

---

# Amazon Kinesis Data Streams

Guia do desenvolvedor



## Amazon Kinesis Data Streams: Guia do desenvolvedor

Copyright © 2020 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

## Table of Contents

O que é o Amazon Kinesis Data Streams? .....	1
O que posso fazer com o Kinesis Data Streams? .....	1
Benefícios do uso do Kinesis Data Streams .....	2
Serviços relacionados .....	2
Streams de dados .....	2
Como determinar o tamanho inicial de um Kinesis Data Stream .....	3
Criar um stream .....	3
Como atualizar um stream .....	4
Produtores .....	5
Consumidores .....	5
.....	6
Cotas .....	6
Limites do API .....	6
Aumento de cotas .....	11
Terminologia e conceitos .....	12
Arquitetura de alto nível .....	12
Terminologia do Kinesis Data Streams .....	12
Kinesis Data Stream .....	12
Registro de dados .....	13
Período de retenção .....	13
Produtor .....	13
Consumidor .....	13
Amazon Kinesis Data Streams Application .....	13
Fragmento .....	13
Chave de partição .....	14
Número de sequência .....	14
Kinesis Client Library .....	14
Nome do aplicativo .....	14
Criptografia do lado do servidor .....	14
Configuração .....	16
Cadastre-se na AWS .....	16
Fazer download de bibliotecas e ferramentas .....	16
Configurar seu ambiente de desenvolvimento .....	17
Conceitos básicos .....	18
Instalar e configurar o AWS CLI .....	18
Instalar a AWS CLI .....	18
Configurar a AWS CLI .....	19
Executar operações básicas de fluxo de dados do Kinesis usando a AWS CLI .....	19
Etapa 1 Criar um fluxo .....	20
Etapa 2. Colocar um registro .....	21
Etapa 3 Obter o registro .....	21
Etapa 4. Limpeza .....	23
Exemplos: .....	25
Tutorial: Processar dados de ações em tempo real usando a KCL e a KCL 2.x .....	25
Pré-requisitos .....	26
Etapa 1: Criar um stream de dados .....	26
Etapa 2: Criar uma política e um usuário do IAM .....	27
Etapa 3: Fazer download e criar o código .....	30
Etapa 4: Implementar o produtor .....	31
Etapa 5: Implementar o consumidor .....	34
Etapa 6: (Opcional) Estender o consumidor .....	37
Etapa 7: Finalizar .....	38
Tutorial: Processar dados de ações em tempo real usando a KPL e a KCL 1.x .....	39
Pré-requisitos .....	40

Etapa 1: Criar um stream de dados .....	41
Etapa 2: Criar uma política e um usuário do IAM .....	42
Etapa 3: Fazer download e criar o código de implementação .....	45
Etapa 4: Implementar o produtor .....	46
Etapa 5: Implementar o consumidor .....	49
Etapa 6: (Opcional) Estender o consumidor .....	52
Etapa 7: Finalizar .....	53
Tutorial: Analisar dados de ações em tempo real usando o Kinesis Data Analytics for Java Applications ..	54
Pré-requisitos .....	54
Etapa 1: Configurar uma conta .....	55
Etapa 2: Configurar aAWS CLI .....	57
Etapa 3: criar um aplicativo .....	58
Tutorial: Usar o AWS Lambda com o Amazon Kinesis Data Streams .....	72
AWS Solução de transmissão de dados .....	72
Criar e gerenciar streamings .....	73
Criar um stream .....	73
Criar o cliente do Kinesis Data Streams .....	73
Criar o stream .....	73
Listar streams .....	74
Listagem de estilhaços .....	75
API de teclado - Recomendado .....	76
cvbestream API - Preterido .....	77
Excluir um stream .....	78
Reestilhar um stream .....	78
Estratégias para reestilhamento .....	79
Dividir um estilhaço .....	79
Mesclar dois estilhaços .....	80
Após o reestilhamento .....	81
Alterar o período de retenção de dados .....	83
Marcação dos streamings .....	84
Conceitos básicos de tags .....	84
Monitoramento de custos com marcação .....	85
Restrições de tag .....	85
Atribuir tags usando o console do Kinesis Data Streams .....	85
Atribuir tags a streams usando o AWS CLI .....	86
Atribuir tags a stream usando a API do Kinesis Data Streams .....	86
Gerenciamento de streamings usando o console .....	87
Gravar em fluxos de dados .....	89
Usar o KPL .....	89
Função da KPL .....	90
Vantagens do uso da KPL .....	90
Quando não usar a KPL .....	91
Instalação do KPL .....	91
Transição da Kinesis Producer Library para os certificados do Amazon Trust Services (ATS) .....	91
Plataformas compatíveis com a KPL .....	92
Conceitos principais da KPL .....	92
Integração da KPL com código de produtor .....	94
Gravação no Kinesis data stream .....	95
Configurar a KPL .....	97
Desagregação de consumidor .....	97
Uso da KPL com o Kinesis Data Firehose .....	100
Uso da API .....	100
Adicionar dados a um stream .....	100
Uso do agente .....	105
Pré-requisitos .....	105
Download e instalação do agente .....	106
Configuração e inicialização do agente .....	106

Configurações do agente .....	107
Monitoramento de vários diretórios de arquivos e gravação em vários streams .....	109
Uso do agente para pré-processar dados .....	110
Comandos da CLI do agente .....	113
Solução de problemas .....	113
Aplicativo produtor está gravando a uma taxa menor que a esperada .....	113
Erro de permissão de chave mestra do KMS não autorizada .....	115
Tópicos avançados .....	115
Limitação de taxas e de novas tentativas .....	115
Considerações ao usar a agregação do KPL .....	116
Leitura de fluxos de dados .....	117
O uso do AWS Lambda .....	118
O uso do Kinesis Data Analytics .....	118
O uso do Kinesis Data Firehose .....	118
Utilizar a Biblioteca de Clientes da Kinesis .....	119
O que é o Kinesis Client Library? .....	119
KCL Versões disponíveis .....	120
Conceitos do KCL .....	120
Utilizar uma tabela de arrendamento para rastrear os painéis processados pela aplicação do consumidor KCL .....	121
Processar vários fluxos de dados com o mesmo KCL 2.x para aplicação do consumidor Java .....	128
Desenvolver consumidores personalizados com taxa de transferência compartilhada .....	131
Desenvolver consumidores personalizados com taxa de transferência compartilhada usando a KCL .....	131
Desenvolver consumidores personalizados com taxa de transferência compartilhada usando o AWS SDK for Java .....	157
Desenvolver consumidores personalizados com taxa de transferência dedicada (distribuição avançada) .....	161
Desenvolver consumidores de distribuição avançada com o KCL 2.x .....	163
Desenvolver consumidores de distribuição avançada com a API do Kinesis Data Streams .....	167
Gerenciar consumidores de distribuição avançada com o Console de gerenciamento da AWS .....	169
Migrar consumidores do KCL 1.x para o KCL 2.x .....	169
Migrar o processador de registros .....	170
Migrar a Fábrica do Processador de Registros .....	173
Migração do operador .....	174
Configurar o Cliente Amazon Kinesis .....	175
Remoção do tempo ocioso .....	178
Remoções de configuração de cliente .....	178
Solução de problemas de consumidores do Kinesis Data Streams .....	179
Alguns registros do Kinesis Data Streams são ignorados ao usar a biblioteca de cliente do Kinesis .....	179
Registros pertencentes ao mesmo estilo são processados por processadores de registros diferentes ao mesmo tempo .....	179
Aplicativo consumidor está lendo a uma taxa menor que a esperada .....	180
GetRecords retorna um array de registros vazios mesmo quando não há dados no stream .....	180
Iterador do estilo expira inesperadamente .....	181
Processamento de registros de consumidores ficando atrasados .....	181
Erro de permissão de chave mestra do KMS não autorizada .....	182
Tópicos avançados .....	182
Processamento de baixa latência .....	182
Usar o AWS Lambda com a Kinesis Producer Library .....	183
Reestilhecimento, escalabilidade e processamento paralelo .....	183
Tratar registros duplicados .....	184
Tratar inicialização, desligamento e limitação .....	186
Monitorar fluxos de dados .....	188
Monitoramento do serviço com o CloudWatch .....	188
Dimensões e métricas do Amazon Kinesis Data Streams .....	189

Acessar métricas do Amazon CloudWatch para o Kinesis Data Streams .....	197
Monitoramento do agente com o CloudWatch .....	198
Monitorar com o CloudWatch .....	198
Registro em log de chamadas de API do Amazon Kinesis Data Streams com o AWS CloudTrail .....	199
Kinesis Data Streams Informações em CloudTrail .....	199
Exemplo: Kinesis Data Streams Entradas dos arquivos de log .....	200
Monitoramento da KCL com o CloudWatch .....	203
Métricas e namespace .....	203
Dimensões e níveis de métricas .....	203
Configuração da métrica .....	204
Lista de métricas do .....	204
Monitoramento da KPL com o CloudWatch .....	212
Métricas, dimensões e namespaces .....	213
Granularidade e nível de métrica .....	213
Acesso local e carregamento do Amazon CloudWatch .....	214
Lista de métricas do .....	214
Segurança .....	217
Proteção de dados .....	217
O que é a criptografia no lado do servidor para o Kinesis Data Streams? .....	218
Custos, regiões e considerações sobre desempenho .....	218
Como começo a usar a criptografia no lado do servidor? .....	219
Criação e uso de chaves mestras do KMS geradas pelo usuário .....	220
Permissões para usar as chaves mestras do KMS geradas pelo usuário .....	221
Verificação e solução de problemas de permissões de chaves do KMS .....	222
Uso de VPC endpoints de interface .....	222
Controle do acesso .....	225
Sintaxe da política .....	225
Ações do Kinesis Data Streams .....	226
Nomes de recurso da Amazon (ARNs) para o Kinesis Data Streams .....	226
Exemplo de políticas para o Kinesis Data Streams .....	227
Validação de conformidade .....	228
Resiliência .....	229
Recuperação de desastres .....	229
Segurança da infraestrutura .....	230
Práticas recomendadas de segurança .....	230
Implemente o privilégio de acesso mínimo .....	231
Usar funções do IAM .....	231
Implemente a criptografia do lado do servidor em recursos dependentes .....	231
Use o CloudTrail para monitorar chamadas de API .....	231
Histórico do documento .....	232
AWS Glossary .....	234
.....	CCXXXV

# O que é o Amazon Kinesis Data Streams?

Você pode usar o Amazon Kinesis Data Streams para coletar e processar grandes [streamings](#) de registros de dados em tempo real. Você pode criar aplicativos de processamento de dados, conhecidos como aplicativos do Kinesis Data Streams. Um aplicativo típico do Kinesis Data Streams lê dados de um stream de dados como registros de dados. Esses aplicativos podem usar a Kinesis Client Library e serem executados em instâncias do Amazon EC2. Você pode enviar os registros processados para painéis, usá-los para gerar alertas, alterar dinamicamente as estratégias de definição de preços e publicidade ou enviar dados para uma variedade de outros serviços da AWS. Para obter informações sobre recursos e definição de preços do Kinesis Data Streams, consulte [Amazon Kinesis Data Streams](#).

O Kinesis Data Streams faz parte da plataforma de streaming de dados do Kinesis, junto com o [Kinesis Data Firehose](#), o [Streams de vídeo do Kinesis](#) e o [Kinesis Data Analytics](#).

Para obter mais informações sobre as soluções de Big Data da AWS, consulte [Big Data na AWS](#). Para obter mais informações sobre as soluções de streaming de dados da AWS, consulte [O que é streaming de dados?](#).

## Tópicos

- [O que posso fazer com o Kinesis Data Streams? \(p. 1\)](#)
- [Benefícios do uso do Kinesis Data Streams \(p. 2\)](#)
- [Serviços relacionados \(p. 2\)](#)
- [Criar e atualizar streams de dados \(p. 2\)](#)
- [Produtores do Kinesis Data Streams \(p. 5\)](#)
- [Consumidores do Kinesis Data Streams \(p. 5\)](#)
- [Cotas e limites do Kinesis Data Streams \(p. 6\)](#)

## O que posso fazer com o Kinesis Data Streams?

Você pode usar o Kinesis Data Streams para entrada e agregação de dados rápidas e contínuas. 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.

A seguir, 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 evita que os dados de log sejam perdidos se o servidor de front-end ou de aplicativos falhar. O Kinesis Data Streams fornece uma alimentação de dados acelerada, porque você não organiza os dados em lotes nos servidores antes de enviá-los para entrada.

### Métricas e relatórios em tempo real

Você pode usar os dados coletados no Kinesis Data Streams para simples análise 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 do site em tempo real e, em seguida, analisar o envolvimento da capacidade de uso do site usando vários aplicativos diferentes do Kinesis Data Streams sendo executados em paralelo.

#### Complexo processamento de stream

Você pode criar Directed Acyclic Graphs (DAGs – Gráficos acíclicos dirigidos) de aplicativos do Kinesis Data Streams e de streams de dados. Normalmente, isso envolve colocar dados de vários aplicativos do Kinesis Data Streams em outro streaming para processamento downstream por outro aplicativo do Kinesis Data Streams.

## 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 a agregação em tempo real de dados seguida do carregamento de dados agregados para um data warehouse ou cluster de redução de mapa.

Os dados são colocados em streams de dados do Kinesis, o que garante a durabilidade e a elasticidade. O atraso entre o momento em que um registro é colocado no streaming e o momento em que ele pode ser recuperado (atraso entre put e get) normalmente é de menos de 1 segundo. Em outras palavras, um aplicativo do Kinesis Data Streams pode começar a consumir os dados do streaming quase que imediatamente após a adição dos dados. O aspecto de serviço gerenciado do Kinesis Data Streams libera você do peso operacional de criação e execução de um pipeline de entrada de dados. Você pode criar aplicativos do tipo de redução de mapa de streaming. A elasticidade do Kinesis Data Streams permite escalar o streaming, de maneira que você nunca perca registros de dados antes que eles expirem.

Vários aplicativos do Kinesis Data Streams podem consumir dados de um streaming, de modo que várias ações, como arquivamento e processamento, podem ocorrer de maneira simultânea e independente. Por exemplo, dois aplicativos podem ler dados do mesmo stream. O primeiro aplicativo calcula executando agregados e atualiza uma tabela do Amazon DynamoDB, e o segundo aplicativo compacta e arquiva dados em um armazenamento de dados, como o Amazon Simple Storage Service (Amazon S3). A tabela DynamoDB com agregados em execução é, então, lida por um painel de relatórios atualizados.

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

## Serviços relacionados

Para obter informações sobre como usar clusters do Amazon EMR para ler e processar streams de dados do Kinesis diretamente, consulte [Kinesis Connector](#).

## Criar e atualizar streams de dados

O Amazon Kinesis Data Streams consome uma grande quantidade de dados em tempo real, armazena os dados de forma durável e os torna disponíveis para consumo. A unidade de dados armazenada pelo Kinesis Data Streams é um registro de dados. 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. Quando você cria um stream, você especifica o número de estilhaços desse stream. A capacidade total de um stream é a soma das capacidades de seus estilhaços. Você pode aumentar ou diminuir o número de estilhaços em um stream de acordo com a necessidade. No entanto, você é cobrado por estilhaço. Para obter informações sobre as capacidades e limites de um estilhaço, consulte [Limites de streams de dados do Kinesis \(p. 6\)](#).

Um [produtor \(p. 5\)](#) coloca registros de dados em estilhaços e um [consumidor \(p. 5\)](#) obtém registros de dados a partir dos estilhaços.

## Como determinar o tamanho inicial de um Kinesis Data Stream

Antes de criar um stream, você precisa determinar um tamanho inicial para ele. Depois de criar o fluxo, você pode dinamicamente escalonar a capacidade do seu estilhaço para cima ou para baixo usando o Console de gerenciamento da AWS ou a API [do UpdateShardCount](#). Você poderá fazer atualizações enquanto houver um aplicativo do Kinesis Data Streams consumindo dados do streaming.

Para determinar o tamanho inicial de um stream, você precisa dos seguintes valores de entrada:

- O tamanho médio do registro de dados gravado no stream em kilobytes (KB) arredondando para o próximo 1 KB, o tamanho dos dados (`average_data_size_in_KB`).
- O número de registros de dados gravados e lidos no stream por segundo (`records_per_second`).
- O número de aplicativos do Kinesis Data Streams que consomem dados de forma simultânea e independente do streaming, ou seja, os consumidores (`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 inicial de estilhaços (`number_of_shards`) de que o streaming precisará usando os valores de entrada na seguinte fórmula:

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

## Criar um stream

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

Para criar um stream de dados usando o console

1. Faça login no Console de gerenciamento da AWS 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 Create data stream (Criar stream de dados).
4. Na página Create Kinesis stream (Criar streaming do Kinesis), insira um nome para o streaming e o número de estilhaços necessários e, em seguida, clique em Create Kinesis stream (Criar streaming 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 a API do Kinesis Data Streams

- Para obter informações sobre a criação de um streaming usando a API do Kinesis Data Streams, consulte [Criar um stream \(p. 73\)](#)

Para criar um stream usando a AWS CLI

- Para obter informações sobre a criação de um streaming usando a AWS CLI, consulte o comando [create-stream](#).

## Como atualizar um stream

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

### Note

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

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 editar o número de estilhaços, selecione Edit (Editar) na seção Shards (Estilhaços) e insira uma nova contagem de estilhaços.
5. 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 chave do KMS para usar como a chave mestra de criptografia ou use a chave mestra padrão, aws/kinesis, gerenciada pelo Kinesis. Se você habilitar a criptografia para um streaming e usar a própria chave mestra do AWS KMS, verifique se os aplicativos de produtor e de consumidor têm acesso à chave mestra do AWS KMS 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 as chaves mestras do KMS geradas pelo usuário” \(p. 221\)](#).
6. 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.
7. 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 “Monitoramento do serviço com o CloudWatch” \(p. 188\)](#).

## Como atualizar um stream usando a API

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

- [AddTagsToStream](#)

- [DecreaseStreamRetentionPeriod](#)
- [DisableEnhancedMonitoring](#)
- [EnableEnhancedMonitoring](#)
- [IncreaseStreamRetentionPeriod](#)
- [RemoveTagsFromStream](#)
- [StartStreamEncryption](#)
- [StopStreamEncryption](#)
- [UpdateShardCount](#)

## Atualizar um streaming usando a AWS CLI

Para obter informações sobre atualização de um streaming usando a AWS CLI, consulte a [Referência da CLI do Kinesis](#).

## Produtores do Kinesis Data Streams

Um produtor envia registros de dados para streams de dados do Amazon Kinesis. Por exemplo, servidor web que envia dados de log para um Kinesis data stream é um produtor. Um [consumidor \(p. 5\)](#) processa os registros de dados obtidos de um streaming.

### Important

O Kinesis Data Streams aceita alterações feitas no período de retenção de registros de dados do stream de dados. Para obter mais informações, consulte [Alterar o período de retenção de dados \(p. 83\)](#).

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.

Para obter mais informações, consulte [Adicionar dados a um stream \(p. 100\)](#) (inclui o código Java de exemplo), as operações [PutRecords](#) e [PutRecord](#) na API do Kinesis Data Streams ou o comando [put-record](#).

## Consumidores do Kinesis Data Streams

Um consumidor, conhecido como um Amazon Kinesis Data Streams application, é um aplicativo criado para ler e processar registros de dados de streams de dados do Kinesis.

Para enviar registros de streaming diretamente para serviços, como o Amazon Simple Storage Service (Amazon S3), o Amazon Redshift, o Amazon Elasticsearch Service (Amazon ES) ou o Splunk, você pode usar um fluxo de entrega do Kinesis Data Firehose, em vez de criar um aplicativo consumidor. Para obter mais informações, consulte [Criar um fluxo de entrega do Kinesis Data Firehose](#) no Guia do

desenvolvedor do Kinesis Data Firehose. No entanto, se você precisar processar registros de dados de maneira personalizada, consulte [Ler dados do Amazon Kinesis Data Streams \(p. 117\)](#) para orientação sobre como criar um consumidor.

Ao criar um consumidor, você poderá implantá-lo em uma instância do Amazon EC2, adicionando-o a uma das Amazon Machine Images (AMIs – Imagens de máquina da Amazon). Você pode escalonar o consumidor executando-o em várias instâncias do Amazon EC2 em um grupo do Auto Scaling. Usar um grupo do Auto Scaling ajudará a iniciar automaticamente novas instâncias se houver uma falha na instância do EC2. Ele também pode escalonar de maneira elástica o número de instâncias conforme a carga do aplicativo muda ao longo do tempo. Os grupos do Auto Scaling garantem que um determinado número de instâncias do EC2 esteja sempre em execução. Para acionar eventos de escalabilidade no grupo do Auto Scaling, você pode especificar métricas como utilização de CPU e memória para aumentar ou diminuir o número de instâncias do EC2 que processam dados do stream. Para obter mais informações, consulte o [Guia do usuário do Amazon EC2 Auto Scaling](#).

## Cotas e limites do Kinesis Data Streams

O Amazon Kinesis Data Streams tem as seguintes cotas e limites de streamings e fragmentos.

- Não há uma cota máxima para o número de streamings que você pode ter em uma conta.
- A cota padrão é de 500 fragmentos por stream de dados para as seguintes regiões da AWS: Leste dos EUA (Norte da Virgínia), Oeste dos EUA (Oregon) e Europa (Irlanda). Para todas as outras regiões, a cota padrão é de 200 fragmentos por stream de dados.

Para solicitar um aumento de cota de fragmentos por stream de dados, siga o procedimento descrito em [Solicitar um aumento de cota](#).

- Um único estilhaço pode consumir até 1 MB de dados por segundo (incluindo as chaves de partição) ou 1.000 registros por segundo para gravações. Da mesma forma, se você escalar o streaming para 5.000 estilhaços, ele poderá ingerir até 5 GB por segundo ou 5 milhões de registros por segundo. Se precisar de mais capacidade de consumo, você poderá aumentar facilmente o número de estilhaços no streaming usando o Console de gerenciamento da AWS ou a API [UpdateShardCount](#).
- O tamanho máximo da carga útil de dados de um registro antes da codificação em base64 é de até 1 MB.
- [GetRecords](#) pode recuperar até 10 MB de dados por chamada de um único estilhaço 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.
- Cada estilhaço pode oferecer suporte a uma taxa total de leitura de dados máxima de até 2 MB por segundo por meio de [GetRecords](#). Se uma chamada para [GetRecords](#) retornar 10 MB, as chamadas subsequentes feitas nos próximos 5 segundos lançarão uma exceção.

## Limites do API

Assim como a maioria das APIs da AWS, as operações da API do Kinesis Data Streams são limitadas por taxa. Os limites a seguir se aplicam por conta da AWS por região. Para obter mais informações sobre as APIs do Kinesis Data Streams, consulte o [Amazon Kinesis API Reference](#).

## Limites de API do plano de controle do KDS

A seção a seguir descreve os limites das APIs do plano de controle do KDS. As APIs do plano de controle do KDS permitem que você crie e gerencie os streams de dados. Esses limites se aplicam por conta da AWS por região.

Limites de API do plano de controle

API	Limite de chamada de API	Limite do nível de streaming	Outros detalhes
AddTagsToStream	5 transações por segundo (TPS)	50 tags por stream de dados por conta por região	
CreateStream	5 TPS	Não há um cota máxima para o número de streamings que você pode ter em uma conta.	Você recebe um <code>LimitExceededException</code> ao fazer uma solicitação <code>CreateStream</code> ao tentar executar um dos seguintes procedimentos: <ul style="list-style-type: none"> <li>• Ter mais de cinco streams no estado <code>CREATING</code> em qualquer momento.</li> <li>• Criar mais fragmentos do que o autorizado para sua conta.</li> </ul>
DecreaseStreamRetentionPeriod	5 TPS	O valor mínimo do período de retenção de um stream de dados é de 24 horas.	
DeleteStream	5 TPS	N/D	
DeregisterStreamConsumer	5 TPS	N/D	
DescribeLimits	1 TPS		
DescribeStream	10 TPS	N/D	
DescribeStreamConsumer	20 TPS	N/D	
DescribeStreamSummary	20 TPS	N/D	
DisableEnhancedMonitoring	5 TPS	N/D	
EnableEnhancedMonitoring	5 TPS	N/D	
IncreaseStreamRetentionPeriod	5 TPS	O valor máximo do período de retenção de um stream é de 168 horas (7 dias).	
ListShards	100 TPS	N/D	
ListStreamConsumers	5 TPS	N/D	
ListStreams	5 TPS	N/D	
ListTagsForStream	5 TPS	N/D	
MergeShards	5 TPS	N/D	

API	Limite de chamada de API	Limite do nível de streaming	Outros detalhes
RegisterStreamConsumer	5 TPS	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 em um status CRIANDO ao mesmo tempo. Registrar um sexto consumidor enquanto há cinco em um status CRIANDO resultará em um <code>LimitExceededException</code> .	
RemoveTagsFromStream	5 TPS	N/D	
SplitShard	5 TPS	N/D	
StartStreamEncryption			É possível aplicar com êxito uma nova chave do AWS KMS para criptografia no lado do servidor 25 vezes em um período contínuo de 24 horas.
StopStreamEncryption			É possível desabilitar com êxito a criptografia no lado do servidor 25 vezes em um período contínuo de 24 horas.

API	Limite de chamada de API	Limite do nível de streaming	Outros detalhes
UpdateShardCount		<ul style="list-style-type: none"> <li>• Dimensionar mais de dez vezes por um período contínuo de 24 horas por stream</li> <li>• Expandir para mais do que o dobro da contagem de fragmentos atual de um stream</li> <li>• Reduzir abaixo da metade da contagem de fragmentos atual de um stream</li> <li>• Expandir para até mais de 10000 fragmentos em um stream</li> <li>• Reduzir um stream com mais de 10000 fragmentos, a menos que o resultado seja inferior a 10000 fragmentos</li> <li>• Expandir para além do limite de fragmentos da sua conta</li> </ul>	

## Limites de API do plano de dados do KDS

A seção a seguir descreve os limites para as APIs do plano de dados do KDS. As APIs do plano de dados do KDS permitem que você use os streams de dados para coletar e processar registros de dados em tempo real. Esses limites se aplicam por fragmento dentro dos streams de dados.

### Limites de API do plano de dados

API	Limite de chamada de API	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 houver uma taxa de transferência provisionada insuficiente no stream, as chamadas subsequentes

API	Limite de chamada de API	Limite de carga útil	Outros detalhes
			feitas no próximo segundo gerarão <code>ProvisionedThroughputExceededException</code>
<code>GetShardIterator</code>	5 TPS		Um iterador de fragmentos expira cinco minutos depois que é retornado ao solicitante. Se uma solicitação <code>GetShardIterator</code> for feita com muita frequência, você receberá um <code>ProvisionedThroughputExceededException</code>
<code>PutRecord</code>	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.	
<code>PutRecords</code>		Cada solicitação <code>PutRecords</code> 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. 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.	
<code>SubscribeToShard</code>	É possível fazer uma chamada para <code>SubscribeToShard</code> por segundo por consumidor registrado por fragmento.		Se você chamar <code>SubscribeToShard</code> novamente com o mesmo <code>ConsumerARN</code> e <code>ShardId</code> em até 5 segundos de uma chamada bem-sucedida, você receberá um <code>ResourceInUseException</code> .



## Aumento de cotas

É possível usar cotas de serviço 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. É possível 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 [O que são cotas de serviço?](#)

Para solicitar um aumento de cota de serviço, siga o procedimento descrito em [Solicitar um aumento de cota](#).

# Terminologia e conceitos do Amazon Kinesis Data Streams

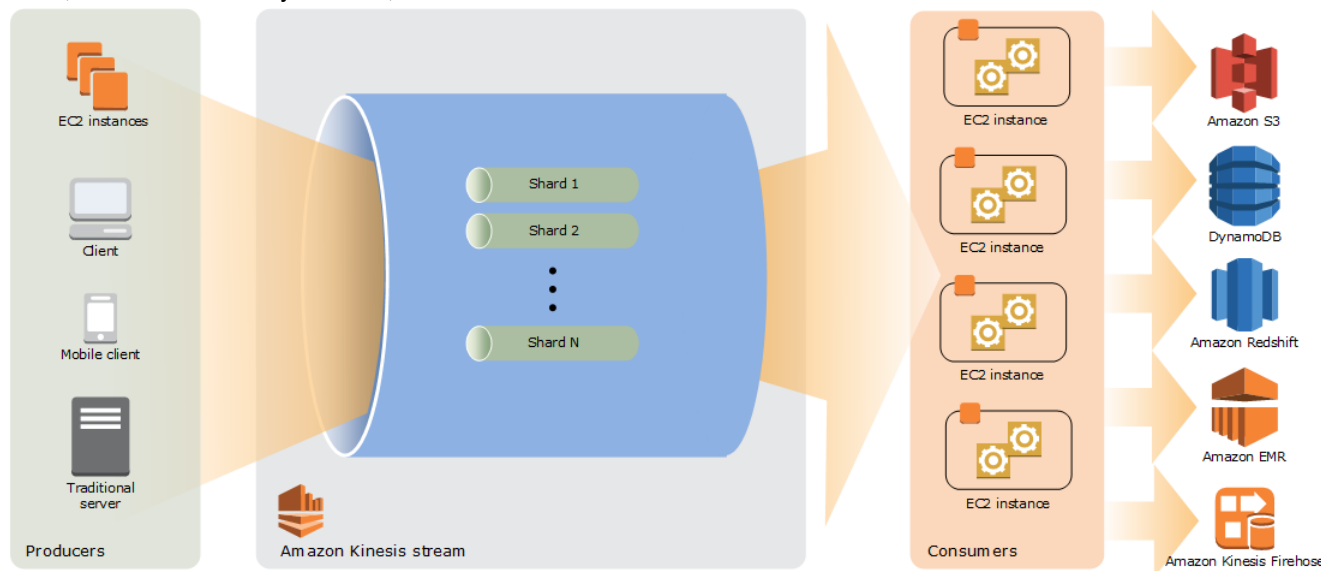
Conforme começar a usar o Amazon Kinesis Data Streams, você poderá se beneficiar da compreensão de sua arquitetura e terminologia.

## Tópicos

- [Arquitetura de alto nível do Kinesis Data Streams \(p. 12\)](#)
- [Terminologia do Kinesis Data Streams \(p. 12\)](#)

## 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, e os consumidores processam os dados em tempo real. Os consumidores (como um aplicativo personalizado em execução no Amazon EC2 ou um fluxo de entrega do Amazon Kinesis Data Firehose) pode armazenar os resultados usando um serviço da AWS, como o Amazon DynamoDB, o Amazon Redshift ou o Amazon S3.



## Terminologia do Kinesis Data Streams

### Kinesis Data Stream

Um Kinesis data stream é um conjunto de [estilhaços \(p. 13\)](#). Cada estilhaço tem uma sequência de registros de dados. Cada registro de dados tem um [número de sequência \(p. 14\)](#) atribuído pelo Kinesis Data Streams.

## Registro de dados

Um registro de dados é a unidade de dados armazenada em um [Kinesis data stream \(p. 12\)](#). Os registros de dados são compostos por um [número de sequência \(p. 14\)](#), uma chave [chave de partição \(p. 14\)](#) e um blob de dados, que é uma sequência de bytes imutável. O Kinesis Data Streams não inspeciona, interpreta ou altera os dados no blob. Um blob de dados pode ser até 1 MB.

## 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 168 horas (7 dias) usando a operação [IncreaseStreamRetentionPeriod](#) e diminuir o período de retenção até um mínimo de 24 horas usando a operação [DecreaseStreamRetentionPeriod](#). Encargos adicionais incidem sobre streams com período de retenção definido acima de 24 horas. Para obter mais informações, consulte [Amazon Kinesis Data Streams Pricing](#).

## Produtor

Os produtores inserem 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 os registros do Amazon Kinesis Data Streams e os processam. Esses consumidores são conhecidos como [Amazon Kinesis Data Streams Application \(p. 13\)](#).

## Amazon Kinesis Data Streams Application

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

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 [Ler dados do Amazon Kinesis Data Streams \(p. 117\)](#).

A saída de um aplicativo do Kinesis Data Streams pode ser inserida em outro stream, permitindo que você crie topologias complexas que processam dados em tempo real. Um aplicativo também pode enviar dados para vários outros serviços da AWS. 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 estilhaço pode oferecer suporte a até 5 transações por segundo para leituras, até uma taxa total máxima de leitura de dados de 2 MB por segundo e até 1.000 registros por segundo para gravações, até uma taxa total máxima de gravação de dados de 1 MB por segundo (inclusive 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 [Reestilhaçar um stream \(p. 78\)](#).

## Chave de partição

Uma chave de partição é usada para agrupar os dados por estilhaço dentro de um streaming. O Kinesis Data Streams segrega os registros de dados pertencentes a um streaming em vários estilhaços. 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 de hash MD5 é usada para mapear chaves de partição para valores inteiros de 128 bits e para mapear registros de dados associados para estilhaços usando os intervalos de chaves de hash dos estilhaços. 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 que é exclusivo por chave de partição em seu estilhaço. O Kinesis Data Streams atribui o número de sequência depois que você grava 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 seu aplicativo para permitir o consumo de dados tolerante a falhas do stream. A Kinesis Client Library garante que para cada estilhaço exista um processador de registros em execução e processando esse estilhaço. A biblioteca também simplifica a leitura de dados do stream. A Kinesis Client Library usa uma tabela do Amazon DynamoDB para armazenar os 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 [Ler dados do Amazon Kinesis Data Streams \(p. 117\)](#).

## Nome do aplicativo

O nome de um Amazon Kinesis Data Streams application identifica o aplicativo. Cada um dos aplicativos deve ter um nome exclusivo delimitado para a conta da AWS e a região usada pelo aplicativo. Esse nome é usado para a tabela de controle no Amazon DynamoDB, e o namespace para as métricas do 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 streaming. O Kinesis Data Streams usa as chaves mestras do [AWS KMS](#) para criptografia. Para obter mais informações, consulte [Proteção de dados no Amazon Kinesis Data Streams \(p. 217\)](#).

### 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 as chaves mestras do KMS geradas pelo usuário” \(p. 221\)](#).

**Note**

O uso de criptografia no lado do servidor incorre em custos do AWS Key Management Service (AWS KMS). Para mais informações, consulte [Definição de preço do AWS Key Management Service](#).

# Configuração do Amazon Kinesis Data Streams

Antes de usar o Amazon Kinesis Data Streams pela primeira vez, conclua as seguintes tarefas.

## Tarefas

- [Cadastre-se na AWS](#) (p. 16)
- [Fazer download de bibliotecas e ferramentas](#) (p. 16)
- [Configurar seu ambiente de desenvolvimento](#) (p. 17)

## Cadastre-se na AWS

Quando você se cadastra na Amazon Web Services (AWS), a conta da AWS é cadastrada automaticamente em todos os serviços da AWS, incluindo o Kinesis Data Streams. Você será cobrado apenas pelos serviços que usar.

Caso você já tenha uma conta da AWS, passe para a próxima tarefa. Se você ainda não possui uma conta da AWS, use o procedimento a seguir para criar uma.

Para se cadastrar em uma conta AWS

1. Abra <https://portal.aws.amazon.com/billing/signup>.
2. Siga as instruções online.

Parte do procedimento de cadastro envolve uma chamada telefônica e a digitação de um código de verificação usando o teclado do telefone.

## Fazer download de bibliotecas e ferramentas

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

- O [Amazon Kinesis API Reference](#) é o conjunto básico de operações com suporte do Kinesis Data Streams. Para obter mais informações sobre a execução de operações básicas usando código Java, consulte o seguinte:
  - [Desenvolvimento de produtores usando a API do Amazon Kinesis Data Streams com o AWS SDK for Java](#) (p. 100)
  - [Desenvolver consumidores personalizados com taxa de transferência compartilhada usando o AWS SDK for Java](#) (p. 157)
  - [Criar e gerenciar streamings](#) (p. 73)
- Os SDKs da AWS para [Go](#), [Java](#), [JavaScript](#), [.NET](#), [Node.js](#), [PHP](#), [Python](#), e [Ruby](#) incluem suporte ao Kinesis Data Streams e exemplos. Se sua versão do AWS SDK for Java não incluir exemplos do Kinesis Data Streams, você poderá fazer download deles no [GitHub](#).
- A Kinesis Client Library (KCL) fornece um modelo de programação fácil de usar para o processamento de dados. A KCL pode ajudar você a começar a usar rapidamente o Kinesis Data Streams em Java, Node.js, .NET, Python e Ruby. Para obter mais informações, consulte [Ler dados de streamings](#) (p. 117).

- O [AWS Command Line Interface](#) oferece suporte a Kinesis Data Streams. A AWS CLI permite que você controle vários serviços da AWS na linha de comando e os automatize por meio de scripts.

## Configurar seu ambiente de desenvolvimento

Para usar a KCL, certifique-se de que seu ambiente de desenvolvimento Java atende aos seguintes requisitos:

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

Observe que o [AWS SDK for Java](#) inclui o Apache Commons e o Jackson na pasta de terceiros. No entanto, o SDK para Java funciona com o Java 1.6, enquanto a Kinesis Client Library exige o Java 1.7.

# Conceitos básicos do Amazon Kinesis Data Streams

As informações desta seção ajudam você a começar a usar o Amazon Kinesis Data Streams. Se você for novo no Kinesis Data Streams, comece familiarizando-se com os conceitos e a terminologia apresentados em [Terminologia e conceitos do Amazon Kinesis Data Streams](#) (p. 12).

Esta seção mostra como executar operações básicas do Amazon Kinesis Data Streams usando a 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 Kinesis data stream.

## Tópicos

- [Instalar e configurar o AWS CLI](#) (p. 18)
- [Executar operações básicas de fluxo de dados do Kinesis usando a AWS CLI](#) (p. 19)

For CLI access, you need an access key ID and secret access key. Use IAM user access keys instead of Usuário raiz da conta da AWS access keys. IAM lets you securely control access to AWS services and resources in your AWS account. For more information about creating access keys, see [Understanding and Getting Your Security Credentials](#) in the AWS General Reference.

Você pode encontrar instruções detalhadas para a configuração do IAM e da chave de segurança em [Criar um usuário do IAM](#).

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 uma das [regiões onde o Kinesis Data Streams é compatível](#).

## Instalar e configurar o AWS CLI

### Instalar a AWS CLI

Para obter etapas detalhadas sobre como instalar a AWS CLI para Windows e para sistemas operacionais Linux, OS X e Unix, consulte [Instalação da CLI da AWS](#).

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

```
aws help
```

Você usará o serviço do Kinesis Data Streams para poder analisar os subcomandos da AWS CLI 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 à API do Kinesis Data Streams documentada na [Referência da API do serviço Amazon Kinesis](#). Por exemplo, o comando `create-stream` corresponde à ação `CreateStream` da API.

A AWS CLI agora está instalada, mas não configurada. Isso é mostrado na próxima seção.

## Configurar a AWS CLI

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

# Executar operações básicas de fluxo de dados do Kinesis usando a AWS CLI

Esta seção descreve o uso básico de um Kinesis data stream 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](#) (p. 12).

### Note

Depois de criar um streaming, sua conta incorre em cobranças nominais pelo uso do Kinesis Data Streams, pois o Kinesis Data Streams não está qualificado para o nível gratuito da AWS. Quando você concluir este tutorial, exclua seus recursos da AWS para parar de gerar cobranças. Para obter mais informações, consulte [Etapa 4. Limpeza](#) (p. 23).

### Tópicos

- [Etapa 1 Criar um fluxo](#) (p. 20)
- [Etapa 2. Colocar um registo](#) (p. 21)

- [Etapa 3 Obter o registro \(p. 21\)](#)
- [Etapa 4. Limpeza \(p. 23\)](#)

## Etapa 1 Criar um fluxo

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 --shard-count 1
```

O parâmetro `--shard-count` é obrigatório e, para esta parte do tutorial, você usa um estilhaço no seu stream. Em seguida, emita o comando a seguir para verificar o andamento da criação do stream:

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

Você deve obter uma saída semelhante ao exemplo a seguir:

```
{
  "StreamDescription": {
    "StreamStatus": "CREATING",
    "StreamName": "Foo",
    "StreamARN": "arn:aws:kinesis:us-west-2:account-id:stream/Foo",
    "Shards": []
  }
}
```

Neste exemplo, o stream tem o status `CREATING`, o que significa que não está pronto para uso. Verifique novamente em alguns instantes para ver uma saída semelhante ao exemplo a seguir:

```
{
  "StreamDescription": {
    "StreamStatus": "ACTIVE",
    "StreamName": "Foo",
    "StreamARN": "arn:aws:kinesis:us-west-2:account-id:stream/Foo",
    "Shards": [
      {
        "ShardId": "shardId-000000000000",
        "HashKeyRange": {
          "EndingHashKey": "170141183460469231731687303715884105727",
          "StartingHashKey": "0"
        },
        "SequenceNumberRange": {
          "StartingSequenceNumber":
            "49546986683135544286507457935754639466300920667981217794"
        }
      }
    ]
  }
}
```

Há informações nessa saída com que você não precisa se preocupar neste tutorial. O principal agora é `"StreamStatus": "ACTIVE"`, que mostra que o stream está pronto para uso, e as informações sobre o único estilhaço que você solicitou. Você também pode verificar a existência do novo stream usando o comando `list-streams`, como mostrado aqui:

```
aws kinesis list-streams
```

## Resultado

```
{
  "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 precisa obter o iterador referente ao estilhaço do seu interesse. 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 comando `get-shard-iterator` 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 comandos `aws kinesis` têm o suporte de uma API do Kinesis Data Streams, portanto, caso você tenha curiosidade sobre algum parâmetro mostrado, leia sobre ele no tópico de referência da API [GetShardIterator](#). A execução bem-sucedida gerará uma saída semelhante ao exemplo a seguir (role no sentido horizontal para ver a saída inteira):

```
{
  "ShardIterator": "AAAAAAAAAAHSywljv0zEgPX4NyKdZ5wryMzP9yALs8NeKbUjplIxtZs1Sp
+KEd9I6AJ9ZG4lNR1EMi+9Md/nHvtLyxpfhEzYvktZ4D9DQVz/mBYWRO6OTZRKnW9gd+efGN2aHFdkH1rJl4BL9Wyrk
+ghYG22D2T1Da2EyNSH1+LAbK33gQweTJADBdyMwlo5r6PqcP2dzhg="
}
```

A string longa de caracteres aparentemente aleatórios é o iterador de estilhaços (a sua será diferente). Você precisará copiar/colar o iterador de estilhaços 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 precisará remover quaisquer novas linhas do seu iterador de estilhaços antes de colá-lo no próximo comando. Se você receber uma mensagem de erro de que o iterador de estilhaços não é mais válido, basta executar o comando `get-shard-iterator` novamente.

getrecods

O comando `get-records` recebe dados do stream, resultando em uma chamada a [GetRecords](#) na API do Kinesis Data Streams. 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. Poderá demorar várias chamadas para se chegar a uma parte do estilhaço que contenha registros.

