Kinesis Data Streams
- Exemplos de políticas para o Kinesis Data Streams

- Compartilhe seu fluxo de dados com outra conta
- Configurar uma AWS Lambda função para ler do Kinesis Data Streams em outra conta
- Compartilhe o acesso usando políticas baseadas em recursos

Sintaxe da política

Uma IAM política é um JSON documento que consiste em uma ou mais declarações. Cada instrução é estruturada da seguinte maneira:

```
{
  "Statement":[{
    "Effect":"effect",
    "Action":"action",
    "Resource":"arn",
    "Condition":{
      "condition":{
        "key":"value"
        }
      }
    }
  ]
}
```

Existem vários elementos que compõem uma instrução:

- Efeito: o efeito pode ser `Allow` ou `Deny`. Por padrão, IAM os usuários não têm permissão para usar recursos e API ações, então todas as solicitações são negadas. Um permitir explícito substitui o padrão. Uma negar explícito substitui todas as permissões.
- Ação: A ação é a API ação específica para a qual você está concedendo ou negando permissão.
- Recurso: o recurso afetado pela ação. Para especificar um recurso na declaração, você precisa usar seu Amazon Resource Name (ARN).
- Condição: condições são opcionais. Elas podem ser usadas para controlar quando as políticas entrarão em vigor.

Ao criar e gerenciar IAM políticas, talvez você queira usar o IAMPolicy Generator e o IAM Policy Simulator.

Ações para o Kinesis Data Streams

Em uma declaração IAM de política, você pode especificar qualquer API ação de qualquer serviço que ofereça suporteIAM. Para Kinesis Data Streams, use o seguinte prefixo com o nome API da ação:. `kinesis:` Por exemplo: `kinesis:CreateStream`, `kinesis:ListStreams` e `kinesis:DescribeStreamSummary`.

Para especificar várias ações em uma única instrução, separe-as com vírgulas, como segue:

```
"Action": ["kinesis:action1", "kinesis:action2"]
```

Também é possível especificar várias ações usando asteriscos. Por exemplo, você pode especificar todas as ações cujo nome começa com a palavra "Obter", conforme o seguinte:

```
"Action": "kinesis:Get*"
```

Para especificar todas as operações do Kinesis Data Streams, use o curinga *, como a seguir:

```
"Action": "kinesis:*"
```

Para ver a lista completa das ações do Kinesis API Data Streams, consulte a Referência do Amazon API Kinesis.

Nomes de recursos da Amazon (ARNs) para Kinesis Data Streams

Cada declaração IAM de política se aplica aos recursos que você especifica usando seusARNs.

Use o seguinte formato ARN de recurso para streams de dados do Kinesis:

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

Por exemplo:

```
"Resource": arn:aws:kinesis:*:111122223333:stream/my-stream
```

Exemplos de políticas para o Kinesis Data Streams

As políticas de exemplo a seguir demonstram como você pode controlar o acesso do usuário aos fluxos de dados do Kinesis.

Example 1: Allow users to get data from a stream

Example

Esta política permite que um usuário ou grupo execute as operações `DescribeStreamSummary`, `GetShardIterator` e `GetRecords` no stream especificado e `ListStreams` em qualquer stream. Esta política pode ser aplicada a usuários que devem conseguir obter dados de um stream específico.

```
{
    "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 que um usuário ou grupo use a operação `PutRecord` com qualquer um dos streams da conta. Esta política pode ser aplicada a usuários que devem conseguir adicionar registros de dados a todos os streams em uma conta.

```
{
```

```
    "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 que um usuário ou grupo use qualquer operação do Kinesis Data Streams no fluxo especificado. Esta política poderia ser aplicada a usuários que devem ter controle administrativo por meio de um stream específico.

```
{
    "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 que um usuário ou grupo use qualquer operação do Kinesis Data Streams em qualquer fluxo em uma conta. Como esta política concede acesso total a todos os streams, você deve restringi-la somente aos administradores.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": [
                "arn:aws:kinesis:*:111122223333:stream/*"
            ]
        }
    ]
}
```

Compartilhe seu fluxo de dados com outra conta

> **Note**
>
> Atualmente, a Kinesis Producer Library não suporta a especificação de um stream ARN ao gravar em um stream de dados. Use o AWS SDK se quiser gravar em um fluxo de dados entre contas.