No exemplo a seguir do comando `get-records` (role horizontalmente para ver o comando inteiro):

```
aws kinesis get-records --shard-iterator
AAAAAAAAAHSywljv0zEgPX4NyKdZ5wryMzP9yALs8NeKbUjplIxtZs1Sp+KEd9I6AJ9ZG4lNR1EMi
+9Md/nHvtLyxpfhEzYvkTZ4D9DQVz/mBYWRO6OTZRKnW9gd+efGN2aHFdkH1rJl4BL9Wyrk
+ghYG22D2T1Da2EyNSH1+LabK33gQweTJADBdyMwlo5r6PqcP2dzhg=
```

Se você estiver executando este tutorial a partir de um processador de comando do tipo Unix, como `bash`, poderá automatizar a aquisição de iterador de estilhaços usando um comando aninhado, como este (role horizontalmente para ver o comando inteiro):

```
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 a partir de um sistema compatível com PowerShell, poderá automatizar a aquisição do iterador de estilhaços usando um comando como este (role horizontalmente para ver o comando inteiro):

```
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 comando `get-records` solicitará registros do seu stream para o estilhaço que você especificou quando obteve o iterador de estilhaços, como no exemplo a seguir (role horizontalmente para ver a saída inteira):

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

  "NextShardIterator": "AAAAAAAAAEDOW3ugseWPE4503kqN1yN1UaodY8unE0sYslMUmC6lX9hlig5+t4RtZM0/
tALfiI4QGjunVgJvQsjxjh2aLyxaAaPr
+LaoENQ7eVs4EdYXgKyThTZGPcca2fVXYJWL3yafv9dsDwsYVedI66dbMZFC8rPMWc797zxQkv4pSKvPOZvrUIudb8UkH3VMzx58Is=
"
}
```

Observe que `get-records` é descrito acima como uma solicitação, ou seja, você pode receber zero ou mais registros mesmo que haja registros em seu stream, e os registros retornados podem não representar todos os registros que existem atualmente no stream. Isso é perfeitamente normal, e o código de produção simplesmente sondará o stream quanto aos registros em intervalos apropriados (essa velocidade de sondagem varia de acordo com os requisitos de design específicos do aplicativo).

A primeira coisa que você provavelmente observará sobre seu registro nesta parte do tutorial é que os dados parecem ser lixo – e não o texto limpo `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 na AWS CLI não fornece decodificação Base64, pois essa decodificação

para conteúdo binário bruto impresso em stdout pode acarretar comportamento indesejado e possíveis problemas de segurança em algumas plataformas e terminais. Se utilizar um decodificador Base64 (por exemplo, <https://www.base64decode.org/>) para decodificar manualmente dGVzdGRhdGE= verá que é, de facto, testdata. Isto é suficiente para este tutorial porque, na prática, o AWS CLI é raramente utilizado para consumir dados, mas mais frequentemente para monitorizar o estado do fluxo e obter informações, conforme mostrado anteriormente (describe-stream e list-streams). Tutoriais futuros mostrarão como criar aplicativos de consumidor de qualidade de produção usando a Kinesis Client Library (KCL), que se encarregará da Base64 para você. Para mais informações sobre o KCL, consulte [Desenvolver consumidores personalizados com rendimento partilhado utilizando KCL](#).

Nem sempre get-records retornará todos os registros no stream/estilhaço especificado. Quando isso acontecer, use o NextShardIterator a partir do último resultado para obter o próximo conjunto de registros. Portanto, se mais dados estavam sendo colocados no stream (situação normal em aplicativos de produção), você pode continuar sondando dados usando get-records todas as vezes. No entanto, se você não chamar get-records usando o próximo iterador de estilhaços dentro do tempo de vida de 300 segundos do iterador de estilhaços, receberá uma mensagem de erro e precisará usar o comando get-shard-iterator para obter um novo iterador de estilhaços.

Essa saída também fornece MillisBehindLatest, que é o número de milissegundos em que a resposta da operação GetRecords está da ponta do stream, indicando o atraso do consumidor em relação ao 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. Isso não é problema, pois os registros de dados permanecerão em um stream por 24 horas, aguardando você recuperá-los. Esse período, chamado de período de retenção, é configurável para até 168 horas (7 dias).

Observe que um resultado bem-sucedido de get-records sempre terá um NextShardIterator, mesmo que não haja mais nenhum registro atualmente no stream. Este é um modelo de sondagem que assume que um produtor colocará mais registros no stream em um determinado momento. Você pode criar suas próprias rotinas de sondagem, porém, se usar a KCL mencionada anteriormente para desenvolver aplicativos de consumidor, ela se encarregará dessa sondagem para você.

Se você chamar get-records até que não haja mais registros no stream e o estilhaço do qual você está extraindo, verá uma saída com registros vazios, semelhante ao exemplo a seguir (role horizontalmente para ver a saída inteira):

```
{
  "Records": [],
  "NextShardIterator": "AAAAAAAAAAGCJ5jzQNjmdhO6B/YDIDE56jmZmrmmA/r1WjoHXC/
kPJXc1rckt3TFL55dENfe5meNgdkyCRpUPGzJpMgYHaJ53C3nCnAjq6s7ZupjXeJGoUFS5oCuFwhP+Wul/
EhyNeS5DYXLSSC5XCcapmCAYGFjYER69QsdQjxMmbPE/hiybFDi5qtkT6/PsZNz6kFoqtDk="
}
```

## Etapa 4. Limpeza

Por fim, convém excluir seu stream para liberar recursos e evitar cobranças indesejadas à sua conta, como observado anteriormente. Faça isso efetivamente sempre que tiver criado um stream que não será usado, pois as cobranças incidem por stream mesmo que ele não seja usado para colocar e obter dados. O comando de limpeza é simples:

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

Como o êxito do comando não gera saída, você pode usar describe-stream para verificar o andamento da exclusão:

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

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

```
{
  "StreamDescription": {
    "StreamStatus": "DELETING",
    "StreamName": "Foo",
    "StreamARN": "arn:aws:kinesis:us-west-2:account-id:stream/Foo",
    "Shards": []
  }
}
```

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

```
A client error (ResourceNotFoundException) occurred when calling the DescribeStream
operation:
Stream Foo under account 112233445566 not found.
```

# Exemplos de tutoriais do Amazon Kinesis Data Streams

Os exemplos de tutoriais desta seção são projetados para oferecer mais auxílio na compreensão da funcionalidade e de conceitos do Amazon Kinesis Data Streams.

## Tópicos

- [Tutorial: Processar dados de ações em tempo real usando a KCL e a KCL 2.x \(p. 25\)](#)
- [Tutorial: Processar dados de ações em tempo real usando a KPL e a KCL 1.x \(p. 39\)](#)
- [Tutorial: Analisar dados de ações em tempo real usando o Kinesis Data Analytics for Java Applications \(p. 54\)](#)
- [Tutorial: Usar o AWS Lambda com o Amazon Kinesis Data Streams \(p. 72\)](#)
- [AWS Solução de dados de transmissão para Amazon Kinesis \(p. 72\)](#)

## Tutorial: Processar dados de ações em tempo real usando a KCL e a KCL 2.x

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

### Important

Depois de criar um streaming, sua conta incorre em cobranças nominais pelo uso do Kinesis Data Streams, pois o Kinesis Data Streams não está qualificado para o nível gratuito da AWS. Depois que for iniciado, o aplicativo consumidor também incorrerá em cobranças nominais pelo uso do Amazon DynamoDB. O aplicativo consumidor usa o DynamoDB para rastrear o estado de processamento. Ao terminar de usar este aplicativo, exclua seus recursos da AWS para parar de gerar cobranças. Para obter mais informações, consulte [Etapa 7: Finalizar \(p. 38\)](#).

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 uma das [regiões da AWS](#) que oferecem suporte ao Kinesis Data Streams.

### Tarefas

- [Pré-requisitos \(p. 26\)](#)
- [Etapa 1: Criar um stream de dados \(p. 26\)](#)
- [Etapa 2: Criar uma política e um usuário do IAM \(p. 27\)](#)
- [Etapa 3: Fazer download e criar o código \(p. 30\)](#)
- [Etapa 4: Implementar o produtor \(p. 31\)](#)
- [Etapa 5: Implementar o consumidor \(p. 34\)](#)
- [Etapa 6: \(Opcional\) Estender o consumidor \(p. 37\)](#)
- [Etapa 7: Finalizar \(p. 38\)](#)

## Pré-requisitos

É necessário atender aos seguintes requisitos para concluir este tutorial:

### Conta do Amazon Web Services

Antes de começar, familiarize-se com os conceitos abordados em [Terminologia e conceitos do Amazon Kinesis Data Streams \(p. 12\)](#), especialmente com fluxos, estilhaços, produtores e consumidores. Também é útil concluir as etapas no seguinte guia: [Instalar e configurar o AWS CLI \(p. 18\)](#).

São necessários uma conta da AWS e um navegador da web para acessar o Console de gerenciamento da AWS.

Para acesso ao console, use seu nome de usuário e senha para do IAM para fazer login no [Console de gerenciamento da AWS](#) a partir da [página de login do IAM](#). O IAM permite controlar com segurança o acesso a serviços e recursos da AWS na sua conta da AWS. Para obter detalhes sobre credenciais programáticas e do console, consulte [Compreensão e obtenção de suas credenciais de segurança](#) no AWS General Reference.

Para obter mais informações sobre o IAM e instruções de configuração de chave de segurança, consulte [Criar um usuário do IAM](#).

### Requisitos de software do sistema

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

Se você tiver um Java IDE, como o [Eclipse](#), será possível usá-lo para abrir, editar, compilar e executar o código-fonte.

Você precisa da versão mais recente do [AWS SDK for Java](#). Se você usar o Eclipse como o IDE, poderá instalar o [AWS Toolkit for Eclipse](#).

O aplicativo consumidor exige a Kinesis Client Library (KCL) versão 2.2.9 ou posterior, que é possível obter do GitHub em <https://github.com/aws-labs/amazon-kinesis-client/tree/master>.

### Próximas etapas

[Etapa 1: Criar um stream de dados \(p. 26\)](#)

## Etapa 1: Criar um stream 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 Console de gerenciamento da AWS e abra o console do Kinesis em <https://console.aws.amazon.com/kinesis>.
2. Selecione Data Streams (Streams 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 AK).
5. Insira um nome para seu fluxo de dados (por exemplo, **StockTradeStream**).
6. Digite **1** como o número de estilhaços, mas deixe Estimate the number of shards you'll need (Estimar o número de estilhaços de que você precisará) recolhido.
7. Escolha Create Kinesis stream (Criar streaming do AK).

Na página Kinesis Stream List (Lista de fluxos do Kinesis), o status do seu fluxo é exibido 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

[Etapa 2: Criar uma política e um usuário do IAM \(p. 27\)](#)

## Etapa 2: Criar uma política e um usuário do IAM

As melhores práticas de segurança da AWS determinam o uso de permissões granulares para controlar o acesso a recursos diferentes. O AWS Identity and Access Management (IAM) permite gerenciar usuários e permissões de usuário na AWS. Uma [Política do IAM](#) lista explicitamente as ações 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
<code>DescribeStream</code> , <code>DescribeStreamSummary</code> , <code>DescribeStreamConsumer</code>	Kinesis data stream	Antes de tentar ler registros, o consumidor verifica se o fluxo de dados está contido no fluxo de dados.
<code>SubscribeToShard</code> , <code>RegisterStreamConsumer</code>	Kinesis data stream	Assina e registra os consumidores em um estilhaço.
<code>PutRecord</code> , <code>PutRecords</code>	Kinesis data stream	Grava registros no Kinesis Data Streams.

### Consumidor

Ações	Recurso	Finalidade
<code>DescribeStream</code>	Kinesis data stream	Antes de tentar ler registros, o consumidor verifica se o fluxo de dados está contido no fluxo de dados.

Ações	Recurso	Finalidade
GetRecords, GetShardIterator	Kinesis data stream	Lê registros de um estilhaço.
CreateTable, DescribeTable, GetItem, PutItem, Scan, UpdateItem	Tabela do Amazon DynamoDB	Se o consumidor é desenvolvido usando a Kinesis Client Library, as permissões para uma tabela do DynamoDB para rastrear o progresso.
DeleteItem	Tabela do Amazon DynamoDB	Para quando o consumidor executar operações de divisão de Streams.
PutMetricData	Log do Amazon CloudWatch	A KCL também faz upload de métricas para o CloudWatch.

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

### Para criar uma política do IAM

1. Localize o nome de recurso da Amazon (ARN) para o novo fluxo de dados criado na etapa acima. Você pode encontrar esse ARN listado como Stream ARN (ARN do streaming) na parte superior da guia Details (Detalhes). O formato do ARN é o seguinte:

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

região

Código da região da AWS; por exemplo, us-west-2. Para obter mais informações, consulte [Conceitos de região e zona de disponibilidade](#).

account

O ID da conta da AWS, conforme mostrado em [Configurações da conta](#).

name

O nome do fluxo de dados criado na etapa acima, que é `StockTradeStream`.

2. Determine o ARN da tabela do DynamoDB a ser usada pelo consumidor (e a ser criada pela primeira instância de 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 no ARN do fluxo de dados que você está usando neste tutorial, mas o nome é o nome da tabela do DynamoDB criada e usada pelo aplicativo consumidor. A KCL usa o nome do aplicativo como nome da tabela. Nesta etapa, use `StockTradesProcessor` para o nome da tabela do DynamoDB, porque esse é o nome do aplicativo usado nas etapas posteriores deste tutorial.

3. No console do IAM, em Políticas (Políticas) (<https://console.aws.amazon.com/iam/home#policies>), selecione Create policy (Criar política). Se esta for a primeira vez que você trabalha com políticas do IAM, escolha Get Started (Conceitos básicos), Create Policy (Criar política).
4. Escolha Select (Selecionar) ao lado de Policy Generator (Gerador de políticas).
5. Escolha Amazon Kinesis como o serviço da 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:

Serviço da AWS	Ações	ARN
Amazon DynamoDB	CreateTable, DeleteItem, DescribeTable, GetItem, PutItem, Scan, UpdateItem	O ARN da tabela do DynamoDB criada na Etapa 2 deste procedimento.
Amazon CloudWatch	PutMetricData	*

O asterisco (\*) é usado quando não é necessário especificar um ARN. Nesse caso, é porque não há nenhum recurso específico no CloudWatch para o qual a ação PutMetricData é 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 console do IAM em <https://console.aws.amazon.com/iam/>.
2. Na página Users (Usuários), selecione Add user (Adicionar usuário).
3. Para User name, digite `StockTradeStreamUser`.
4. Em Access type (Tipo de acesso), escolha Programmatic access (Acesso programático) e, em seguida, escolha Next: Permissions (Próximo: permissões).
5. Selecione Attach existing policies directly.
6. Pesquise por nome a política criada no procedimento acima (`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 política do IAM a um usuário

1. No console do IAM, abra [Políticas \(Políticas\)](#) e selecione Policy Actions (Ações da política).
2. Escolha `StockTradeStreamPolicy` e Attach (Anexar).
3. Escolha `StockTradeStreamUser` e Attach Policy (Anexar política).

## Próximas etapas

[Etapa 3: Fazer download e criar o código \(p. 30\)](#)

## Etapa 3: Fazer download e criar 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. Faça download do código-fonte do repositório do GitHub <https://github.com/aws-samples/amazon-kinesis-learning> no seu computador.

2. Crie um projeto no IDE com o código-fonte, respeitando a estrutura de diretório fornecida.
3. Adicione as seguintes bibliotecas ao projeto:
  - Amazon Kinesis Client Library (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
  - Jackson Dataformat: CBOR
  - Joda Time
4. Dependendo do seu IDE, o projeto pode ser compilado automaticamente. Se não, compile o projeto usando as etapas apropriadas para o seu IDE.

Se concluiu essas etapas com êxito, você agora está pronto para ir para a próxima seção, [the section called “Etapa 4: Implementar o produtor” \(p. 31\)](#).

## Próximas etapas

(p. 31)

## Etapa 4: Implementar 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.

### Classe StockTrade

Uma transação de ações 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 instância StockTrade no formato JSON. Por exemplo:

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

### Classe StockTradeGenerator

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

### Classe StockTradesWriter

O método `main` do produtor, `StockTradesWriter` recupera continuamente uma transação aleatória e a envia ao 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 região, credenciais e 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:

- A API `PutRecord` espera uma matriz de bytes, e é necessário converter a transação para o formato JSON. 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.

```
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();
```

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 [Gravar dados no Amazon Kinesis Data Streams \(p. 89\)](#).

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 taxa de transferência e ficar limitado. É recomendado considerar cuidadosamente sua política de tentativa para operações `put` a fim de evitar perda de dados, por exemplo, usando como uma simples 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 da API do Kinesis 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 [Gravar dados no Amazon Kinesis Data Streams \(p. 89\)](#).

Para executar o produtor

1. Verifique se a chave de acesso e o par de chaves secretas recuperados em [Etapa 2: Criar uma política e um usuário do IAM \(p. 27\)](#) 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 transações de ações estão sendo ingeridas pelo Kinesis Data Streams.

## Próximas etapas

[Etapa 5: Implementar o consumidor \(p. 34\)](#)

## Etapa 5: Implementar 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 compilado sobre a Kinesis Client Library (KCL), que faz grande parte do trabalho pesado comum a aplicativos consumidores. Para obter mais informações, consulte [Utilizar a Biblioteca de Clientes da Kinesis \(p. 119\)](#).

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

Classe `StockTradesProcessor`

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

- Lê o aplicativo, o fluxo de dados e os nomes de região passados como argumentos.
- Cria uma instância de `KinesisAsyncClient` com o nome da região.
- Cria uma instância de `StockTradeRecordProcessorFactory` que veicula instâncias de `ShardRecordProcessor`, implementadas por uma instância de `StockTradeRecordProcessor`.
- Cria uma instância de `ConfigsBuilder` com a instância de `KinesisAsyncClient`, `StreamName`, `ApplicationName` e `StockTradeRecordProcessorFactory`. Isso é útil para criar todas as configurações com valores padrão.
- Cria um programador da KCL (anteriormente, nas versões 1.x da KCL, era conhecido como o operador da KCL) com a instância de `ConfigsBuilder`.



- 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.

#### Classe `StockTradeRecordProcessor`

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

Os métodos `initialize` e `shutdownRequested` são usados pela KCL para permitir que o processador de registros saiba quando ele deve estar pronto para começar a receber registros e quando ele deve esperar parar de receber registros, respectivamente, para que ele possa executar qualquer configuração específica do aplicativo e tarefas de encerramento. `leaseLost` e `shardEnded` são usados para implementar qualquer lógica para o que fazer quando um contrato de aluguel é perdido ou um processamento chegou ao fim de um estilhaço. Neste exemplo, simplesmente registramos 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 definição de pontos de verificação, consulte ???

#### Classe `StockStats`

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 corrente 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 método `reportStats` simples. Sinta-se à vontade para modificar o formato de saída mais adequado às 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 interface `ShardRecordProcessor`

```
@Override
public void leaseLost(LeaseLostInput leaseLostInput) {
    log.info("Lost lease, so terminating.");
}

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

@Override
public void shutdownRequested(ShutdownRequestedInput shutdownRequestedInput) {
    log.info("Scheduler is shutting down, checkpointing.");
    checkpoint(shutdownRequestedInput.checkpointer());
}

private void checkpoint(RecordProcessorCheckpointer checkpointer) {
    log.info("Checkpointing shard " + kinesisShardId);
    try {
        checkpointer.checkpoint();
    } catch (ShutdownException se) {
        // Ignore checkpoint if the processor instance has been shutdown (fail over).
        log.info("Caught shutdown exception, skipping checkpoint.", se);
    } catch (ThrottlingException e) {
        // Skip checkpoint when throttled. In practice, consider a backoff and retry
        policy.
    }
}
```

```
        log.error("Caught throttling exception, skipping checkpoint.", e);
    } catch (InvalidStateException e) {
        // This indicates an issue with the DynamoDB table (check for table,
        // provisioned IOPS).
        log.error("Cannot save checkpoint to the DynamoDB table used by the Amazon
        Kinesis Client Library.", e);
    }
}
```

Para executar o consumidor

1. Execute o produtor escrito em [\(p. 31\)](#) para injetar registros de negociações de ações no streaming.
2. Verifique se o par chave de acesso e chave secreta recuperado anteriormente (durante a criação do usuário do IAM) foi salvo no arquivo `~/.aws/credentials`.
3. Execute a classe `StockTradesProcessor` com os seguintes argumentos:

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

Observe que, se você criou o fluxo em uma região diferente de `us-west-2`, será necessário 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-0000000000001 stats for last 1 minute *****
Most popular stock being bought: WMT, 27 buys.
Most popular stock being sold: PTR, 14 sells.
*****
```

## Próximas etapas

[Etapa 6: \(Opcional\) Estender o consumidor \(p. 37\)](#)

## Etapa 6: (Opcional) Estender 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

[Etapa 7: Finalizar \(p. 38\)](#)

## Etapa 7: Finalizar

Como você está pagando para usar o Kinesis data stream, 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

Para processar uma grande quantidade de dados quase em tempo real, não é preciso escrever nenhum código mágico nem desenvolver uma imensa infraestrutura. É tão simples quanto escrever lógica para processar uma pequena quantidade de dados (como escrever `processRecord(record)`), só que usando o Kinesis Data Streams para escalonar para 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 disso para você. 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 operadores em instâncias do Amazon EC2 e usar os grupos do Auto Scaling.

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

Como um stream é processado continuamente, sua saída pode ser enviada a outros destinos. A AWS fornece [conectores](#) para integrar o Kinesis Data Streams a outros serviços da AWS e ferramentas de terceiros.

# Tutorial: Processar dados de ações em tempo real usando a KPL e a KCL 1.x

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

### Important

Depois de criar um streaming, sua conta incorre em cobranças nominais pelo uso do Kinesis Data Streams, pois o Kinesis Data Streams não está qualificado para o nível gratuito da AWS. Depois que for iniciado, o aplicativo consumidor também incorrerá em cobranças nominais pelo uso do Amazon DynamoDB. O aplicativo consumidor usa o DynamoDB para rastrear o estado de processamento. Ao terminar de usar este aplicativo, exclua seus recursos da AWS para parar de gerar cobranças. Para obter mais informações, consulte [Etapa 7: Finalizar \(p. 53\)](#).

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 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, como o Amazon Elastic Compute Cloud (Amazon EC2).

Os exemplos mostrados usam a região Oeste dos EUA (Oregon), mas funcionam em qualquer uma das [Regiões da AWS que oferecem suporte ao Kinesis Data Streams](#).

#### Tarefas

- [Pré-requisitos](#) (p. 40)
- [Etapa 1: Criar um stream de dados](#) (p. 41)
- [Etapa 2: Criar uma política e um usuário do IAM](#) (p. 42)
- [Etapa 3: Fazer download e criar o código de implementação](#) (p. 45)
- [Etapa 4: Implementar o produtor](#) (p. 46)
- [Etapa 5: Implementar o consumidor](#) (p. 49)
- [Etapa 6: \(Opcional\) Estender o consumidor](#) (p. 52)
- [Etapa 7: Finalizar](#) (p. 53)

## Pré-requisitos

Estes são os requisitos para concluir o [Tutorial: Processar dados de ações em tempo real usando a KPL e a KCL 1.x](#) (p. 39).

## Conta do Amazon Web Services

Antes de começar, familiarize-se com os conceitos abordados em [Terminologia e conceitos do Amazon Kinesis Data Streams](#) (p. 12), especialmente fluxos, estilhaços, produtores e consumidores. Também é útil ter concluído [Instalar e configurar o AWS CLI](#) (p. 18).

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

Para acesso ao console, use seu nome de usuário e senha para do IAM para fazer login no [Console de gerenciamento da AWS](#) a partir da [página de login do IAM](#). O IAM permite controlar com segurança o acesso a serviços e recursos da AWS na sua conta da AWS. Para obter detalhes sobre credenciais programáticas e do console, consulte [Compreensão e obtenção de suas credenciais de segurança](#) no AWS General Reference.

Para obter mais informações sobre o IAM e instruções de configuração de chave de segurança, consulte [Criar um usuário do IAM](#).

## Requisitos de software do sistema

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

Se você tiver um Java IDE, 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 usar o Eclipse como o IDE, poderá instalar o [AWS Toolkit for Eclipse](#).

O aplicativo consumidor requer a Kinesis Client Library (KCL) versão 1.2.1 ou superior, que você pode obter no GitHub em [Kinesis Client Library \(Java\)](#).

## Próximas etapas

[Etapa 1: Criar um stream de dados \(p. 41\)](#)

## Etapa 1: Criar um stream de dados

Na primeira etapa do [Tutorial: Processar dados de ações em tempo real usando a KPL e a KCL 1.x \(p. 39\)](#), você cria o streaming que usará em etapas subsequentes.

Para criar um fluxo

1. Faça login no Console de gerenciamento da AWS e abra o console do Kinesis em <https://console.aws.amazon.com/kinesis>.
2. Selecione Data Streams (Streams 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 AK).
5. Insira um nome para seu streaming (por exemplo, **StockTradeStream**).
6. Digite **1** como o número de estilhaços, mas deixe Estimate the number of shards you'll need (Estimar o número de estilhaços de que você precisará) recolhido.
7. Escolha Create Kinesis stream (Criar streaming do AK).

Na página Kinesis Stream List (Lista de streamings do AK), o status do seu streaming aparece como **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 estilhaços

Ao começar a usar o Kinesis Data Streams fora deste tutorial, você poderá precisar planejar o processo de criação do streaming 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 taxa de transferência. Na página Create Kinesis Stream (Criar streaming do AK), expanda Estimate the number of shards you'll need (Estimar o número de estilhaços de que você precisará). 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

Leve em conta o número de entidades que fornecem dados e o número aproximado de registros por segundo produzidos por cada um. 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 estilhaços, use o [Formulário de limites do Kinesis Data Streams](#). Para obter mais informações sobre streams e estilhaços, consulte [Criar e atualizar streams de dados \(p. 2\)](#) e [Criar e gerenciar streamings \(p. 73\)](#).

## Próximas etapas

[Etapa 2: Criar uma política e um usuário do IAM \(p. 42\)](#)

## Etapa 2: Criar uma política e um usuário do IAM

As melhores práticas de segurança da AWS determinam o uso de permissões granulares para controlar o acesso a recursos diferentes. O AWS Identity and Access Management (IAM) permite gerenciar usuários e permissões de usuário na AWS. Uma [Política do IAM](#) lista explicitamente as ações permitidas e os recursos aos quais as ações são aplicáveis.

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

### Produtor

Ações	Recurso	Finalidade
<code>DescribeStream</code> , <code>DescribeStreamSummary</code> , <code>DescribeStreamConsumer</code>	Kinesis data stream	Antes de tentar gravar registros, o produtor verifica se o stream contém registros e se o stream tem um consumidor.
<code>SubscribeToShard</code> , <code>RegisterStreamConsumer</code>	Kinesis data stream	Faz a inscrição e registra um consumidor em um estilhaço.
<code>PutRecord</code> , <code>PutRecords</code>	Kinesis data stream	Gravar registros no Kinesis Data Streams.

### Consumidor

Ações	Recurso	Finalidade
<code>DescribeStream</code>	Kinesis data stream	Antes de tentar ler registros, o consumidor verifica se o stream contém registros.
<code>GetRecords</code> , <code>GetShardIterator</code>	Kinesis data stream	Ler registros de um estilhaço do Kinesis Data Streams.
<code>CreateTable</code> , <code>DescribeTable</code> , <code>GetItem</code> , <code>PutItem</code> , <code>Scan</code> , <code>UpdateItem</code>	Tabela do Amazon DynamoDB	Se o consumidor é desenvolvido usando a Kinesis Client Library, cria a tabela do DynamoDB para rastrear o estado de processamento.
<code>DeleteItem</code>	Tabela do Amazon DynamoDB	Para quando o consumidor executar operações de divisão de Streams.



Ações	Recurso	Finalidade
PutMetricData	Log do Amazon CloudWatch	A KCL também faz upload de métricas para o CloudWatch

Para esse aplicativo, você cria uma única política do IAM 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 Nome de recurso da Amazon (ARN) para o novo stream. Você pode encontrar esse ARN listado como Stream ARN (ARN do streaming) na parte superior da guia Details (Detalhes). O formato do ARN 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](#).

account

O ID da conta da AWS, conforme mostrado em [Configurações da conta](#).

name

Nome do stream de [Etapa 1: Criar um stream de dados \(p. 41\)](#), que é StockTradeStream.

2. Determine o ARN da tabela do DynamoDB a ser usada pelo consumidor (e criada pela primeira instância de 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. A KCL usada pelo consumidor usa o nome do aplicativo como o nome da tabela. Use o nome do aplicativo que será usado mais tarde, StockTradesProcessor.

3. No console do IAM, em Policies (Políticas) (<https://console.aws.amazon.com/iam/home#policies>), selecione Create policy (Criar política). Se esta for a primeira vez que você trabalha com políticas do IAM, escolha Get Started (Conceitos básicos), Create Policy (Criar política).
4. Escolha Select (Selecionar) ao lado de Policy Generator (Gerador de políticas).
5. Escolha Amazon Kinesis como o serviço da AWS.
6. Selecione DescribeStream, GetShardIterator, GetRecords, PutRecord e PutRecords como ações permitidas.
7. Digite o ARN criado na Etapa 1.
8. Use Add Statement (Adicionar instrução) para cada um dos seguintes:

Serviço da AWS	Ações	ARN
Amazon DynamoDB	CreateTable, DeleteItem, DescribeTable, GetItem, PutItem, Scan, UpdateItem	O ARN que você criou na etapa 2

Serviço da AWS	Ações	ARN
Amazon CloudWatch	PutMetricData	*

O asterisco (\*) é usado quando não é necessário especificar um ARN. Nesse caso, é porque não há nenhum recurso específico no CloudWatch para o qual a ação PutMetricData é 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 console do IAM em <https://console.aws.amazon.com/iam/>.
2. Na página Users (Usuários), selecione Add user (Adicionar usuário).
3. Para User name, digite `StockTradeStreamUser`.
4. Em Access type (Tipo de acesso), escolha Programmatic access (Acesso programático) e, em seguida, escolha Next: Permissions (Próximo: permissões).
5. Selecione Attach existing policies directly.
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 política do IAM a um usuário

1. No console do IAM, abra [Políticas \(Políticas\)](#) e selecione Policy Actions (Ações da política).
2. Escolha `StockTradeStreamPolicy` e Attach (Anexar).
3. Escolha `StockTradeStreamUser` e Attach Policy (Anexar política).

## Próximas etapas

[Etapa 3: Fazer download e criar o código de implementação \(p. 45\)](#)

## Etapa 3: Fazer download e criar o código de implementação

O código esqueleto é fornecido para o [the section called “Tutorial: Processar dados de ações em tempo real usando a KPL e a KCL 1.x” \(p. 39\)](#). 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 as implementações.

Para fazer download e compilação do código de implementação

1. Faça download do [código-fonte](#) no computador.
2. Crie um projeto no IDE favorito com o código-fonte, respeitando a estrutura de diretório fornecida.
3. Adicione as seguintes bibliotecas ao projeto:
  - Amazon Kinesis Client Library (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
  - Jackson Dataformat: CBOR
  - Joda Time
4. Dependendo do seu IDE, o projeto pode ser compilado automaticamente. Se não, compile o projeto usando as etapas apropriadas para o seu IDE.

Se concluiu essas etapas com êxito, você agora está pronto para ir para a próxima seção, [the section called “Etapa 4: Implementar o produtor” \(p. 46\)](#). Se a compilação gerar erros em qualquer estágio, investigue e os corrija antes de continuar.

## Próximas etapas

(p. 46)

## Etapa 4: Implementar o produtor

O aplicativo no [Tutorial: Processar dados de ações em tempo real usando a KPL e a KCL 1.x \(p. 39\)](#) 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.

### Classe StockTrade

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 instância `StockTrade` no formato JSON. Por exemplo:

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

### Classe StockTradeGenerator

`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.

### Classe StockTradesWriter

O método `main` do produtor, `StockTradesWriter`, recupera continuamente uma negociação aleatória e, em seguida, 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:

- A API de `PutRecord` espera uma matriz de bytes, e você precisa converter `trade` no formato JSON. 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 \(p. 100\)](#).

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 streaming pode atingir o limite de taxa de transferência e ficar limitado. Recomendamos considerar cuidadosamente sua política de retentativa para operações `put` a fim de evitar perda de dados, por exemplo, usando como uma simples retentativa.

- O registro de status é útil mas opcional:

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

O produtor mostrado aqui usa a funcionalidade de registro único da API do Kinesis 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 \(p. 100\)](#).

Para executar o produtor

1. Verifique se o par chave de acesso e chave secreta recuperado anteriormente (durante a criação do usuário do IAM) foi salvo 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 stream de negociações de ações agora está sendo ingerido pelo Kinesis Data Streams.

## Próximas etapas

[Etapa 5: Implementar o consumidor \(p. 49\)](#)

## Etapa 5: Implementar o consumidor

O aplicativo consumidor no [Tutorial: Processar dados de ações em tempo real usando a KPL e a KCL 1.x \(p. 39\)](#) processa continuamente o streaming de negociações de ações criado em [\(p. 46\)](#).

Em seguida, ele produz as ações mais populares compradas e vendidas a cada minuto. O aplicativo é compilado sobre a Kinesis Client Library (KCL), que faz grande parte do trabalho pesado comum a aplicativos consumidores. Para obter mais informações, consulte [Desenvolver consumidores do KCL 1.x \(p. 131\)](#).

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

### Classe StockTradesProcessor

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 operador da KCL com a instância de `RecordProcessorFactory` e uma configuração padrão, inclusive o nome do streaming, as credenciais e o nome do aplicativo.
- O operador cria um novo thread para cada estilhaço (atribuído a essa instância de consumidor), que faz loop continuamente para ler registros do Kinesis Data Streams. Em seguida, ele invoca a instância de `RecordProcessor` para processar cada lote de registros recebidos.

### Classe StockTradeRecordProcessor

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 que possa realizar qualquer tarefa de configuração e encerramento específica ao aplicativo. 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 \(p. 51\)](#).

#### Classe StockStats

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 corrente 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 " + kinesisisShardId + " 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 [\(p. 46\)](#) para injetar registros de negociações de ações no streaming.



2. Verifique se o par chave de acesso e chave secreta recuperado anteriormente (durante a criação do usuário do IAM) foi salvo 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-0000000000001 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 estiver familiarizado com as vantagens da Kinesis Client Library, abordada em [Desenvolver consumidores do KCL 1.x \(p. 131\)](#) e em outros lugares, você poderá se perguntar por que deve usá-la aqui. Embora você use apenas um streaming de estilhaço e uma instância de consumidor para processá-la, ainda é mais fácil implementar o consumidor usando a 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 a KCL fornece.

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 do Kinesis Data Streams. A KCL obtém os registros e invoca 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 definição de pontos de verificação significa registrar o ponto no streaming até os registros de dados que foram consumidos e processados até o momento, de modo que, se o aplicativo falhar, o streaming será lido a partir daquele ponto e não do início do streaming. 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.

Conforme visto no [\(p. 46\)](#), as operações `put` na API do Kinesis 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 estilhaços (quando há mais de um estilhaço 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.

Neste aplicativo, como o processamento de registros do consumidor não é intensivo, você pode usar um estilhaço e fazer o processamento no mesmo thread da KCL. 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, a KCL pode obter novos registros mais rapidamente, enquanto outros threads podem processar os registros em paralelo. O design multithread não é trivial e deve ser planejado com técnicas avançadas, portanto, aumentar a contagem de estilhaços costuma ser a maneira mais eficiente e econômica de escalar.

## Próximas etapas

[Etapa 6: \(Opcional\) Estender o consumidor \(p. 52\)](#)

## Etapa 6: (Opcional) Estender o consumidor

O aplicativo no [Tutorial: Processar dados de ações em tempo real usando a KPL e a KCL 1.x \(p. 39\)](#) 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

[Etapa 7: Finalizar \(p. 53\)](#)

## Etapa 7: Finalizar

Como você está pagando para usar o Kinesis data stream, 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

Para processar uma grande quantidade de dados quase em tempo real, não é preciso escrever nenhum código mágico nem desenvolver uma imensa infraestrutura. É tão simples quanto escrever lógica para processar uma pequena quantidade de dados (como escrever `processRecord(Record)`), só que usando o Kinesis Data Streams para escalonar para 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 disso para você. 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 operadores em instâncias do Amazon EC2 e usar os grupos do Auto Scaling.

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

Como um stream é processado continuamente, sua saída pode ser enviada a outros destinos. A AWS fornece [conectores](#) para integrar o Kinesis Data Streams a outros serviços da AWS e ferramentas de terceiros.

## Próximas etapas

- Para obter mais informações sobre o uso das operações da API do Kinesis Data Streams, consulte [Desenvolvimento de produtores usando a API do Amazon Kinesis Data Streams com o AWS SDK for](#)

[Java](#) (p. 100), [Desenvolver consumidores personalizados com taxa de transferência compartilhada usando o AWS SDK for Java](#) (p. 157) e [Criar e gerenciar streamings](#) (p. 73).

- Para obter mais informações sobre o Kinesis Client Library, consulte [Desenvolver consumidores do KCL 1.x](#) (p. 131).
- Para obter mais informações sobre como otimizar seu aplicativo, consulte [Tópicos avançados](#) (p. 182).

## Tutorial: Analisar dados de ações em tempo real usando o Kinesis Data Analytics for Java Applications

O cenário deste tutorial envolve ingerir negociações de ações em um fluxo de dados e gravar um aplicativo do [Amazon Kinesis Data Analytics](#) simples que realize cálculos no fluxo. Você aprenderá a enviar um streaming de registros para o Kinesis Data Streams e implementar um aplicativo que consome e processa os registros quase em tempo real.

Com o Amazon Kinesis Data Analytics para aplicativos Java, você pode usar o Java para processar e analisar dados de streaming. O serviço permite que você crie e execute o código Java em origens de streaming para executar análises de séries temporais, alimentar painéis em tempo real e criar métricas em tempo real.

Você pode criar aplicativos Java no Kinesis Data Analytics 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 uso do Kinesis Data Streams e do Kinesis Data Analytics, porque eles não estão qualificados para o nível gratuito da AWS. Ao terminar de usar este aplicativo, exclua seus recursos da AWS para parar de gerar 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 deslizando 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 uma das [regiões da AWS que oferecem suporte ao Kinesis Data Analytics](#).