Anexe uma política baseada em recursos ao seu fluxo de dados para conceder acesso a outra conta, IAM usuário ou IAM função. Políticas baseadas em recursos são documentos JSON de política que você anexa a um recurso, como um fluxo de dados. Essas políticas concedem permissão para a entidade principal especificada executar ações específicas nesse recurso e definem sob quais condições isso se aplica. Uma política pode ter várias declarações. Você deve especificar uma entidade principal em uma política baseada em recursos. Os diretores podem incluir contas, usuários, funções, usuários federados ou AWS serviços. Você pode configurar políticas no console do Kinesis Data Streams API ou. SDK

Observe que compartilhar o acesso a consumidores registrados, como o Enhanced Fan Out, exige uma política tanto sobre o fluxo de dados ARN quanto sobre o consumidorARN.

Habilitar o acesso entre contas

Para habilitar o acesso entre contas, você pode especificar uma conta ou IAM entidades inteiras em outra conta como principal em uma política baseada em recursos. Adicionar uma entidade principal entre contas à política baseada em recurso é apenas metade da tarefa de estabelecimento

da relação de confiança. Quando o principal e o recurso estão em AWS contas separadas, você também deve usar uma política baseada em identidade para conceder ao principal acesso ao recurso. No entanto, se uma política baseada em recurso conceder acesso a uma entidade principal na mesma conta, nenhuma política baseada em identidade adicional será necessária.

Para obter mais informações sobre o uso de políticas baseadas em recursos para acesso entre contas, consulte Acesso a recursos entre contas em. IAM

Os administradores de fluxo de dados podem usar AWS Identity and Access Management políticas para especificar quem tem acesso ao quê. Ou seja, qual entidade principal pode executar ações em quais recursos, e em que condições. O `Action` elemento de uma JSON política descreve as ações que você pode usar para permitir ou negar acesso em uma política. As ações de política geralmente têm o mesmo nome da AWS API operação associada.

Ações do Kinesis Data Streams que podem ser compartilhadas:

Ação	Nível de acesso
DescribeStreamConsumer	Consumidor
DescribeStreamSummary	Fluxo de dados
GetRecords	Fluxo de dados
GetShardIterator	Fluxo de dados
ListShards	Fluxo de dados
PutRecord	Fluxo de dados
PutRecords	Fluxo de dados
SubscribeToShard	Consumidor

Veja a seguir exemplos do uso de uma política baseada em recursos para conceder acesso entre contas ao fluxo de dados ou ao consumidor registrado.

Para realizar uma ação entre contas, você deve especificar o fluxo ARN para acesso ao fluxo de dados e o consumidor ARN para acesso de consumidor registrado.

Exemplo de políticas baseadas em recursos para fluxos de dados do Kinesis

Compartilhar um consumidor registrado envolve uma política de fluxo de dados e uma política de consumidor devido às ações necessárias.

> **Note**
>
> Veja os seguintes exemplos de valores válidos para `Principal`:
>
> - `{"AWS": "123456789012"}`
> - IAMUsuário — `{"AWS": "arn:aws:iam::123456789012:user/user-name"}`
> - IAMFunção — `{"AWS":["arn:aws:iam::123456789012:role/role-name"]}`
> - Várias entidades principais (pode ser uma combinação de contas, usuários, perfis): `{"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

Declaração de política de fluxo de dados:

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

Declaração 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"
        }
    ]
}
```

Não há suporte para o caractere curinga (*) no campo de ações ou entidade principal, a fim de manter o princípio do privilégio mínimo.

Gerencie a política do seu fluxo de dados de forma programática

Além do AWS Management Console, o Kinesis Data Streams APIS tem três para gerenciar sua política de stream de dados:

- PutResourcePolicy

- GetResourcePolicy
- DeleteResourcePolicy

Use `PutResourePolicy` para anexar ou substituir uma política de um fluxo de dados ou consumidor. Use `GetResourcePolicy` para verificar e visualizar uma política do fluxo de dados ou consumidor especificado. Use `DeleteResourcePolicy` para excluir uma política do fluxo de dados ou consumidor especificado.

Limites de políticas

As políticas de recursos do Kinesis Data Streams têm as restrições a seguir.

- Não há suporte para curingas (*) para ajudar a impedir que um amplo acesso seja concedido por meio das políticas de recursos diretamente vinculadas a um fluxo de dados ou a um consumidor registrado. Além disso, inspecione cuidadosamente as seguintes políticas para confirmar se elas não concedem amplo acesso:
  - Políticas baseadas em identidade vinculadas aos AWS diretores associados (por exemplo, funções) IAM
  - Políticas baseadas em recursos anexadas aos AWS recursos associados (por exemplo, AWS Key Management Service KMS chaves)
- AWS Os diretores de serviço não recebem suporte para diretores para evitar possíveis deputados confusos.
- Não há suporte para entidades principais federadas.
- Usuários canônicos não IDs são suportados.
- O tamanho da política não pode exceder 20 kB.

Compartilhe o acesso a dados criptografados

Se você habilitou a criptografia do lado do servidor para um fluxo de dados com KMS chave AWS gerenciada e deseja compartilhar o acesso por meio de uma política de recursos, deve passar a usar a chave gerenciada pelo cliente (). CMK Para obter mais informações, consulte O que é criptografia do lado do servidor para o Kinesis Data Streams?. Além disso, você deve permitir que suas entidades principais de compartilhamento tenham acesso às suasCMK, usando recursos de compartilhamento KMS entre contas. Certifique-se também de fazer a alteração nas IAM políticas das entidades principais de compartilhamento. Para obter mais informações, consulte Permitir que usuários de outras contas usem uma KMS chave.

Configurar uma AWS Lambda função para ler do Kinesis Data Streams em outra conta

Para ver um exemplo de como configurar uma função do Lambda para ler do Kinesis Data Streams em outra conta, consulte Compartilhe o acesso com funções entre contas AWS Lambda.

Compartilhe o acesso usando políticas baseadas em recursos

> **Note**
>
> Atualizar uma política existente baseada em recursos significa substituir a atual, portanto, certifique-se de incluir todas as informações necessárias em sua nova política.

Compartilhe o acesso com funções entre contas AWS Lambda

Operador do Lambda

1. Acesse o IAMconsole para criar uma IAM função que será usada como função de execução do Lambda para sua AWS Lambda função. Adicione a IAM política gerenciada `AWSLambdaKinesisExecutionRole` que tem as permissões de invocação necessárias do Kinesis Data Streams e do Lambda. Essa política também concede acesso a todos os possíveis recursos do Kinesis Data Streams aos quais você possa ter acesso.
2. No AWS Lambda console, crie uma AWS Lambda função para processar registros em um stream de dados do Kinesis Data Streams e, durante a configuração da função de execução, escolha a função que você criou na etapa anterior.
3. Forneça o perfil de execução ao proprietário do recurso do Kinesis Data Streams para configurar a política de recursos.
4. Conclua a configuração da função do Lambda.

Proprietário do recurso do Kinesis Data Streams

1. Obtenha o perfil de execução entre contas do Lambda que invocará a função do Lambda.