### Tarefas

- [Pré-requisitos para concluir os exercícios](#) (p. 54)
- [Etapa 1: configurar uma conta da AWS e criar um usuário administrador](#) (p. 55)
- [Etapa 2: configurar a AWS Command Line Interface \(AWS CLI\)](#) (p. 57)
- [Etapa 3: criar e executar um aplicativo do Kinesis Data Analytics para Java](#) (p. 58)

## 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 do ambiente `JAVA_HOME` para apontar para o local de instalação do JDK.
- 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
```

#### Note

Kinesis Data Analytics for Java Applications só é compatível com aplicativos Java criados com o Apache Maven.

Para começar a usar, vá até [Etapa 1: configurar uma conta da AWS e criar um usuário administrador](#) (p. 55).

## Etapa 1: configurar uma conta da AWS e criar um usuário administrador

Antes de usar o Amazon Kinesis Data Analytics for Java Applications pela primeira vez, conclua as seguintes tarefas:

1. [Cadastre-se na AWS](#) (p. 55)
2. [Criar um usuário do IAM](#) (p. 56)

### Cadastre-se na AWS

Ao se cadastrar na Amazon Web Services (AWS), sua conta da AWS é automaticamente cadastrada em todos os serviços da AWS, incluindo o Amazon Kinesis Data Analytics. Você será cobrado apenas pelos serviços que usar.

Com o Kinesis Data Analytics, você paga apenas pelos recursos que usa. Se você for um cliente novo da AWS, poderá começar a usar o Kinesis Data Analytics gratuitamente. Para obter mais informações, consulte [Nível gratuito da AWS](#).

Caso você já tenha uma conta da AWS, passe à próxima tarefa. Se você não tiver uma conta da AWS, siga estas etapas para criar uma.

Para criar uma conta da AWS

1. Abra <https://portal.aws.amazon.com/billing/signup>.
2. Siga as instruções online.

Parte do procedimento de cadastro envolve uma chamada telefônica e a digitação de um código de verificação usando o teclado do telefone.

Observe o ID da conta da AWS porque você precisará dele na próxima tarefa.

## Criar um usuário do IAM

Os serviços na AWS, como o Amazon Kinesis Data Analytics, exigem que credenciais sejam fornecidas ao acessá-los. Dessa maneira, o serviço pode determinar se você tem permissões para acessar os recursos próprios desse serviço. O Console de gerenciamento da AWS exige que você insira sua senha.

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

Se tiver se cadastrado na AWS, mas não tiver criado um usuário do IAM para si, você poderá criar um usando o console do IAM.

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 Console de gerenciamento da AWS e abra o console do IAM 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 política AdministratorAccess. 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.

Como criar um usuário do IAM para si mesmo, adicioná-lo ao grupo Administradores e criar 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. Selecione Programmatic access (Acesso programático) e AWS Management Console access (Acesso ao Console de Gerenciamento da AWS).
4. Escolha Próximo: Permissões.
5. Marque a caixa de seleção ao lado do grupo Administrators (Administradores). Então, escolha Próximo: Análise.
6. Escolha Criar usuário.

Como fazer login como o novo usuário do IAM

1. Saia do Console de gerenciamento da AWS.
2. Use o seguinte formato de URL para fazer login no console:

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

O `aws_account_number` é o ID de sua conta da AWS sem hífens. Por exemplo, se o ID da conta da AWS for 1234-5678-9012, substitua `aws_account_number` por **123456789012**. Para obter

informações sobre como localizar o número da conta, consulte [O ID da sua conta da AWS e o alias no Guia do usuário do IAM](#).

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

#### Note

Se não quiser que o URL da página de cadastro contenha o ID da sua conta da AWS, crie um alias da conta.

Para criar ou remover um alias de conta

1. Abra o console do IAM em <https://console.aws.amazon.com/iam/>.
2. No painel de navegação, selecione Dashboard (Painel).
3. Encontre o link de login de usuário do IAM.
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 Customize (Personalizar) e, em seguida, selecione Yes, Delete (Sim, excluir). A URL de login será revertida para o ID da sua conta da AWS.

Para fazer o login depois de criar o alias de uma conta, use o seguinte URL:

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

Para verificar o link de login para usuários do IAM em sua conta, abra o console do IAM e procure em IAM users sign-in link (Link de login de usuários do IAM) no painel.

Para obter mais informações sobre IAM, consulte o seguinte:

- [AWS Identity and Access Management \(IAM\)](#)
- [Conceitos básicos](#)
- [Guia do usuário do IAM](#)

## Próxima etapa

[Etapa 2: configurar a AWS Command Line Interface \(AWS CLI\) \(p. 57\)](#)

## Etapa 2: configurar a AWS Command Line Interface (AWS CLI)

Nesta etapa, você faz download e configura a AWS CLI para usá-la com o Amazon Kinesis Data Analytics for Java Applications.

#### 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 tiver a AWS CLI instalada, pode ser necessário atualizá-la para obter as funcionalidades mais recentes. Para obter mais informações, consulte [Instalar a interface de](#)

[linha de comando da AWS](#) no Guia do usuário do AWS Command Line Interface. Para verificar a versão da AWS CLI, execute o seguinte comando:

```
aws --version
```

Os exercícios neste tutorial requerem a seguinte versão da AWS CLI ou posterior:

```
aws-cli/1.16.63
```

Para configurar a AWS CLI

1. Baixe e configure a AWS CLI. Para obter instruções, consulte os tópicos a seguir 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 configuração da AWS CLI. Você pode usar esse perfil ao executar os comandos da AWS CLI. Para obter mais informações sobre perfis nomeados, consulte [Perfis nomeados](#) no Guia do usuário do 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 regiões da AWS disponíveis, consulte [Regiões e endpoints da AWS](#) no Referência geral do Amazon Web Services.

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

```
aws help
```

Depois de configurar uma conta da AWS e a AWS CLI, você pode experimentar o próximo exercício, no qual configura um aplicativo de amostra e testa a configuração completa.

## Próxima etapa

[Etapa 3: criar e executar um aplicativo do Kinesis Data Analytics para Java \(p. 58\)](#)

# Etapa 3: criar e executar um aplicativo do Kinesis Data Analytics para Java

Neste exercício, você cria um Kinesis Data Analytics for Java application com fluxos de dados como uma origem e um destino.

Esta seção contém as seguintes etapas:

- [Criar dois fluxos de dados do Amazon Kinesis \(p. 59\)](#)
- [Gravação de registros de amostra no fluxo de entrada \(p. 59\)](#)
- [Baixar e examinar o código Java Apache Flink Streaming \(p. 60\)](#)
- [Compilar o código do aplicativo \(p. 60\)](#)
- [Fazer upload do código Java Apache Flink Streaming \(p. 63\)](#)
- [Criar e executar o aplicativo do Kinesis Data Analytics \(p. 63\)](#)



## Criar dois fluxos de dados do Amazon Kinesis

Antes de criar um Kinesis Data Analytics for Java application para este exercício, crie dois fluxos de dados do Kinesis (`ExampleInputStream` e `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. Use o seguinte comando da AWS CLI do Amazon Kinesis `create-stream` para criar o primeiro fluxo (`ExampleInputStream`).

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

2. Execute o mesmo comando, alterando o nome do fluxo para `ExampleOutputStream`, a fim de criar o segundo fluxo que o aplicativo usará para gravar a saída.

```
$ 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 seguinte conteúdo:

```
import json
import boto3
import random
import datetime

kinesis = boto3.client('kinesis')
def getReferrer():
    data = {}
    now = datetime.datetime.now()
    str_now = now.isoformat()
    data['EVENT_TIME'] = str_now
    data['TICKER'] = random.choice(['AAPL', 'AMZN', 'MSFT', 'INTC', 'TBV'])
    price = random.random() * 100
    data['PRICE'] = round(price, 2)
    return data

while True:
    data = json.dumps(getReferrer())
    print(data)
```

```
kinesis.put_record(  
    StreamName="ExampleInputStream",  
    Data=data,  
    PartitionKey="partitionkey")
```

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

```
$ python stock.py
```

## Baixar e examinar o código Java Apache Flink Streaming

O código de aplicativo Java destas amostras está disponível no 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:

- O aplicativo usa uma origem do Kinesis para ler do fluxo de origem. O trecho a seguir cria o destino 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 como instalar o Apache Maven e o Java Development Kit (JDK), consulte [Pré-requisitos para concluir os exercícios \(p. 54\)](#).

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 Kinesis Data Analytics for Java Applications.
- Um método `main` que contém a lógica do aplicativo.

### Note

Para usar o conector do Kinesis para o aplicativo a seguir, você precisa fazer download do código-fonte do conector e compilá-lo como 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 seu código de aplicativo, compile-o e empacote-o em um arquivo JAR. Há duas formas de compilar e empacotar o código:
- Use a ferramenta de linha de comando do Maven. Crie seu arquivo JAR executando o seguinte comando no diretório que contém o arquivo `pom.xml`:

```
mvn package
```

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

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

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

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

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

## Fazer upload do código Java Apache Flink Streaming

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

Como fazer upload do código do aplicativo

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

O código do aplicativo agora está armazenado em um bucket do Amazon S3 em que o aplicativo pode acessá-lo.

## Criar e executar o aplicativo do Kinesis Data Analytics

Você pode criar e executar um Kinesis Data Analytics for Java application usando o console ou a AWS CLI.

### Note

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

### Tópicos

- [Criar e executar o aplicativo \(console\)](#) (p. 63)
- [Criar e executar o aplicativo \(AWS CLI\)](#) (p. 68)

## Criar e executar 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 Create analytics application (Criar aplicativo de análise).
3. Na página Kinesis Analytics - Create application (Kinesis Analytics – Criar aplicativo), forneça os detalhes do aplicativo da seguinte forma:
  - Em Application name (Nome do aplicativo), insira **MyApplication**.
  - Em Description (Descrição), insira **My java test app**.
  - Em Runtime (Tempo de execução), escolha Apache Flink 1.6.
4. Em Access permissions (Permissões de acesso), escolha Create / update IAM role **kinesis-analytics-MyApplication-us-west-2** (Criar/atualizar função do IAM de **kinesis-analytics-MyApplication-us-west-2**).

### Kinesis Analytics - Create application

Kinesis Analytics applications continuously read and analyze data from a connected streaming source in real-time. To enable interactivity with your data during configuration you will be prompted to run your application. Kinesis Analytics resources are not covered under the [AWS Free Tier](#), and **usage-based charges** apply. For more information, see [Kinesis Analytics pricing](#).

Application name\* MyApplication

Description My java test app

Runtime ☐ SQL ☒ Apache Flink 1.6

Access permissions

Create or choose IAM role with the required permissions. Kinesis Analytics will assume this role when you later execute your code. [Learn more](#)

Access permissions\* ☒ Create / update IAM role **kinesis-analytics-MyApplication-us-west-2** ☐ Choose from IAM roles that Kinesis Analytics can assume

\* Required Cancel Create application

5. Selecione Criar aplicativo.

## Note

Quando cria um Kinesis Data Analytics for Java application usando o console, você tem a opção de criar uma função e política do IAM para o seu aplicativo. O aplicativo usa essa função e política para acessar os recursos dependentes. Esses recursos do IAM são nomeados com 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**

## Editar a política do IAM

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

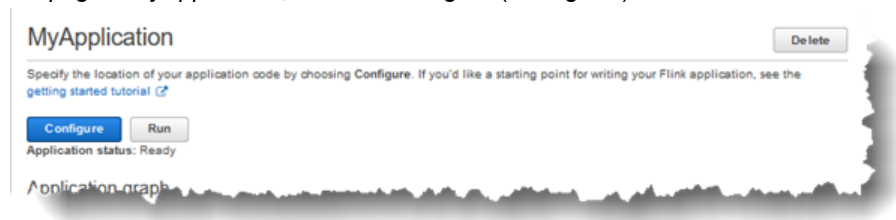
1. Abra o console do IAM em <https://console.aws.amazon.com/iam/>.
2. Escolha Policies (Políticas). Escolha a política **kinesis-analytics-service-MyApplication-us-west-2** que o console criou para você na seção anterior.
3. Na página Summary (Resumo), escolha Edit policy (Editar política). Selecione a guia JSON.
4. Adicione a seção destacada do exemplo de política a seguir à política. Substitua os exemplos de IDs de conta (**012345678901**) pelo ID da 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 página MyApplication, escolha Configure (Configurar).



2. Na página Configure application (Configurar aplicativo), forneça o Code location (Local do código):
  - Em Amazon S3 bucket (Bucket do Amazon S3), insira **ka-app-code-*<username>***.
  - Em Path to Amazon S3 object (Caminho até o objeto do Amazon S3), insira **java-getting-started-1.0.jar**.
3. Em Access to application resources (Acesso aos recursos do aplicativo), Access permissions (Permissões de acesso), escolha Create / update IAM role **kinesis-analytics-MyApplication-us-west-2** (Criar/atualizar função do IAM de **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
<b>flink.inputstream.initpos</b>	<b>LATEST</b>
<b>aws:region</b>	<b>us-west-2</b>
<b>AggregationEnabled</b>	<b>false</b>

6. Em Monitoring (Monitoramento), confirme se Monitoring metrics level (Nível de monitoramento de métricas) está definido como Application (Aplicativo).
7. Em CloudWatch logging (Registro em log do CloudWatch), marque a caixa de seleção Enable (Habilitar).
8. Escolha Update.



Configure application

In this section, you specify your application's code location, IAM role, application properties, snapshots, and monitoring.

**Code location**

Choose the S3 bucket that contains your application's JAR file. Your application code is reloaded from the S3 bucket every time you update your application using this page. To update other application settings without reloading the application code, use the AWS CLI. [Learn more](#)

Amazon S3 bucket: ks-app-code-yourlog Create new

View: [ks-app-code-yourlog](#) in Amazon S3

Path to Amazon S3 object: java-getting-started-1.0.jar

Specify the path and object name of the JAR file containing your application code. [View specified object in Amazon S3](#)

**Access to application resources**

Access permissions: ☒ Create / update IAM role `kinesis-analytics-MyApplication-us-east-2`  
☐ Choose from IAM roles that Kinesis Analytics can assume

► Properties  
► Snapshots  
▼ Monitoring

Kinesis Analytics can log record delivery errors to CloudWatch logs. If enabled, a CloudWatch log group and corresponding log stream are created on your behalf. [Learn more](#)

Standard rules apply for using CloudWatch logs. For more information, see [CloudWatch pricing](#).

**Monitoring metrics level** ☒ Application  
Includes job-level metrics such as uptime, downtime, checkpoints, and MillsBehindLatest.

☐ Task  
Flink Task-level metrics such as numRecordsInPerSecond for the application.

☐ Operator  
Operator-level metrics such as numRecordsInPerSecond for an operator.

☐ Parallelism  
Task and operator level metrics at the thread level.

**CloudWatch logging** ☒ Enable  
Kinesis Analytics can log record delivery errors to CloudWatch logs. If enabled, a CloudWatch log group and corresponding log stream are created on your behalf.

**Monitoring log level**

The Flink-Log4j log level. For more information, see [how to use logging](#).

**Monitoring log level** ☒ Error  
Includes error.

☐ Warn  
Includes error and warn.

☐ Info  
Includes error, warn, and info.

☐ Debug  
Includes error, warn, and info.

\* Required Cancel Update

## Note

Quando você opta por habilitar o registro em log do CloudWatch, o Kinesis Data Analytics cria um grupo de logs e um fluxo de logs para você. Os nomes desses recursos são os seguintes:

- Grupo de logs: `/aws/kinesis-analytics/MyApplication`
- Fluxo de logs: `kinesis-analytics-log-stream`

## Executar o aplicativo

1. Na página `MyApplication`, escolha `Run` (Executar). Confirme a ação.

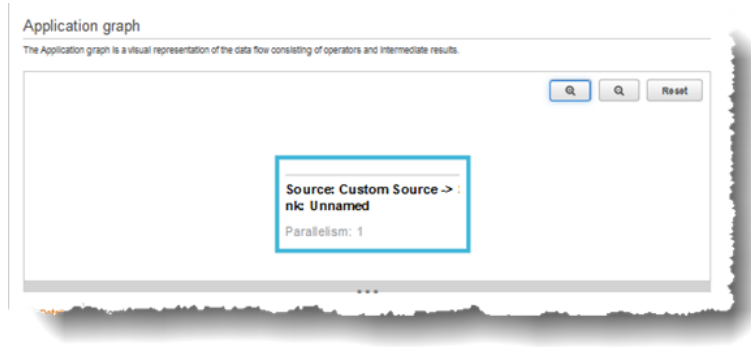
MyApplication Delete

Specify the location of your application code by choosing [Configure](#). If you'd like a starting point for writing your Flink application, see the [getting started tutorial](#)

Configure Run

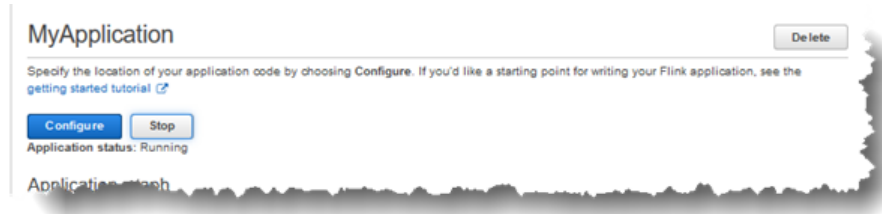
Application status: Ready

2. Quando o aplicativo estiver em execução, atualize a página. O console mostra o Application graph (Gráfico do aplicativo).



### Interromper o aplicativo

Na página MyApplication, escolha Stop (Interromper). Confirme a ação.



### Atualizar o aplicativo

Usando o console, você pode atualizar configurações do aplicativo, como as propriedades do aplicativo, as configurações de monitoramento e a localização ou o nome do arquivo JAR do aplicativo. Você também pode recarregar o JAR do aplicativo do bucket do Amazon S3 se precisar atualizar o código do aplicativo.

Na página MyApplication, escolha Configure (Configurar). Atualize as configurações do aplicativo e escolha Update (Atualizar).

## Criar e executar o aplicativo (AWS CLI)

Nesta seção, você usa a AWS CLI para criar e executar o aplicativo Kinesis Data Analytics. O Kinesis Data Analytics for Java Applications usa o comando da AWS CLI `kinesisanalyticsv2` para criar e interagir com aplicativos do Kinesis Data Analytics.

### 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, anexe a política a uma função do IAM (que você cria na próxima seção). Assim, quando Kinesis Data Analytics assumir a função, o serviço terá as permissões necessárias para ler a partir do fluxo de origem e gravar no fluxo de destino.

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 para armazenar o código do aplicativo. Substitua o ID da conta nos Nomes de recurso da Amazon (ARNs) (`012345678901`) pelo ID da conta.

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "Action": "kinesis:Read",  
      "Resource": "arn:aws:kinesis:us-east-1:012345678901:stream/SourceStream",  
      "Condition": {"ArnEquals": {"aws:PrincipalArn": "arn:aws:iam::012345678901:user:username"}}  
    },  
    {  
      "Effect": "Allow",  
      "Action": "kinesis:Write",  
      "Resource": "arn:aws:kinesis:us-east-1:012345678901:stream/DestinationStream",  
      "Condition": {"ArnEquals": {"aws:PrincipalArn": "arn:aws:iam::012345678901:user:username"}}  
    }  
  ]  
}
```

```
"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 obter instruções passo a passo para criar uma política de permissões, consulte [Tutorial: crie e anexe a sua primeira política gerenciada pelo cliente](#) no Guia do usuário do IAM.

#### Note

Para acessar outros serviços da AWS, use o AWS SDK for Java. O Kinesis Data Analytics define automaticamente as credenciais exigidas pelo SDK para usar as da função do IAM de execução de serviço que está associada ao seu aplicativo. Não é necessária nenhuma etapa adicional.

### Criar uma função do IAM

Nesta seção, você cria uma função do IAM que o Kinesis Data Analytics for Java application pode assumir para ler um fluxo de origem e gravar no fluxo de destino.

O Kinesis Data Analytics não pode acessar seu fluxo sem permissões. Conceda essas permissões por meio de uma função do IAM. Cada função do IAM tem duas políticas anexadas. A política de confiança concede ao Kinesis Data Analytics permissão para assumir a função e a política de permissões determina o que o Kinesis Data Analytics 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 console do IAM em <https://console.aws.amazon.com/iam/>.
2. No painel de navegação, escolha Roles (Funções) e Create Role (Criar função).
3. Em Select type of trusted identity (Selecionar tipo de identidade confiável), escolha AWS service (Serviço da AWS). Em Choose the service that will use this role (Escolher o serviço que usará esta função), escolha Kinesis. Em Select your use case (Selecionar seu caso de uso), escolha Kinesis Analytics.

Escolha Próximo: Permissões.

4. Na página Attach permissions policies, escolha 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** em Role name (Nome da função). Selecione Create role.

Agora você criou uma função do IAM 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 Kinesis Data Analytics assume essa função 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” \(p. 68\)](#).

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

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

Para obter instruções passo a passo sobre como criar uma função, consulte [Criar uma função do IAM \(console\)](#) no Guia do usuário do IAM.

### Criar o aplicativo do Kinesis Data Analytics

1. Salve o seguinte código JSON em um arquivo chamado `create_request.json`. Substitua o ARN da função de amostra pelo ARN da função que você criou anteriormente. Substitua o sufixo do ARN 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 kinesisanalyticstv2 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 seguinte código JSON 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 kinesisanalyticstv2 start-application --cli-input-json file://start_request.json
```

O aplicativo agora está em execução. Você pode verificar as métricas do Kinesis Data Analytics no console do Amazon CloudWatch 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 seguinte código JSON em um arquivo chamado `stop_request.json`.

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

2. Execute a ação [StopApplication](#) com a seguinte solicitação para interromper o aplicativo:

```
aws kinesisanalyticstv2 stop-application --cli-input-json file://stop_request.json
```

O aplicativo agora está interrompido.

## Tutorial: Usar o 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 fluxo de dados do Kinesis. O AWS Lambda sonda esse fluxo de dados e, quando detecta novos registros de dados, invoca sua função do Lambda. O AWS Lambda executa a função do Lambda assumindo a função de execução especificada quando você criou a função do Lambda.

Para obter instruções detalhadas passo a passo, consulte [Tutorial: Usar o AWS Lambda com o Amazon Kinesis](#).

### Note

Este tutorial presume que você tenha algum conhecimento de operações básicas do Lambda e do console do Lambda. Caso ainda não tenha feito isso, siga as instruções em [Conceitos básicos do AWS Lambda](#) para criar sua primeira função do Lambda.

## AWS Solução de dados de transmissão para Amazon Kinesis

O AWS Solução de dados de transmissão para Amazon Kinesis configura automaticamente o AWS serviços necessários para capturar, armazenar, processar e fornecer dados de streaming facilmente. A solução fornece várias opções para resolver casos de utilização de dados de transmissão que utilizam vários AWS incluindo Kinesis Fluxos de dados, AWS Lambda, Amazon API Gateway, e Amazon Kinesis Data Analytics.

Cada solução inclui os seguintes componentes:

- Um AWS CloudFormation pacote para implementar o exemplo completo.
- Um CloudWatch painel para visualizar as métricas da aplicação.
- CloudWatch alarmes nas métricas de aplicação mais relevantes.
- Todos necessários IAM funções e políticas.

A solução pode ser encontrada aqui: [Solução de dados de transmissão para Amazon Kinesis](#)

# Criar e gerenciar streamings

Estes exemplos discutem a [API do Amazon Kinesis Data Streams](#) e usam o [AWS SDK para Java](#) para criar, excluir e trabalhar com um Kinesis data stream.

O código Java de exemplo neste capítulo demonstra como executar operações básicas da API do Kinesis Data Streams e se divide 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 a [API do Kinesis Data Streams](#) usando outras linguagens de programação. Para obter mais informações sobre todos os SDKs da AWS disponíveis, consulte [Comece a desenvolver usando a Amazon Web Services](#).

## Tópicos

- [Criar um stream \(p. 73\)](#)
- [Listar streams \(p. 74\)](#)
- [Listagem de estilhaços \(p. 75\)](#)
- [Excluir um stream \(p. 78\)](#)
- [Reestilhar um stream \(p. 78\)](#)
- [Alterar o período de retenção de dados \(p. 83\)](#)
- [Atribuir tags nos streams no Amazon Kinesis Data Streams \(p. 84\)](#)
- [Gerenciar streaming de dados do Kinesis usando o console \(p. 87\)](#)

## Criar um stream

Use as etapas a seguir para criar seu Kinesis data stream.

### Criar o cliente do Kinesis Data Streams

Para trabalhar com o Kinesis data streams, 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 [Regiões e endpoints do Kinesis Data Streams](#) no AWS General Reference.

### Criar o stream

Agora que criou seu cliente do Kinesis Data Streams, você pode criar um streaming com o qual trabalhar. Isso pode ser feito com o console do Kinesis Data Streams ou de forma programática. Para criar um stream de forma programática, instancie um objeto `CreateStreamRequest`, depois especifique um nome para o stream e o número de estilhaços para o stream usar.

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

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

A taxa de transferência do stream depende do número de estilhaços: mais estilhaços são necessários para uma maior taxa de transferência provisionada. Mais estilhaços também aumentam o custo que a AWS cobra pelo stream. Para obter mais informações sobre como calcular um número apropriado de estilhaços para o aplicativo, consulte [Como determinar o tamanho inicial de um Kinesis Data Stream \(p. 3\)](#).

Depois que o objeto `createStreamRequest` é configurado, crie um stream chamando o método `createStream` para o 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" );
}
```

## Listar streams

Conforme descrito na seção anterior, os streams são delimitados pela conta da AWS associada às credenciais da AWS usadas para criar a instância do cliente do Kinesis Data Streams e também pela região especificada para o cliente. Uma conta da AWS pode ter muitos streams ativos ao mesmo tempo.



Você pode listar seus streams no console do Kinesis Data Streams ou de forma programática. O código desta seção mostra como listar todos os streams para sua conta da AWS.

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

Este exemplo de código cria primeiro uma nova instância de `ListStreamsRequest` e liga para `setLimit` para especificar que deve ser devolvido um máximo de 20 fluxos para cada chamada para `listStreams`. Se não especificar um valor para `setLimit`, Kinesis Data Streams devolve um número de fluxos iguais ou inferiores 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 streams do que o limite especificado, mesmo se houver mais streams do que isso na conta e 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());
}
```

Este código chama para o `getHasMoreStreams` método em `listStreamsRequest` para verificar se existem fluxos adicionais disponíveis para além dos devolvidos na chamada inicial para `listStreams`. Se sim, o código chama para o `setExclusiveStartStreamName` com o nome do último fluxo que foi devolvido na chamada anterior para `listStreams`. O `setExclusiveStartStreamName` causa a próxima chamada para `listStreams` começar após esse fluxo. 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, [Criar um stream \(p. 73\)](#).

## Listagem de estilhaços

Um fluxo de dados pode ter um ou mais caracteres. Existem dois métodos para listar (ou recuperar) derivações de um fluxo de dados.

Tópicos

- [API de teclado - Recomendado \(p. 76\)](#)

- [cvbestream API - Preterido \(p. 77\)](#)

## API de teclado - Recomendado

O método recomendado para listar ou recuperar as derivações de um fluxo de dados é utilizar o [listas](#) API. O exemplo seguinte mostra como pode obter uma lista das derivações num 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 arquivo POM, 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, pode utilizar o [filtro](#) parâmetro para filtrar a resposta da API. Só pode especificar um filtro de cada vez.

Se utilizar o `ShardFilter` parâmetro ao invocar a API de lista, a `Type` é a propriedade necessária e tem de ser especificada. Se especificar o `AT_TRIM_HORIZON`, `FROM_TRIM_HORIZON`, ou `AT_LATEST` não é necessário especificar o `ShardId` ou o `Timestamp` propriedades opcionais.

Se especificar o `AFTER_SHARD_ID` introduza também o valor para o `ShardId` propriedade. O `ShardId` a propriedade é idêntica em termos de fucalidade para `ExclusiveStartShardId` parâmetro da API de lista de marcações. Quando `ShardId` a propriedade é especificada, a resposta inclui as derivações começando pelo compartimento cuja ID segue imediatamente `ShardId` que forneceu.

Se especificar o `AT_TIMESTAMP` ou `FROM_TIMESTAMP_ID` introduza também o valor para o `Timestamp` propriedade. Se especificar o `AT_TIMESTAMP` tipo, então todas as derivações que estavam abertas no carimbo de data/hora fornecida são devolvidas. Se especificar o `FROM_TIMESTAMP` e, em seguida, todas as derivações que começam desde o carimbo de data/hora fornecida à TIP são devolvidas.

## cvbestream API - Preterido

### Important

A informação abaixo descodifica uma forma actualmente preterida de recuperar derivações de um fluxo de dados através da API `describeStream`. É actualmente altamente recomendado que utilize o `ListShards` API para recuperar os painéis que compõem o fluxo de dados.

O objeto de resposta retornado pelo método `describeStream` permite que você recupere informações sobre os estilhaços que compõem o stream. Para recuperar os estilhaços, chame o método `getShards` para esse objeto. Esse método pode não retornar todos os estilhaços do stream em uma única chamada. No código a seguir, verificamos o método `getHasMoreShards` em `getStreamDescription` para ver se há outros estilhaços que não foram retornados. Se houver, ou seja, se esse método retornar `true`, continuaremos chamando `getShards` em um loop, adicionando cada novo lote de estilhaços retornados à nossa lista de estilhaços. O loop é encerrado quando `getHasMoreShards` retorna `false`, ou seja, todos os estilhaços tiverem sido retornados. Observe que `getShards` não retorna estilhaços que estão no estado `EXPIRED`. Para obter mais informações sobre estados de estilhaço, incluindo o estado `EXPIRED`, consulte [Roteamento de dados, persistência de dados e estado do estilhaço após um reestilhaçamento](#) (p. 82).

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

```
}  
} while ( exclusiveStartShardId != null );
```

## Excluir um stream

Você pode excluir um stream 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 subsequentemente você criar um novo streaming com o mesmo nome do streaming anterior e, se os aplicativos que estavam operando nele ainda estiverem em execução, esses aplicativos poderão tentar interagir com o streaming novo como se fosse o anterior — com resultados imprevisíveis.

## Reestilhar um stream

### Important

Você pode reestilhar o streaming usando a API [UpdateShardCount](#). Caso contrário, é possível continuar executando divisões e mesclagens, como explicado aqui.

O Amazon Kinesis Data Streams oferece suporte ao reestilhamento, o que permite ajustar o número de estilhaços no streaming para se adaptar às alterações na taxa do fluxo de dados pelo streaming. O reestilhamento é considerado uma operação avançada. Se não estiver familiarizado com o Kinesis Data Streams, volte a este assunto depois de se familiarizar com todos os outros aspectos do Kinesis Data Streams.

Há dois tipos de operações de reestilhamento: 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 reestilhamento 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 reestilhamento é chamado de estilhaço pai. O estilhaço (ou o par de estilhaços) resultante da operação de novo estilhamento 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. De forma semelhante, a mesclagem reduz o número de estilhaços no streaming e, portanto, diminui a capacidade de dados — e o custo — do streaming.

O reestilhamento costuma ser executado por um aplicativo administrativo, que é diferente dos aplicativos de produtor (put) e dos aplicativos de consumidor (get). Um aplicativo administrativo assim monitora o desempenho geral do streaming com base em métricas fornecidas pelo Amazon CloudWatch ou com base em métricas coletadas dos produtores e consumidores. O aplicativo administrativo também precisa de um conjunto de permissões do IAM maior, além de consumidores ou produtores, porque consumidores e produtores não costumam precisar acessar as APIs usadas para reestilhamento. Para obter mais informações sobre permissões do IAM para o Kinesis Data Streams, consulte [Controle do acesso aos recursos do Amazon Kinesis Data Streams usando o IAM \(p. 225\)](#)

## Tópicos

- [Estratégias para reestilhaçamento \(p. 79\)](#)
- [Dividir um estilhaço \(p. 79\)](#)
- [Mesclar dois estilhaços \(p. 80\)](#)
- [Após o reestilhaçamento \(p. 81\)](#)

## Estratégias para reestilhaçamento

A finalidade do reestilhaçamento no Amazon Kinesis Data Streams é permitir que o streaming se adapte às 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 reestilhaçamento pode ser dividir cada estilhaço do streaming — o que dobra a capacidade do streaming. No entanto, isso pode fornecer mais capacidade adicional do que você realmente precisa e, portanto, gerar um custo desnecessário.

Você também pode usar as métricas para determinar quais são os estilhaços "quentes" ou "frios", ou seja, os estilhaços que estavam 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 performance do streaming nas métricas do Amazon CloudWatch publicadas pelo Kinesis Data Streams. No entanto, você mesmo também pode coletar algumas métricas dos seus streams. Uma abordagem é registrar 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 Streams usa o [MD5](#) para calcular a chave de hash a partir da chave de partição. Como você especifica a chave de partição do registro, pode usar MD5 para calcular o valor de chave de hash desse registro e registrá-lo.

Você também pode registrar os IDs dos estilhaços aos quais são atribuídos seus registros de dados. 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 IDs de estilhaço e os valores de chave de hash, você pode determinar quais estilhaços e chaves de hash estão recebendo a maior ou menor parte do tráfego. Em seguida, você pode usar o reestilhaçamento para fornecer mais ou menos capacidade, conforme apropriado para essas chaves.

## Dividir um estilhaço

Para dividir um estilhaço no Amazon Kinesis Data Streams, você precisa especificar como valores de chave de hash dos estilhaços pais devem ser redistribuídos para os estilhaços filhos. 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 hash do [MD5](#) da chave de partição que você especifica para o registro de dados no momento em que adiciona o registro de dados ao streaming. 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 estilhaços filhos 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 [Aguardar um stream ficar ativo novamente \(p. 82\)](#).

## Mesclar dois estilhaços

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 estilhaços OPEN em um streaming — como um grupo — sempre abrange o intervalo inteiro de valores de chaves de hash do MD5. Para obter mais informações sobre estados de estilhaços —, como CLOSED — consulte [Roteamento de dados, persistência de dados e estado do estilhaço após um reestilhaçamento](#) (p. 82)

Para identificar os estilhaços candidatos para mesclagem, você deve filtrar todos os estilhaços que estão no estado CLOSED. Placas que são OPEN—ou seja, não CLOSED—tem um número de sequência final de null. Pode testar o número de sequência final para um rodapé utilizando:

```
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 [Aguardar um stream ficar ativo novamente](#) (p. 82).

## Após o reestilhaçamento

Após qualquer tipo de procedimento de reestilhaçamento no Amazon Kinesis Data Streams, e antes da conclusão do processamento normal de registro, outros procedimentos e considerações são necessários. As seções a seguir descrevem esses itens.

Tópicos

- [Aguardar um stream ficar ativo novamente](#) (p. 82)
- [Roteamento de dados, persistência de dados e estado do estilhaço após um reestilhaçamento](#) (p. 82)

## Aguardar um stream ficar ativo novamente

Depois de chamar uma operação de reestilhamento, `splitShard` ou `mergeShards`, você precisa esperar o stream ficar ativo novamente. O código a ser usado é o mesmo de quando você espera que um streaming se torne ativo após a [criação de um streaming \(p. 73\)](#). 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" );
}
```

## Roteamento de dados, persistência de dados e estado do estilhaço após um reestilhamento

O Kinesis Data Streams é um serviço de streaming de dados em tempo real, então, seus aplicativos devem pressupor que os dados estão fluindo continuamente pelos estilhaços 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 reestilhamento permanecem nesses estilhaços. Em outras palavras, os estilhaços pais não desaparecem quando o reestilhamento ocorre. Eles persistem com os dados que continham antes do reestilhamento. Os registros de dados nos estilhaços pai podem ser acessados usando as operações [getShardIterator](#) e [getRecords](#) (p. 157) na API do Kinesis Data 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 \(p. 83\)](#).



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

- **ABERTO:** Antes de um funcionamento resduro, um nó principal está no `OPEN` estado, o que significa que os registros de dados podem ser ambos adicionados ao rodapé e recuperados do compartimento.
- **FECHADO:** Após um funcionamento resduro, as principais transições de transferência de dados `CLOSED` estado. 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.
- **EXPIRADO:** Após o período de retenção do fluxo ter expirado, todos os registros de dados no cartão principal expiraram e já não estão 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](#) (p. 83).

Depois que o reestilhecimento ocorreu e o stream está novamente em um estado `ACTIVE`, você pode iniciar imediatamente a ler dados dos estilhaços filhos. No entanto, os estilhaços pais que permanecem após o reestilhecimento ainda podem conter dados que ainda não foram lidos e foram adicionados ao stream antes do reestilhecimento. 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 reestilhecimento, 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 está lendo dados usando Kinesis Client Library, a biblioteca garante que você receberá os dados em ordem, mesmo se ocorrer um reestilhecimento.

## Alterar o período de retenção de dados

O Amazon Kinesis Data Streams aceita alterações feitas no período de retenção de registros de dados do streaming. Kinesis data stream é uma sequência ordenada de registros de dados na qual gravar e da qual ler 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. Um Kinesis data stream armazena registros de 24 horas por padrão (o máximo é 168 horas).

Você pode aumentar o período de retenção até 168 horas usando a operação [IncreaseStreamRetentionPeriod](#). Você pode diminuir o período de retenção até um mínimo de 24 horas usando a operação [DecreaseStreamRetentionPeriod](#). A sintaxe de solicitação das duas operações inclui o nome do stream e o período de retenção em horas. Finalmente, você pode verificar o período de retenção atual de um stream chamando a operação [DescribeStream](#).

Ambas as operações são fáceis de usar. 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 os registros inacessíveis no período de retenção antigo vários minutos após aumentar o período de retenção. 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.

Quase imediatamente, o Kinesis Data Streams torna inacessíveis os registros mais antigos que o novo período de retenção após a diminuição do período de retenção. Portanto, deve-se tomar muito cuidado ao chamar a operação [DecreaseStreamRetentionPeriod](#).

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 operações da API de período de retenção permitem que você configure isso de forma proativa ou responda a eventos operacionais reativamente.

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 Streams](#).

## Atribuir tags nos streams no Amazon Kinesis Data Streams

Você pode atribuir seus próprios metadados aos streamings que cria no Amazon Kinesis Data Streams na forma de tags. Tag é um par de chave-valor que você define para um stream. O uso de marcas é uma forma simples, mas eficiente, de gerenciar os recursos da AWS e organizar os dados, incluindo dados de faturamento.

### Tópicos

- [Conceitos básicos de tags \(p. 84\)](#)
- [Monitoramento de custos com marcação \(p. 85\)](#)
- [Restrições de tag \(p. 85\)](#)
- [Atribuir tags usando o console do Kinesis Data Streams \(p. 85\)](#)
- [Atribuir tags a streams usando o AWS CLI \(p. 86\)](#)
- [Atribuir tags a stream usando a API do Kinesis Data Streams \(p. 86\)](#)

## Conceitos básicos de tags

Você usa o console do Kinesis Data Streams, a AWS CLI ou a API do Kinesis Data Streams para executar 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
- Aplicativo: nome do aplicativo

- Ambiente: produção

## Monitoramento de custos com marcação

Você pode usar marcas para categorizar e monitorar seus custos da AWS. Quando você aplica tags aos recursos da AWS, incluindo streams, o relatório de alocação de custos da AWS inclui o uso e os custos agrupados por tags. É possível aplicar tags que representem categorias de negócios (como centros de custos, nomes de aplicativos ou proprietários) para organizar seus custos de vários serviços. Para obter mais informações, consulte [Usar tags de alocação de custos para relatórios de faturamento personalizados](#) no Guia do usuário do AWS Billing and Cost Management.

## Restrições de tag

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 marca com `aws:` porque esse prefixo é reservado para uso pela AWS. A AWS cria marcas 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: `_ . / = + - @`.

## Atribuir tags 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 Stream Details (Detalhes do streaming), clique na 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 Stream Details (Detalhes do streaming), clique na guia Tags.
4. Especifique a chave da tag no campo Key (Chave). Opcionalmente, especifique um valor de tag no campo Value (Valor) e, em seguida, clique em Add Tag (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 obter mais informações, consulte [Restrições de tag](#) (p. 85).

5. Para visualizar a nova tag na lista da guia Tags, clique no 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 Stream Details (Detalhes do streaming), clique na guia Tags e, em seguida, clique no ícone Remove (Remover) da tag.
4. Na caixa de diálogo Delete Tag (Excluir tag), clique em Yes, Delete (Sim, excluir).

## Atribuir tags a streams 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.

## Atribuir tags a stream usando a API do Kinesis Data Streams

Você pode adicionar, listar e remover tags usando a API do Kinesis 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.

## Gerenciar streaming de dados do Kinesis usando o console

Os procedimentos a seguir mostram como criar, excluir e trabalhar com um streaming de dados do Amazon Kinesis usando o Console de gerenciamento da AWS.

### Para criar um fluxo

1. Faça login no Console de gerenciamento da AWS e abra o console do Kinesis em <https://console.aws.amazon.com/kinesis>.
2. Escolha Data Streams (Streamings de dados) na barra de navegação.
3. Escolha Create Kinesis stream (Criar streaming do Kinesis).
4. Digite um nome para o streaming (por exemplo, **StockTradeStream**).
5. Especifique o número de estilhaços. Se precisar de ajuda, expanda Estimate the number of shards you'll need (Estimar o número de estilhados que serão necessários).
6. Escolha Create Kinesis stream (Criar streaming do Kinesis).

### Para listar seus streams

1. Abra o console do Kinesis em <https://console.aws.amazon.com/kinesis>.
2. Escolha Data Streams (Streamings de dados) na barra de navegação.
3. (Opcional) Para visualizar mais detalhes de um stream, escolha o nome dele.

### Para editar um stream

1. Abra o console do Kinesis em <https://console.aws.amazon.com/kinesis>.
2. Escolha Data Streams (Streamings de dados) na barra de navegação.
3. Escolha o nome do stream.
4. Para escalar a capacidade do estilhaço, faça o seguinte:
  - a. Em Shards (Estilhaços), escolha Edit (Editar).
  - b. Especifique o novo número de estilhaços.
  - c. Escolha Salvar.
5. Para editar o período de retenção de dados, faça o seguinte:
  - a. Em Data retention period (Período de retenção de dados), escolha Edit (Editar).
  - b. Especifique um período entre 24 e 168 horas. Os registros são armazenados no stream durante esse período. Cobranças adicionais se aplicam a períodos maiores que 24 horas. Para obter mais informações, consulte [Definição de preço do Amazon Kinesis Data Streams](#).
  - c. Escolha Salvar.
6. Para habilitar ou desabilitar as métricas no nível do estilhaço, faça o seguinte:
  - a. Em Shard level metrics (Métricas em nível de estilhaços), escolha Edit (Editar).
  - b. Selecione as métricas a monitorar. Para obter mais informações, consulte [Métricas do nível de estilhado aprimoradas \(p. 194\)](#).

c. Escolha Salvar.

#### Para excluir seus streams

1. Abra o console do Kinesis em <https://console.aws.amazon.com/kinesis>.
2. Escolha Data Streams (Streamings de dados) na barra de navegação.
3. Marque a caixa de seleção ao lado dos streams a excluir.
4. Escolha Actions, Delete.
5. Quando a confirmação for solicitada, escolha Excluir.

# Gravar dados no Amazon Kinesis Data Streams

Um produtor é um aplicativo que grava dados no Amazon Kinesis Data Streams. Você pode criar produtores para o Kinesis Data Streams usando o AWS SDK for Java e a Kinesis Producer Library.

Se você não estiver familiarizado com o Kinesis Data Streams, comece familiarizando-se com os conceitos e a terminologia apresentados em [O que é o Amazon Kinesis Data Streams? \(p. 1\)](#) e [Conceitos básicos do Amazon Kinesis Data Streams \(p. 18\)](#).

## Tópicos

- [Desenvolver produtores usando a Kinesis Producer Library da Amazon \(p. 89\)](#)
- [Desenvolvimento de produtores usando a API do Amazon Kinesis Data Streams com o AWS SDK for Java \(p. 100\)](#)
- [Gravação no Amazon Kinesis Data Streams por meio do agente do Kinesis \(p. 105\)](#)
- [Solução de problemas de produtores do Amazon Kinesis Data Streams \(p. 113\)](#)
- [Tópicos avançados para produtores do Kinesis Data Streams \(p. 115\)](#)

## Desenvolver produtores usando a Kinesis Producer Library da Amazon

Um produtor do Amazon Kinesis Data Streams é um aplicativo que coloca registros de dados de usuário em um Kinesis data stream (também chamado de consumo de dados). A Kinesis Producer Library (KPL) simplifica o desenvolvimento de aplicativos de produtor permitindo que os desenvolvedores atinjam alta taxa de transferência de gravação para um Kinesis data stream.

Você pode monitorar a KPL com o Amazon CloudWatch. Para obter mais informações, consulte [Monitoramento da Kinesis Producer Library com o Amazon CloudWatch \(p. 212\)](#).

## Tópicos

- [Função da KPL \(p. 90\)](#)
- [Vantagens do uso da KPL \(p. 90\)](#)
- [Quando não usar a KPL \(p. 91\)](#)
- [Instalação do KPL \(p. 91\)](#)
- [Transição da Kinesis Producer Library para os certificados do Amazon Trust Services \(ATS\) \(p. 91\)](#)
- [Plataformas compatíveis com a KPL \(p. 92\)](#)
- [Conceitos principais da KPL \(p. 92\)](#)
- [Integração da KPL com código de produtor \(p. 94\)](#)
- [Gravação no Kinesis Data Stream usando a KPL \(p. 95\)](#)
- [Configurar a Kinesis Producer Library \(p. 97\)](#)
- [Desagregação de consumidor \(p. 97\)](#)

- [Uso da KPL com o Kinesis Data Firehose \(p. 100\)](#)

## Função da KPL

A KPL é uma biblioteca fácil de usar e altamente configurável que ajuda você a gravar em um Kinesis data stream. Ela atua como um intermediário entre o código do seu aplicativo de produtor e as ações da API do Kinesis Data Streams. A KPL executa as seguintes tarefas principais:

- Grava em um ou mais Kinesis data streams com um mecanismo de retentativa automático e configurável
- 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 taxa de transferência
- Integra-se perfeitamente à [Kinesis Client Library \(KCL\)](#) para desagregar registros em lote no consumidor
- Envia métricas do Amazon CloudWatch em seu nome para proporcionar visibilidade sobre o desempenho do produtor

Observe que a KPL é diferente da API do Kinesis Data Streams que está disponível no [AWS SDKs](#). A API do Kinesis Data Streams ajuda você a gerenciar muitos aspectos do Kinesis Data Streams (incluindo a criação de streamings, reestilhamento e colocação e obtenção de registros), enquanto que a KPL fornece uma camada de abstração especificamente para o consumo de dados. Para obter informações sobre a API do Kinesis Data Streams, consulte o [Amazon Kinesis API Reference](#).

## Vantagens do uso da KPL

A lista a seguir representa algumas vantagens principais do uso da KPL para desenvolver produtores do Kinesis Data Streams.

A KPL pode ser usada 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 [Gravação no Kinesis Data Stream usando a KPL \(p. 95\)](#).

### Benefícios de desempenho

A KPL pode ajudar a criar produtores de alto desempenho. Considere uma situação em que suas instâncias do Amazon EC2 atuam como proxy para coletar eventos de 100 bytes de centenas ou milhares de dispositivos de baixa potência e gravar registros em um Kinesis data stream. Cada instância do EC2 precisa gravar milhares de eventos por segundo no seu streaming de dados. Para alcançar a taxa de transferência 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. A KPL executa todas essas tarefas para você.

### Facilidade de uso do lado do consumidor

Para desenvolvedores do lado do consumidor usando a KCL no Java, a KPL integra-se sem esforço adicional. Quando a KCL recupera um registro agregado do Kinesis Data Streams que consiste em vários registros de usuário da KPL, ela chama automaticamente a KPL para extrair os registros individuais do usuário antes de retorná-los ao usuário.

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

### Monitoramento de produtor

Você pode coletar, monitorar e analisar seus produtores do Kinesis Data Streams usando o Amazon CloudWatch e a KPL. A KPL emite métricas de taxa de transferência, de erro e outras para o



CloudWatch em seu nome e pode ser configurada para monitorar no nível de streaming, estilhaço ou produtor.

#### Arquitetura assíncrona

Como a KPL pode armazenar os registros em buffer antes de enviá-los ao Kinesis Data Streams, ela não força o aplicativo chamador a bloquear e aguardar uma confirmação de que o registro chegou no servidor antes de continuar a execução. Uma chamada para colocar um registro na KPL sempre retorna imediatamente, não espera o registro ser enviado ou uma resposta ser recebida do servidor. Em vez disso, é criado um objeto `Future` que recebe o resultado do envio do registro ao Kinesis Data Streams em um momento posterior. É o mesmo comportamento dos clientes assíncronos no AWS SDK.

## Quando não usar a KPL

A KPL pode incorrer em atraso por processamento adicional 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 usar o AWS SDK diretamente. Para obter mais informações sobre o uso do SDK da AWS com o Kinesis Data Streams, consulte [Desenvolvimento de produtores usando a API do Amazon Kinesis Data Streams com o AWS SDK for Java \(p. 100\)](#) Para obter mais informações sobre o `RecordMaxBufferedTime` e outras propriedades configuráveis pelo usuário da KPL, consulte [Configurar a Kinesis Producer Library \(p. 97\)](#)

## Instalação do KPL

A Amazon fornece binários pré-compilados da Kinesis Producer Library (KPL) de C++ para macOS, Windows e distribuições recentes do Linux (para obter detalhes das plataformas com suporte, 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 da KPL e da KCL, use os seguintes links de pesquisa do Maven:

- [KPL](#)
- [KCL](#)

O binários do Linux foram compilados com GNU Compiler Collection (GCC) e vinculados estaticamente à `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 anteriores do Linux podem compilar a KPL usando as instruções de compilação fornecidas junto com o código-fonte no GitHub. Para fazer download da KPL do GitHub, consulte a [Kinesis Producer Library](#).

## Transição da Kinesis Producer Library para os certificados do Amazon Trust Services (ATS)

Em 9 de fevereiro de 2018, às 9:00, horário do Pacífico, o Amazon Kinesis Data Streams instalou os certificados ATS. Para poder continuar a gravar registros no Kinesis Data Streams usando a Kinesis Producer Library (KPL), é necessário atualizar a instalação da KPL para a [versão 0.12.6](#) ou posterior. Essa alteração afeta todas as regiões da AWS.

Para obter informações sobre a mudança para ATS, consulte [Como se preparar para a mudança da AWS para a sua própria autoridade de certificação](#).

Se você encontrar problemas e precisar de suporte técnico, [crie um caso](#) no Centro de atendimento da AWS.

## Plataformas compatíveis com a KPL

A Kinesis Producer Library (KPL) é escrita em C++ e executada como um processo filho do processo principal do usuário. 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 posterior é compatível com todas as versões do KPL até a versão 0.14.0.

A plataforma Windows NÃO é compatível a partir da versão KPL 0.14.0 ou superior.

Observe que a KPL é de 64 bits apenas.

## Código-fonte

Se os binários fornecidos na instalação da KPL não forem suficientes para seu ambiente, o núcleo da KPL será escrito como um módulo C++. O código-fonte do módulo C++ e a interface do Java são lançados sob a Amazon Public License e disponibilizados no GitHub na [Kinesis Producer Library](#). Embora seja possível usar a KPL em qualquer plataforma para a qual haja disponibilidade de JRE e um compilador C++ compatível com os padrões recentes, a Amazon não aceita oficialmente qualquer plataforma que não esteja na lista de plataformas compatíveis.

## Conceitos principais da KPL

As seções a seguir contêm os conceitos e a terminologia necessários para entender e tirar proveito da Kinesis Producer Library (KPL).

### Tópicos

- [Registros \(p. 92\)](#)
- [Agrupamento em lotes \(p. 93\)](#)
- [Agregação \(p. 93\)](#)
- [Coleta \(p. 93\)](#)

## Registros

Este guia faz distinção entre registros de usuário da KPL e registros do Kinesis Data Streams. Ao usar o termo registro sem um qualificador, fazemos referência a um registro de usuário da KPL. Quando fazemos referência a um registro do Kinesis Data Streams, mencionamos explicitamente um registro do Kinesis Data Streams.

Um registro de usuário da KPL é um blob de dados que tem significado específico para o usuário. Os exemplos incluem um blob JSON que representa um evento de interface do usuário em um site ou uma entrada de log proveniente de um servidor da web.

Um registro do Kinesis Data Streams é uma instância da estrutura de dados `Record` definida pela API de serviço do Kinesis Data Streams. 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.

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

A KPL aceita dois tipos de agrupamento em lotes:

- Agregação – armazenamento de vários registros em um único registro do Kinesis Data Streams.
- Coleta – uso da operação `PutRecords` da API para enviar vários registros do Kinesis Data Streams a um ou mais estilhaços em seu Kinesis data stream.

Os dois tipos de agrupamento em lotes da KPL são projetados para coexistir e podem ser ativados ou desativados de forma independente. Por padrão, ambos estão ativados.

## Agregação

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 chamada de API, o que aumenta efetivamente a taxa de transferência do produtor.

Os estilhaços do Kinesis Data Streams oferecem suporte a até 1.000 registros do Kinesis Data Streams por segundo, ou uma taxa de transferência de 1 MB. O limite de registros do Kinesis Data Streams por segundo vincula os clientes aos 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 taxa de transferência 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 a agregação da KPL é possível empacotar 1.000 registros em apenas 10 registros do Kinesis Data Streams reduzindo o RPS para 10 (a 50 KB cada).

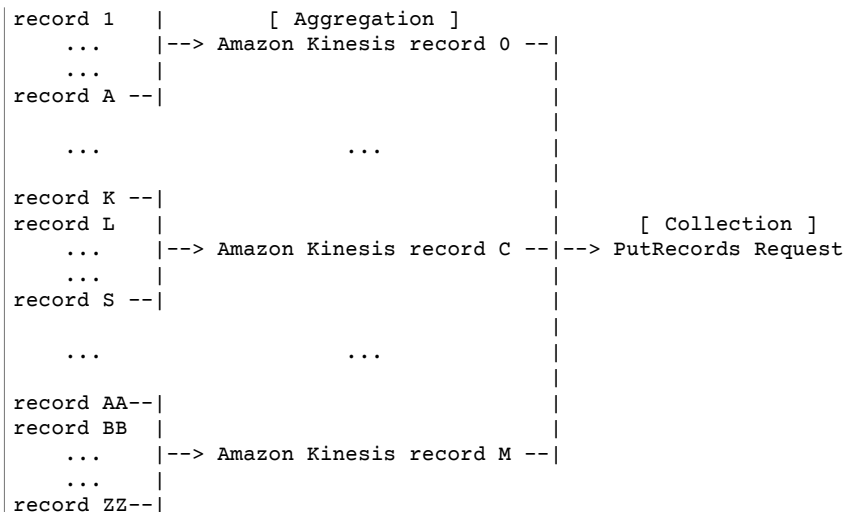
## Coleta

Coleta refere-se ao envio em lotes de vários registros do Kinesis Data Streams e ao envio dos mesmos em uma única solicitação HTTP com uma chamada à operação `PutRecords` da API, em vez de enviar cada registro do Kinesis Data Streams em uma solicitação HTTP própria.

Isso aumenta a taxa de transferência em comparação com o uso de nenhuma coleção, pois reduz a sobrecarga de fazer muitas solicitações HTTP separadas. Na verdade, `PutRecords`, por si só, foi projetado especificamente para essa finalidade.

A coleta difere da agregação porque trabalha 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 --|
```



## Integração da KPL com código de produtor

A Kinesis Producer Library (KPL) é executada em um processo separado e se comunica com o processo de usuário pai usando IPC. Essa arquitetura, às vezes chamada de [microserviço](#), é escolhida por dois motivos principais:

### 1) O processo do usuário não falhará mesmo que a KPL falhe

Seu processo pode ter tarefas não relacionadas ao Kinesis Data Streams e continuar a operação mesmo que a KPL falhe. Também é possível ao processo de usuário pai reiniciar a KPL e recuperar um estado totalmente funcional (essa funcionalidade está nos wrappers oficiais).

Um exemplo é um servidor da web que envia métricas ao Kinesis Data Streams. O servidor pode continuar atendendo páginas mesmo que a parte do Kinesis Data Streams tenha parado de funcionar. Portanto, falhar todo o servidor devido a um erro na KPL causaria 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 usar a KPL facilmente.

## Matriz de uso recomendado

A matriz de uso a seguir enumera as configurações recomendadas para diferentes usuários e aconselha se e como você deve usar a 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	Versão da KCL	Lógica do ponto de verificação	Você consegue ver a KPL?	Advertências
Tudo menos Java	*	*	*	Não	N/D
Java	Java	Usa o Java SDK diretamente	N/D	Sim	Se a agregação for usada, você precisará usar

Linguagem do lado do produtor	Linguagem do lado do consumidor	Versão da KCL	Lógica do ponto de verificação	Você consegue ver a KPL?	Advertências
					a biblioteca de desagregação fornecida após as chamadas a GetRecords.
Java	Tudo menos Java	Usa o 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 verificação sem argumento algum	Sim	Nenhum
Java	Java	1.4.x	Chama o ponto de verificação com um número sequencial explícito	Sim	Desative a agregação ou altere o 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.

## Gravação no Kinesis Data Stream usando a KPL

As seções a seguir mostram o código de exemplo em uma progressão desde o produtor de "barebones" mais simples possível até o código totalmente assíncrono.

### Código de produtor de barebones

Só é preciso o código a seguir para gravar um produtor de trabalho mínimo. Os registros de 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 ...
```

## Responder aos resultados de forma síncrona

No exemplo anterior, o código não verificou se os registros do usuário da KPL foram bem-sucedidos. A KPL executa todas as novas tentativas necessárias para dar conta das 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
        }
    }
}
}
```

## Responder 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 os parâmetros usando os 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 dos parâmetros de configuração e limites de valor, consulte a [arquivo de propriedades de configuração de amostra 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.

## Desagregação de consumidor

A partir da versão 1.4.0, a KCL oferece suporte à desagregação automática dos registros de usuário da KPL. O código do aplicativo do consumidor escrito com as versões anteriores da KCL será compilado sem qualquer modificação depois que você atualizar a KCL. No entanto, caso a agregação da KPL esteja sendo usada no lado do produtor, há uma sutileza envolvendo definição de pontos de verificação: todos os sub-registros dentro de um registro agregado têm o mesmo número sequencial. Então, os dados adicionais precisarão ser armazenados com o ponto de verificação se for preciso distinguir os sub-registros. Esses dados adicionais são chamados de números de subsequência.

## Migração das versões anteriores da KCL

Não é necessário alterar as chamadas existentes para que definam pontos de verificação em conjunto com a agregação. Ainda é certo que você pode recuperar com êxito todos os registros armazenados no Kinesis

Data Streams. A KCL agora oferece duas novas operações de ponto de verificação para aceitar casos de uso específicos, descritos abaixo.

Caso seu código existente tenha sido escrito para uma versão da KCL anterior ao suporte à KPL, e a operação de ponto de verificação seja chamada sem argumentos, isso será equivalente a definir o ponto de verificação do número de sequência do último registro de usuário da KPL 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 ponto de verificação `checkpoint()` da KCL sem argumentos é semanticamente equivalente a definir o ponto de verificação do número de sequência da última chamada a `Record` no lote, junto com o número de subsequência 0 (zero) implícito.

Chamar a nova operação de ponto de verificação `checkpoint(Record record)` da KCL é semanticamente equivalente a definir o ponto de verificação do número de sequência de `Record` conhecido junto com o número de subsequência 0 (zero) implícito. 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 de ponto de verificação `checkpoint(String sequenceNumber, long subSequenceNumber)` da KCL explicitamente verifica 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 ponto de verificação do Amazon DynamoDB, a KCL 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.

## Extensões da KCL para a desagregação da KPL

Como discutido anteriormente, a desagregação da KPL pode envolver definição de ponto de verificação de subsequência. Para facilitar o uso da definição de ponto de verificação de subsequência, foi adicionada uma classe `UserRecord` à 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 agregados com exatamente um sub-registro.

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

```
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);
```

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

Durante o processamento de registros de usuário da KPL, a KCL grava o número de subsequência no Amazon DynamoDB como um campo extra para cada linha. As versões anteriores da KCL usavam `AFTER_SEQUENCE_NUMBER` para obter registros ao retomar pontos de verificação. Em vez disso, a KCL com suporte à KPL usa `AT_SEQUENCE_NUMBER`. 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.

## Usar GetRecords diretamente

Em vez de usar a KCL, você pode optar por chamar a operação da API `GetRecords` diretamente para recuperar os registros do Kinesis Data Streams. Para desempacotar esses registros recuperados nos registros de usuário originais da KPL, 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 a lista de registros conhecidos do Kinesis Data Streams em uma lista de registros de usuário da KPL. Qualquer registro de usuário da KPL cuja chave de hash explícita ou chave de partição estiver fora do intervalo de `startingHashKey` (inclusive) e de `endingHashKey` (inclusive) será descartado da lista de registros retornados.

## Uso da KPL com o Kinesis Data Firehose

Se você usa a Kinesis Producer Library (KPL) para gravar dados em um streaming de dados do Kinesis, é possível usar a agregação para combinar os registros que você grava no streaming de dados do Kinesis. Se você usar esse streaming de dados como origem para o fluxo de entrega do Kinesis Data Firehose, o Kinesis Data Firehose desagregará os registros antes de entregá-los ao destino. Se você configurar o fluxo de entrega para transformar os dados, o Kinesis Data Firehose desagregará os registros antes de entregá-los ao AWS Lambda. Para obter mais informações, consulte [Gravar no Kinesis Data Firehose usando o Kinesis Data Streams](#).

# Desenvolvimento de produtores usando a API do Amazon Kinesis Data Streams com o AWS SDK for Java

Você pode desenvolver produtores usando a API do Amazon Kinesis Data Streams com o AWS SDK para Java. Se você não estiver familiarizado com o Kinesis Data Streams, comece familiarizando-se com os conceitos e a terminologia apresentados em [O que é o Amazon Kinesis Data Streams? \(p. 1\)](#) e [Conceitos básicos do Amazon Kinesis Data Streams \(p. 18\)](#).

Esses exemplos discutem a [API do Kinesis Data Streams](#) e usam o [AWS SDK para Java](#) para adicionar (put) dados a um streaming. Contudo, na maioria dos casos de uso, é melhor usar a biblioteca KPL do Kinesis Data Streams. Para obter mais informações, consulte [Desenvolver produtores usando a Kinesis Producer Library da Amazon \(p. 89\)](#).

O código Java de exemplo neste capítulo demonstra como executar operações básicas da API do Kinesis Data Streams e está 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. Você também pode chamar a [API do Kinesis Data Streams](#) usando outras linguagens de programação. Para obter mais informações sobre todos os SDKs da AWS disponíveis, consulte [Comece a desenvolver usando 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 [Criar e gerenciar streamings \(p. 73\)](#).

## 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 ou altera dados de forma alguma. Cada registro também tem um número sequencial e uma chave de partição associados.

Há duas operações diferentes na API do Kinesis Data Streams que adicionam dados a um stream: [PutRecords](#) e [PutRecord](#). A operação `PutRecords` envia vários registros ao stream por solicitação HTTP e a operação singular `PutRecord` envia registros ao stream um por vez (uma solicitação HTTP separada é necessária para cada registro). Talvez convenha usar `PutRecords` para a maioria dos

aplicativos, pois ele atingirá uma taxa de transferência mais alta por produtor de dados. Para obter mais informações sobre cada uma dessas operações, consulte as subseções abaixo.

#### Tópicos

- [Adicionar vários registros com PutRecords \(p. 101\)](#)
- [Adicionar um único registro com PutRecord \(p. 103\)](#)

Sempre lembre-se de que, como o aplicativo de origem está adicionando dados ao stream usando a API do Kinesis Data Streams, provavelmente há um ou mais aplicativos de consumidor que estão processando dados fora do stream simultaneamente. Para obter informações sobre como os consumidores obtêm dados usando a API do Kinesis Data Streams, consulte [Como obter dados de um stream \(p. 157\)](#)

#### Important

[Alterar o período de retenção de dados \(p. 83\)](#)

## Adicionar 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 taxa de transferência mais alta ao enviar dados para seu Kinesis data stream. Cada solicitação de [PutRecords](#) pode dar suporte a até 500 registros. Cada registro na solicitação pode ser tão grande quanto 1 MB, até um limite de 5 MB para toda a solicitação, inclusive chaves de partição. Assim como a operação única [PutRecord](#) descrita abaixo, [PutRecords](#) usa números sequenciais 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 sequência é atribuído pelo Kinesis Data Streams após a chamada a `client.putRecords` para adicionar os registros de dados ao streaming. 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 taxa de transferência.

## Exemplo de PutRecords

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.

## Tratar 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 Sintaxe da solicitação PutRecords

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

### Example Sintaxe da resposta a PutRecords

```
{
  "FailedRecordCount": 1,
```

```
"Records": [
  {
    "SequenceNumber": "21269319989900637946712965403778482371",
    "ShardId": "shardId-000000000001"

  },
  {
    "ErrorCode": "ProvisionedThroughputExceededException",
    "ErrorMessage": "Rate exceeded for shard shardId-000000000001 in stream
exampleStreamName under account 111111111111."

  },
  {
    "SequenceNumber": "21269319989999637946712965403778482985",
    "ShardId": "shardId-000000000002"
  }
]
}
```

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 Manipulador de falhas `PutRecords`

```
PutRecordsRequest putRecordsRequest = new PutRecordsRequest();
putRecordsRequest.setStreamName(myStreamName);
List<PutRecordsRequestEntry> putRecordsRequestEntryList = new ArrayList<>();
for (int j = 0; j < 100; j++) {
    PutRecordsRequestEntry putRecordsRequestEntry = new PutRecordsRequestEntry();
    putRecordsRequestEntry.setData(ByteBuffer.wrap(String.valueOf(j).getBytes()));
    putRecordsRequestEntry.setPartitionKey(String.format("partitionKey-%d", j));
    putRecordsRequestEntryList.add(putRecordsRequestEntry);
}

putRecordsRequest.setRecords(putRecordsRequestEntryList);
PutRecordsResult putRecordsResult = amazonKinesisClient.putRecords(putRecordsRequest);

while (putRecordsResult.getFailedRecordCount() > 0) {
    final List<PutRecordsRequestEntry> failedRecordsList = new ArrayList<>();
    final List<PutRecordsResultEntry> putRecordsResultEntryList =
putRecordsResult.getRecords();
    for (int i = 0; i < putRecordsResultEntryList.size(); i++) {
        final PutRecordsRequestEntry putRecordRequestEntry =
putRecordsRequestEntryList.get(i);
        final PutRecordsResultEntry putRecordsResultEntry =
putRecordsResultEntryList.get(i);
        if (putRecordsResultEntry.getErrorCode() != null) {
            failedRecordsList.add(putRecordRequestEntry);
        }
    }
    putRecordsRequestEntryList = failedRecordsList;
    putRecordsRequest.setRecords(putRecordsRequestEntryList);
    putRecordsResult = amazonKinesisClient.putRecords(putRecordsRequest);
}
```

## Adicionar um único registro com `PutRecord`

Cada chamada para `PutRecord` opera em um único registro. Prefira a operação `PutRecords` descrita em [Adicionar vários registros com `PutRecords` \(p. 101\)](#), a menos que seu aplicativo precise

especificamente enviar sempre registros 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 após a chamada a `client.putRecord` para adicionar o registro de dados ao streaming. 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 sequenciais 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 [Exemplo de PutRecord \(p. 104\)](#).

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 a chave de partição (e dados associados) para um determinado estilhaço.

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.

## Exemplo de PutRecord

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.

## Gravação no Amazon Kinesis Data Streams por meio do agente do Kinesis

O agente do Kinesis é um aplicativo 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 métricas do Amazon CloudWatch para facilitar o monitoramento e a solução de problemas no 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 [Configurações do agente \(p. 107\)](#)).

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

- [Pré-requisitos \(p. 105\)](#)
- [Download e instalação do agente \(p. 106\)](#)
- [Configuração e inicialização do agente \(p. 106\)](#)
- [Configurações do agente \(p. 107\)](#)
- [Monitoramento de vários diretórios de arquivos e gravação em vários streams \(p. 109\)](#)
- [Uso do agente para pré-processar dados \(p. 110\)](#)
- [Comandos da CLI do agente \(p. 113\)](#)

## Pré-requisitos

- O sistema operacional deve ser a AMI do Amazon Linux com versão 2015.09 ou posterior ou o Red Hat Enterprise Linux versão 7 ou posterior.
- Se você está usando o Amazon EC2 para executar o agente, execute a instância do EC2.
- Gerencie suas credenciais da AWS usando um dos seguintes métodos:
  - Especifique uma função do IAM ao executar a instância do EC2.
  - Especifique as credenciais da AWS ao configurar o agente (consulte [awsAccessKeyId \(p. 106\)](#) e [awsSecretAccessKey \(p. 106\)](#)).
  - Edite `/etc/sysconfig/aws-kinesis-agent` para especificar sua região e as chaves de acesso da AWS.
  - Se a instância do EC2 estiver em outra conta da AWS, crie uma função do IAM para fornecer acesso ao serviço do Kinesis Data Streams e especifique essa função ao configurar o agente (consulte [assumeRoleARN \(p. 106\)](#) e [assumeRoleExternalId \(p. 106\)](#)). Use um dos métodos anteriores para especificar as credenciais da AWS de um usuário na outra conta que tem permissão para assumir essa função.
- A função do IAM ou as credenciais da AWS especificadas devem ter permissão para executar a operação do Kinesis Data Streams [PutRecords](#) para que o agente envie dados ao streaming. Se você



habilitar o monitoramento do CloudWatch para o agente, também será necessária permissão para realizar a operação do CloudWatch [PutMetricData](#). Para obter mais informações, consulte [Controle do acesso aos recursos do Amazon Kinesis Data Streams usando o IAM](#) (p. 225), [Monitoramento do Agente de integridade do Kinesis Data Streams com o Amazon CloudWatch](#) (p. 198) e [CloudWatch Access Control](#).

## Download e instalação do agente

Primeiro, conecte-se à instância. Para obter mais informações, consulte o tópico [Conecte-se à sua instância](#) no Guia do usuário do Amazon EC2 para instâncias do Linux. Se você tiver problemas para se conectar, consulte [Resolução de problemas para se conectar à sua instância](#) no Guia do usuário do Amazon EC2 para instâncias do Linux.

Para configurar o agente usando a AMI do Amazon Linux

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.amzn1.noarch.rpm
```

Para configurar o agente usando o GitHub

1. Faça download do agente em [awslabs/amazon-kinesis-agent](#).
2. Instale o agente navegando até o diretório de download e executando o comando a seguir:

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

## Configuração e inicialização do 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 a inicialização 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`.

## 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
<code>assumeRoleARN</code>	O ARN da função a ser assumida pelo usuário. Para obter mais informações, consulte <a href="#">Delegar acesso entre contas da AWS usando funções do IAM</a> no Guia do usuário do IAM.
<code>assumeRoleExternalId</code>	Um identificador opcional que determina quem pode assumir a função. Para obter mais informações, consulte <a href="#">Como usar um ID externo</a> no Guia do usuário do IAM.
<code>awsAccessKeyId</code>	O ID de chave de acesso da AWS que substitui as credenciais padrão. Essa configuração tem precedência sobre todos os outros provedores de credenciais.
<code>awsSecretAccessKey</code>	A chave secreta da AWS que substitui as credenciais padrão. Essa configuração tem precedência sobre todos os outros provedores de credenciais.
<code>cloudwatch.emitMetrics</code>	Permite que o agente emita métricas para o CloudWatch se definidas (true).  Padrão: true
<code>cloudwatch.endpoint</code>	O endpoint regional do CloudWatch.

Definição da configuração	Descrição
	Padrão: <code>monitoring.us-east-1.amazonaws.com</code>
<code>kinesis.endpoint</code>	O endpoint regional do Kinesis Data Streams.  Padrão: <code>kinesis.us-east-1.amazonaws.com</code>

Estas são as configurações de fluxo.

Definição da configuração	Descrição
<code>dataProcessingOptions</code>	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 obter mais informações, consulte <a href="#">Uso do agente para pré-processar dados (p. 110)</a> .
<code>kinesisStream</code>	[Obrigatório] O nome do stream.
<code>filePattern</code>	[Obrigatório] Um glob para os arquivos que precisam ser monitorados pelo agente. Qualquer arquivo que corresponda a esse padrão é selecionado pelo agente automaticamente e monitorado. Para todos os arquivos correspondentes a esse padrão, deve ser concedida uma permissão de leitura a <code>aws-kinesis-agent-user</code> . Para o diretório que contém os arquivos, devem ser concedidas permissões de leitura e execução a <code>aws-kinesis-agent-user</code> .
<code>initialPosition</code>	A posição em que o arquivo começou a ser analisado. Os valores válidos são <code>START_OF_FILE</code> e <code>END_OF_FILE</code> .  Padrão: <code>END_OF_FILE</code>
<code>maxBufferAgeMillis</code>	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)
<code>maxBufferSizeBytes</code>	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)
<code>maxBufferSizeRecords</code>	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
<code>minTimeBetweenFilePolls</code>	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

Definição da configuração	Descrição
	Padrão: 100
<code>multiLineStartPattern</code>	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.
<code>partitionKeyOption</code>	O método para gerar a chave de partição. Os valores válidos são <code>RANDOM</code> (inteiro gerado aleatoriamente) e <code>DETERMINISTIC</code> (um valor de hash calculado a partir dos dados).  Padrão: <code>RANDOM</code>
<code>skipHeaderLines</code>	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)
<code>truncatedRecordTerminationString</code>	A string que o agente utiliza para truncar um registro analisado quando o tamanho do registro excede o limite de tamanho de registro do Kinesis Data Streams. (1,000 KB)  Padrão: <code>'\n'</code> (nova linha)

## Monitoramento de vários diretórios de arquivos e gravação em vários streams

Ao especificar vários fluxos de configurações, você pode configurar o agente para monitorar vários diretórios de arquivos e enviar dados a vários streams. Na configuração de exemplo a seguir, o agente monitora dois diretórios de arquivos e envia dados a um streaming de dados do Kinesis e a um fluxo de entrega do Kinesis Data Firehose, respectivamente. Observe que você pode especificar diferentes endpoints para o Kinesis Data Streams e o Kinesis Data Firehose para que o streaming do Kinesis e o fluxo de entrega do Kinesis Data 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 como usar o agente com o Kinesis Data Firehose, consulte [Writing to Amazon Kinesis Data Firehose with Kinesis Agent](#).

## Uso do 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 no [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 com formato separado por delimitador em um registro com formato JSON.

```
{
  "optionName": "CSVTOJSON",
  "customFieldNames": [ "field1", "field2", ... ],
  "delimiter": "yourdelimiter"
}
```

##### customFieldNames

[Obrigatório] Os nomes de campo usados como chaves em cada par de valores de chave JSON. 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 com formato de log em um registro com formato JSON. 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 Apache Common Log. Cada entrada de log tem o seguinte padrão: `"%{host} %{ident} %{authuser} [%{datetime}] \"%{request}\" %{response} %{bytes}"`.

- **COMBINEDAPACHELOG** — O formato Apache Combined Log. Cada entrada de log tem o seguinte padrão: "%{host} %{ident} %{authuser} [%{datetime}] \"%{request}\" %{response} %{bytes} %{referrer} %{agent}".
- **APACHEERRORLOG** — O formato 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 de campo personalizados usados como chaves em cada par de valores de chave JSON. 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 : Configuração LOGTOJSON

Aqui está um exemplo de uma configuração LOGTOJSON para uma entrada Apache Common Log convertida em formato JSON:

```
{
  "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 : Configuração LOGTOJSON 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 Apache Common Log do exemplo anterior é convertida em formato JSON, 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 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 linhas 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 os valores customFieldNames especificados, a fim de 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 : Configuração LOGTOJSON com padrão de correspondência

Aqui está um exemplo de configuração LOGTOJSON referente a uma entrada Apache Common Log convertida em formato JSON, 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"}
```

## Comandos da CLI do agente

Inicie automaticamente o agente durante a inicialização 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
```

## Solução de problemas de produtores do Amazon Kinesis Data Streams

As seções a seguir oferecem soluções para alguns problemas comuns que você pode encontrar ao trabalhar com produtores do Amazon Kinesis Data Streams.

- [Aplicativo produtor está gravando a uma taxa menor que a esperada \(p. 113\)](#)
- [Erro de permissão de chave mestra do KMS não autorizada \(p. 115\)](#)

### Aplicativo produtor está gravando a uma taxa menor que a esperada

Os motivos mais comuns para a taxa de transferência de gravação ser mais lenta do que o esperado são os seguintes:

- [Limites de serviço excedidos \(p. 114\)](#)

- [Otimização do produtor \(p. 114\)](#)

## Limites de serviço excedidos

Para descobrir se os limites de serviço estão sendo excedidos, verifique se o produtor está lançando exceções de taxa de transferência a partir do serviço e valide quais operações da API estão sendo aceleradas. Lembre-se de que há limites diferentes de acordo com a chamada, consulte [Cotas e limites do Kinesis Data Streams \(p. 6\)](#). 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 taxa de transferência de pico e leve em conta o número de estilhaços no seu stream. Para obter mais informações sobre o gerenciamento de streams, consulte [Criar e gerenciar streamings \(p. 73\)](#).

### Tip

Lembre-se de arredondar para o kilobyte mais próximo para cálculos de limitação de taxa de transferência ao usar a operação de um único registro `PutRecord`, enquanto a operação de vários registros `PutRecords` é arredondada na soma cumulativa dos registros em cada chamada. Por exemplo, uma solicitação `PutRecords` com 600 registros com tamanho de 1,1 KB não serão aceleradas.

## Otimização do produtor

Antes de começar a otimizar o produtor, há algumas tarefas importantes a serem concluídas. Primeiro, identifique sua taxa de transferência 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 \(p. 114\)](#)). 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 normalmente está em execução a partir de um servidor no local ou instância do Amazon EC2. Os clientes que precisam de uma taxa de transferência 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 microalocar registros em lote/buffer, use [Kinesis Producer Library](#) (que tem lógica de agregação avançada), a operação de vários registros `PutRecords` ou agregue registros em um arquivo maior antes de usar a operação de um único registro `PutRecord`. Se você não puder alocar em lote/buffer, use vários threads para gravar no serviço do Kinesis Data Streams ao mesmo tempo. O 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 operação `PutRecords` ou o registrador do Kinesis nos SDKs da AWS Mobile. Para obter mais informações, consulte Guia de conceitos básicos do AWS Mobile SDK para Android e AWS Mobile SDK for iOS Getting Started Guide. 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.

## Erro de permissão de chave mestra do KMS não autorizada

Esse erro ocorre quando um aplicativo produtor grava em um stream criptografado sem permissões na chave mestra do KMS. Para atribuir permissões a um aplicativo para que acesse uma chave do KMS, consulte [Como usar políticas de chaves no AWS KMS](#) e [Usar políticas do IAM com o AWS KMS](#).

## Tópicos avançados para produtores do Kinesis Data Streams

Esta seção discute como otimizar seus produtores do Amazon Kinesis Data Streams.

### Tópicos

- [Limitação de taxas e tentativas de KPL \(p. 115\)](#)
- [Considerações ao usar a agregação do KPL \(p. 116\)](#)

## Limitação de taxas e tentativas de KPL

Quando você adiciona registros de usuário da Kinesis Producer Library (KPL) usando a operação `addUserRecord()` da KPL, um registro recebe um time stamp 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 ao Kinesis data stream como registros do Amazon Kinesis Data Streams usando uma chamada à operação da API `PutRecords` do Kinesis Data Streams. A operação

`PutRecords` envia solicitações ao stream que ocasionalmente apresenta falhas parciais ou totais. Os registros com falha são automaticamente adicionados de volta ao buffer do KPL. O novo prazo é definido com base no mínimo destes dois valores:

- Metade da configuração de `RecordMaxBufferedTime` atual
- O valor de tempo de vida do registro

Essa estratégia permite que os registros do usuário da KPL repetidos sejam incluídos em chamadas à API do Kinesis Data Streams subsequentes, para melhorar a taxa de transferência e reduzir a complexidade e, ao mesmo tempo, aplicar o valor de tempo de vida do registro do Kinesis Data Streams. 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 taxas

O KPL inclui um recurso de limitação de taxas, que limita a taxa de transferência por estilhaço enviada de um único produtor. A limitação de taxas é implementada com o uso de um algoritmo de bucket de token com buckets separados para bytes e registros do Kinesis Data Streams. Cada gravação bem-sucedida em um Kinesis data stream adiciona um token (ou vários) 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 taxa de transferência 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.

## Considerações ao usar a agregação do KPL

Embora o esquema do número de sequência dos registros resultantes do Amazon Kinesis Data Streams permaneça o mesmo, a agregação faz com que a indexação de registros de usuários da Kinesis Producer Library (KPL) contidos em um registro do agregado do Kinesis Data Streams se inicie em 0 (zero). No entanto, desde que você não dependa dos números de sequência para identificar exclusivamente os registros de usuários da KPL, o código poderá ignorar isso, já que a agregação (dos registros de usuários do KPL em um registro do Kinesis Data Streams) e a desagregação subsequente (de um registro do Kinesis Data Streams nos registros de usuários do KPL) cuidam disso automaticamente para você. Isso se aplica se o consumidor está usando o SDK da AWS ou do KCL. Para usar essa funcionalidade de agregação, você precisará retirar a parte do Java do KPL de sua compilação se o consumidor foi criado usando a API fornecida no SDK da AWS.

Se você pretender usar números de sequência como identidades exclusivas para os registros de usuários da KPL, recomendamos usar as operações `public int hashCode()` e `public boolean equals(Object obj)` que respeitam os contratos fornecidos em `Record` e `UserRecord` para habilitar a comparação dos registros de usuários da KPL. Além disso, se desejar examinar o número subsequente do registro de usuários da KPL, você poderá convertê-lo em uma instância de `UserRecord` e recuperar o número subsequente.

Para obter mais informações, consulte [Desagregação de consumidor \(p. 97\)](#).

# Ler dados do Amazon Kinesis Data Streams

Um consumidor é um aplicativo que processa todos os dados de um streaming de dados do Kinesis. Quando um consumidor usa distribuição avançada, ele recebe sua própria alocação de taxa de transferência de leitura de 2 MB/s permitindo que vários consumidores leiam dados do mesmo streaming em paralelo, sem disputa com outros consumidores por taxa de transferência de leitura. Para usar o recurso de distribuição avançada de estilhaços, consulte [Desenvolver consumidores personalizados com taxa de transferência dedicada \(distribuição avançada\)](#) (p. 161).

Por padrão, os estilhaços em um streaming fornecem 2 MB/s de taxa de transferência de leitura por estilhaço. Essa taxa de transferência 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 taxa de transferência por estilhaço é fixo, ainda que haja vários consumidores fazendo a leitura pelo estilhaço. Para usar essa taxa de transferência padrão de estilhaços, consulte [Desenvolver consumidores personalizados com taxa de transferência compartilhada](#) (p. 131).