2. No console do Amazon Kinesis Data Streams, escolha o fluxo de dados. Escolha a guia Compartilhamento de fluxo de dados e clique no botão Criar política de compartilhamento para iniciar o editor visual de políticas. Para compartilhar um consumidor registrado em um fluxo de

dados, escolha o consumidor e, em seguida, escolha Criar política de compartilhamento. Você também pode escrever a JSON política diretamente.

3. Especifique o perfil de execução entre contas do Lambda como a entidade principal e as ações exatas do Kinesis Data Streams às quais você está compartilhando o acesso. Certifique-se de incluir a ação `kinesis:DescribeStream`. Para obter mais informações sobre exemplos de políticas de recursos do Kinesis Data Streams, consulte Exemplo de políticas baseadas em recursos para fluxos de dados do Kinesis.
4. Escolha Criar política ou use o PutResourcePolicypara anexar a política ao seu recurso.

Compartilhe o acesso com consumidores de várias contas KCL

- Se você estiver usando KCL 1.x, verifique se está usando KCL 1.15.0 ou superior.
- Se você estiver usando KCL 2.x, verifique se está usando KCL 2.5.3 ou superior.

KCLoperadora

1. Forneça seu IAM usuário ou IAM função que executará o KCL aplicativo para o proprietário do recurso.
2. Pergunte ao proprietário do recurso sobre o fluxo de dados ou o consumidorARN.
3. Certifique-se de especificar o fluxo fornecido ARN como parte de sua KCL configuração.
   - Para KCL 1.x: use o KinesisClientLibConfigurationconstrutor e forneça o fluxo. ARN
   - Para KCL 2.x: você pode fornecer apenas o stream ARN ou StreamTrackerpara a Biblioteca de cliente Kinesis. ConfigsBuilder Para StreamTracker, forneça o stream ARN e o Epoch de criação a partir da tabela de lease do DynamoDB que é gerada pela biblioteca. Se você quiser ler de um consumidor registrado compartilhado, como o Enhanced Fan-Out, use StreamTracker e também forneça ao consumidor. ARN

Proprietário do recurso do Kinesis Data Streams

1. Obtenha o IAM usuário ou a IAM função de várias contas que executará o KCL aplicativo.
2. No console do Amazon Kinesis Data Streams, escolha o fluxo de dados. Escolha a guia Compartilhamento de fluxo de dados e clique no botão Criar política de compartilhamento para iniciar o editor visual de políticas. Para compartilhar um consumidor registrado em um fluxo de

dados, escolha o consumidor e, em seguida, escolha Criar política de compartilhamento. Você também pode escrever a JSON política diretamente.

3. Especifique o IAM usuário ou a IAM função do KCL aplicativo entre contas como principal e as ações exatas do Kinesis Data Streams às quais você está compartilhando o acesso. Para obter mais informações sobre exemplos de políticas de recursos do Kinesis Data Streams, consulte Exemplo de políticas baseadas em recursos para fluxos de dados do Kinesis.
4. Escolha Criar política ou use o PutResourcePolicypara anexar a política ao seu recurso.

Compartilhe o acesso a dados criptografados

Se você habilitou a criptografia do lado do servidor para um fluxo de dados com KMS chave AWS gerenciada e deseja compartilhar o acesso por meio de uma política de recursos, deve passar a usar a chave gerenciada pelo cliente (). CMK Para obter mais informações, consulte O que é criptografia do lado do servidor para o Kinesis Data Streams?. Além disso, você deve permitir que suas entidades principais de compartilhamento tenham acesso às suasCMK, usando recursos de compartilhamento KMS entre contas. Certifique-se também de fazer a alteração nas IAM políticas das entidades principais de compartilhamento. Para obter mais informações, consulte Permitir que usuários de outras contas usem uma KMS chave.

Validação de conformidade para Amazon Kinesis Data Streams

Auditores terceirizados avaliam a segurança e a conformidade do Amazon Kinesis Data Streams como parte de vários programas de conformidade. AWS Isso incluiSOC, PCI RAMPHIPAA, Fed e outros.

Para obter uma lista de AWS serviços no escopo de programas de conformidade específicos, consulte AWS Serviços no escopo por programa de conformidade. Para obter informações gerais, consulte Programas de conformidade da AWS.

Você pode baixar relatórios de auditoria de terceiros usando AWS Artifact. Para obter mais informações, consulte Baixando relatórios no AWS Artifact.

Sua responsabilidade de conformidade ao usar o Kinesis Data Streams é determinada pela confidencialidade dos dados, pelos objetivos de conformidade da empresa e pelos regulamentos e leis aplicáveis. Se seu uso do Kinesis Data Streams estiver sujeito à conformidade com padrões HIPAA PCI como,, RAMP ou o AWS Fed, fornece recursos para ajudar a:

- Guias de início rápido sobre segurança e conformidade — Esses guias de implantação discutem considerações arquitetônicas e fornecem etapas para a implantação de ambientes básicos focados em segurança e conformidade em. AWS
- Documento técnico sobre arquitetura para HIPAA segurança e conformidade — Este whitepaper descreve como as empresas podem usar AWS para criar aplicativos compatíveis. HIPAA
- AWS Recursos de conformidade — esta coleção de pastas de trabalho e guias que podem ser aplicados ao seu setor e localização
- AWS Config— Esse AWS serviço que avalia o quão bem suas configurações de recursos estão em conformidade com as práticas internas, as diretrizes e os regulamentos do setor.
- AWS Security Hub— Esse AWS serviço fornece uma visão abrangente do seu estado de segurança interno, AWS que ajuda você a verificar sua conformidade com os padrões e as melhores práticas do setor de segurança.

Resiliência no Amazon Kinesis Data Streams

A infraestrutura AWS global é construída em torno de AWS regiões e zonas de disponibilidade. AWS As regiões fornecem várias zonas de disponibilidade fisicamente separadas e isoladas, conectadas a redes de baixa latência, alta taxa de transferência e alta redundância. Com as Zonas de Disponibilidade, é possível projetar e operar aplicações e bancos de dados que executem o failover automaticamente entre as Zonas de Disponibilidade sem interrupção. As zonas de disponibilidade são mais altamente disponíveis, tolerantes a falhas e escaláveis que uma ou várias infraestruturas de datacenter tradicionais.

Para obter mais informações sobre AWS regiões e zonas de disponibilidade, consulte Infraestrutura AWS global.

Além da infraestrutura AWS global, o Kinesis Data Streams oferece vários recursos para ajudar a suportar suas necessidades de resiliência e backup de dados.

Recuperação de desastres no Amazon Kinesis Data Streams

Quando você usa uma aplicação do Amazon Kinesis Data Streams para processar dados de um fluxo, podem ocorrer falhas nos seguintes níveis:

- Um processador de registros pode falhar
- Um operador pode falhar ou a instância do aplicativo que instanciou o operador pode falhar

- Uma EC2 instância que esteja hospedando uma ou mais instâncias do aplicativo pode falhar

Falha no processador de registros

O trabalhador invoca métodos do processador de registros usando tarefas Java ExecutorService. Se uma tarefa falhar, o operador manterá o controle do estilhaço que o processador de registros estava processando. O operador inicia uma nova tarefa de processador de registros para processar esse estilhaço. Para obter mais informações, consulte Limitação de leitura.

Falha no funcionário ou no aplicativo

Se um operador (ou uma instância) da aplicação do Amazon Kinesis Data Streams falhar, você deverá detectar e resolver a situação. Por exemplo, se o método `Worker.run` lançar uma exceção, você deverá identificá-la e tratá-la.

Se o próprio aplicativo falhar, você deverá detectar isso e reiniciá-lo. Quando o aplicativo é iniciado, ele instancia um novo operador, que, por sua vez, instancia novos processadores de registros aos quais são atribuídos estilhaços automaticamente para processamento. Podem ser os mesmos estilhaços que esses processadores de registros estavam processando antes da falha ou estilhaços novos para esses processadores.