A tabela a seguir compara a taxa de transferência padrão para a distribuição avançada. O atraso de propagação da mensagem é definido como o tempo, em milissegundos, que uma carga enviada usando as APIs de envio de carga (como PutRecord e PutRecords) demora para chegar ao aplicativo do consumidor por meio das APIs de consumo de carga (como GetRecords e SubscribeToShard).

Características	Consumidores não registrados sem distribuição avançada	Consumidores registrados com distribuição avançada
Taxa de transferência 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 taxa de transferência. 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 taxa de transferência 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 cerca de 1000 ms se você tiver cinco consumidores.	Normalmente, uma média de 70 ms se você tiver um ou cinco consumidores.
Custos:	N/D	Há um custo de recuperação de dados e um custo de hora de estilhaço por consumidor. Para obter mais informações, consulte <a href="#">Definição de preço do Amazon Kinesis Streams</a> .
Registro de modelo de entrega	Modelo Pull por HTTP usando <a href="#">GetRecords</a> .	O Kinesis Data Streams envia os registros para você por HTTP/2 usando <a href="#">SubscribeToShard</a> .

#### Tópicos

- [Desenvolver consumidores usando o AWS Lambda \(p. 118\)](#)
- [Desenvolver consumidores usando o Amazon Kinesis Data Analytics \(p. 118\)](#)
- [Desenvolver consumidores usando o Amazon Kinesis Data Firehose \(p. 118\)](#)
- [Utilizar a Biblioteca de Clientes da Kinesis \(p. 119\)](#)
- [Desenvolver consumidores personalizados com taxa de transferência compartilhada \(p. 131\)](#)
- [Desenvolver consumidores personalizados com taxa de transferência dedicada \(distribuição avançada\) \(p. 161\)](#)
- [Migrar consumidores do KCL 1.x para o KCL 2.x \(p. 169\)](#)
- [Solução de problemas de consumidores do Kinesis Data Streams \(p. 179\)](#)
- [Tópicos avançados para consumidores do Amazon Kinesis Data Streams \(p. 182\)](#)

## Desenvolver consumidores usando o AWS Lambda

Você pode usar uma função AWS Lambda para processar registros em um fluxo de dados. O AWS Lambda é um serviço de computação que permite executar código sem o provisionamento ou o gerenciamento de 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 o AWS Lambda, você pode executar código para praticamente qualquer tipo de aplicativo ou serviço de back-end, tudo sem 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 [Usar o AWS Lambda com o Amazon Kinesis](#).

## Desenvolver consumidores usando o Amazon Kinesis Data Analytics

Você pode usar um aplicativo do Amazon Kinesis Data Analytics para processar e analisar dados em um fluxo do Kinesis usando o SQL ou Java. Os aplicativos do Kinesis Data Analytics podem enriquecer os dados usando fontes de referência, agregar dados ao longo do tempo ou usar machine learning para encontrar anomalias de dados. Depois, você pode gravar os resultados da análise em outro fluxo do Kinesis, um fluxo de entrega do Kinesis Data Firehose ou uma função Lambda. Para obter mais informações, consulte o [Guia do desenvolvedor do Kinesis Data Analytics para aplicativos SQL](#) ou o [Guia do desenvolvedor do Kinesis Data Analytics para aplicativos Java](#).

## Desenvolver consumidores usando o Amazon Kinesis Data Firehose

É possível usar um Kinesis Data Firehose para ler e processar registros de um streaming do Kinesis. O Kinesis Data Firehose é um serviço totalmente gerenciado para fornecimento de dados de streaming em tempo real a destinos como o Amazon S3, o Amazon Redshift, o Amazon Elasticsearch Service e o Splunk. Você também pode configurar o Kinesis Data Firehose para transformar seus registros de dados e para converter o formato do registro antes de entregar os dados para seu destino. Para obter mais informações, consulte [Gravar no Kinesis Data Firehose usando o Kinesis Data Streams](#).

# Utilizar a Biblioteca de Clientes da Kinesis

Um dos métodos de desenvolvimento de aplicações de consumidor personalizadas que podem processar dados dos fluxos de dados KDS é utilizar o Kinesis Client Library (KCL).

## Tópicos

- [O que é o Kinesis Client Library? \(p. 119\)](#)
- [KCL Versões disponíveis \(p. 120\)](#)
- [Conceitos do KCL \(p. 120\)](#)
- [Utilizar uma tabela de arrendamento para rastrear os painéis processados pela aplicação do consumidor KCL \(p. 121\)](#)
- [Processar vários fluxos de dados com o mesmo KCL 2.x para aplicação do consumidor Java \(p. 128\)](#)

## O que é o Kinesis Client Library?

KCL ajuda-o a consumir e processar dados de um fluxo de dados da Kinesis, ao tomar conta de muitas das tarefas complexas associadas ao cálculo distribuído. Estes incluem balanceamento de carga em vários casos de aplicações de consumidor, resposta a falhas de instância de aplicação de consumidor, verificação de registos processados e reactuação para ressalto. O KCL cuida de todas estas subtarefas para que possa concentrar os seus esforços ao escrever a sua lógica de processamento de registos personalizada.

O KCL é diferente das API de Dados da Kinesis que estão disponíveis nas entidades AWS. As API de Dados da Kinesis ajudam-no a gerir muitos aspetos da Kinesis Data Streams, incluindo a criação de fluxos, a redefinição e a colocação e obtenção de registos. O KCL fornece uma camada de abstracção em torno de todas estas subtarefas, especificamente para que possa concentrar-se na lógica de processamento de dados personalizados da sua aplicação de consumidor. Para obter informações sobre a API de Dados da Kinesis, consulte o [Referência da Amazon Kinesis API](#).

### Important

O KCL é uma biblioteca Java. O suporte para idiomas que não o Java é fornecido utilizando uma interface multilíngue chamada multilangdaemon. Este daemon é baseado em Java e funciona em segundo plano quando está a utilizar um idioma KCL que não o Java. Por exemplo, se instalar o KCL para Python e escrever totalmente a sua aplicação de consumidor na Python, ainda precisa de Java instalado no seu sistema devido ao multilangdaemon. Além disso, o multilangdaemon tem algumas predefinições que poderá ter de personalizar para o seu caso de utilização, por exemplo, a região AWS a que se conecta. Para mais informações sobre o multilangdaemon no github, consulte [Projecto KCL multilangdaemon](#).

A KCL atua como um intermediário entre a lógica de processamento de registro e o Kinesis Data Streams. A KCL executa as seguintes tarefas:

- Liga-se ao fluxo de dados
- Enumera os sombreados dentro do fluxo de dados
- Utiliza locações para coordenar as associações de depósitos com os seus trabalhadores
- Cria uma instância de um processador de registro para cada estilhaço que gerencia
- Recolhe registos de dados do fluxo de dados
- Envia os registos ao processador de registos correspondente
- Registros processados pelos pontos de verificação
- Equilibra associações de trabalhadores de shard (locações) quando a contagem de instâncias de trabalhadores muda ou quando o fluxo de dados é regravado (os painéis são divididos ou fundidos)

## KCL Versões disponíveis

Atualmente, pode utilizar qualquer uma das seguintes versões suportadas do KCL para criar as suas aplicações personalizadas para o consumidor:

- KCL 1.x

Para obter mais informações, consulte [Desenvolver consumidores do KCL 1.x](#) (p. 131)

- KCL 2.x

Para obter mais informações, consulte [Desenvolver consumidores do KCL 2.x](#) (p. 146)

Pode utilizar KCL 1.x ou KCL 2.x para construir aplicações de consumidor que utilizam o rendimento partilhado. Para obter mais informações, consulte [Desenvolver consumidores personalizados com taxa de transferência compartilhada usando a KCL](#) (p. 131).

Para criar aplicações para consumidores que utilizem um rendimento dedicado (consumidores de fãs melhorados), só pode utilizar KCL 2.x. Para obter mais informações, consulte [Desenvolver consumidores personalizados com taxa de transferência dedicada \(distribuição avançada\)](#) (p. 161).

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 [Migrar consumidores do KCL 1.x para o KCL 2.x](#) (p. 169).

## Conceitos do KCL

- Aplicação de consumidor KCL – uma aplicação personalizada com o KCL e concebida para ler e processar registos dos fluxos de dados.
- Instância de aplicação de consumidor - As aplicações de consumidor KCL são normalmente distribuídas, com uma ou mais instâncias de aplicação a funcionar simultaneamente para coordenar falhas e carregar o processamento de registos de dados de equilíbrio de carga.
- Trabalhador – uma classe de alto nível que uma instância de aplicação de consumidor KCL utiliza para iniciar o processamento de dados.

### Important

Cada instância de aplicação do consumidor KCL tem um trabalhador.

O trabalhador inicializa e supervisiona várias tarefas, incluindo a sincronização de informações sobre o balanço e a localização, o rastreamento de atribuições de títulos e o processamento de dados dos painéis. Um trabalhador fornece KCL com as informações de configuração para a aplicação do consumidor, tais como o nome do fluxo de dados cujos registos de dados, esta aplicação de consumidor KCL vai processar e as credenciais AWS necessárias para aceder a este fluxo de dados. O trabalhador também inicia essa instância específica de aplicação de consumidor KCL para fornecer registos de dados do fluxo de dados aos processadores de registos.

### Important

Na KCL 1.x, esta aula é chamada Trabalhador. Para mais informações, (estes são repositórios Java KCL), consulte <https://github.com/aws-labs/amazon-kinesis-client/blob/v1.x/src/main/java/com/amazonaws/services/kinesis/clientlibrary/lib/worker/Worker.java>. Na KCL 2.x, esta aula é chamada Programador. O objectivo do Scheduler in KCL 2.x é idêntico ao objectivo do trabalhador em KCL 1.x. Para mais informações sobre a classe Scheduler (Agendador) em KCL 2.x, consulte <https://github.com/aws-labs/amazon-kinesis-client/blob/master/amazon-kinesis-client/src/main/java/software/amazon/kinesis/coordinator/Scheduler.java>.

- Localização – dados que definem a ligação entre um trabalhador e um “shard”. As aplicações de consumidor KCL distribuído utilizam localizações para o processamento de registos de dados de partição

numa frota de trabalhadores. Em qualquer altura, cada depósito de registos de dados está vinculado a um trabalhador em particular por um arrendamento identificado pelo chave pendente variável.

Por predefinição, um trabalhador pode deter uma ou mais locações (sujeito ao valor do `maxleasesfortrabalhador` variável) ao mesmo tempo.

#### Important

Todos os trabalhadores irão terminar para manter todas as locações disponíveis para todas as placas disponíveis num fluxo de dados. Mas apenas um trabalhador irá realizar com sucesso cada locação em qualquer momento.

Por exemplo, se tiver uma instância de aplicação do consumidor A com o trabalhador A que está a processar um fluxo de dados com 4 derivações, o trabalhador A pode suportar os alugueres para as colunas 1, 2, 3 e 4 ao mesmo tempo. Mas se tiver duas instâncias de aplicação do consumidor: A e B com o trabalhador A e o trabalhador B, e estas instâncias estão a processar um fluxo de dados com 4 derivações, o trabalhador A e o trabalhador B não podem ambos manter o aluguer para 1.º andar ao mesmo tempo. Um trabalhador mantém o aluguer num determinado ponto até estar pronto para parar de processar este registo de dados ou até falha. Quando um funcionário parar de manter o arrendamento, outro trabalhador ocupa e mantém o arrendamento.

Para mais informações, (estes são repositórios Java KCL), consulte <https://github.com/aws-labs/amazon-kinesis-client/blob/v1.x/src/main/java/com/amazonaws/services/kinesis/leases/impl/Lease.java> para KCL 1.x e <https://github.com/aws-labs/amazon-kinesis-client/blob/master/amazon-kinesis-client/src/main/java/software/amazon/kinesis/leases/Lease.java> para KCL 2.x.

- Tabela de locação - uma tabela exclusiva de `dynamodb` da Amazon que é usada para manter o registo das derivações num fluxo de dados KDS que está a ser alugado e processado pelos trabalhadores da aplicação do consumidor KCL. A tabela de arrendamento deve permanecer em sincronia (dentro de um trabalhador e em todos os trabalhadores) com as informações mais recentes sobre o fluxo de dados, enquanto a aplicação do consumidor KCL está a funcionar. Para obter mais informações, consulte [Utilizar uma tabela de arrendamento para rastrear os painéis processados pela aplicação do consumidor KCL](#) (p. 121).
- Processador de registo – a lógica que define como a sua aplicação de consumidor KCL processa os dados que obtém dos fluxos de dados. Em execução, uma instância de aplicação de consumidor KCL instanciar um trabalhador, e este trabalhador instanciar um processador de registo para cada ponto onde detém um arrendamento.

## Utilizar uma tabela de arrendamento para rastrear os painéis processados pela aplicação do consumidor KCL

### Tópicos

- [O que é uma Tabela de Arrendamento](#) (p. 121)
- [Throughput](#) (p. 123)
- [Como é que uma tabela de arrendamento é sincronizada com os Shards num fluxo de dados KDS](#) (p. 123)

## O que é uma Tabela de Arrendamento

Para cada Amazon Kinesis Data Streams application, KCL utiliza uma tabela de locação única (armazenada em Amazon DynamoDB tabela) para manter o registo das derivações num fluxo de dados KDS que estão a ser alugados e processados pelos trabalhadores da aplicação do consumidor KCL.



### Important

KCL utiliza o nome da aplicação do consumidor para criar o nome da tabela de arrendamento que esta aplicação do consumidor utiliza, portanto, cada nome da aplicação do consumidor deve ser único.

Pode ver a tabela de arrendamento utilizando o [Amazon DynamoDB consola](#) enquanto a aplicação do consumidor está a funcionar.

Se a tabela de locação para a sua aplicação de consumidor KCL não existir quando a aplicação é iniciada, um dos trabalhadores cria a tabela de locação para esta aplicação.

### 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 arrendamento representa um ponto que está a ser processado pelos trabalhadores da sua aplicação de consumidor. Se a sua aplicação de consumidor KCL processa apenas um fluxo de dados, então `leaseKey` que é a tecla cardinal para a tabela de locação, é a identificação de referência. Se estiver [Processar vários fluxos de dados com o mesmo KCL 2.x para aplicação do consumidor Java](#) (p. 128), então a estrutura da chave de locação parece 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:** O número de sequência de ponto de verificação mais recente para o rodapé. Este valor é único em todas as derivações no fluxo de dados.
- **padrãodeverificação:** Ao utilizar a funcionalidade de agregação da Biblioteca da Kinesis, esta é uma extensão para domínio que rastreia registos de utilizadores individuais dentro do Kinesis registo.
- **contador de pagamento:** Utilizado para o aluguer de arrendamento para que os trabalhadores possam detectar que o seu aluguer foi realizado por outro trabalhador.
- **chave de entrega:** Um identificador único para um aluguer. Cada locação é específica para um compartimento no fluxo de dados e é mantido por um trabalhador de cada vez.
- **proprietário:** O trabalhador que está a realizar este aluguer.
- **o seu nome de utilizador:** Quantas vezes este aluguer mudou os trabalhadores desde a última vez em que um ponto de controlo foi escrito.
- **parentshardid:** Utilizado para garantir que o compartimento principal é totalmente processado antes de o processamento começar nos painéis descendentes. Isso garante que os registos sejam processados na mesma ordem em que foram colocados no stream.
- **intervalo analógico:** Utilizado pelo `PeriodicShardSyncManager` para executar os sinc periódicos para encontrar as derivações em falta na tabela de arrendamento e criar locações para eles, se necessário.

### Note

Estes dados estão presentes na tabela de arrendamento para cada ponto inicial, começando pela KCL 1.14 e KCL 2.3. Para mais informações sobre `PeriodicShardSyncManager` e sincronização periódica entre locações e ganchos, consulte [Como é que uma tabela de arrendamento é sincronizada com os Shards num fluxo de dados KDS](#) (p. 123).

- **crianças:** Utilizado pelo `LeaseCleanupManager` para rever o estado de processamento do posto de trabalho da criança e decidir se o “shard” principal pode ser eliminado da tabela de arrendamento.

### Note

Estes dados estão presentes na tabela de arrendamento para cada ponto inicial, começando pela KCL 1.14 e KCL 2.3.



- shardid: A ID do rodapé.

#### Note

Estes dados só estão presentes na tabela de aluguer se estiver [Processar vários fluxos de dados com o mesmo KCL 2.x para aplicação do consumidor Java \(p. 128\)](#). Isto só é suportado em KCL 2.x para Java, começando com KCL 2.3 para Java e para além de.

- nome do fluxo O identificador do fluxo de dados no seguinte formato: account-id:StreamName:streamCreationTimestamp.

#### Note

Estes dados só estão presentes na tabela de aluguer se estiver [Processar vários fluxos de dados com o mesmo KCL 2.x para aplicação do consumidor Java \(p. 128\)](#). Isto só é suportado em KCL 2.x para Java, começando com KCL 2.3 para Java e para além de.

## Throughput

Se seu Amazon Kinesis Data Streams application recebe exceções de taxa de transferência provisionada, você deve aumentar a taxa de transferência provisionada para a tabela do DynamoDB. A 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 o seu aplicativo. Por exemplo, se o seu Amazon Kinesis Data Streams application define pontos de verificação com frequência ou opera em um stream que é composto por vários estilhaços, você pode precisar de uma taxa de transferência maior.

Para obter informações sobre a taxa de transferência provisionada no DynamoDB, consulte [Modo de capacidade de leitura/gravação](#) e [Trabalhar com tabelas e dados](#) no Guia do desenvolvedor do Amazon DynamoDB.

## Como é que uma tabela de arrendamento é sincronizada com os Shards num fluxo de dados KDS

Os trabalhadores nas aplicações de consumidor da KCL utilizam locações para processar as derivações de um dado fluxo de dados. A informação sobre o que o trabalhador está a pagar em qualquer altura é armazenada numa tabela de arrendamento. A tabela de arrendamento deve permanecer em sincronização com as últimas informações de trecho do fluxo de dados enquanto a aplicação do consumidor KCL estiver a funcionar. A KCL sincroniza a tabela de arrendamento com a informação de shards adquirida do serviço de Tesouraria de Dados da Kinesis durante o arranque da aplicação do consumidor (quer quando a aplicação do consumidor seja inicializada ou reiniciada) e também sempre que um ponto que esteja a ser processado atingir um fim (ressalto). Por outras palavras, os trabalhadores ou uma aplicação de consumidor KCL são sincronizados com o fluxo de dados que estão a processar durante a faixa de reforço inicial da aplicação do consumidor e sempre que a aplicação do consumidor encontrar um fluxo de dados reshard event.

#### Tópicos

- [Sincronização em KCL 1.0 - 1.13 e KCL 2.0 - 2.2 \(p. 123\)](#)
- [Sincronização em KCL 2.x, começando com KCL 2.3 e Beyond \(p. 124\)](#)
- [Sincronização em KCL 1.x, começando com KCL 1.14 e posterior \(p. 126\)](#)

## Sincronização em KCL 1.0 - 1.13 e KCL 2.0 - 2.2

Nos KCL 1.0 - 1.13 e KCL 2.0 - 2.2, durante a criação de arranque da aplicação do consumidor e também durante cada fluxo de dados, o KCL sincroniza a tabela de arrendamento com as informações de rodapé adquiridas no serviço de Tesouraria de Dados da Kinesis, ao invocar o `ListShards` ou o `DescribeStream` API Discovery. Em todas as versões de KCL listadas acima, cada trabalhador de uma

aplicação de consumidor KCL completa os seguintes passos para realizar o processo de sincronização de arrendamento/variação durante o arranque da aplicação do consumidor e em cada fluxo reshard event:

- Recupera todas as derivações para os dados que estão a ser processados
- Recupera todas as locações da tabela de arrendamento
- Filtra cada ponto aberto que não tenha um aluguer na tabela de arrendamento
- Itere sobre todas as placas abertas e para cada barra aberta sem a abertura principal:
  - Atravesse a árvore da hierarquia através do caminho do antepassados para determinar se o fragmento é um descensor. Um fragmento é considerado descensivo, se for processado um "sling" anteauposto (entrada de arrendamento para a presidência de predecessão existente na tabela de arrendamento) ou se for processada um predador anteauposto (por exemplo, se a posição inicial for TRIM\_HORIZON ou AT\_TIMESTAMP)
  - Se o ponto livre em contexto for um descensivo, o KCL verifica o ponto de controlo com base na posição inicial e cria locações para os seus pais, se necessário

## Sincronização em KCL 2.x, começando com KCL 2.3 e Beyond

Começando com as versões mais recentes suportadas do KCL 2.x (KCL 2.3) e para além da biblioteca, a biblioteca suporta agora as seguintes alterações ao processo de sincronização. Estas alterações de sincronização de locação/locação reduzem significativamente o número de chamadas API efectuadas pelas aplicações de consumidor KCL para o serviço de Fluxos de Dados da Kinesis e otimizam a gestão de arrendamento na sua aplicação de consumidor KCL.

- Durante o arranque da aplicação, se a mesa de locação estiver vazia, o KCL utiliza o `ListShard` Opção de filtragem da API (a `ShardFilter` parâmetro de pedido opcional) para recuperar e criar locações apenas para uma captura de painéis abertos no momento especificado pelo `ShardFilter` parâmetro. O `ShardFilter` permite-lhe filtrar a resposta do `ListShards` API. A única propriedade necessária da `ShardFilter` parâmetro é `Type`. KCL utiliza o `Type` propriedade do filtro e o seguinte dos valores válidos para identificar e devolver uma captura de placas abertas que possam exigir novas locações:
  - `AT_TRIM_HORIZON` - a resposta inclui todas as derivações abertas em `TRIM_HORIZON`.
  - `AT_LATEST` - a resposta inclui apenas os sombreados abertos do fluxo de dados.
  - `AT_TIMESTAMP` - a resposta inclui todas as derivações cujo carimbo de data/hora de início seja igual ou inferior ao carimbo de data/hora e data de hora de fim que é igual ou superior à data de carimbo de hora indicada ou ainda aberto.

`ShardFilter` é utilizado ao criar locações para uma tabela de locação vazia para inicializar locações para uma captura de derivações especificadas em `RetrievalConfig#initialPositionInStreamExtended`.

Para mais informações sobre `ShardFilter`, consulte [https://docs.aws.amazon.com/kinesis/latest/APIReference/API\\_ShardFilter.html](https://docs.aws.amazon.com/kinesis/latest/APIReference/API_ShardFilter.html).

- Em vez de todos os trabalhadores que efectuem a sincronização de arrendamento/shard, para manter a tabela de arrendamento atualizada com as últimas paletas no fluxo de dados, um único líder de trabalhadores eleita realiza a sincronização de arrendamento/variação.
- KCL 2.3 utiliza o `ChildShards` parâmetro de devolução do `GetRecords` e o `SubscribeToShard` para realizar a sincronização de arrendamento/variação que ocorre em `SHARD_END` para as placas fechadas, permitindo que um trabalhador KCL crie apenas locações para as derivações descendentes do engate que terminou o processamento. Para a partilha em todas as aplicações do consumidor, esta otimização da sincronização de arrendamento/variação utiliza o `ChildShards` parâmetro do `GetRecords` API. Para as aplicações de consumo dedicadas (ventilador melhorado), esta otimização da sincronização de arrendamento/variação utiliza o `ChildShards` parâmetro do `SubscribeToShard` API. Para mais informações, consulte [petrecabos](#), [subscrevamenteshards](#), e [bebé](#).
- Com as alterações acima, o comportamento do KCL está a passar do modelo de todos os trabalhadores a aprender sobre os painéis existentes para o modelo de trabalhadores que aprendem apenas sobre

as crianças que são portadoras de cada trabalhador. Por conseguinte, além da sincronização que ocorre durante a criação de uma aplicação de aplicações de consumo de consumidores e de eventos resrígidos, a KCL também realiza análises periódicas adicionais de shard/locação para identificar quaisquer potenciais buracos na tabela de arrendamento (por outras palavras, para saber mais sobre todos os novos painéis) para garantir que o intervalo hash completo do fluxo de dados está a ser processado e crie locações para eles, se necessário. `PeriodicShardSyncManager` é o componente que é responsável por executar exames periódicos de renting/shard.

Para mais informações sobre `PeriodicShardSyncManager` em KCL 2.3, consulte <https://github.com/aws-labs/amazon-kinesis-client/blob/master/amazon-kinesis-client/src/main/java/software/amazon/kinesis/leases/LeaseManagementConfig.java#L201-L213>.

No KCL 2.3, estão disponíveis novas opções de configuração para configurar `PeriodicShardSyncManager` em `LeaseManagementConfig`:

Name (Nome): .	Valor padrão: ""	Description (Descrição)
recuperadode recuperaçãoauditor executionononilis	120 000 (2 minutos)	Frequência (em milis) do trabalho do auditor a digitalizar para locações parciais na tabela de arrendamento. Se o auditor detectar qualquer buraco nos arrendamentos para um fluxo, então iria desencadear a sincronização de um comutador com base em leasesRecoveryAuditorInco

Name (Nome): .	Valor padrão: ""	Description (Descrição)
baixaçãode recuperaçãoauditãode segurança	3. diverência	Limite de confiança para o trabalho de auditor periódico para determinar se os alugueres para um fluxo de dados na tabela de locação são inconsistentes. Se o auditor encontrar o mesmo conjunto de inconsistências consecutivas para um fluxo de dados durante este período, então iria desencadear uma sincronização de "shard".

Também são emitidas novas métricas turdwatch para monitorizar a saúde do `PeriodicShardSyncManager`. Para mais informações, consulte [PeriodicShardSyncManager \(p. 207\)](#).

- Incluindo uma otimização para `HierarchicalShardSyncer` para criar apenas locações para uma camada de painéis.

## Sincronização em KCL 1.x, começando com KCL 1.14 e posterior

Começando com as versões mais recentes suportadas do KCL 1.x (KCL 1.14) e para além disso, a biblioteca suporta agora as seguintes alterações ao processo de sincronização. Estas alterações de sincronização de locação/locação reduzem significativamente o número de chamadas API efectuadas pelas aplicações de consumidor KCL para o serviço de Fluxos de Dados da Kinesis e otimizam a gestão de arrendamento na sua aplicação de consumidor KCL.

- Durante o arranque da aplicação, se a mesa de locação estiver vazia, o KCL utiliza o `ListShard` Opção de filtragem da API (a `ShardFilter` parâmetro de pedido opcional) para recuperar e criar locações apenas para uma captura de painéis abertos no momento especificado pelo `ShardFilter` parâmetro. O `ShardFilter` permite-lhe filtrar a resposta do `ListShards` API. A única propriedade necessária da `ShardFilter` parâmetro é `Type`. KCL utiliza o `Type` propriedade do filtro e o seguinte dos valores válidos para identificar e devolver uma captura de placas abertas que possam exigir novas locações:
  - `AT_TRIM_HORIZON` - a resposta inclui todas as derivações abertas em `TRIM_HORIZON`.
  - `AT_LATEST` - a resposta inclui apenas os sombreados abertos do fluxo de dados.
  - `AT_TIMESTAMP` - a resposta inclui todas as derivações cujo carimbo de data/hora de início seja igual ou inferior ao carimbo de data/hora e data de hora de fim que é igual ou superior à data de carimbo de hora indicada ou ainda aberto.

`ShardFilter` é utilizado ao criar locações para uma tabela de locação vazia para inicializar locações para uma captura de derivações especificadas em `KinesisClientLibConfiguration#initialPositionInStreamExtended`.

Para mais informações sobre `ShardFilter`, consulte [https://docs.aws.amazon.com/kinesis/latest/APIReference/API\\_ShardFilter.html](https://docs.aws.amazon.com/kinesis/latest/APIReference/API_ShardFilter.html).

- Em vez de todos os trabalhadores que efectuem a sincronização de arrendamento/shard, para manter a tabela de arrendamento atualizada com as últimas paletas no fluxo de dados, um único líder de trabalhadores eleita realiza a sincronização de arrendamento/variação.
- KCL 1.14 utiliza o `ChildShards` parâmetro de devolução do `GetRecords` e o `SubscribeToShard` para realizar a sincronização de arrendamento/variação que ocorre em `SHARD_END` para as placas fechadas, permitindo que um trabalhador KCL crie apenas locações para as derivações descendentes do engate que terminou o processamento. Para mais informações, consulte [petrecabos](#) e [bebé](#).
- Com as alterações acima, o comportamento do KCL está a passar do modelo de todos os trabalhadores a aprender sobre os painéis existentes para o modelo de trabalhadores que aprendem apenas sobre as crianças que são portadoras de cada trabalhador. Por conseguinte, além da sincronização que ocorre durante a criação de uma aplicação de aplicações de consumo de consumidores e de eventos resrígidos, a KCL também realiza análises periódicas adicionais de shard/locação para identificar quaisquer potenciais buracos na tabela de arrendamento (por outras palavras, para saber mais sobre todos os novos painéis) para garantir que o intervalo hash completo do fluxo de dados está a ser processado e crie locações para eles, se necessário. `PeriodicShardSyncManager` é o componente que é responsável por executar exames periódicos de renting/shard.

Quando `KinesisClientLibConfiguration#shardSyncStrategyType` está definido para `ShardSyncStrategyType.SHARD_END`, `PeriodicShardSyncLeasesRecoveryAuditorInconsistencyConfidenceThreshold` é utilizado para determinar o limiar para o número de digitalizações consecutivas que contêm orifícios na tabela de arrendamento, após o que aplicar uma sincronização de "shard". Quando `KinesisClientLibConfiguration#shardSyncStrategyType` está definido para `ShardSyncStrategyType.PERIODIC`, `LeasesRecoveryAuditorInconsistencyConfidenceThreshold` é ignorado.

Para mais informações sobre `PeriodicShardSyncManager` em KCL 1.14, consulte <https://github.com/aws-labs/amazon-kinesis-client/blob/v1.x/src/main/java/com/amazonaws/services/kinesis/clientlibrary/lib/worker/KinesisClientLibConfiguration.java#L987-L999>.

No KCL 1.14, a nova opção de configuração está disponível para configurar `PeriodicShardSyncManager` em `LeaseManagementConfig`:

Name (Nome): .	Valor padrão: ""	Description (Descrição)
baixaçãode recuperaçãoauditãodediverênciade segurança	3.	Limite de confiança para o trabalho de auditor periódico para determinar se os alugueres para um fluxo de dados na tabela de locação são inconsistentes. Se o auditor encontrar o mesmo conjunto de inconsistências consecutivas para um fluxo de dados durante este período, então iria

Name (Nome): .	Valor padrão: ""	Description (Descrição)
		desencadear uma sincronização de "shard".

Também são emitidas novas métricas turdwatch para monitorizar a saúde do `PeriodicShardSyncManager`. Para mais informações, consulte [PeriodicShardSyncManager](#) (p. 207).

- KCL 1.14 agora também suporta uma limpeza de arrendamento diferida. Os alugueres são eliminados de forma assíncrona por `LeaseCleanupManager` ao chegar `SHARD_END`, quando um compartimento expirou para além do período de retenção do fluxo de dados ou foi fechado como resultado de uma operação de reabastecimento.

Estão disponíveis novas opções de configuração para configurar LeaseCleanupManager.

Name (Nome):	Valor padrão: ""	Description (Descrição)
leasecleanupinterval	111 segundos	Intervalo para executar a sequência de limpeza de locação.
concluiratualizaratualizaçãolimpeza	5 minutos.	Intervalo para verificar se um renting está concluído ou não.
bagagemdelimpezasol	30 minutos	Intervalo para verificar se um aluguer é lixo (ou seja, encurtado para além do período de retenção do fluxo de dados) ou não.

- Incluindo uma otimização para `KinesisShardSyncer` para criar apenas locações para uma camada de painéis.

## Processar vários fluxos de dados com o mesmo KCL 2.x para aplicação do consumidor Java

Esta secção descreve as seguintes alterações em KCL 2.x para Java que lhe permite criar KCL aplicações de consumo que podem processar mais do que 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 para além de.

O processamento de múltiplos fluxos NÃO é suportado para quaisquer outros idiomas nos quais o KCL 2.x possa ser implementado.

O processamento multistream NÃO é suportado em nenhuma versão do KCL 1.x.

- interface multistreamtracker

Para construir uma aplicação de consumidor que possa processar vários fluxos ao mesmo tempo, tem de implementar uma nova interface chamada [multidinamizador](#). Esta interface inclui o `streamConfigList` que devolve a lista de fluxos de dados e as suas configurações a serem processadas pela aplicação do consumidor KCL. Repare que os fluxos de dados que estão a ser processados podem ser alterados durante o período de execução da aplicação do consumidor. `streamConfigList` é chamado periodicamente pelo KCL para aprender sobre as alterações nos fluxos de dados a processar.

O `streamConfigList` o método preenche o [otimização](#) lista.

```
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;
}
```

Note que o `StreamIdentifier` e `InitialPositionInStreamExtended` são campos obrigatórios, enquanto `consumerArn` é opcional. Tem de fornecer o `consumerArn` apenas se estiver a utilizar o KCL 2.x para implementar uma aplicação de consumidor melhorada.

Para mais informações sobre `StreamIdentifier`, consulte <https://github.com/aws-labs/amazon-kinesis-client/blob/0c5042dadf794fe988438436252a5a8fe70b6b0b/amazon-kinesis-client/src/main/java/software.amazon.kinesis.common/StreamIdentifier.java#L29>. Pode criar uma instância multistream para o `StreamIdentifier` a partir do identificador de fluxo serializado. O identificador de fluxo serializado deve ser do seguinte formato: `account-id:StreamName:streamCreationTimestamp`.

```
* @param streamIdentifierSer
* @return StreamIdentifier
*/
public static StreamIdentifier multiStreamInstance(String streamIdentifierSer) {
    if (PATTERN.matcher(streamIdentifierSer).matches()) {
        final String[] split = streamIdentifierSer.split(DELIMITER);
        return new StreamIdentifier(split[0], split[1], Long.parseLong(split[2]));
    } else {
        throw new IllegalArgumentException("Unable to deserialize StreamIdentifier
from " + streamIdentifierSer);
    }
}
```

`MultistreamTracker` também inclui uma estratégia para eliminar locações de fluxos antigos na tabela de arrendamento (`formerStreamsLeasesDeletionStrategy`). Note que a estratégia NÃO PODE ser alterada durante o período de execução da aplicação do consumidor. Para mais informações, consulte <https://github.com/aws-labs/amazon-kinesis-client/blob/0c5042dadf794fe988438436252a5a8fe70b6b0b/amazon-kinesis-client/src/main/java/software.amazon.kinesis.processor/FormerStreamsLeasesDeletionStrategy.java>

- **construtor de confisco** é uma classe de aplicações que pode utilizar para especificar todas as definições de configuração KCL 2.x a utilizar ao construir a sua aplicação de consumidor KCL. `ConfigsBuilder` a turma tem agora apoio para `MultiStreamTracker` interface. Pode inicializar o construtor de configograma com o nome do fluxo de dados único para consumir registos de:

```
/**
 * 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 pode inicializar o construtor de configs com `MultiStreamTracker` se pretender implementar uma aplicação de consumidor KCL que processa 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 suporte multistream implementado para a sua aplicação de consumidor KCL, cada linha da tabela de locação da aplicação contém agora a identificação de rede e o nome do fluxo dos vários fluxos de dados que este processo de aplicação.
- Quando o suporte multistream para a sua aplicação de consumidor KCL é implementado, a chave de locação assume a seguinte estrutura: `account-id:StreamName:streamCreationTimestamp:ShardId`. Por exemplo, `111111111:multiStreamTest-1:12345:shardId-000000000336`.

#### Important

Quando a sua aplicação de consumidor KCL existente está configurada para processar apenas um fluxo de dados, a tecla de transferência (que é a tecla cardinal para a tabela de locação) é a ID de referência. Se reconfigurar esta aplicação de consumidor KCL existente para processar vários fluxos de dados, esta quebra a sua tabela de locação, porque com suporte multistream, a estrutura de transferência deve ser a seguinte: `account-id:StreamName:StreamCreationTimestamp:ShardId`.

## Desenvolver consumidores personalizados com taxa de transferência compartilhada

Se não precisar de taxa de transferência específica ao receber dados do Kinesis Data Streams, nem de atrasos de propagação de leitura em 200 ms, você poderá criar aplicativos de consumidor, conforme descrito nos tópicos a seguir.

#### Tópicos

- [Desenvolver consumidores personalizados com taxa de transferência compartilhada usando a KCL](#) (p. 131)
- [Desenvolver consumidores personalizados com taxa de transferência compartilhada usando o AWS SDK for Java](#) (p. 157)

Para obter informações sobre a criação de consumidores que podem receber fluxos de dados do Kinesis com taxa de transferência dedicada, consulte [Desenvolver consumidores personalizados com taxa de transferência dedicada \(distribuição avançada\)](#) (p. 161).

## Desenvolver consumidores personalizados com taxa de transferência compartilhada usando a KCL

Um dos métodos de desenvolvimento de uma aplicação de consumidor personalizada com rendimento partilhado é utilizar o Kinesis Client Library (KCL).

#### Tópicos

- [Desenvolver consumidores do KCL 1.x](#) (p. 131)
- [Desenvolver consumidores do KCL 2.x](#) (p. 146)

## Desenvolver consumidores do KCL 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 sobre o KCL, consulte [O que é o Kinesis Client Library?](#) (p. 119).

#### Tópicos

- [Desenvolver um consumidor de Kinesis Client Library em Java \(p. 132\)](#)
- [Desenvolver um consumidor de Kinesis Client Library em Node.js \(p. 137\)](#)
- [Desenvolver um consumidor de Kinesis Client Library em .NET \(p. 140\)](#)
- [Desenvolver um consumidor de Kinesis Client Library em Python \(p. 143\)](#)
- [Desenvolver um consumidor de Kinesis Client Library em Ruby \(p. 145\)](#)

## Desenvolver um consumidor de Kinesis Client Library em Java

Você pode usar a Kinesis Client Library (KCL) para criar aplicativos que processam dados dos streamings de dados do Kinesis. A Kinesis Client Library está disponível em vários idiomas. Este tópico discute Java. Para visualizar a referência do Javadoc, consulte o [tópico sobre o Javadoc da AWS da classe AmazonKinesisClient](#).

Para fazer download da KCL de Java no GitHub, acesse a [Kinesis Client Library \(Java\)](#). Para localizar a KCL de Java no Apache Maven, acesse a página [Resultados da pesquisa do KCL](#). Para fazer download do código de exemplo de um aplicativo de consumidor da KCL de Java no GitHub, acesse a página [Projeto de exemplo da KCL para Java](#) no 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 Java da AWS e Log4j, consulte [Registro em log com Log4j](#) no AWS SDK for Java Developer Guide.

Você precisa concluir as tarefas a seguir ao implementar um aplicativo de consumidor da KCL em Java:

### Tarefas

- [Implementar os métodos `IRecordProcessor` \(p. 132\)](#)
- [Implementar uma fábrica de classe para a interface `IRecordProcessor` \(p. 135\)](#)
- [Criar um operador \(p. 135\)](#)
- [Modificar as propriedades de configuração \(p. 135\)](#)
- [Migrar para a versão 2 da interface do processador de registros \(p. 136\)](#)

### Implementar os métodos `IRecordProcessor`

Atualmente, a KCL oferece suporte a duas versões da interface `IRecordProcessor`: a interface original está disponível com a primeira versão da KCL e a versão 2 está disponível desde a versão 1.5.0 da 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.

#### Versões do `IRecordProcessor`

- [Interface original \(versão 1\) \(p. 132\)](#)
- [Interface atualizada \(versão 2\) \(p. 134\)](#)

#### 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, IRecordProcessorCheckpointier checkpointier)
public void shutdown(IRecordProcessorCheckpointier checkpointier, ShutdownReason reason)
```

#### initialize

A KCL chama o método `initialize` quando o processador de registros é instanciado, passando um ID de estilhaço específico como um 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 o Kinesis Data Streams tem semântica pelo menos uma vez, o que significa que cada registro de dados de um estilhaço 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 [Reestilhaçamento, escalabilidade e processamento paralelo](#) (p. 183).

```
public void initialize(String shardId)
```

#### processRecords

A KCL chama o método `processRecords` passando uma lista de registro de dados do estilhaço especificado pelo método `initialize(shardId)`. 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, IRecordProcessorCheckpointier
    checkpointier)
```

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 o processador de registros para rastrear os registros que já foram processados em um estilhaço. O KCL cuida deste seguimento por passar um seletor de verificação (`IRecordProcessorCheckpointier`) a `processRecords`. O processador de registros chama para o `checkpoint` neste interface para informar o KCL até que ponto progrediu no processamento dos registros no rodapé. Caso o operador falhe, a KCL usa essas informações para reiniciar o processamento do estilhaço no último registro processado conhecido.

Em uma operação de divisão ou mesclagem, a KCL não começará a processar os novos estilhaços até que os processadores dos estilhaços originais tenham chamado `checkpoint` para indicar que o processamento dos estilhaços originais está completo.

Se você não passar um parâmetro, a KCL assumirá 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 de contactar

checkpoint em cada chamada para `processRecords`. Um processador poderia, por exemplo, ligar `checkpoint` em cada terceira chamada para `processRecords`. Opcionalmente, pode especificar o número de sequência exacto de um registo como um parâmetro para `checkpoint`. Neste caso, o KCL assume que todos os registos foram processados apenas até esse registo.

No exemplo, o método privado `checkpoint` mostra como chamar `IRecordProcessorCheckpointier.checkpoint` usando a lógica de novas tentativas e o tratamento de exceções apropriados.

A KCL depende do `processRecords` para gerenciar todas as exceções que surjam do processamento de registos de dados. Se ocorrer uma exceção em `processRecords`, a KCL ignorará os registos de dados que foram passados antes da exceção. Ou seja, esses registos não serão reenviados para o processador de registos que lançou a exceção ou para qualquer outro processador de registos no consumidor.

shutdown

A KCL chama o método `shutdown` quando o processamento termina (o motivo do desligamento é `TERMINATE`) ou quando o operador não está mais respondendo (a razão do desligamento é `ZOMBIE`).

```
public void shutdown(IRecordProcessorCheckpointier checkpointier, ShutdownReason reason)
```

O processamento termina quando o processador de registos não recebe mais registos do estiloção porque ele foi dividido ou intercalado, ou o stream foi excluído.

O KCL também passa um `IRecordProcessorCheckpointier` interface para `shutdown`. Se o motivo do encerramento for `TERMINATE`, o processador de registos deve terminar o processamento de quaisquer registos de dados e, em seguida, ligar para o `checkpoint` método 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 registos 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 registos em `processRecords()`, você pode usar `processRecordsInput.getRecords()`.

A partir da versão 2 dessa interface (KCL 1.5.0 e posterior), as novas entradas a seguir 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 registos seriam fornecidos à instância do processador de registos. Esse é o número de sequência que foi verificado pela última vez pela instância do processador de registos que processou anteriormente o mesmo estiloção. 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 registos anterior parasse.

## Implementar uma fábrica de classe para a interface IRecordProcessor

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();
    }
}
```

## Criar um operador

Conforme discutido em [Implementar os métodos IRecordProcessor \(p. 132\)](#), há duas versões da interface do processador de registros da KCL para escolha, o que afeta como você cria um operador. 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();
```

## Modificar 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 do aplicativo

A KCL exige um nome de aplicativo que seja exclusivo nos 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 executa uma instância adicional do mesmo código de aplicativo, mas com um nome de aplicativo diferente, a KCL trata a segunda instância como um aplicativo totalmente separado que também opera no mesmo stream.
- A KCL cria uma tabela do DynamoDB com o nome do aplicativo e usa essa tabela para manter informações de estado (como pontos de verificação e mapeamento de operador-estilhaço) para o aplicativo. Cada aplicativo tem sua própria tabela do DynamoDB. Para obter mais informações, consulte [Utilizar uma tabela de arrendamento para rastrear os painéis processados pela aplicação do consumidor KCL](#) (p. 121).

## Configurar credenciais

Você precisa disponibilizar as credenciais da AWS para um dos provedores de credenciais na cadeia de provedores de credencial padrão. Por exemplo, se você estiver executando o consumidor em uma instância do EC2, recomendamos que você inicie a instância com uma função do IAM. As credenciais da AWS que refletem as permissões associadas à função do IAM são disponibilizadas aos aplicativos na instância por meio de metadados da instância. Essa é a maneira mais segura de gerenciar credenciais para um consumidor em execução em uma instância do EC2.

O aplicativo de exemplo primeiro tentará recuperar as credenciais do IAM 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 metadados da instância, consulte [Metadados da instância](#) no Guia do usuário do Amazon EC2 para instâncias do Linux.

## Usar o ID do operador 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. Alterar a construção do trabalhador para utilizar `Worker.Builder`. Por exemplo:

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

## Desenvolver um consumidor de Kinesis Client Library em Node.js

Você pode usar a Kinesis Client Library (KCL) para criar aplicativos que processam dados dos streamings de dados do Kinesis. A Kinesis Client Library está disponível em vários idiomas. Este tópico discute Node.js.

O KCL é uma biblioteca Java; o suporte a linguagens que não sejam Java é dado usando-se uma interface multilíngue chamada `MultiLangDaemon`. Este daemon se baseia em Java e é executado em segundo plano quando você usa uma linguagem KCL diferente de Java. Portanto, se você instalar o KCL para Node.js e escrever o aplicativo de consumidor inteiramente em Node.js, ainda precisará do Java instalado no sistema por causa do `MultiLangDaemon`. Além disso, o `MultiLangDaemon` tem algumas configurações padrão que você pode personalizar para o caso de uso. Por exemplo, a região da AWS à qual ele se conecta. Para obter mais informações sobre o `MultiLangDaemon` no GitHub, acesse a página Projeto `MultiLangDaemon` da [KCL](#).

Para fazer download da KCL de Node.js no GitHub, acesse a [Kinesis Client Library \(Node.js\)](#).

### Downloads de códigos de exemplo

Há dois exemplos de código disponíveis para KCL em Node.js:

- [basic-sample](#)

Usado nas seções a seguir para ilustrar os conceitos básicos de criação de um aplicativo de consumidor da KCL em 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 há um arquivo README com mais informações.

Você precisa concluir as tarefas a seguir ao implementar um aplicativo de consumidor da KCL em Node.js:

### Tarefas

- [Implementar o processador de registros \(p. 138\)](#)
- [Modificar as propriedades de configuração \(p. 139\)](#)



## Implementar o processador de registros

O consumidor mais simples possível utilizando o KCL para o Node.js deve implementar um `recordProcessor` função, que por sua vez contém as funções `initialize`, `processRecords`, e `shutdown`. A amostra fornece uma implementação que pode utilizar como ponto de partida (consulte `sample_kcl_app.js`).

```
function recordProcessor() {  
  // return an object that implements initialize, processRecords and shutdown functions.}
```

### `initialize`

A KCL chama a função `initialize` quando o processador de registros 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 o Kinesis Data Streams tem semântica pelo menos uma vez, o que significa que cada registro de dados de um estilhaço 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 [Reestilhaçamento, escalabilidade e processamento paralelo](#) (p. 183).

```
initialize: function(initializeInput, completeCallback)
```

### `processRecords`

A KCL chama essa função com entrada que contém uma lista de registros de dados do estilhaço especificado para a função `initialize`. 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 o processador de registros para rastrear os registros que já foram processados em um estilhaço. O KCL cuida deste seguimento com um `checkpoint` objeto passou como `processRecordsInput.checkpointer`. O seu processador de registro chama para o `checkpointer.checkpoint` para informar o KCL até que ponto progrediu no processamento dos registros no rodapé. Caso o operador falhe, a KCL usa essas informações quando você reinicia o processamento do estilhaço para que ele continue a partir do último registro processado conhecido.



Em uma operação de divisão ou mesclagem, a KCL não começa a processar os novos estilhaços até que os processadores dos estilhaços originais tenham chamado `checkpoint` para indicar que o processamento dos estilhaços originais está completo.

Se você não passar o número de sequência para a função `checkpoint`, a KCL assumirá que a chamada para `checkpoint` indica que todos os registros foram processados, até o último registro que foi passado ao 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 de contactar `checkpoint` em cada chamada para `processRecords`. Um processador poderia, por exemplo, ligar `checkpoint` em cada terceira chamada, ou algum evento externo ao seu processador de registro, tal como um serviço de verificação/validação personalizado que implementou.

Opcionalmente, pode especificar o número de sequência exacto de um registro como um parâmetro para `checkpoint`. Neste caso, o KCL assume que todos os registros foram processados apenas até esse registro.

O aplicativo de exemplo básico mostra a chamada mais simples possível para a função `checkpoint`. Você pode adicionar outra lógica de verificação que precisar para o consumidor neste ponto da função.

`shutdown`

A KCL chama a função `shutdown` quando o processamento termina (`shutdownInput.reason` é `TERMINATE`) ou quando o operador não está mais respondendo (o `shutdownInput.reason` é `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` para `shutdown`. Se o motivo do encerramento for `TERMINATE`, deve certificar-se de que o processador de registros terminou o processamento de quaisquer registros de dados e, em seguida, ligue para o `checkpoint` função nesta interface.

## Modificar 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 do aplicativo

A KCL exige um aplicativo exclusivo entre seus aplicativos e entre suas 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 executa uma instância adicional do mesmo código de aplicativo, mas com um nome de aplicativo diferente, a KCL trata a segunda instância como um aplicativo totalmente separado que também opera no mesmo stream.
- A KCL cria uma tabela do DynamoDB com o nome do aplicativo e usa essa tabela para manter informações de estado (como pontos de verificação e mapeamento de operador-estilhaço) para o aplicativo. Cada aplicativo tem sua própria tabela do DynamoDB. Para obter mais informações, consulte [Utilizar uma tabela de arrendamento para rastrear os painéis processados pela aplicação do consumidor KCL](#) (p. 121).

## Configurar credenciais

Você precisa disponibilizar as credenciais da AWS para um dos provedores de credenciais na cadeia de provedores de credencial 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 o consumidor em uma instância do Amazon EC2, recomendamos configurar a instância com uma função do IAM. As credenciais da AWS que refletem as permissões associadas à função do IAM são disponibilizadas aos aplicativos na instância por meio de metadados da instância. Essa é a maneira mais segura de gerenciar credenciais para um aplicativo de consumidor em execução em uma instância do EC2.

O exemplo a seguir configura a KCL para processar um Kinesis data stream chamado `kclnodejssample` 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
```

## Desenvolver um consumidor de Kinesis Client Library em .NET

Você pode usar a Kinesis Client Library (KCL) para criar aplicativos que processam dados dos streamings de dados do Kinesis. A Kinesis Client Library está disponível em vários idiomas. Este tópico discute .NET.

O KCL é uma biblioteca Java; o suporte a linguagens que não sejam Java é dado usando-se uma interface multilíngue chamada `MultiLangDaemon`. Este daemon se baseia em Java e é executado em segundo plano quando você usa uma linguagem KCL diferente de Java. Portanto, se você instalar o KCL para .NET e escrever o aplicativo de consumidor inteiramente em .NET, ainda precisará do Java instalado no sistema por causa do `MultiLangDaemon`. Além disso, o `MultiLangDaemon` tem algumas configurações padrão que você pode personalizar para o caso de uso. Por exemplo, a região da AWS à qual ele se conecta. Para obter mais informações sobre o `MultiLangDaemon` no GitHub, acesse a página [Projeto MultiLangDaemon da KCL](#).

Para fazer download da KCL de .NET no GitHub, acesse a [Kinesis Client Library \(.NET\)](#). Para fazer download do código de exemplo para um aplicativo de consumidor da KCL de .NET, acesse a [página de projeto de consumidor de exemplo da KCL de .NET](#) no GitHub.

Você precisa concluir as tarefas a seguir ao implementar um aplicativo de consumidor da KCL em .NET:

### Tarefas

- [Implemente os métodos de classe `IRecordProcessor` \(p. 140\)](#)
- [Modificar as propriedades de configuração \(p. 142\)](#)

### Implemente os métodos de classe `IRecordProcessor`

O consumidor deve implementar os seguintes métodos para `IRecordProcessor`. O consumidor de amostras fornece implementações que pode utilizar como ponto de partida (consulte o `SampleRecordProcessor` classe em `SampleConsumer/AmazonKinesisSampleConsumer.cs`).

```
public void Initialize(InitializationInput input)
```

```
public void ProcessRecords(ProcessRecordsInput input)
public void Shutdown(ShutdownInput input)
```

### Initialize

A KCL chama esse método quando o processador de registros é instanciado, passando um ID de estilhaço específico no parâmetro `input` (`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 o Kinesis Data Streams tem semântica pelo menos uma vez, o que significa que cada registro de dados de um estilhaço 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 [Reestilhaçamento, escalabilidade e processamento paralelo](#) (p. 183).

```
public void Initialize(InitializationInput input)
```

### ProcessRecords

A KCL chama esse método, passando uma lista de registros de dados no parâmetro `input` (`input.Records`) do estilhaço especificado pelo método `Initialize`. 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 o processador de registros para rastrear os registros que já foram processados em um estilhaço. A KCL cuida desse rastreamento para você passando um objeto `Checkpoint` para `ProcessRecords` (`input.Checkpointer`). O processador de registros chama o método `Checkpoint` para informar à KCL sobre o progresso do processamento dos registros no estilhaço. Caso o operador falhe, a KCL usa essas informações para reiniciar o processamento do estilhaço no último registro processado conhecido.

Em uma operação de divisão ou mesclagem, a KCL não começa a processar os novos estilhaços até que os processadores dos estilhaços originais tenham chamado `Checkpoint` para indicar que o processamento dos estilhaços originais está completo.

Se você não passar um parâmetro, a KCL assumirá que a chamada para `Checkpoint` indica 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 de contactar `Checkpoint` em cada chamada para `ProcessRecords`. Um processador poderia, por exemplo, ligar `Checkpoint` em cada terceiro ou quarto

telefone. Opcionalmente, pode especificar o número de sequência exacto de um registo como um parâmetro para `Checkpointter.Checkpoint`. Neste caso, o KCL assume que os registos foram processados apenas até esse registo.

No exemplo, o método privado `Checkpoint(Checkpointer checkpointer)` mostra como chamar o método `Checkpointter.Checkpoint` usando a lógica de novas tentativas e o tratamento de exceções apropriados.

A KCL para .NET lida com exceções de maneira diferente das outras bibliotecas de idioma da KCL, pois não lida com nenhuma exceção que surja do processamento de registos de dados. As exceções não detectadas do código do usuário causam uma falha no programa.

### Shutdown

A KCL chama o método `Shutdown` quando o processamento termina (o motivo do desligamento é `TERMINATE`) ou quando o operador não está mais respondendo (o valor de `input.Reason` de desligamento é `ZOMBIE`).

```
public void Shutdown(ShutdownInput input)
```

O processamento termina quando o processador de registos não recebe mais registos do estilhaço porque ele foi dividido ou intercalado, ou o stream foi excluído.

O KCL também passa um `Checkpointter` para `shutdown`. Se o motivo do encerramento for `TERMINATE`, o processador de registos deve terminar o processamento de quaisquer registos de dados e, em seguida, ligar para o `checkpoint` método nesta interface.

### Modificar 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 do aplicativo

A KCL exige um aplicativo exclusivo entre seus aplicativos e entre suas 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 executa uma instância adicional do mesmo código de aplicativo, mas com um nome de aplicativo diferente, a KCL trata a segunda instância como um aplicativo totalmente separado que também opera no mesmo stream.
- A KCL cria uma tabela do DynamoDB com o nome do aplicativo e usa essa tabela para manter informações de estado (como pontos de verificação e mapeamento de operador-estilhaço) para o aplicativo. Cada aplicativo tem sua própria tabela do DynamoDB. Para obter mais informações, consulte [Utilizar uma tabela de arrendamento para rastrear os painéis processados pela aplicação do consumidor KCL](#) (p. 121).

### Configurar credenciais

Você precisa disponibilizar as credenciais da AWS para um dos provedores de credenciais na cadeia de provedores de credencial 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 o aplicativo de consumidor em uma instância do EC2, recomendamos que configure a instância com uma

função do IAM. As credenciais da AWS que refletem as permissões associadas à função do IAM são disponibilizadas aos aplicativos na instância por meio de metadados da instância. Essa é a maneira mais segura de gerenciar credenciais para um consumidor em execução em uma instância do EC2.

O arquivo de propriedades do exemplo configura a KCL para processar um Kinesis data stream chamado "words" usando o processador de registros fornecido em `AmazonKinesisSampleConsumer.cs`.

## Desenvolver um consumidor de Kinesis Client Library em Python

Você pode usar a Kinesis Client Library (KCL) para criar aplicativos que processam dados dos streamings de dados do Kinesis. A Kinesis Client Library está disponível em vários idiomas. Este tópico discute Python.

O KCL é uma biblioteca Java; o suporte a linguagens que não sejam Java é dado usando-se uma interface multilíngue chamada MultiLangDaemon. Este daemon se baseia em Java e é executado em segundo plano quando você usa uma linguagem KCL diferente de Java. Portanto, se você instalar o KCL para Python e escrever o aplicativo de consumidor inteiramente em Python, ainda precisará do Java instalado no sistema por causa do MultiLangDaemon. Além disso, o MultiLangDaemon tem algumas configurações padrão que você pode personalizar para o caso de uso. Por exemplo, a região da AWS à qual ele se conecta. Para obter mais informações sobre o MultiLangDaemon no GitHub, acesse a página Projeto MultiLangDaemon da [KCL](#).

Para fazer download da [Kinesis Client Library \(Python\)](#) do Python no GitHub, acesse KCL. Para fazer download do código de exemplo para um aplicativo de consumidor da KCL do Python, acesse a página do [projeto de exemplo da KCL para Python](#) no GitHub.

Você precisa concluir as tarefas a seguir ao implementar um aplicativo de consumidor da KCL em Python:

### Tarefas

- [Implemente os métodos de classe RecordProcessor \(p. 143\)](#)
- [Modificar as propriedades de configuração \(p. 145\)](#)

### Implemente os métodos de classe RecordProcessor

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, checkpoint)
def shutdown(self, checkpoint, reason)
```

#### initialize

A KCL chama o método `initialize` quando o processador de registros é instanciado, passando um ID de estilhaço específico como um 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 o Kinesis Data Streams tem semântica pelo menos uma vez, o que significa que cada registro de dados de um estilhaço 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 [Reestilhaçamento, escalabilidade e processamento paralelo \(p. 183\)](#).

```
def initialize(self, shard_id)
```

## process\_records

A KCL chama esse método, passando uma lista de registros de dados do estilhaço especificado pelo método `initialize`. 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, checkpoint)
```

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 o processador de registros para rastrear os registros que já foram processados em um estilhaço. O KCL cuida deste seguimento por passar um `Checkpoint` para `process_records`. O processador de registros chama para o `checkpoint` neste objeto para informar o KCL até que ponto progrediu no processamento dos registros no rodapé. Caso o operador falhe, a KCL usa essas informações para reiniciar o processamento do estilhaço no último registro processado conhecido.

Em uma operação de divisão ou mesclagem, a KCL não começa a processar os novos estilhaços até que os processadores dos estilhaços originais tenham chamado `checkpoint` para indicar que o processamento dos estilhaços originais está completo.

Se você não passar um parâmetro, a KCL assumirá 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 de contactar `checkpoint` em cada chamada para `process_records`. Um processador poderia, por exemplo, ligar `checkpoint` em cada terceira chamada. Opcionalmente, pode especificar o número de sequência exacto de um registro como um parâmetro para `checkpoint`. Neste caso, o KCL assume que todos os registros foram processados apenas até esse registro.

No exemplo, o método privado `checkpoint` mostra como chamar o método `Checkpoint.checkpoint` usando a lógica de novas tentativas e o tratamento de exceções apropriados.

A KCL depende do `process_records` para gerenciar todas as exceções que surjam do processamento de registros de dados. Se ocorrer uma exceção em `process_records`, a KCL ignorará os registros de dados que foram passados para `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

A KCL chama o método `shutdown` quando o processamento termina (o motivo do desligamento é `TERMINATE`) ou quando o operador não está mais respondendo (a `reason` do desligamento é `ZOMBIE`).

```
def shutdown(self, checkpoint, reason)
```

O processamento termina quando o processador de registros não recebe mais registros do estiloção porque ele foi dividido ou intercalado, ou o stream foi excluído.

O KCL também passa um `Checkpoint` para `shutdown`. Se o encerramento `reason` é `TERMINATE`, o processador de registros deve terminar o processamento de quaisquer registros de dados e, em seguida, ligar para o `checkpoint` método nesta interface.

### Modificar 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 do aplicativo

A KCL exige um nome do 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 executa uma instância adicional do mesmo código de aplicativo, mas com um nome de aplicativo diferente, a KCL trata a segunda instância como um aplicativo totalmente separado que também opera no mesmo stream.
- A KCL cria uma tabela do DynamoDB com o nome do aplicativo e usa essa tabela para manter informações de estado (como pontos de verificação e mapeamento de operador-estiloção) para o aplicativo. Cada aplicativo tem sua própria tabela do DynamoDB. Para obter mais informações, consulte [Utilizar uma tabela de arrendamento para rastrear os painéis processados pela aplicação do consumidor KCL](#) (p. 121).

### Configurar credenciais

Você precisa disponibilizar as credenciais da AWS para um dos provedores de credenciais na cadeia de provedores de credencial 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 o aplicativo de consumidor em uma instância do Amazon EC2, recomendamos que configure a instância com uma função do IAM. As credenciais da AWS que refletem as permissões associadas à função do IAM são disponibilizadas aos aplicativos na instância por meio de metadados da instância. Essa é a maneira mais segura de gerenciar credenciais para um aplicativo de consumidor em execução em uma instância do EC2.

O arquivo de propriedades do exemplo configura a KCL para processar um Kinesis data stream chamado "words" usando o processador de registros fornecido em `sample_kclpy_app.py`.

### Desenvolver um consumidor de Kinesis Client Library em Ruby

Você pode usar a Kinesis Client Library (KCL) para criar aplicativos que processam dados dos streamings de dados do Kinesis. A Kinesis Client Library está disponível em vários idiomas. Este tópico discute Ruby.

O KCL é uma biblioteca Java; o suporte a linguagens que não sejam Java é dado usando-se uma interface multilíngue chamada `MultiLangDaemon`. Este daemon se baseia em Java e é executado em segundo plano quando você usa uma linguagem KCL diferente de Java. Portanto, se você instalar o KCL para Ruby e escrever o aplicativo de consumidor inteiramente em Ruby, ainda precisará do Java instalado no sistema por causa do `MultiLangDaemon`. Além disso, o `MultiLangDaemon` tem algumas configurações padrão que você pode personalizar para o caso de uso. Por exemplo, a região da AWS à qual ele se conecta. Para



obter mais informações sobre o MultiLangDaemon no GitHub, acesse a página Projeto MultiLangDaemon da [KCL](#).

Para fazer download da KCL do Ruby no GitHub, acesse [Kinesis Client Library \(Ruby\)](#). Para fazer download do código de exemplo para um aplicativo de consumidor da KCL do Ruby, acesse a página do [Projeto de exemplo da KCL para Ruby](#) no GitHub.

Para obter mais informações sobre a biblioteca de suporte do Ruby da KCL, consulte [KCL Ruby Gems Documentation](#).

## Desenvolver consumidores do KCL 2.x

Este tópico mostra como usar a versão 2.0 da Kinesis Client Library (KCL). Para obter mais informações sobre a KCL, consulte a visão geral fornecida em [Desenvolver consumidores usando a Kinesis Client Library 1.x](#).

### Tópicos

- [Desenvolver um consumidor da Kinesis Client Library em Java \(p. 146\)](#)
- [Desenvolver um consumidor da Kinesis Client Library em Python \(p. 151\)](#)

## Desenvolver um consumidor da Kinesis Client Library em Java

O código seguinte mostra uma implementação de exemplo em Java de `ProcessorFactory` e `RecordProcessor`. Se quiser tirar partido da funcionalidade de ventilador melhorada, consulte [Utilização de consumidores com ventilador melhorado](#).

```
/*
 * 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);
    }
}
}
```

## Desenvolver um consumidor da Kinesis Client Library em Python

Você pode usar a Kinesis Client Library (KCL) para criar aplicativos que processam dados dos streamings de dados do Kinesis. A Kinesis Client Library está disponível em vários idiomas. Este tópico discute Python.

O KCL é uma biblioteca Java; o suporte a linguagens que não sejam Java é dado usando-se uma interface multilíngue chamada MultiLangDaemon. Este daemon se baseia em Java e é executado em segundo plano quando você usa uma linguagem KCL diferente de Java. Portanto, se você instalar o KCL para Python e escrever o aplicativo de consumidor inteiramente em Python, ainda precisará do Java instalado no sistema por causa do MultiLangDaemon. Além disso, o MultiLangDaemon tem algumas configurações padrão que você pode personalizar para o caso de uso. Por exemplo, a região da AWS à qual ele se

conecta. Para obter mais informações sobre o MultiLangDaemon no GitHub, acesse a página [Projeto MultiLangDaemon da KCL](#).

Para fazer download da [Kinesis Client Library \(Python\)](#) do Python no GitHub, acesse KCL. Para fazer download do código de exemplo para um aplicativo de consumidor da KCL do Python, acesse a página do [projeto de exemplo da KCL para Python](#) no GitHub.

Você precisa concluir as tarefas a seguir ao implementar um aplicativo de consumidor da KCL em Python:

#### Tarefas

- [Implemente os métodos de classe RecordProcessor \(p. 143\)](#)
- [Modificar as propriedades de configuração \(p. 145\)](#)

### Implemente os métodos de classe RecordProcessor

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 record.
    :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 checkpoint
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()
```

## Modificar 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 may 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 RecordProcessorCheckpoint#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 do aplicativo

A KCL exige um nome do 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 executa uma instância adicional do mesmo código de aplicativo, mas com um nome de aplicativo diferente, a KCL trata a segunda instância como um aplicativo totalmente separado que também opera no mesmo stream.
- A KCL cria uma tabela do DynamoDB com o nome do aplicativo e usa essa tabela para manter informações de estado (como pontos de verificação e mapeamento de operador-estilho) para o

aplicativo. Cada aplicativo tem sua própria tabela do DynamoDB. Para obter mais informações, consulte [Utilizar uma tabela de arrendamento para rastrear os painéis processados pela aplicação do consumidor KCL](#) (p. 121).

## Credentials

Você precisa disponibilizar as credenciais da AWS para um dos provedores de credenciais na [cadeia de provedores de credencial padrão](#). Você pode usar a propriedade `AWSCredentialsProvider` para definir um provedor de credenciais. Se você executar o aplicativo de consumidor em uma instância do Amazon EC2, recomendamos que configure a instância com uma função do IAM. As credenciais da AWS que refletem as permissões associadas à função do IAM são disponibilizadas aos aplicativos na instância por meio de metadados da instância. Essa é a maneira mais segura de gerenciar credenciais para um aplicativo de consumidor em execução em uma instância do EC2.

# Desenvolver consumidores personalizados com taxa de transferência compartilhada usando o AWS SDK for Java

Um dos métodos de desenvolvimento personalizado Kinesis Data Streams os consumidores com partilha em todo o mundo devem utilizar o Amazon Kinesis Data Streams API. Esta secção descreve a utilização do Kinesis Data Streams API com o AWS SDK para Java. O código de amostra Java nesta secção demonstra como realizar operações básicas de API KDS 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.

Pode ligar para o Kinesis Data Streams utilizando outros idiomas de programação diferentes. Para obter mais informações sobre todos os SDKs da AWS disponíveis, consulte [Comece a desenvolver usando a Amazon Web Services](#).

## Important

O método recomendado para o desenvolvimento personalizado Kinesis Data Streams os consumidores com partilha em todo o mundo são utilizar a Kinesis Client Library (KCL). A KCL ajuda-o a consumir e processar dados de um fluxo de dados da Kinesis, ao tomar conta de muitas das tarefas complexas associadas ao cálculo distribuído. Para mais informações, consulte [Desenvolver consumidores personalizados com rendimento partilhado utilizando KCL](#).

## Tópicos

- [Como obter dados de um stream](#) (p. 157)
- [Como usar iteradores de estilhaços](#) (p. 158)
- [Como usar o GetRecords](#) (p. 159)
- [Como adaptar a uma refragmentação](#) (p. 160)

## Como obter dados de um stream

O Kinesis Data Streams incluindo as `getShardIterator` e `getRecords` métodos que pode invocar para recuperar registos de um fluxo de dados. Este é o modelo de pull, onde o seu código recolhe registos de dados diretamente dos sombreados do fluxo de dados.

## Important

Recomendamos que utilize o apoio do processador de registo fornecido por KCL para recuperar registos dos seus fluxos de dados. Este é o modelo de push, onde implementa o código que

processa os dados. O KCL recupera registros de dados do fluxo de dados e entrega-os ao seu código de aplicação. Além disso, a KCL fornece as funcionalidades de failover, recuperação e balanceamento de carga. Para mais informações, consulte [Desenvolver consumidores personalizados com rendimento partilhado utilizando KCL](#).

Contudo, em alguns casos poderá preferir utilizar o Kinesis Data Streams API. Por exemplo, para implementar ferramentas personalizadas para monitorizar ou depurar os seus fluxos de dados.

#### Important

O Kinesis Data Streams aceita alterações feitas no período de retenção de registros de dados do stream de dados. Para obter mais informações, consulte [Alterar o período de retenção de dados \(p. 83\)](#).

## Como usar iteradores de estilhaços

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). Para trabalhar com o iterador de estilhaços, você precisa recuperar o estilhaço, conforme abordado em [cvbestream API - Preterido \(p. 77\)](#).

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 na sua conta da AWS, consulte [Listar streams \(p. 74\)](#). Para obter informações sobre como obter os estilhaços em um stream, consulte [cvbestream API - Preterido \(p. 77\)](#).

```
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 os registros devem ser retornados a partir do primeiro registro adicionado ao estilhaço — em vez de começar com o registro adicionado mais recentemente, também conhecido como a ponta. 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 dos registros. Para obter mais informações, consulte o exemplo de código em [Exemplo de PutRecord \(p. 104\)](#).

## Como usar o 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 o 10 MB dos 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, o aplicativo deve aguardar um tempo adequado para as fontes de dados do streaming, mas de pelo menos 1 segundo. Em seguida, tente obter novamente os dados do rodapé utilizando o analisador de derivação devolvido pela chamada anterior para `getRecords`. Existe cerca de uma latência de 3 segundos desde o momento em que um registro é adicionado ao fluxo até ao momento em que está disponível `getRecords`.

Passa 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 os dados. Esse comportamento é projetado e não indica um problema com a KCL ou seus dados.

## Como adaptar a uma refragmentação

Se `getRecordsResult.getNextShardIterator` devoluções `null`, indica que ocorreu uma divisão ou fusão que envolvia este shard. Esta camisola está agora numa `CLOSED` estado e leu todos os registos de dados disponíveis desta pasta.

Neste cenário, pode utilizar `getRecordsResult.childShards` para saber mais sobre os canais de criança do rodapé que estão a ser processados que foram criados pela divisão ou junção. Para mais informações, consulte [bebé](#).

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. Esta é a base para a qual `getRecordsResult.getNextShardIterator` devolvido `null`. Se a ordem dos dados for importante para a sua aplicação, certifique-se de que lê também todos os dados do outro folheto principal antes de ler quaisquer dados novos da pasta de criança criada pela junção.

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 reestilhaçamento, incluindo uma discussão sobre estados de estilhaços — como `CLOSED` — consulte [Reestilhaçar um stream \(p. 78\)](#).

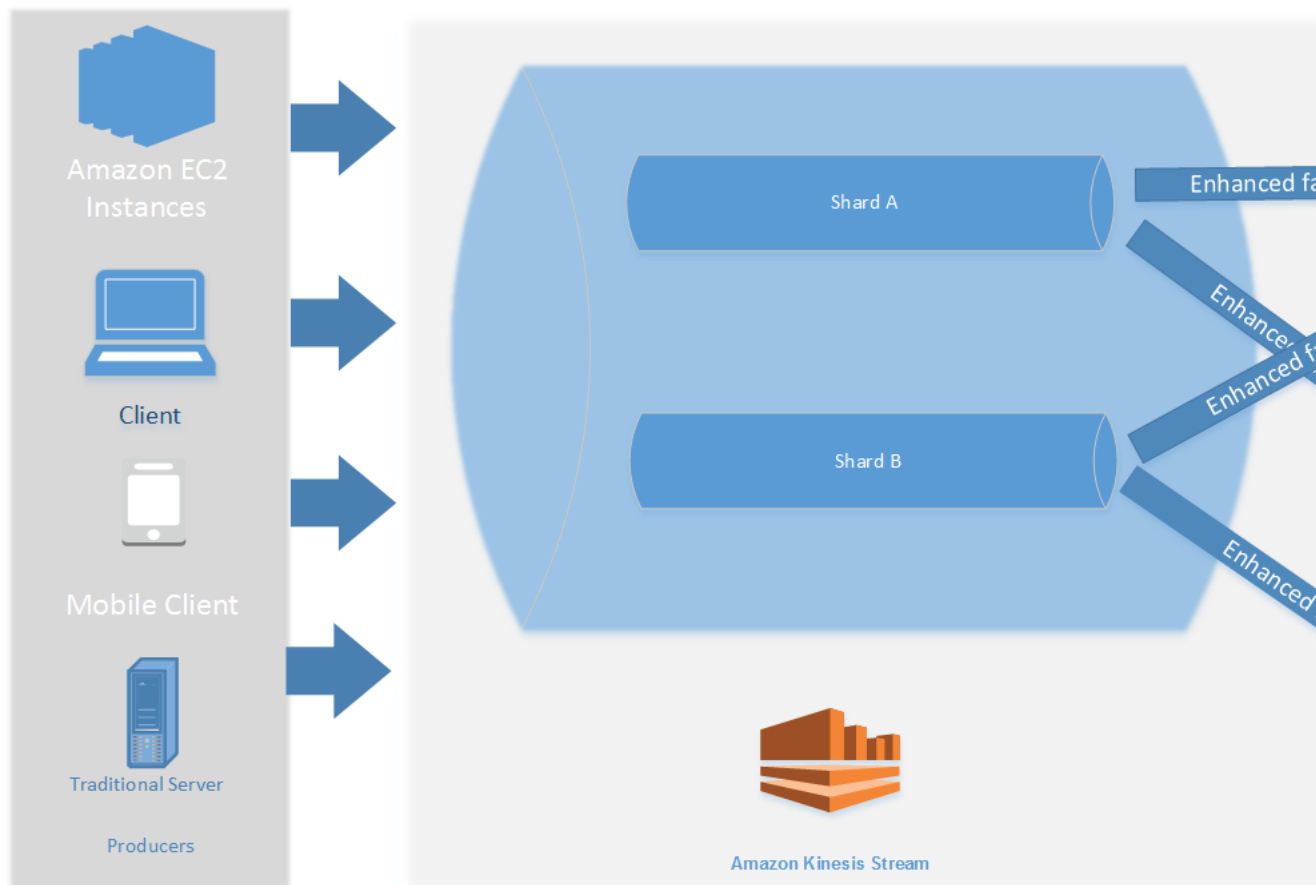
## Desenvolver consumidores personalizados com taxa de transferência dedicada (distribuição avançada)

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 taxa de transferência de até 2 MB de dados por segundo por estilhaço. Essa taxa de transferência é dedicada, o que significa que os consumidores que usam distribuição avançada não precisam lidar com outros consumidores que estejam recebendo dados do streaming. O Kinesis Data Streams envia os registros de dados por push do streaming para os 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 usar a versão 2.0 ou posterior da Amazon Kinesis Client Library (KCL) para criar um consumidor, a KCL configurará o consumidor para usar a distribuição avançada para receber dados de todos os estilhaços do streaming. Se usar a API para criar um consumidor que use distribuição avançada, você poderá se inscrever em estilhaços individuais.



O diagrama mostra o seguinte:

- Um streaming com dois estilhaços.
- Dois consumidores que utilizam um ventilador melhorado para receber dados do fluxo: Consumidor X e Consumidor Y. Cada um dos dois consumidores é subscrito para todos os ganchos e todos os registros do fluxo. Se você usar a versão 2.0 ou posterior da KCL para criar um consumidor, a KCL inscreverá automaticamente esse consumidor em todos os estilhaços do streaming. Por outro lado, se usar a API para criar um consumidor, você poderá se inscrever em estilhaços 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

- [Desenvolver consumidores de distribuição avançada com o KCL 2.x \(p. 163\)](#)
- [Desenvolver consumidores de distribuição avançada com a API do Kinesis Data Streams \(p. 167\)](#)
- [Gerenciar consumidores de distribuição avançada com o Console de gerenciamento da AWS \(p. 169\)](#)



## Desenvolver consumidores de distribuição avançada com o KCL 2.x

Os consumidores que usam distribuição avançada no Amazon Kinesis Data Streams podem receber registros de um fluxo de dados com taxa de transferência dedicada de até 2 MB de dados por segundo por estilhaço. Esse tipo de consumidor não precisa lidar com outros consumidores que estejam recebendo dados do streaming. Para obter mais informações, consulte [Desenvolver consumidores personalizados com taxa de transferência dedicada \(distribuição avançada\)](#) (p. 161).

Você pode usar a versão 2.0 ou posterior da Kinesis Client Library (KCL) para desenvolver aplicativos que usam distribuição avançada para receber dados de streamings. A KCL inscreve automaticamente o aplicativo em todos os estilhaços de um streaming e garante que o aplicativo consumidor possa ler com um valor da taxa de transferência de 2 MB/s por estilhaço. Se você quiser usar a KCL sem ativar a distribuição avançada, consulte [Desenvolver consumidores usando a Kinesis Client Library 2.0](#).

### Tópicos

- [Desenvolver consumidores de distribuição avançada usando o KCL 2.x em Java](#) (p. 163)

## Desenvolver consumidores de distribuição avançada usando o KCL 2.x em Java

É possível usar a versão 2.0 ou posterior da Kinesis Client Library (KCL) para desenvolver aplicativos no Amazon Kinesis Data Streams para receber dados de streamings usando a distribuição avançada. 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);
    }
}
}
```

## Desenvolver consumidores de distribuição avançada com a API do Kinesis Data Streams

Distribuição avançada é um recurso do Amazon Kinesis Data Streams que permite que os consumidores recebam registros de um fluxo de dados com taxa de transferência dedicada de até 2 MB de dados por segundo por estilhaço. Um consumidor que usa distribuição avançada não precisa lidar com outros consumidores que estejam recebendo dados do streaming. Para obter mais informações, consulte [Desenvolver consumidores personalizados com taxa de transferência dedicada \(distribuição avançada\)](#) (p. 161).

Você pode usar operações de API para criar um consumidor que use distribuição avançada no Kinesis Data Streams.

Para registrar um consumidor com distribuição avançada usando a API do Kinesis Data Streams

1. Chame [RegisterStreamConsumer](#) para registrar seu aplicativo como um consumidor que usa distribuição avançada. O Kinesis Data Streams gera um nome de recurso da Amazon (ARN) para o consumidor e o retorna na resposta.
2. Para começar a ouvir a um determinado estilhaço, passe o ARN do consumidor em uma chamada para [SubscribeToShard](#). O Kinesis Data Streams, então, começa a enviar os registros do estilhaço a você, na forma de eventos do tipo [SubscribeToShardEvent](#) por meio de uma conexão HTTP/2. A conexão permanece aberta por até 5 minutos. Chame [SubscribeToShard](#) novamente caso você queira continuar a receber registros do estilhaço após *future*, que é retornado pela chamada para [SubscribeToShard](#), for concluído normal ou excepcionalmente.

### Note

[SubscribeToShard](#) A API também devolve a lista das derivações descendentes da corrente atual quando o final do impulso actual for atingido.

3. Para cancelar o registro de um consumidor que esteja usando a distribuição avançada, chame [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);
}
```

Se `event.ContinuationSequenceNumber` devoluções `null`, indica que ocorreu uma divisão ou fusão que envolvia este shard. Esta camisa está agora numa `CLOSED` estado e leu todos os registros de dados disponíveis desta pasta. Neste cenário, por exemplo acima, pode utilizar `event.childShards` para saber mais sobre os canais de criança do rodapé que estão a ser processados que foram criados pela divisão ou junção. Para mais informações, consulte [bebé](#).

## Gerenciar consumidores de distribuição avançada com o Console de gerenciamento da AWS

Os consumidores que usam distribuição avançada no Amazon Kinesis Data Streams podem receber registros de um fluxo de dados com taxa de transferência dedicada de até 2 MB de dados por segundo por estilhaço. Para obter mais informações, consulte [Desenvolver consumidores personalizados com taxa de transferência dedicada \(distribuição avançada\)](#) (p. 161).

Você pode usar o Console de gerenciamento da AWS para ver uma lista de todos os consumidores registrados para usar a distribuição avançada com um streaming específico. Para cada consumidor, você pode ver detalhes, como ARN, status e data de criação, além das métricas de monitoramento e as tags associadas ao consumidor.

Para visualizar consumidores registrados para usar distribuição avançada, o status, a data de criação e as métricas no console

1. Faça login no Console de gerenciamento da AWS e abra o console do Kinesis em <https://console.aws.amazon.com/kinesis>.
2. Selecione Data Streams (Streams de dados) no painel de navegação.
3. Escolha um streaming de dados do Kinesis para ver os 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 (Streams de dados) no painel de navegação.
3. Escolha um streaming de dados do Kinesis para ver os 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).

## Migrar consumidores do KCL 1.x para o KCL 2.x

Este tópico explica as diferenças entre as versões 1.x e 2.x da Kinesis Client Library (KCL). Ele também mostra como migrar o consumidor da versão 1.x para a versão 2.x da 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 da KCL apresenta as seguintes alterações de interface:

## Alterações de interface da KCL

## Tópicos

## Migrar o processador de registros

```
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.IRecordProcessorCheckpoint;
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.getCheckpoint().checkpoint();
            } catch (ShutdownException | InvalidStateException e) {
            }
        }
    }
}
```



```
        throw new RuntimeException(e);
    }
}

@Override
public void shutdownRequested(IRecordProcessorCheckpoint checkpoint) {
    try {
        checkpoint.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.IShutdownNotificationAware`  
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. Substituir o `shutdown` com os seguintes métodos novos: `leaseLost`, `shardEnded`, e `shutdownRequested`.