Em uma situação em que o trabalhador ou o aplicativo falham, a falha não é detectada e há outras instâncias do aplicativo em execução em outras EC2 instâncias, os trabalhadores dessas outras instâncias lidam com a falha. Eles criam processadores de registro adicionais para processar os estilhaços que não estão mais sendo processados pelo operador com falha. A carga nessas outras EC2 instâncias aumenta de acordo.

O cenário descrito aqui pressupõe que, embora o trabalhador ou o aplicativo tenha falhado, a EC2 instância de hospedagem ainda está em execução e, portanto, não é reiniciada por um grupo do Auto Scaling.

Falha na EC2 instância da Amazon

Recomendamos que você execute as EC2 instâncias do seu aplicativo em um grupo de Auto Scaling. Dessa forma, se uma das EC2 instâncias falhar, o grupo Auto Scaling iniciará automaticamente uma nova instância para substituí-la. Você deve configurar as instâncias para iniciar a aplicação do Amazon Kinesis Data Streams na inicialização.

Segurança da infraestrutura no Kinesis Data Streams

Como um serviço gerenciado, o Amazon Kinesis Data Streams é protegido AWS pelos procedimentos globais de segurança de rede descritos no whitepaper Amazon Web Services: Visão geral dos processos de segurança.

Você usa API chamadas AWS publicadas para acessar o Kinesis Data Streams pela rede. Os clientes devem oferecer suporte ao Transport Layer Security (TLS) 1.2 ou posterior. Os clientes também devem oferecer suporte a pacotes de criptografia com sigilo direto perfeito (), como Ephemeral Diffie-Hellman (PFS) ou Elliptic Curve Ephemeral Diffie-Hellman (). DHE ECDHE A maioria dos sistemas modernos, como Java 7 e versões posteriores, comporta esses modos.

Além disso, as solicitações devem ser assinadas usando uma ID de chave de acesso e uma chave de acesso secreta associada a um IAM principal. Ou você pode usar o AWS Security Token Service (AWS STS) para gerar credenciais de segurança temporárias para assinar solicitações.

Melhores práticas de segurança para o Kinesis Data Streams

O Amazon Kinesis Data Streams fornece uma série de recursos de segurança a serem considerados no desenvolvimento e na implementação das suas próprias políticas de segurança. As melhores práticas a seguir são diretrizes gerais e não representam uma solução completa de segurança. Como essas práticas recomendadas podem não ser adequadas ou suficientes no seu ambiente, trate-as como considerações úteis em vez de requisitos.

Implemente o acesso de privilégio mínimo

Ao conceder permissões, você decide quem receberá quais permissões para quais recursos do Kinesis Data Streams. Você habilita ações específicas que quer permitir nesses recursos. Portanto, você deve conceder somente as permissões necessárias para executar uma tarefa. A implementação do privilégio de acesso mínimo é fundamental para reduzir o risco de segurança e o impacto que pode resultar de erros ou usuários mal-intencionados.

Use IAM funções

Aplicações de clientes e produtores precisam ter credenciais válidas para acessar fluxos de dados do Kinesis. Você não deve armazenar AWS credenciais diretamente em um aplicativo cliente ou em um bucket do Amazon S3. Essas são credenciais de longo prazo que não são automaticamente alternadas e podem ter um impacto comercial significativo se forem comprometidas.

Em vez disso, você deve usar uma IAM função para gerenciar credenciais temporárias para que seus aplicativos de produtor e cliente acessem os fluxos de dados do Kinesis. Quando você usa uma função, não precisa usar credenciais de longo prazo (como um nome de usuário e uma senha ou chaves de acesso) para acessar outros recursos.

Para obter mais informações, consulte os seguintes tópicos no Guia IAM do usuário:

- IAMFunções
- Cenários comuns para funções: usuários, aplicativos e serviços

Implemente criptografia do lado do servidor em recursos dependentes

É possível criptografar dados em repouso e dados em trânsito no Kinesis Data Streams. Para obter mais informações, consulte Proteção de dados no Amazon Kinesis Data Streams.