```
// @Override
// public void shutdownRequested(IRecordProcessorCheckpoint checkpoint) {
//
//     //
//     // This is moved to shardEnded(...)
//     //
//     try {
//         checkpoint.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(ShutdownRequestedInput)  
//     //  
//     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();  
    }  
}  
}
```

## Migrar a Fábrica do Processador de Registros

A fábrica do processador de registros é responsável por criar processadores de registro quando uma concessão é realizada. Veja a seguir um exemplo de 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.IRecordProcessorFactory` 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();
    }
}
```

## Migração do operador

Na versão 2.0 da KCL, uma nova classe, chamada Scheduler, substitui a classe Worker. Veja a seguir um exemplo de um operador da 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 foi movida de uma única classe de configuração (`KinesisClientLibConfiguration`) para seis classes de configuração. A tabela a seguir descreve a migração.

### Campos de Configuração e suas Novas Classes

Campo Original	Nova classe de configuração	Description (Descrição)
<code>applicationName</code>	<code>ConfigsBuilder</code>	O nome do aplicativo KCL. Usado como padrão para o <code>tableName</code> e o <code>consumerName</code> .
<code>tableName</code>	<code>ConfigsBuilder</code>	Permite substituir o nome da tabela usada para a tabela de concessão do Amazon DynamoDB.
<code>streamName</code>	<code>ConfigsBuilder</code>	O nome do stream a partir do qual esse aplicativo processa registros.
<code>kinesisEndpoint</code>	<code>ConfigsBuilder</code>	Essa opção não existe mais. Consulte remoções de configuração de cliente.

Campo Original	Nova classe de configuração	Description (Descrição)
dynamoDBEndpoint	ConfigsBuilder	Essa opção não existe mais. Consulte remoções de configuração de cliente.
initialPositionInStreamExtended	StreamExtendedConfig	A localização no estilhaço a partir da qual a KCL começa a obter 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	ConfigdeGerencia	Tempo em segundos 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	ConfigdeGerencia	Tempo em segundos entre chamadas de sincronização de estilhaços.
maxRecords	PollingConfig	Permite definir o número máximo de registros que o Kinesis retorna.
idleTimeBetweenReadsInMillis	StreamListenerConfig	Essa opção não existe mais. Consulte a remoção do tempo ocioso.
callProcessRecordsEvenIfEmpty	StreamListenerConfig	Quando definido, o processador de registros é solicitado mesmo quando nenhum registro foi fornecido pelo Kinesis.
parentShardPollIntervalInMillis	StreamListenerConfig	Com que frequência um processador de registros deve sondar a conclusão de estilhaços pai.
cleanupLeasesUponShardCompletion	ShardConsumer	Quando definidas, as concessões são removidas assim que as concessões filho iniciam o processamento.
ignoreUnexpectedChildShards	ShardConsumer	Quando definidos, estilhaços filho que possuem um estilhaço aberto são ignorados. Isso se refere principalmente ao DynamoDB Streams.
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.
metricsBufferTimeInMillis	MetricsConfig	Controla a publicação de métricas do CloudWatch.
metricsMaxQueueSize	MetricsConfig	Controla a publicação de métricas do CloudWatch.

Campo Original	Nova classe de configuração	Description (Descrição)
metricsLevel	MetricsConfig	Controla a publicação de métricas do CloudWatch.
metricsEnabledDimensions	MetricsConfig	Controla a publicação de métricas do CloudWatch.
validateSequenceNumberBeforeFetch	ClientConfiguration	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	LeaseManagement	O número máximo de concessões que uma única instância do aplicativo deve aceitar.
maxLeasesToStealAtOneTime	LeaseManagement	O número máximo de concessões que um aplicativo deve tentar roubar de uma só vez.
initialLeaseTableReaderName	LeaseManagement	A ID de leitura do DynamoDB que é usada se a Kinesis Client Library precisar criar uma nova tabela de concessão do DynamoDB.
initialLeaseTableWriterName	LeaseManagement	A ID de leitura do DynamoDB que é usada se a Kinesis Client Library precisar criar uma nova tabela de concessão do DynamoDB.
initialPositionInStream	ConfigGenerator	A posição inicial do aplicativo no stream. Isso é usado somente durante a criação da concessão inicial.
skipShardSyncAtWorkerInitialization	CoordinatorConfig	Desative a sincronização de dados de estilhaço se a tabela de concessão tiver concessões existentes. TODO kinesidry-438
shardPrioritization	CoordinatorConfig	A priorização de estilhaços a ser usada.
shutdownGraceMillis	N/D	Essa opção não existe mais. Consulte as remoções MultiLang.
timeoutInSeconds	N/D	Essa opção não existe mais. Consulte as remoções MultiLang.
retryGetRecordsInSeconds	PollingConfig	Configura o atraso entre as tentativas GetRecords para falhas.
maxGetRecordsThreads	PollingConfig	O tamanho do grupo de threads usado para GetRecords.
maxLeaseRenewalThreads	LeaseManagement	O tamanho do grupo de threads de renovação de concessão. Quanto mais concessões seu aplicativo aceitar, maior esse grupo deve ser.
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.
listShardsBackoffTimeInMills	LifeCycleConfig	O número de milissegundos de espera entre as chamadas para ListShards em caso de falha.

Campo Original	Nova classe de configuração	Description (Descrição)
<code>maxListShardsRetryAttempts</code>	<code>CoordinatorConfig</code>	O número máximo de novas tentativas de <code>ListShards</code> antes de desistir.

## Remoção do tempo ocioso

Na versão 1.x da KCL, 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#shardConsumerDispatchPollIntervalInMillis`.
- A quantidade de tempo inativo quando nenhum registro foi 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 estilohaço só ocorre quando uma solicitação é enviada.

## Remoções de configuração de cliente

Na versão 2.0, a 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
<code>kinesisEndpoint</code>	Configure o SDK <code>KinesisAsyncClient</code> com parâmetro de avaliação preferido: <code>KinesisAsyncClient.builder().endpointOverride(URI.create("https://&lt;kinesis endpoint&gt;")).build()</code> .
<code>dynamoDBEndpoint</code>	Configure o SDK <code>DynamoDbAsyncClient</code> com parâmetro de avaliação preferido: <code>DynamoDbAsyncClient.builder().endpointOverride(URI.create("https://&lt;dynamodb endpoint&gt;")).build()</code> .
<code>kinesisClientConfiguration</code>	Configure o SDK <code>KinesisAsyncClient</code> com a configuração necessária: <code>KinesisAsyncClient.builder().overrideConfiguration(&lt;your configuration&gt;).build()</code> .
<code>dynamoDBClientConfiguration</code>	Configure o SDK <code>DynamoDbAsyncClient</code> com a configuração necessária: <code>DynamoDbAsyncClient.builder().overrideConfiguration(&lt;your configuration&gt;).build()</code> .
<code>cloudWatchClientConfiguration</code>	Configure o SDK <code>CloudWatchAsyncClient</code> com a configuração necessária: <code>CloudWatchAsyncClient.builder().overrideConfiguration(&lt;your configuration&gt;).build()</code> .
<code>regionName</code>	Configure o SDK com a Região de sua preferência. Ela é a mesma para todos os clientes do SDK. Por exemplo, <code>KinesisAsyncClient.builder().region(Region.US_WEST_2).build()</code> .



# Solução de problemas de consumidores do Kinesis Data Streams

As seções a seguir oferecem soluções para alguns problemas comuns que você pode encontrar ao trabalhar com consumidores do Amazon Kinesis Data Streams.

- [Alguns registros do Kinesis Data Streams são ignorados ao usar a biblioteca de cliente do Kinesis \(p. 179\)](#)
- [Registros pertencentes ao mesmo estilhaço são processados por processadores de registros diferentes ao mesmo tempo \(p. 179\)](#)
- [Aplicativo consumidor está lendo a uma taxa menor que a esperada \(p. 180\)](#)
- [GetRecords retorna um array de registros vazios mesmo quando não há dados no stream \(p. 180\)](#)
- [Iterador do estilhaço expira inesperadamente \(p. 181\)](#)
- [Processamento de registros de consumidores ficando atrasados \(p. 181\)](#)
- [Erro de permissão de chave mestra do KMS não autorizada \(p. 182\)](#)

## 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 tratada emitida a partir de `processRecords`. O Kinesis Client Library (KCL) depende do seu `processRecords` código para tratar quaisquer exceções que surjam do processamento dos registros de dados. Qualquer exceção lançada em `processRecords` é absorvida pela KCL. Para evitar infinitas tentativas em uma falha recorrente, a KCL não reenvia o lote de registros processados no momento da exceção. A KCL chama `processRecords` para 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 estilhaço são processados por processadores de registros diferentes ao mesmo tempo

Para qualquer aplicativo da Kinesis Client Library (KCL) em execução, um estilhaço só tem um proprietário. No entanto, vários processadores de registro pode temporariamente processar o mesmo estilhaço. No caso de uma instância de operador que perde a conectividade com a rede, a KCL pressupõe que o operador inacessível não está mais processando registros, após o tempo de failover expirar, e instrui outras instâncias de operador a assumir. 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 estilhões executados por outro processador de registros, ele deve lidar com os dois casos seguintes para executar o encerramento normal:

1. Após a chamada atual a `processRecords` ser concluída, a KCL chama o método de desligamento no processador de registros com o motivo de desligamento 'ZOMBIE'. Espera-se que seus processadores de registros limpem quaisquer recursos conforme apropriado e, em seguida, saiam.
2. Quando tenta o ponto de controle de um trabalhador "zombie", o KCL a `ShutdownException`. Depois de receber esta exceção, espera-se que o seu código saia do método atual de forma limpa.

Para obter mais informações, consulte [Tratar registros duplicados \(p. 184\)](#).

## Aplicativo consumidor está lendo a uma taxa menor que a esperada

Os motivos mais comuns para a taxa de transferência 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 estilhão. Para obter mais informações, consulte [Cotas e limites do Kinesis Data Streams \(p. 6\)](#). Nesse caso, aumente o número de estilhões no Kinesis data stream.
2. O [limite](#) que especifica o número máximo de `GetRecords` por chamada pode ter sido configurado com um valor baixo. Se você estiver usando a KCL, poderá configurar o operador com um valor baixo para a propriedade `maxRecords`. Em geral, recomendamos o uso dos padrões do sistema para essa propriedade.
3. A lógica em sua chamada `processRecords` está demorando mais do que o esperado por várias razões possíveis. A lógica pode usar muitos recursos da CPU, bloquear a E/S ou estar afunilada na sincronização. Para testar se isso é verdadeiro, teste a execução de processadores de registros vazios e comparar a taxa de transferência de leitura. Para obter informações sobre como acompanhar os dados de entrada, consulte [Reestilhamento, escalabilidade e processamento paralelo \(p. 183\)](#).

Se você tem apenas um aplicativo consumidor, sempre é possível ler pelo menos duas vezes mais rápido do que a taxa de colocação. Isso porque você pode gravar até 1.000 registros por segundo para gravações, até uma taxa total máxima de gravação de dados de 1 MB por segundo (inclusive chaves de partição). Cada estilhão aberto pode suportar até 5 transações por segundo para leituras, até uma taxa total máxima 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 estilhão. 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 um array de registros vazios mesmo quando não há dados no stream

O consumo, ou a obtenção, de registros é um modelo de envio. Os desenvolvedores devem chamar [GetRecords](#) em um loop contínuo sem retrocesso. 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 estilhão.
2. Não há dados perto da parte do estilhão 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.

Num cenário de produção, a única altura em que o ciclo contínuo deve ser saído é quando `NextShardIterator` o valor é `NULL`. Quando `NextShardIterator` é `NULL`, significa que o engate atual foi fechado e o `ShardIterator` o valor iria de outra forma apontar para o último registro. Se o aplicativo consumidor nunca chamar `SplitShard` ou `MergeShards`, o estilo permanecerá aberto e as chamadas a `GetRecords` nunca retornarão um valor de `NextShardIterator` que seja `NULL`.

Se você usar a Kinesis Client Library (KCL), o padrão de consumo acima será abstraído para você. Isso inclui a manipulação automática de um conjunto de estilos que mudam dinamicamente. Com a KCL, o desenvolvedor fornece apenas a lógica para processar registros de entrada. Isso é possível porque a biblioteca faz chamadas contínuas a `GetRecords` para você.

## Iterador do estilo expira inesperadamente

Um novo iterador de estilo é retornado por toda solicitação a `GetRecords` (como `NextShardIterator`), que você usa na próxima solicitação `GetRecords` (como `ShardIterator`). Normalmente, esse iterador do estilo não expira antes de você usá-lo. No entanto, você pode descobrir que iteradores de estilo 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 estilo 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 locação. Essa situação tem maior probabilidade de ocorrer se você tiver um grande número de estilos. Para solucionar esse problema, aumente a capacidade de gravação atribuída à tabela do estilo. Para obter mais informações, consulte [Utilizar uma tabela de arrendamento para rastrear os painéis processados pela aplicação do consumidor KCL](#) (p. 121).

## Processamento de registros de consumidores ficando atrasados

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 estilos e consumidores no stream. Observe que, 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. Uma solução provisória rápida é aumentar o período de retenção. Isso interrompe a perda de dados importantes enquanto você soluciona o problema. Para obter mais informações, consulte [Monitoramento do serviço do Amazon Kinesis Data Streams com o Amazon CloudWatch](#) (p. 188). Em seguida, identifique até que ponto a sua aplicação do consumidor está a ler de cada placa utilizando um CloudWatch métrico emitido pelo Kinesis Client Library (KCL), `MillisBehindLatest`. Para mais informações, consulte [Monitoramento da Kinesis Client Library com o Amazon CloudWatch](#) (p. 203).

Veja os motivos mais comuns para consumidores ficarem atrasados:

- Grandes aumentos súbitos em `GetRecords.IteratorAgeMilliseconds` ou `MillisBehindLatest` geralmente indicam um problema transitório, como falhas de operação da API para um aplicativo de downstream. 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 taxa de transmissão do stream. Você pode verificar esse comportamento analisando as outras métricas personalizadas do CloudWatch emitidas pela KCL associadas à operação `processTask`, incluindo `RecordProcessor.processRecords.Time.Success` e `RecordsProcessed`.

- Caso veja um aumento na métrica `processRecords.Time` correlacionada ao aumento na taxa de transferência, você deve analisar a lógica do processamento de registros para identificar por que ela não está se dimensionando de acordo com a maior taxa de transferência.
- Caso veja um aumento nos valores de `processRecords.Time` que não esteja correlacionado com aumento na taxa de transferência, 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. Finalmente, confirme se você tem uma quantidade adequada de recursos físicos (memória, utilização da CPU etc.) nos nós de processamento subjacentes durante o pico de demanda.

## Erro de permissão de chave mestra do KMS não autorizada

Esse erro ocorre quando um aplicativo consumidor lê em um stream criptografado sem permissões na chave mestra do KMS. Para atribuir permissões a um aplicativo para que acesse uma chave do KMS, consulte [Como usar políticas de chaves no AWS KMS](#) e [Usar políticas do IAM com o AWS KMS](#).

# Tópicos avançados para consumidores do Amazon Kinesis Data Streams

Saiba como otimizar seu consumidor do Amazon Kinesis Data Streams.

### Tópicos

- [Processamento de baixa latência \(p. 182\)](#)
- [Usar o AWS Lambda com a Kinesis Producer Library \(p. 183\)](#)
- [Reestilhçamento, escalabilidade e processamento paralelo \(p. 183\)](#)
- [Tratar registros duplicados \(p. 184\)](#)
- [Tratar inicialização, desligamento e limitação \(p. 186\)](#)

## Processamento de baixa latência

Atraso de propagação é definido como a latência de ponta a ponta desde o momento em que um registro é gravado no stream até ser lido por um aplicativo de 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 vários aplicativos de consumidor processando um stream simultaneamente sem atingir os limites do Amazon Kinesis Data Streams de cinco chamadas a `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 padrões da KCL estão definidos para seguir a melhor prática de sondagem 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 sobrepondo as configurações padrão da KCL para sondar com mais frequência, como mostrado nos exemplos a seguir.

Código de configuração Java da KCL:

```
kinesisClientLibConfiguration = new
    KinesisClientLibConfiguration(applicationName,
        streamName,
        credentialsProvider,

    workerId).withInitialPositionInStream(initialPositionInStream).withIdleTimeBetweenReadsInMillis(250);
```

Configuração de arquivo de propriedades para Python e Ruby KCL:

```
idleTimeBetweenReadsInMillis = 250
```

#### Note

Como o Kinesis Data Streams tem um limite de cinco chamadas a `GetRecords` por segundo por estilhaço, configurar a propriedade `idleTimeBetweenReadsInMillis` abaixo de 200 ms pode fazer com que seu aplicativo 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.

## Usar o AWS Lambda com a Kinesis Producer Library

A [Kinesis Producer Library](#) (KPL) agrega pequenos registros formatados pelo usuário em registros maiores de até 1 MB para usar melhor a taxa de transferência do Amazon Kinesis Data Streams. Embora a KCL para Java ofereça suporte à desagregação desses registros, você precisa usar um módulo especial para desagregar registros ao usar o AWS Lambda como o consumidor de seus streams. Você pode obter o código de projeto e as instruções necessários no GitHub em [Módulos de desagregação da Kinesis Producer Library para o AWS Lambda](#). Os componentes deste projeto fornecem a capacidade de processar dados serializados da KPL no AWS Lambda em Java, Node.js e Python. Esses componentes também podem ser usados como parte de um [aplicativo de várias linguagens da KCL](#).

## Reestilhaçamento, escalabilidade e processamento paralelo

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 a própria KCL não inicie as operações de reestilhaçamento, ela é projetada para se adaptar às alterações no número de estilhaços resultantes do reestilhaçamento.

Conforme observado em [Utilizar uma tabela de arrendamento para rastrear os painéis processados pela aplicação do consumidor KCL](#) (p. 121), a KCL rastreia os estilhaços no streaming usando uma tabela do Amazon DynamoDB. Quando novos estilhaços são criados em consequência do reestilhaçamento, a KCL descobre os novos estilhaços e preenche novas linhas na tabela. Os operadores descobrem automaticamente os novos estilhaços e criam processadores para tratar os dados provenientes deles. A KCL também distribui os estilhaços no stream entre todos os operadores e processadores de registros disponíveis.

A KCL garante que os dados existentes nos estilhaços antes do reestilhaçamento sejam processados primeiro. Depois do processamento dos dados, os dados dos novos estilhaços são enviados para processadores de registros. Dessa forma, a KCL preserva a ordem em que os registros de dados foram adicionados ao stream para uma determinada chave de partição.

## Exemplo: Reestilhaçamento, escalabilidade e processamento paralelo

O exemplo a seguir ilustra como a KCL ajuda você a lidar com escalabilidade e reestilhaçamento:

- Por exemplo, se o aplicativo é executado em uma instância do EC2 e processa um Kinesis data stream que tem quatro estilhaços. Esta instância única tem um operador da KCL e quatro processadores de registros (um processador de registros para cada estilhaço). 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 operador da KCL é iniciado na segunda instância, ele faz balanceamento de carga com a primeira instância, de modo que cada instância passa a processar dois estilhaços.
- Se você decidir dividir os quatro estilhaços em cinco, a KCL novamente coordenará o processamento entre as instâncias: uma instância processará três estilhaços e a outra processará dois estilhaços. Uma coordenação semelhante ocorre quando você mescla estilhaços.

Normalmente, quando você usa a KCL, deve garantir que o número de instâncias não exceda o número de estilhaços (exceto para fins de espera de falha). Cada estilhaço é processado por exatamente um operador da KCL e tem exatamente um processador de registros correspondente, para que você nunca precise de várias instâncias para processar apenas um estilhaço. 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 Auto Scaling para escalar automaticamente suas instâncias com base em métricas apropriadas. Para obter mais informações, consulte [Guia do usuário do Amazon EC2 Auto Scaling](#).

Quando o reestilhaçamento aumenta o número de estilhaços no stream, o aumento correspondente no número de processadores de registros aumenta a carga nas instâncias do EC2 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 suas instâncias para iniciar seu Amazon Kinesis Data Streams application na inicialização, para que os operadores e processadores de registros adicionais fiquem ativos na nova instância imediatamente.

Para obter mais informações sobre a refragmentação, consulte [Reestilhaçar um stream \(p. 78\)](#).

## Tratar registros duplicados

Há dois principais motivos pelos quais os registros podem ser entregues mais de uma vez ao seu Amazon Kinesis Data Streams application: retentativas de produtor e retentativas de consumidor. Seu aplicativo precisa prever e tratar devidamente o processamento de registros individuais várias vezes.



## Retentativas de produtor

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 usar o `PutRecord` do AWS SDK, a configuração padrão [tentará novamente até três vezes uma chamada `PutRecord` com falha](#).

## Retentativas de consumidor

As retentativas de consumidor (aplicativo de processamento de dados) acontecem quando os processadores de registros são reiniciados. Os processadores de registros do mesmo estilo reiniciam nos seguintes casos:

1. Um operador é encerrado inesperadamente
2. Instâncias de operador são adicionadas ou removidas
3. Os estilos são mesclados ou divididos
4. O aplicativo é implantado

Em todos esses casos, o mapeamento de estilos para operador para processador de registros é atualizado continuamente para balancear a carga do processamento. Processadores de estilos 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 [Reestilhecimento, escalabilidade e processamento paralelo](#) (p. 183).

### Exemplo: Tentativas do consumidor resultando em registros reentregues

Neste exemplo, você tem um aplicativo que lê continuamente registros de um stream, agrega registros em um arquivo local e faz upload do arquivo para o Amazon S3. Para simplificar, suponha que haja apenas 1 estilo e 1 operador processando o estilo. 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 estilo, 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 a retentativas de 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 [Elasticsearch](#), você pode usar uma combinação de versionamento e IDs exclusivos para impedir processamento duplicado.

O aplicativo de exemplo da seção anterior lê continuamente os registros de um stream, agrega os registros em um arquivo local e faz upload do arquivo para o Amazon S3. Conforme ilustrado, os registros de 10001 a 20000 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 utiliza este esquema: Amazon S3 prefixo, identificação de identificação, e `First-Sequence-Num`. Neste caso, pode ser algo como `sample-shard000001-10001`.
3. Depois de carregar o Amazon S3 arquivo, ponto de controle especificando `Last-Sequence-Num`. Neste caso, teria o ponto de controle no registro número 15000.

Com esse esquema, mesmo se os registros forem 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 reestilhecimento, o número de registros deixados no estilhaço pode ser menor que o número fixo necessário. Neste caso, o método `shutdown()` precisa esvaziar o arquivo para o Amazon S3 e definir o ponto de verificação no último número de sequência. O esquema acima também é compatível com as operações de reestilhecimento.

## Tratar inicialização, desligamento e limitação

Veja aqui algumas considerações adicionais para incorporar ao projeto do seu Amazon Kinesis Data Streams application.

### Tópicos

- [Inicializar produtores de dados e consumidores de dados \(p. 186\)](#)
- [Desligar um Amazon Kinesis Data Streams Application \(p. 187\)](#)
- [Limitação de leitura \(p. 187\)](#)

## Inicializar produtores de dados e consumidores de dados

Por padrão, a KCL começa a ler registros da ponta do fluxo, 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 stream, defina o seguinte valor no arquivo de propriedades do seu Amazon Kinesis Data Streams application:

```
initialPositionInStream = TRIM_HORIZON
```



O Amazon Kinesis Data Streams mantém registros por 24 a 168 horas. 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 consumidor lendo do stream e usar a métrica `GetRecords.IteratorAgeMilliseconds` do CloudWatch 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 por meio do stream para testar se o stream está funcionando de ponta a ponta como 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 [Como usar iteradores de estilhaços](#) (p. 158).

## Desligar um Amazon Kinesis Data Streams Application

Quando seu Amazon Kinesis Data Streams application tiver concluído a tarefa pretendida, você deverá desligá-lo encerrando as instâncias do EC2 em que está em execução. É possível encerrar as instâncias usando o [Console de gerenciamento da AWS](#) ou a [AWS CLI](#).

Depois de desligar o Amazon Kinesis Data Streams application, você deve excluir a tabela do Amazon DynamoDB que a KCL usou para rastrear o estado do aplicativo.

## Limitação de leitura

A taxa de transferência de um stream é provisionada no nível do estilhaço. Cada estilhaço tem uma taxa de transferência de leitura de até 5 transações por segundo para leituras, até uma taxa total máxima de leitura de dados de 2 MB por segundo. Se um aplicativo (ou grupo de aplicativos que opera no mesmo stream) tentar obter dados de um estilhaço a uma taxa mais rápida, o Kinesis Data Streams limitará as operações `Get` correspondentes.

Em um Amazon Kinesis Data Streams application, se um processador de registros estiver processando dados mais rapidamente do que o limite — como no caso de um failover — ocorrerá controle de utilização. Como o KCL gerencia as interações entre o aplicativo e o Kinesis Data Streams, as exceções de limitação ocorrem no código do KCL, e não no código do aplicativo. No entanto, como a KCL registra essas exceções, você as vê nos logs.

Se você achar que o seu aplicativo fica limitado de forma consistente, considere aumentar o número de estilhaços do stream.

# Monitorar o Amazon Kinesis Data Streams

Você pode monitorar seus streaming de dados no Amazon Kinesis Data Streams usando os seguintes recursos:

- [Métricas do CloudWatch \(p. 188\)](#) — o Kinesis Data Streams envia métricas personalizadas do Amazon CloudWatch com monitoramento detalhado para cada streaming.
- [Agente do Kinesis \(p. 198\)](#) — o agente do Kinesis publica métricas personalizadas do CloudWatch para ajudar a avaliar se o agente está funcionando conforme o esperado.
- [Registro em log da API \(p. 199\)](#) — o Kinesis Data Streams usa o AWS CloudTrail para registrar chamadas à API e armazenar os dados em um bucket do Amazon S3.
- [A Biblioteca de cliente do Kinesis \(p. 203\)](#) — Kinesis Client Library (KCL) fornece métricas por estilhaço, operador e aplicativo da KCL.
- [A Biblioteca de produtor do Kinesis \(p. 212\)](#) — Kinesis Producer Library (KPL) fornece métricas por estilhaço, operador e aplicativo da KPL.

## Monitoramento do serviço do Amazon Kinesis Data Streams com o Amazon CloudWatch

O Amazon Kinesis Data Streams e o Amazon CloudWatch são integrados, permitindo coletar, visualizar e analisar métricas do CloudWatch para os streamings 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 configuradas para os streams são coletadas e enviadas automaticamente ao CloudWatch a cada minuto. As métricas são arquivadas por duas semanas. Depois desse período, os dados serão descartados.

A tabela a seguir descreve o monitoramento de nível de estilhaço avançado e nível de stream básico para streaming de dados do Kinesis.

Type	Description (Descrição)
Básico (nível de stream)	Dados de nível de stream são enviados automaticamente a cada minuto, sem custo adicional.
Avançado (nível de estilhaço)	<p>Dados de nível de estilhaço são enviados a cada minuto por um custo adicional. Para obter esse nível de dados, você precisa habilitá-lo especificamente para o stream usando a operação <a href="#">EnableEnhancedMonitoring</a>.</p> <p>Para obter informações sobre a definição de preço, consulte a <a href="#">página do produto do Amazon CloudWatch</a>.</p>

## Dimensões e métricas do Amazon Kinesis Data Streams

O Kinesis Data Streams envia métricas ao CloudWatch em dois níveis: no nível de streaming e, opcionalmente, de estiloço. As métricas no nível do stream se destinam aos casos de uso de monitoramento mais comuns em condições normais. As métricas em nível de estiloço são para tarefas de monitoramento específicas, geralmente relacionadas à solução de problemas, e são habilitadas com o uso da operação [EnableEnhancedMonitoring](#).

Para ver uma explicação sobre as estatísticas reunidas com base nas métricas do CloudWatch, consulte [CloudWatch Statistics](#) no Guia do usuário do Amazon CloudWatch.

### Tópicos

- [Métricas no nível do fluxo básicas \(p. 189\)](#)
- [Métricas do nível de estiloço aprimoradas \(p. 194\)](#)
- [Dimensões para métricas do Amazon Kinesis Data Streams \(p. 196\)](#)
- [Métricas recomendadas do Amazon Kinesis Data Streams \(p. 197\)](#)

## Métricas no nível do fluxo básicas

O namespace `AWS/Kinesis` inclui métricas de nível do fluxo a seguir.

O Kinesis Data Streams envia essas métricas em nível de streaming ao CloudWatch a cada minuto. Essas métricas estão sempre disponíveis.

Métrica	Description (Descrição)
<code>GetRecords.Bytes</code>	<p>O número de bytes recuperados do stream do Kinesis, medido ao longo do período de tempo especificado. As estatísticas mínima, máxima e média representam os bytes em uma única operação <code>GetRecords</code> para o fluxo no período especificado.</p> <p>Nome da métrica do nível de estiloço: <code>OutgoingBytes</code></p> <p>Dimensões <code>StreamName</code></p> <p>Estatísticas Mínimo, Máximo, Média, Soma, Amostras</p> <p>Unidades Bytes</p>
<code>GetRecords.IteratorAge</code>	<p>Essa métrica foi substituída. Use <code>GetRecords.IteratorAgeMilliseconds</code>.</p>
<code>GetRecords.IteratorAgeMilliseconds</code>	<p>Idade do último registro em todas as chamadas <code>GetRecords</code> feitas com relação a um fluxo do Kinesis, medida ao longo do período especificado. Idade é a diferença entre a hora atual e quando o último registro da chamada <code>GetRecords</code> foi gravado no stream. As estatísticas de mínimo e máximo podem ser usadas para acompanhar o progresso dos aplicativos de consumidores do Kinesis. Um valor zero indica que os registros lidos estão em completa sincronização com o stream.</p>

Métrica	Description (Descrição)
	<p>Nome da métrica do nível de estilhaço: <code>IteratorAgeMilliseconds</code></p> <p>Dimensões <code>StreamName</code></p> <p>Estatísticas Mínimo, Máximo, Média, Amostras</p> <p>Unidades Milissegundos</p>
<code>GetRecords.Latency</code>	<p>O tempo gasto por operação <code>GetRecords</code>, medido ao longo do período de tempo especificado.</p> <p>Dimensões <code>StreamName</code></p> <p>Estatísticas Mínimo, Máximo, Média</p> <p>Unidades Milissegundos</p>
<code>GetRecords.Records</code>	<p>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 <code>GetRecords</code> para o fluxo no período especificado.</p> <p>Nome da métrica do nível de estilhaço: <code>OutgoingRecords</code></p> <p>Dimensões <code>StreamName</code></p> <p>Estatísticas Mínimo, Máximo, Média, Soma, Amostras</p> <p>Unidades Contagem</p>
<code>GetRecords.Success</code>	<p>O número de operações <code>GetRecords</code> bem-sucedidas por stream, medido ao longo do período de tempo especificado.</p> <p>Dimensões <code>StreamName</code></p> <p>Estatísticas Média, soma, amostras</p> <p>Unidades Contagem</p>
<code>IncomingBytes</code>	<p>O número de bytes colocados com sucesso no stream do Kinesis ao longo do período de tempo especificado. Essa métrica inclui bytes das operações <code>PutRecord</code> e <code>PutRecords</code>. As estatísticas mínima, máxima e média representam os bytes em uma única operação <code>put</code> para o fluxo no período especificado.</p> <p>Nome da métrica do nível de estilhaço: <code>IncomingBytes</code></p> <p>Dimensões <code>StreamName</code></p> <p>Estatísticas Mínimo, Máximo, Média, Soma, Amostras</p> <p>Unidades Bytes</p>

Métrica	Description (Descrição)
<code>IncomingRecords</code>	<p>O número de registros colocados com sucesso no stream do Kinesis ao longo do período de tempo especificado. Essa métrica inclui registros das operações <code>PutRecord</code> e <code>PutRecords</code>. As estatísticas mínima, máxima e média representam os registros em uma única operação put para o fluxo no período especificado.</p> <p>Nome da métrica do nível de estilhaço: <code>IncomingRecords</code></p> <p>Dimensões <code>StreamName</code></p> <p>Estatísticas Mínimo, Máximo, Média, Soma, Amostras</p> <p>Unidades Contagem</p>
<code>PutRecord.Bytes</code>	<p>O número de bytes colocados no streaming do Kinesis usando a operação <code>PutRecord</code> ao longo do período especificado.</p> <p>Dimensões <code>StreamName</code></p> <p>Estatísticas Mínimo, Máximo, Média, Soma, Amostras</p> <p>Unidades Bytes</p>
<code>PutRecord.Latency</code>	<p>O tempo gasto por operação <code>PutRecord</code>, medido ao longo do período de tempo especificado.</p> <p>Dimensões <code>StreamName</code></p> <p>Estatísticas Mínimo, Máximo, Média</p> <p>Unidades Milissegundos</p>
<code>PutRecord.Success</code>	<p>O número de operações <code>PutRecord</code> bem-sucedidas por streaming do Kinesis, medido ao longo do período especificado. A média reflete a porcentagem de gravações bem-sucedidas em um stream.</p> <p>Dimensões <code>StreamName</code></p> <p>Estatísticas Média, soma, amostras</p> <p>Unidades Contagem</p>
<code>PutRecords.Bytes</code>	<p>O número de bytes colocados no streaming do Kinesis usando a operação <code>PutRecords</code> ao longo do período especificado.</p> <p>Dimensões <code>StreamName</code></p> <p>Estatísticas Mínimo, Máximo, Média, Soma, Amostras</p> <p>Unidades Bytes</p>

Métrica	Description (Descrição)
<code>PutRecords.Latency</code>	<p>O tempo gasto por operação <code>PutRecords</code>, medido ao longo do período de tempo especificado.</p> <p>Dimensões <code>StreamName</code></p> <p>Estatísticas Mínimo, Máximo, Média</p> <p>Unidades Milissegundos</p>
<code>PutRecords.Records</code>	<p>O número de registros bem-sucedidos em uma operação <code>PutRecords</code> por stream do Kinesis medido ao longo do período especificado.</p> <p>Dimensões <code>StreamName</code></p> <p>Estatísticas Mínimo, Máximo, Média, Soma, Amostras</p> <p>Unidades Contagem</p>
<code>PutRecords.Success</code>	<p>O número de operações <code>PutRecords</code> com pelo menos um registro bem-sucedido, por streaming do Kinesis medido ao longo do período especificado.</p> <p>Dimensões <code>StreamName</code></p> <p>Estatísticas Média, soma, amostras</p> <p>Unidades Contagem</p>
<code>ReadProvisionedThroughputExceeded</code>	<p>O número de chamadas <code>GetRecords</code> limitadas para o fluxo ao longo do período especificado. A estatística mais usada para essa métrica é Média.</p> <p>Quando a estatística mínima tem um valor de 1, todos os registros foram limitados ao fluxo durante o período especificado.</p> <p>Quando a estatística máxima tem um valor de 0 (zero), nenhum registro foi limitado ao fluxo durante o período especificado.</p> <p>Nome da métrica do nível de estilhaço: <code>ReadProvisionedThroughputExceeded</code></p> <p>Dimensões <code>StreamName</code></p> <p>Estatísticas Mínimo, Máximo, Média, Soma, Amostras</p> <p>Unidades Contagem</p>
<code>SubscribeToShard.RateExceeded</code>	<p>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.</p> <p>Dimensões nome, nome de utilizador</p>

Métrica	Description (Descrição)
<code>SubscribeToShard.Success</code>	<p>Essa métrica registra se a assinatura <code>SubscribeToShard</code> foi estabelecida com êxito. A assinatura se mantém ativa por no máximo 5 minutos. Portanto, essa métrica é emitida pelo menos uma vez a cada 5 minutos.</p> <p>Dimensões nome, nome de utilizador</p>
<code>SubscribeToShardEvent.Bytes</code>	<p>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.</p> <p>Nome da métrica do nível de estilhaço: <code>OutgoingBytes</code></p> <p>Dimensões nome, nome de utilizador</p> <p>Estatísticas Mínimo, Máximo, Média, Soma, Amostras</p> <p>Unidades Bytes</p>
<code>SubscribeToShardEvent.MillisBehind</code>	<p>A diferença entre a hora atual e o momento em que o último registro do evento <code>SubscribeToShard</code> foi gravado no stream.</p> <p>Dimensões nome, nome de utilizador</p> <p>Estatísticas Mínimo, Máximo, Média, Amostras</p> <p>Unidades Milissegundos</p>
<code>SubscribeToShardEvent.Records</code>	<p>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.</p> <p>Nome da métrica do nível de estilhaço: <code>OutgoingRecords</code></p> <p>Dimensões nome, nome de utilizador</p> <p>Estatísticas Mínimo, Máximo, Média, Soma, Amostras</p> <p>Unidades Contagem</p>
<code>SubscribeToShardEvent.Success</code>	<p>Essa métrica é emitida sempre que um evento é publicado com êxito. Ela será emitida somente quando houver uma assinatura ativa.</p> <p>Dimensões nome, nome de utilizador</p> <p>Estatísticas Mínimo, Máximo, Média, Soma, Amostras</p> <p>Unidades Contagem</p>

Métrica	Description (Descrição)
<code>WriteProvisionedThroughputExceeded</code>	<p>O número de registros rejeitados por causa da limitação para o fluxo ao longo do período especificado. Essa métrica inclui a limitação das operações <code>PutRecord</code> e <code>PutRecords</code>. A estatística mais usada para essa métrica é Média.</p> <p>Quando a estatística mínima tem um valor diferente de zero, nenhum registro é limitado ao fluxo durante o período especificado.</p> <p>Quando a estatística máxima tem um valor de 0 (zero), nenhum registro foi limitado ao fluxo durante o período especificado.</p> <p>Nome da métrica do nível de estilhaço: <code>WriteProvisionedThroughputExceeded</code></p> <p>Dimensões <code>StreamName</code></p> <p>Estatísticas Mínimo, Máximo, Média, Soma, Amostras</p> <p>Unidades Contagem</p>

## Métricas do nível de estilhado aprimoradas

O namespace `AWS/Kinesis` inclui métricas de nível do estilhaço a seguir.

O Kinesis envia as seguintes métricas em nível de estilhaço ao CloudWatch a cada minuto. Cada dimensão métrica cria uma métrica do CloudWatch e faz aproximadamente 43.200 chamadas de API `PutMetricData` por mês. Essas métricas não são permitidas por padrão. Há uma cobrança para as métricas aprimoradas emitidas pelo Kinesis. Para mais informações, consulte [Amazon CloudWatch Preços](#) sob o cabeçalho Amazon CloudWatch Custom Metrics. As cobranças são indicadas por estilhaço pela métrica por mês.

Métrica	Description (Descrição)
<code>IncomingBytes</code>	<p>O número de bytes colocados com sucesso no estilhaço ao longo do período especificado. Essa métrica inclui bytes das operações <code>PutRecord</code> e <code>PutRecords</code>. As estatísticas mínima, máxima e média representam os bytes em uma única operação put para o estilhaço no período especificado.</p> <p>Nome da métrica no nível do fluxo: <code>IncomingBytes</code></p> <p>Dimensões nome do software, <code>shardid</code></p> <p>Estatísticas Mínimo, Máximo, Média, Soma, Amostras</p> <p>Unidades Bytes</p>
<code>IncomingRecords</code>	<p>O número de registros colocados com sucesso no estilhaço ao longo do período especificado. Essa métrica inclui registros das operações <code>PutRecord</code> e <code>PutRecords</code>. As estatísticas mínima, máxima e média representam os registros em uma única operação put para o estilhaço no período especificado.</p>



Métrica	Description (Descrição)
	<p>Nome da métrica no nível do fluxo: <code>IncomingRecords</code></p> <p>Dimensões nome do software, shardid</p> <p>Estatísticas Mínimo, Máximo, Média, Soma, Amostras</p> <p>Unidades Contagem</p>
<code>IteratorAgeMilliseconds</code>	<p>A idade do último registro em todas as chamadas <code>GetRecords</code> 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 <code>GetRecords</code> foi gravado no stream. As estatísticas de mínimo e máximo podem ser usadas para acompanhar o progresso dos aplicativos de consumidores do Kinesis. Um valor de 0 (zero) indica que os registros lidos estão em completa sincronização com o fluxo.</p> <p>Nome da métrica no nível do fluxo: <code>GetRecords.IteratorAgeMilliseconds</code></p> <p>Dimensões nome do software, shardid</p> <p>Estatísticas Mínimo, Máximo, Média, Amostras</p> <p>Unidades Milissegundos</p>
<code>OutgoingBytes</code>	<p>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 <code>GetRecords</code> ou publicados em um único evento <code>SubscribeToShard</code> para o estilhaço no período especificado.</p> <p>Nome da métrica no nível do fluxo: <code>GetRecords.Bytes</code></p> <p>Dimensões nome do software, shardid</p> <p>Estatísticas Mínimo, Máximo, Média, Soma, Amostras</p> <p>Unidades Bytes</p>
<code>OutgoingRecords</code>	<p>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 <code>GetRecords</code> ou publicados em um único evento <code>SubscribeToShard</code> para o estilhaço no período especificado.</p> <p>Nome da métrica no nível do fluxo: <code>GetRecords.Records</code></p> <p>Dimensões nome do software, shardid</p> <p>Estatísticas Mínimo, Máximo, Média, Soma, Amostras</p> <p>Unidades Contagem</p>

Métrica	Description (Descrição)
<code>ReadProvisionedThroughputExceeded</code>	<p>O número de chamadas <code>GetRecords</code> 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.</p> <p>Quando a estatística mínima tem um valor de 1, todos os registros foram limitados ao estilhaço durante o período especificado.</p> <p>Quando a estatística máxima tem um valor de 0 (zero), nenhum registro foi limitado ao estilhaço durante o período especificado.</p> <p>Nome da métrica no nível do fluxo: <code>ReadProvisionedThroughputExceeded</code></p> <p>Dimensões nome do software, shardid</p> <p>Estatísticas Mínimo, Máximo, Média, Soma, Amostras</p> <p>Unidades Contagem</p>
<code>WriteProvisionedThroughputExceeded</code>	<p>O número de registros rejeitados por causa da limitação para o estilhaço ao longo do período especificado. Esta métrica inclui limitação das operações <code>PutRecord</code> e <code>PutRecords</code> 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.</p> <p>Quando a estatística mínima tem um valor diferente de zero, nenhum registro é limitado ao estilhaço durante o período especificado.</p> <p>Quando a estatística máxima tem um valor de 0 (zero), nenhum registro foi limitado ao estilhaço durante o período especificado.</p> <p>Nome da métrica no nível do fluxo: <code>WriteProvisionedThroughputExceeded</code></p> <p>Dimensões nome do software, shardid</p> <p>Estatísticas Mínimo, Máximo, Média, Soma, Amostras</p> <p>Unidades Contagem</p>

## Dimensões para métricas do Amazon Kinesis Data Streams

Dimension	Description
<code>StreamName</code>	The name of the Kinesis stream. All available statistics are filtered by <code>StreamName</code> .