Use CloudTrail para monitorar API chamadas

O Kinesis Data Streams é AWS CloudTrail integrado com, um serviço que fornece um registro das ações realizadas por um usuário, função AWS ou serviço no Kinesis Data Streams.

Usando as informações coletadas por CloudTrail, você pode determinar a solicitação que foi feita ao Kinesis Data Streams, o endereço IP a partir do qual a solicitação foi feita, quem fez a solicitação, quando ela foi feita e detalhes adicionais.

Para obter mais informações, consulte the section called “Registre chamadas do Amazon Kinesis API Data Streams com AWS CloudTrail”.

Histórico do documento

A tabela a seguir descreve as mudanças importantes na documentação do Amazon Kinesis Data Streams.

Alteração	Descrição	Alterado em
Foi adicionado o suporte para compartilhar fluxos de dados entre contas.	Adição do Compartilhe seu fluxo de dados com outra conta.	22 de novembro de 2023
Adição de suporte para os modos de capacidade de fluxo de dados provisionado e sob demanda.	Adição do Escolha o modo de capacidade do fluxo de dados.	29 de novembro de 2021
Novo conteúdo de criptografia do lado do servidor	Adição do Proteção de dados no Amazon Kinesis Data Streams.	7 de julho de 2017
Novo conteúdo para CloudWatch métricas aprimoradas.	Atualizado Monitore o Kinesis Data Streams.	19 de abril de 2016
Novo conteúdo para agente aprimorado do Kinesis.	Atualizado Grave no Amazon Kinesis Data Streams usando o Kinesis Agent.	11 de abril de 2016
Novo conteúdo para o uso de agentes do Kinesis.	Adição do Grave no Amazon Kinesis Data Streams usando o Kinesis Agent.	2 de outubro de 2015

Alteração	Descrição	Alterado em
Atualize KPL o conteúdo da versão 0.10.0.	Adição do Desenvolva produtores usando a Amazon Kinesis Producer Library () KPL.	15 de julho de 2015
Tópico de atualização de KCL métricas para métricas configuráveis.	Adição do Monitore a biblioteca de cliente do Kinesis com a Amazon CloudWatch.	9 de julho de 2015
Reorganização do conteúdo.	Reorganização significativa dos tópicos do conteúdo para obter visualização em árvore mais concisa e agrupamento mais lógico.	01 de julho de 2015
Novo tópico do guia para KPL desenvolvedores.	Adição do Desenvolva produtores usando a Amazon Kinesis Producer Library () KPL.	02 de junho de 2015
Novo tópico sobre KCL métricas.	Adição do Monitore a biblioteca de cliente do Kinesis com a Amazon CloudWatch.	19 de maio de 2015
Support forKCL. NET	Adição do Desenvolva um consumidor da Kinesis Client Library em. NET.	1 de maio de 2015
Support for KCL Node.js	Adição do Desenvolva um consumidor da Kinesis Client Library em Node.js.	26 de março de 2015
Support para KCL Ruby	Links adicionados à biblioteca KCL Ruby.	12 de janeiro de 2015
Novo API PutRecords	Foram adicionadas informações sobre novos PutRecords API nothe section called “Adicione vários registros com PutRecords”.	15 de dezembro de 2014
Compatibilidade com atribuição de tags	Adição do Marque seus streams no Amazon Kinesis Data Streams.	11 de setembro de 2014

Alteração	Descrição	Alterado em
Nova CloudWatch métrica	Adição da métrica `GetRecords.IteratorAgeMilliseconds` a Dimensões e métricas do Amazon Kinesis Data Streams.	3 de setembro de 2014
Novo capítulo de monitoramento	Adicionadas Monitore o Kinesis Data Streams e Monitore o serviço Amazon Kinesis Data Streams com a Amazon CloudWatch.	30 de julho de 2014
Limite padrão de estilhaço	Atualização do Cotas e limites: o limite padrão de estilhaço foi elevado de 5 para 10.	25 de fevereiro de 2014
Limite padrão de estilhaço	Atualização do Cotas e limites: o limite padrão de estilhaço foi elevado de 2 para 5.	28 de janeiro de 2014
APIatualizações de versão	Atualizações para a versão 2013-12-02 do Kinesis Data Streams. API	12 de dezembro de 2013
Lançamento inicial	Versão inicial do Guia do desenvolvedor do Amazon Kinesis.	14 de novembro de 2013

As traduções são geradas por tradução automática. Em caso de conflito entre o conteúdo da tradução e da versão original em inglês, a versão em inglês prevalecerá.