## Métricas recomendadas do Amazon Kinesis Data Streams

Várias métricas do Amazon Kinesis Data Streams podem ser de interesse específico para os clientes do Kinesis Data Streams. A lista a seguir oferece métricas recomendadas e suas utilizações.

Métrica	Observações sobre o uso
<code>GetRecords.IteratorAge</code>	Recomendamos a monitorização 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 alarmes do CloudWatch na estatística Máxima para alertá-lo antes que essa perda se torne um risco. Para ver um exemplo de cenário que usa essa métrica, consulte <a href="#">Processamento de registros de consumidores ficando atrasados (p. 181)</a> .
<code>ReadProvisionedThroughputExceeded</code>	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 taxa de transferência de leitura. A estatística mais usada para essa métrica é Média.
<code>WriteProvisionedThroughputExceeded</code>	Elabore a mesma finalidade da métrica <code>ReadProvisionedThroughputExceeded</code> , mas para o lado do produtor (put) do stream. A estatística mais usada para essa métrica é Média.
<code>PutRecord.Success</code> , <code>PutRecords.Success</code>	Recomendamos o uso de alarmes do CloudWatch na estatística Média para indicar se os registros estão falhando no stream. Escolha um ou ambos os tipos put, dependendo do que o produtor usa. Se estiver usando a Kinesis Producer Library (KPL), use <code>PutRecords.Success</code> .
<code>GetRecords.Success</code>	Recomendamos o uso de alarmes do CloudWatch na estatística Média para indicar se os registros estão falhando a partir do stream.

## Acessar métricas do Amazon CloudWatch para o Kinesis Data Streams

Você pode monitorar as métricas do Kinesis Data Streams usando o console do CloudWatch, a linha de comando ou a API do CloudWatch. Os procedimentos a seguir mostram como acessar as métricas usando os seguintes métodos:

Para acessar métricas usando o console do CloudWatch

1. Abra o console do CloudWatch em <https://console.aws.amazon.com/cloudwatch/>.
2. Na barra de navegação, escolha uma Região.
3. No painel de navegação, selecione Metrics (Métricas).
4. No painel CloudWatch Metrics by Category (Métricas do CloudWatch por categoria), selecione Kinesis Metrics (Métricas do Kinesis).
5. Clique na linha relevante para visualizar as estatísticas para o MetricName e o StreamName especificados.

Observação A maioria dos nomes estatísticos da consola corresponde ao correspondente CloudWatch nomes métricos listados acima, excepto Leitura de leitura e Rendimento de escrita. Estas estatísticas

são calculadas em intervalos de 5 minutos: `Rendimento de escrita` monitoriza o `IncomingBytes` CloudWatch métrica e `Leitura de leitura` monitores `GetRecords.Bytes`.

6. (Opcional) No painel gráfico, selecione uma estatística e um período e, em seguida, crie um alarme do CloudWatch usando essas configurações.

Para acessar as métricas usando a AWS CLI

Use os comandos [list-metrics](#) e [get-metric-statistics](#).

Para acessar as métricas usando a CLI do CloudWatch

Use os comandos [mon-list-metrics](#) e [mon-get-stats](#).

Para acessar as métricas usando a API do CloudWatch

Use as operações [ListMetrics](#) e [GetMetricsStatistics](#).

## Monitoramento do Agente de integridade do Kinesis Data Streams com o Amazon CloudWatch

O agente publica métricas personalizadas do CloudWatch com um namespace de `AWSKinesisAgent`. Essas métricas ajudam você a avaliar se o agente está enviando dados ao Kinesis Data Streams conforme especificado e se ele está íntegro e consumindo a quantidade apropriada de CPU e de recursos de memória no 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. As métricas como consumo de CPU e memória no host e contadores de erros do agente indicam uso de recurso por parte do produtor de dados e fornece informações sobre erros potenciais de configuração ou de host. Por fim, o agente também registra exceções de serviço para ajudar a investigar problemas do agente. Estas métricas são reportadas na região especificada na definição de configuração do agente `cloudwatch.endpoint`. Para mais informações sobre a configuração do agente, consulte [Configurações do agente](#) (p. 107).

### Monitorar com o CloudWatch

O agente do Kinesis Data Streams envia as seguintes métricas ao CloudWatch.

Métrica	Description (Descrição)
<code>BytesSent</code>	O número de bytes enviados para o Kinesis Data Streams ao longo do período de tempo especificado.  Unidades Bytes
<code>RecordSendAttempts</code>	O número de tentativas de registro (primeira vez ou como nova tentativa) em uma chamada para <code>PutRecords</code> no período especificado.  Unidades Contagem
<code>RecordSendErrors</code>	O número de registros que retornaram status de falha em uma chamada para <code>PutRecords</code> , incluindo novas tentativas, no período especificado.  Unidades Contagem

Métrica	Description (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

## Registro em log de chamadas de API do Amazon Kinesis Data Streams com o AWS CloudTrail

O Amazon Kinesis Data Streams é integrado com o AWS CloudTrail, um serviço que fornece um registro das ações realizadas por um usuário, uma função ou um serviço da AWS no Kinesis Data Streams. O CloudTrail captura todas as chamadas de API para o Kinesis Data Streams como eventos. As chamadas capturadas incluem as chamadas do console do Kinesis Data Streams e as chamadas de código para as operações da API do Kinesis Data Streams. Se você criar uma trilha, poderá habilitar a entrega contínua de eventos do CloudTrail a um bucket do Amazon S3, incluindo eventos do Kinesis Data Streams. Se não configurar uma trilha, você ainda poderá visualizar os eventos mais recentes no console do CloudTrail em Event history. Com as informações coletadas pelo CloudTrail, você pode determinar a solicitação feita para o Kinesis Data Streams, o endereço IP do qual a solicitação foi feita, quem fez a solicitação, quando ela foi feita e detalhes adicionais.

Para saber mais sobre o CloudTrail, incluindo como configurá-lo e habilitá-lo, consulte o [AWS CloudTrail User Guide](#).

### Kinesis Data Streams Informações em CloudTrail

O CloudTrail está habilitado na sua conta da AWS ao criá-la. Quando a atividade do evento com suporte ocorre no Kinesis Data Streams, ela é registrada em um evento do CloudTrail junto com outros eventos de serviços da AWS em Event history (Histórico de eventos). Você pode visualizar, pesquisar e fazer download de eventos recentes em sua conta da AWS. Para obter mais informações, consulte [Visualizar eventos com o histórico de eventos do CloudTrail](#).

Para obter um registro contínuo de eventos em sua conta da AWS, incluindo eventos para o Kinesis Data Streams, crie uma trilha. Uma trilha permite CloudTrail para fornecer arquivos de log a um bucket do Amazon S3. Por padrão, quando você cria uma trilha no console, ela é aplicada a todas as regiões da AWS. A trilha registra eventos de todas as regiões na partição da AWS e fornece os arquivos de log para o bucket do Amazon S3 que você especificar. Além disso, é possível configurar outros serviços da AWS para analisar mais profundamente e agir sobre os dados de evento coletados nos logs do CloudTrail. Para obter mais informações, consulte o seguinte: .

- [Visão geral da criação de uma trilha](#)
- [CloudTrail Serviços e integrações suportados](#)
- [Configurar notificações de SNS da Amazon para CloudTrail](#)
- [Receber arquivos de log do CloudTrail de várias regiões e receber arquivos de log do CloudTrail de várias contas](#)

O Kinesis Data Streams oferece suporte ao registro em log das seguintes ações como eventos nos arquivos de log do CloudTrail:

- [AddTagsToStream](#)
- [CreateStream](#)
- [DecreaseStreamRetentionPeriod](#)

- [DeleteStream](#)
- [DeregisterStreamConsumer](#)
- [DescribeStream](#)
- [DescribeStreamConsumer](#)
- [DisableEnhancedMonitoring](#)
- [EnableEnhancedMonitoring](#)
- [IncreaseStreamRetentionPeriod](#)
- [ListStreamConsumers](#)
- [ListStreams](#)
- [ListTagsForStream](#)
- [MergeShards](#)
- [RegisterStreamConsumer](#)
- [RemoveTagsFromStream](#)
- [SplitShard](#)
- [StartStreamEncryption](#)
- [StopStreamEncryption](#)
- [UpdateShardCount](#)

Cada evento ou entrada no log contém informações sobre quem gerou a solicitação. As informações de identidade ajudam a determinar:

- Se a solicitação foi feita com credenciais de usuário da raiz ou do AWS Identity and Access Management (IAM).
- Se a solicitação foi feita com credenciais de segurança temporárias de uma função ou de um usuário federado
- Se a solicitação foi feita por outro serviço da AWS.

Para obter mais informações, consulte [Elemento userIdentity do CloudTrail](#).

## Exemplo: Kinesis Data Streams Entradas dos arquivos de log

Uma trilha é uma configuração que permite a entrega de eventos como arquivos de log em um bucket do Amazon S3 que você especificar. Os arquivos de log do CloudTrail contêm uma ou mais entradas de log. Um evento representa uma única solicitação de qualquer origem e inclui informações sobre a ação solicitada, a data e hora da ação, parâmetros de solicitação, e assim por diante. Os arquivos de log do CloudTrail não são um rastreamento de pilha ordenada das chamadas da API pública. Assim, elas não são exibidas em nenhuma ordem específica.

O exemplo a seguir mostra uma entrada de log do CloudTrail que demonstra as ações `CreateStream`, `DescribeStream`, `ListStreams`, `DeleteStream`, `SplitShard` e `MergeShards`.

```
{
  "Records": [
    {
      "eventVersion": "1.01",
      "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::012345678910:user/Alice",
```

```
        "accountId": "012345678910",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice"
    },
    "eventTime": "2014-04-19T00:16:31Z",
    "eventSource": "kinesis.amazonaws.com",
    "eventName": "CreateStream",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "127.0.0.1",
    "userAgent": "aws-sdk-java/unknown-version Linux/x.xx",
    "requestParameters": {
        "shardCount": 1,
        "streamName": "GoodStream"
    },
    "responseElements": null,
    "requestID": "db6c59f8-c757-11e3-bc3b-57923b443c1c",
    "eventID": "b7acfcd0-6ca9-4ee1-a3d7-c4e8d420d99b"
},
{
    "eventVersion": "1.01",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::012345678910:user/Alice",
        "accountId": "012345678910",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice"
    },
    "eventTime": "2014-04-19T00:17:06Z",
    "eventSource": "kinesis.amazonaws.com",
    "eventName": "DescribeStream",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "127.0.0.1",
    "userAgent": "aws-sdk-java/unknown-version Linux/x.xx",
    "requestParameters": {
        "streamName": "GoodStream"
    },
    "responseElements": null,
    "requestID": "f0944d86-c757-11e3-b4ae-25654b1d3136",
    "eventID": "0b2f1396-88af-4561-b16f-398f8eaea596"
},
{
    "eventVersion": "1.01",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::012345678910:user/Alice",
        "accountId": "012345678910",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice"
    },
    "eventTime": "2014-04-19T00:15:02Z",
    "eventSource": "kinesis.amazonaws.com",
    "eventName": "ListStreams",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "127.0.0.1",
    "userAgent": "aws-sdk-java/unknown-version Linux/x.xx",
    "requestParameters": {
        "limit": 10
    },
    "responseElements": null,
    "requestID": "a68541ca-c757-11e3-901b-cbcfe5b3677a",
    "eventID": "22a5fb8f-4e61-4bee-a8ad-3b72046b4c4d"
},
{
    "eventVersion": "1.01",
```

```
"userIdentity": {
  "type": "IAMUser",
  "principalId": "EX_PRINCIPAL_ID",
  "arn": "arn:aws:iam::012345678910:user/Alice",
  "accountId": "012345678910",
  "accessKeyId": "EXAMPLE_KEY_ID",
  "userName": "Alice"
},
"eventTime": "2014-04-19T00:17:07Z",
"eventSource": "kinesis.amazonaws.com",
"eventName": "DeleteStream",
"awsRegion": "us-east-1",
"sourceIPAddress": "127.0.0.1",
"userAgent": "aws-sdk-java/unknown-version Linux/x.xx",
"requestParameters": {
  "streamName": "GoodStream"
},
"responseElements": null,
"requestID": "f10cd97c-c757-11e3-901b-cbcfe5b3677a",
"eventID": "607e7217-311a-4a08-a904-ec02944596dd"
},
{
  "eventVersion": "1.01",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "EX_PRINCIPAL_ID",
    "arn": "arn:aws:iam::012345678910:user/Alice",
    "accountId": "012345678910",
    "accessKeyId": "EXAMPLE_KEY_ID",
    "userName": "Alice"
  },
  "eventTime": "2014-04-19T00:15:03Z",
  "eventSource": "kinesis.amazonaws.com",
  "eventName": "SplitShard",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "127.0.0.1",
  "userAgent": "aws-sdk-java/unknown-version Linux/x.xx",
  "requestParameters": {
    "shardToSplit": "shardId-000000000000",
    "streamName": "GoodStream",
    "newStartingHashKey": "11111111"
  },
  "responseElements": null,
  "requestID": "a6e6e9cd-c757-11e3-901b-cbcfe5b3677a",
  "eventID": "dcd2126f-c8d2-4186-b32a-192dd48d7e33"
},
{
  "eventVersion": "1.01",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "EX_PRINCIPAL_ID",
    "arn": "arn:aws:iam::012345678910:user/Alice",
    "accountId": "012345678910",
    "accessKeyId": "EXAMPLE_KEY_ID",
    "userName": "Alice"
  },
  "eventTime": "2014-04-19T00:16:56Z",
  "eventSource": "kinesis.amazonaws.com",
  "eventName": "MergeShards",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "127.0.0.1",
  "userAgent": "aws-sdk-java/unknown-version Linux/x.xx",
  "requestParameters": {
    "streamName": "GoodStream",
    "adjacentShardToMerge": "shardId-000000000002",
    "shardToMerge": "shardId-000000000001"
```



```
    },
    "responseElements": null,
    "requestID": "e9f9c8eb-c757-11e3-bf1d-6948db3cd570",
    "eventID": "77cf0d06-ce90-42da-9576-71986fec411f"
  }
}
```

## Monitoramento da Kinesis Client Library com o Amazon CloudWatch

A [Kinesis Client Library](#) (KCL) do Amazon Kinesis Data Streams publica métricas personalizadas do Amazon CloudWatch em seu nome usando o nome do aplicativo da KCL como o namespace. Você pode visualizar essas métricas navegando até o [console do CloudWatch](#) e escolhendo Custom Metrics (Métricas personalizadas). Para obter mais informações sobre métricas personalizadas, consulte [Publicar métricas personalizadas](#) no Guia do usuário do Amazon CloudWatch.

Há uma cobrança nominal para as métricas obtidas por upload no CloudWatch pelo KCL. As Métricas personalizadas do Amazon CloudWatch e as Solicitações da API do Amazon CloudWatch são cobradas especificamente. Para obter mais informações, consulte [Definição de preço do Amazon CloudWatch](#).

### Tópicos

- [Métricas e namespace](#) (p. 203)
- [Dimensões e níveis de métricas](#) (p. 203)
- [Configuração da métrica](#) (p. 204)
- [Lista de métricas do](#) (p. 204)

## Métricas e namespace

O namespace usado para fazer o carregamento de métricas é o nome do aplicativo especificado quando você executar o KCL.

## Dimensões e níveis de métricas

Há duas opções para controlar quais métricas são carregadas para o CloudWatch:

### níveis de métrica

Cada métrica é atribuída a um nível individual. Quando você define um nível de relatório de métricas, as métricas com um nível individual abaixo do nível de relatório não são enviadas ao CloudWatch. Os níveis são: `NONE`, `SUMMARY`, e `DETAILED`. A predefinição 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 do](#) (p. 204).

### dimensões habilitadas

Cada métrica do KCL tem dimensões associadas que também são enviadas ao CloudWatch. No KCL 2.x, se o KCL estiver configurado para processar um único fluxo de dados, todas as dimensões de métrica (`Operation`, `ShardId`, e `WorkerIdentifier`) estão activadas por predefinição. Também, em KCL 2.x, se KCL estiver configurado para processar um único fluxo de dados, `Operation` a dimensão não pode ser desativada. No KCL 2.x, se o KCL estiver configurado para processar vários fluxos de dados, todas as dimensões de métrica (`Operation`, `ShardId`, `StreamId`, e

`WorkerIdentifier`) estão activadas por predefinição. Também, no KCL 2.x, se o KCL estiver configurado para processar vários fluxos de dados, o `Operation` e o `StreamId` As dimensões não podem ser desativadas. `StreamId` a dimensão está disponível apenas para as métricas por impulso.

No KCL 1.x, apenas o `Operation` e o `ShardId` as dimensões são activadas por predefinição e o `WorkerIdentifier` a dimensão está desativada. No KCL 1.x, o `Operation` a dimensão não pode ser desativada.

Para obter mais informações sobre as dimensões de métricas do CloudWatch, consulte a seção [Dimensões](#) no tópico Conceitos do Amazon CloudWatch do Guia do usuário do Amazon CloudWatch.

Quando a dimensão `WorkerIdentifier` está habilitada, se um valor diferente é usado para a propriedade do ID do operador toda vez que um determinado operador da KCL é reiniciado, novos conjuntos de métricas com novos valores de dimensão `WorkerIdentifier` são enviados ao CloudWatch. Se você precisar que o valor da dimensão `WorkerIdentifier` seja o mesmo entre reinicializações de um operador específico da KCL, será necessário especificar explicitamente o mesmo valor do ID do operador durante a inicialização para cada operador. Observe que o valor do ID do operador para cada operador da KCL ativo precisa ser exclusivo em todos os operadores da KCL.

## Configuração da métrica

Os níveis de métrica e as dimensões habilitadas podem ser configurados com a instância `KinesisClientLibConfiguration`, que é passada ao operador na inicialização do aplicativo KCL. No caso do `MultiLangDaemon`, as propriedades `metricsLevel` e `metricsEnabledDimensions` podem ser especificadas no arquivo `.properties` usado para ativar o aplicativo `MultiLangDaemon` da KCL.

Podem ser atribuídos aos níveis métricos um de três valores: NENHUM, RESUMO ou DETALHADO. Os valores de dimensões habilitadas precisam ser strings separadas por vírgulas com a lista de dimensões que são permitidas para as métricas do CloudWatch. As dimensões usadas pelo aplicativo da KCL são `Operation`, `ShardId` e `WorkerIdentifier`.

## Lista de métricas do

As tabelas a seguir listam as métricas do KCL, agrupadas por escopo e operação.

### Tópicos

- [Métricas do aplicativo por KCL \(p. 204\)](#)
- [Métricas por operador \(p. 208\)](#)
- [Métricas por estilhaço \(p. 211\)](#)

## Métricas do aplicativo por KCL

Essas métricas são agregadas em todos os operadores da KCL no escopo do aplicativo, conforme definido pelo namespace do Amazon CloudWatch.

### Tópicos

- [InitializeTask \(p. 205\)](#)
- [ShutdownTask \(p. 205\)](#)
- [ShardSyncTask \(p. 206\)](#)
- [BlockOnParentTask \(p. 207\)](#)
- [PeriodicShardSyncManager \(p. 207\)](#)
- [MultistreamTracker \(p. 208\)](#)

## InitializeTask

A operação `InitializeTask` é responsável por inicializar o processador de registros para o aplicativo da KCL. A lógica dessa operação inclui a obtenção de um iterador de estilhaços do Kinesis Data Streams e a inicialização do processador de registros.

Métrica	Description (Descrição)
<code>KinesisDataFetcher.getIteratorSuccess</code>	Número de operações <code>GetShardIterator</code> bem-sucedidas por aplicativo da KCL.  Nível métrico: Detalhado  Unidades Contagem
<code>KinesisDataFetcher.getIteratorTime</code>	Tempo usado por operação <code>GetShardIterator</code> para o aplicativo da KCL determinado.  Nível métrico: Detalhado  Unidades Milissegundos
<code>RecordProcessor.initializeTime</code>	Tempo percorrido pelo método de inicialização do processador de registros.  Nível métrico: Resumo  Unidades Milissegundos
Bem-sucedida	Número de inicializações bem-sucedidas do processador de registros.  Nível métrico: Resumo  Unidades Contagem
Time (Tempo)	Tempo percorrido pelo operador do KCL para a inicialização do processador de registros.  Nível métrico: Resumo  Unidades 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	Description (Descrição)
<code>CreateLease.Success</code>	O número de vezes que novos estilhaços filho são adicionados com sucesso à tabela do DynamoDB do aplicativo da KCL depois do desligamento do estilhaço pai.  Nível métrico: Detalhado  Unidades Contagem

Métrica	Description (Descrição)
CreateLease.Time	Tempo usado para adicionar informações do novo estilhaço filho à tabela do DynamoDB do aplicativo da KCL.  Nível métrico: Detalhado  Unidades Milissegundos
UpdateLease.Success	Número de pontos de verificação finais bem-sucedidos durante o desligamento do processador de registros.  Nível métrico: Detalhado  Unidades Contagem
UpdateLease.Time	Tempo necessário para a operação de pontos de verificação durante o desligamento do processador de registros.  Nível métrico: Detalhado  Unidades Milissegundos
RecordProcessor.shutdownTime	Tempo percorrido pelo método de desligamento do processador de registros.  Nível métrico: Resumo  Unidades Milissegundos
Bem-sucedida	Número de tarefas de desligamento bem-sucedidas.  Nível métrico: Resumo  Unidades Contagem
Time (Tempo)	Tempo percorrido pelo operador do KCL para a tarefa de desligamento.  Nível métrico: Resumo  Unidades Milissegundos

## ShardSyncTask

A operação `ShardSyncTask` detecta alterações nas informações de estilhaços do Kinesis data stream para que novos estilhaços possam ser processados pelo aplicativo da KCL.

Métrica	Description (Descrição)
CreateLease.Success	Número de tentativas bem-sucedidas para adicionar novas informações de estilhaços à tabela do DynamoDB do aplicativo da KCL.  Nível métrico: Detalhado  Unidades Contagem
CreateLease.Time	Tempo usado para adicionar informações do novo estilhaço à tabela do DynamoDB do aplicativo da KCL.  Nível métrico: Detalhado

Métrica	Description (Descrição)
	Unidades Milissegundos
Bem-sucedida	Número de operações bem-sucedidas de sincronização de estilhaços.  Nível métrico: Resumo  Unidades Contagem
Time (Tempo)	Tempo percorrido para a operação de sincronização de estilhaços.  Nível métrico: Resumo  Unidades Milissegundos

## BlockOnParentTask

Se o estilhaço é dividido ou mesclado com outros, novos estilhaços filhos são criados. A operação `BlockOnParentTask` garante que o processamento de registros dos novos estilhaços não será iniciado até que os estilhaços pai sejam completamente processados pela KCL.

Métrica	Description (Descrição)
Bem-sucedida	Número de verificações bem-sucedidas para a conclusão de estilhaços pai.  Nível métrico: Resumo  Unidades Contagem
Time (Tempo)	Tempo percorrido para a conclusão de estilhaços pai.  Nível métrico: Resumo  Unidade Milissegundos

## PeriodicShardSyncManager

O `PeriodicShardSyncManager` é responsável por examinar os fluxos de dados que estão a ser processados pelo KCL aplicação do consumidor, identificar fluxos de dados com locações parciais e entregá-los para sincronização.

As seguintes métricas estão disponíveis quando o KCL está configurado para processar um único fluxo de dados (então o valor de `numagistosync` e `numcustoswithpartiallocases` está definido para 1) e também quando KCL está configurado para processar vários fluxos de dados.

Métrica	Description (Descrição)
numagistosinalizado	O número de fluxos de dados (por conta AWS) que estão a ser processados pela aplicação do consumidor que contêm locações parciais e que devem ser entregues para sincronização.  Nível métrico: Resumo  Unidades Contagem

Métrica	Description (Descrição)
<code>simplifiedLocations</code>	O número de fluxos de dados (por conta AWS) que a aplicação do consumidor está a processar, que contêm alugueres parciais.  Nível métrico: Resumo  Unidades Contagem
<code>Bem-sucedida</code>	O número de vezes <code>PeriodicShardSyncManager</code> foi capaz de identificar com sucesso locais parciais nos fluxos de dados que a aplicação do consumidor está a processar.  Nível métrico: Resumo  Unidades Contagem
<code>Time (Tempo)</code>	A quantidade do tempo (em milissegundos) que o <code>PeriodicShardSyncManager</code> para examinar os fluxos de dados que a aplicação do consumidor está a processar, de modo a determinar que fluxos de dados requerem sincronização de canais.  Nível métrico: Resumo  Unidades Milissegundos

## MultistreamTracker

O `MultistreamTracker` A interface permite-lhe construir aplicações de consumidor KCL que podem processar vários fluxos de dados ao mesmo tempo.

Métrica	Description (Descrição)
<code>eliminarfluxos.Contagem</code>	O número de fluxos de dados eliminados neste período de tempo.  Nível métrico: Resumo  Unidades Contagem
<code>dicientes.Contagem</code>	O número de fluxos de dados ativos a serem processados.  Nível métrico: Resumo  Unidades Contagem
<code>alisãodeeliminação.Contagem</code>	O número de fluxos de dados que estão pendentes com base em <code>FormerStreamsLeasesDeletionStrategy</code> .  Nível métrico: Resumo  Unidades Contagem

## Métricas por operador

Essas métricas são agregadas em todos os processadores de registros que consomem dados de um Kinesis data stream, como uma instância do Amazon EC2.

#### Tópicos

- [RenewAllLeases](#) (p. 209)
- [TakeLeases](#) (p. 209)

## RenewAllLeases

A operação `RenewAllLeases` renova periodicamente concessões de estilhaços de propriedade de uma determinada instância de operador.

Métrica	Description (Descrição)
RenewLease.Success	Número de renovações bem-sucedidas de concessões por parte do operador.  Nível métrico: Detalhado  Unidades Contagem
RenewLease.Time	O tempo necessário para a operação de renovação de concessões.  Nível métrico: Detalhado  Unidades Milissegundos
CurrentLeases	Número de concessões de estilhaços pertencentes ao operador depois que todas as concessões foram renovadas.  Nível métrico: Resumo  Unidades Contagem
LostLeases	Número de concessões de estilhaços perdidas após uma tentativa de renovar todas as concessões pertencentes ao operador.  Nível métrico: Resumo  Unidades Contagem
Bem-sucedida	Número de vezes que a operação de renovação da concessão foi bem-sucedida para o operador.  Nível métrico: Resumo  Unidades Contagem
Time (Tempo)	Tempo percorrido para renovar todas as concessões para o operador.  Nível métrico: Resumo  Unidades Milissegundos

## TakeLeases

A operação `TakeLeases` equilibra o processamento de registros entre todos os operadores da KCL. Se o operador do KCL atual tem menos concessões de estilhaços que o necessário, usa concessões de estilhaços de outro operador que está sobrecarregado.

Métrica	Description (Descrição)
ListLeases.Success	<p>Número de vezes que todas as concessões de estilhaços foram recuperadas com sucesso da tabela do DynamoDB do aplicativo da KCL.</p> <p>Nível métrico: Detalhado</p> <p>Unidades Contagem</p>
ListLeases.Time	<p>Tempo usado para recuperar todas as concessões de estilhaços da tabela do DynamoDB do aplicativo da KCL.</p> <p>Nível métrico: Detalhado</p> <p>Unidades Milissegundos</p>
TakeLease.Success	<p>Número de vezes que o operador usou com êxito concessões de estilhaços de outros operadores do KCL.</p> <p>Nível métrico: Detalhado</p> <p>Unidades Contagem</p>
TakeLease.Time	<p>Tempo percorrido para atualizar a tabela de concessão com concessões executadas pelo operador.</p> <p>Nível métrico: Detalhado</p> <p>Unidades Milissegundos</p>
NumWorkers	<p>Número total de operadores, conforme identificado por um operador específico.</p> <p>Nível métrico: Resumo</p> <p>Unidades Contagem</p>
NeededLeases	<p>Número de concessões de estilhaços que o operador atual precisa para uma carga de processamento de estilhaços equilibrada.</p> <p>Nível métrico: Detalhado</p> <p>Unidades Contagem</p>
LeasesToTake	<p>O número de concessões que o operador tentará executar.</p> <p>Nível métrico: Detalhado</p> <p>Unidades Contagem</p>
TakenLeases	<p>Número de concessões realizadas com sucesso pelo operador.</p> <p>Nível métrico: Resumo</p> <p>Unidades Contagem</p>
TotalLeases	<p>Número total de estilhaços que o aplicativo KCL está processando.</p> <p>Nível métrico: Detalhado</p> <p>Unidades Contagem</p>



Métrica	Description (Descrição)
ExpiredLeases	Número total de estilhaços que não estão sendo processados por nenhum operador, conforme identificado pelo operador específico.  Nível métrico: Resumo  Unidades Contagem
Bem-sucedida	Número de vezes que a operação <code>TakeLeases</code> foi concluída com sucesso.  Nível métrico: Resumo  Unidades Contagem
Time (Tempo)	Tempo percorrido pela operação <code>TakeLeases</code> para um operador.  Nível métrico: Resumo  Unidades Milissegundos

## Métricas por estilhaço

Essas métricas são agregadas em um único processador de registros.

### ProcessTask

A operação `ProcessTask` chama `GetRecords` com a posição do iterador atual para recuperar registros do streaming e chama a função `processRecords` do processador de registros.

Métrica	Description (Descrição)
<code>KinesisDataFetcher.getRecords</code>	Número de operações <code>GetRecords</code> bem-sucedidas por estilhaço do Kinesis data stream.  Nível métrico: Detalhado  Unidades Contagem
<code>KinesisDataFetcher.getRecordsTime</code>	Tempo percorrido por operação <code>GetRecords</code> para o estilhaço do Kinesis data stream.  Nível métrico: Detalhado  Unidades Milissegundos
<code>UpdateLease.Success</code>	Número de pontos de verificação bem-sucedidos feitos pelo processador de registros para o determinado estilhaço.  Nível métrico: Detalhado  Unidades Contagem
<code>UpdateLease.Time</code>	Tempo percorrido para cada operação de ponto de verificação para o determinado estilhaço.  Nível métrico: Detalhado  Unidades Milissegundos

Métrica	Description (Descrição)
DataBytesProcessed	<p>Tamanho total de registros processados em bytes em cada chamada de <code>ProcessTask</code>.</p> <p>Nível métrico: Resumo</p> <p>Unidades Byte</p>
RecordsProcessed	<p>Número de registros processados em cada chamada de <code>ProcessTask</code>.</p> <p>Nível métrico: Resumo</p> <p>Unidades Contagem</p>
ExpiredIterator	<p>Número de <code>ExpiredIteratorException</code> recebidos durante a chamada de <code>GetRecords</code>.</p> <p>Nível métrico: Resumo</p> <p>Unidades Contagem</p>
MillisBehindLatest	<p>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.</p> <p>Nível métrico: Resumo</p> <p>Unidades Milissegundos</p>
RecordProcessor.processRecordsTime	<p>Tempo decorrido pelo método <code>processRecords</code> do processador de registros.</p> <p>Nível métrico: Resumo</p> <p>Unidades Milissegundos</p>
Bem-sucedida	<p>Número de operações bem-sucedidas de tarefas do processo.</p> <p>Nível métrico: Resumo</p> <p>Unidades Contagem</p>
Time (Tempo)	<p>Tempo percorrido para a operação de tarefas do processo.</p> <p>Nível métrico: Resumo</p> <p>Unidades Milissegundos</p>

## Monitoramento da Kinesis Producer Library com o Amazon CloudWatch

A [Kinesis Producer Library](#) (KPL) para Amazon Kinesis Data Streams publica métricas personalizadas do Amazon CloudWatch em seu nome. Você pode visualizar essas métricas navegando até o [console](#)

do [CloudWatch](#) e escolhendo Custom Metrics (Métricas personalizadas). Para obter mais informações sobre métricas personalizadas, consulte [Publicar métricas personalizadas](#) no Guia do usuário do Amazon CloudWatch.

Há uma cobrança nominal para as métricas obtidas por upload no CloudWatch pela KPL especificamente, aplicam-se cobranças de métricas personalizadas do Amazon CloudWatch e solicitações de API do Amazon CloudWatch. Para obter mais informações, consulte [Definição de preço do Amazon CloudWatch](#). A coleta de métricas locais não gera cobranças do CloudWatch.

#### Tópicos

- [Métricas, dimensões e namespaces](#) (p. 213)
- [Granularidade e nível de métrica](#) (p. 213)
- [Acesso local e carregamento do Amazon CloudWatch](#) (p. 214)
- [Lista de métricas do](#) (p. 214)

## Métricas, dimensões e namespaces

Você pode especificar um nome de aplicativo ao ativar o KPL, que é, então, usado como parte do namespace durante o carregamento de métricas. Isso é opcional; o KPL oferece um valor padrão se um nome de aplicativo não é definido.

Você também pode configurar o KPL para adicionar dimensões adicionais arbitrárias às métricas. Isso é útil se você deseja dados mais refinados nas métricas do 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 definições de configuração da KPL são imutáveis; portanto, não é possível alterar essas dimensões adicionais após a instância da KPL ser inicializada.

## Granularidade e nível de métrica

Há duas opções para controlar o número de métricas carregadas para o CloudWatch:

#### nível de métrica

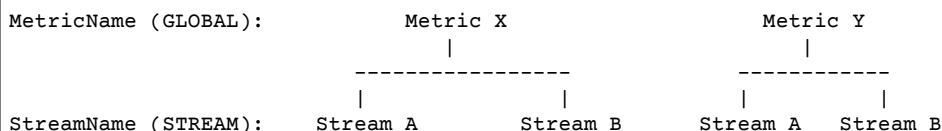
Este é um indicador aproximado da importância de uma métrica. Cada métrica é atribuída a um nível. Ao definir um nível, as métricas com níveis menores não são enviadas ao CloudWatch. Os níveis são `NONE`, `SUMMARY`, e `DETAILED`. A predefinição é `DETAILED` ou seja, todas as métricas. `NONE` significa que não há métricas, de modo que nenhuma métrica é atribuída a esse nível.

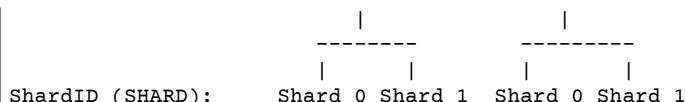
#### granularidade

Controla se a mesma métrica é emitida em níveis adicionais de granularidade. Os níveis são `GLOBAL`, `STREAM`, e `SHARD`. A predefinição 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 streams com dois estilhaços cada produzirão sete métricas do CloudWatch: uma para cada estilhaço, um para cada stream e uma geral; todas descrevem 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:





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 especificar uma dimensão adicional, tem de fornecer valores para `tuple:<DimensionName, DimensionValue, Granularity>`. A granularidade é utilizada para determinar onde a dimensão personalizada é inserida na hierarquia: `GLOBAL` significa que a dimensão adicional é inserida após o nome métrico, `STREAM` significa que é inserido após o nome do fluxo, e `SHARD` significa que é inserido após a identificação do compartimento. Se várias dimensões adicionais são fornecidas por nível de granularidade, elas são inseridas na ordem determinada.

## Acesso local e carregamento do Amazon CloudWatch

Métricas para a instância atual da KPL estão disponíveis localmente em tempo real; é possível consultar a KPL a qualquer momento para obtê-las. A KPL calcula localmente a soma, a média, o mínimo, o máximo, e a contagem de cada métrica, como no 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 carregamento ao 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, a KPL agrega métricas localmente em buckets de 1 minuto e faz upload de um objeto de estatística no CloudWatch uma vez por minuto, por métrica habilitada.

## Lista de métricas do

Métrica	Description (Descrição)
<code>UserRecordsReceived</code>	<p>Contagem de quantos registros de usuário lógicos foram recebidos pelo núcleo do KPL para operações put. Não disponível no nível de estilhaço.</p> <p>Nível métrico: Detalhado</p> <p>Unidade Contagem</p>
<code>UserRecordsPending</code>	<p>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.</p> <p>O KPL oferece um método dedicado para recuperar essa métrica em nível global para os clientes gerenciarem a taxa de colocação.</p>

Métrica	Description (Descrição)
	Nível métrico: Detalhado Unidade Contagem
UserRecordsPut	Contagem de quantos registros de usuário lógicos foram colocados com êxito.  O KPL não conta registros com falhas para essa métrica. 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 métrico: Resumo Unidade Contagem
UserRecordsDataPut	Bytes nos registros de usuário lógicos colocados com êxito.  Nível métrico: Detalhado Unidade Bytes
KinesisRecordsPut	Contagem de quantos registros do Kinesis Data Streams foram colocados com êxito (cada registro do Kinesis Data Streams pode conter vários registros de usuário).  O KPL retorna 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 métrico: Resumo Unidade Contagem
KinesisRecordsDataPut	Bytes nos registros do Kinesis Data Streams.  Nível métrico: Detalhado Unidade Bytes
ErrorsByCode	Contagem de cada tipo de código de erro. Isto introduz uma dimensão adicional de <code>ErrorCode</code> , para além das dimensões normais, como <code>StreamName</code> e <code>ShardId</code> . Nem todos os erros podem ser rastreados para um canalizador. 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 da API do Kinesis 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 métrico: Resumo Unidade Contagem

Métrica	Description (Descrição)
<b>AllErrors</b>	<p>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.</p> <p>Nível métrico: Resumo</p> <p>Unidade Contagem</p>
<b>RetriesPerRecord</b>	<p>Número de tentativas realizadas por registro de usuário. Zero é emitido para registros que são bem-sucedidos em uma tentativa.</p> <p>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 tempo de vida do registro é um valor grande, essa métrica pode ficar significativamente atrasada.</p> <p>Nível métrico: Detalhado</p> <p>Unidade Contagem</p>
<b>BufferingTime</b>	<p>O tempo entre um registro de usuário que chega ao KPL e sai para o back-end. Essas informações são transmitidas de volta ao usuário por registro, mas também estão disponíveis como estatística agregada.</p> <p>Nível métrico: Resumo</p> <p>Unidade Milissegundos</p>
<b>Request Time</b>	<p>O tempo necessário para executar <code>PutRecordsRequests</code>.</p> <p>Nível métrico: Detalhado</p> <p>Unidade Milissegundos</p>
<b>User Records per Kinesis Record</b>	<p>O número de registros de usuário lógicos agregados em um único registro do Kinesis Data Streams.</p> <p>Nível métrico: Detalhado</p> <p>Unidade Contagem</p>
<b>Amazon Kinesis Records per PutRecordsRequest</b>	<p>O número de Kinesis Data Streams registros agregados num único <code>PutRecordsRequest</code>. Não disponível ao nível do compartimento.</p> <p>Nível métrico: Detalhado</p> <p>Unidade Contagem</p>
<b>User Records per PutRecordsRequest</b>	<p>O número total de registros de utilizador contidos num <code>PutRecordsRequest</code>. Isto é aproximadamente equivalente ao produto das duas métricas anteriores. Não disponível no nível de estilhaço.</p> <p>Nível métrico: Detalhado</p> <p>Unidade Contagem</p>

# Segurança no Amazon Kinesis Data Streams

A segurança da nuvem na AWS é a nossa maior prioridade. Como um cliente da AWS, você se beneficiará de um datacenter e uma arquitetura de rede criados para atender os requisitos da maioria das organizações com exigências de segurança.

A segurança é uma responsabilidade compartilhada entre a AWS e você. O [modelo de responsabilidade compartilhada](#) descreve a segurança da nuvem e a segurança na nuvem:

- Segurança da nuvem – a AWS é responsável pela proteção da infraestrutura que executa serviços da AWS na nuvem da AWS. A AWS também fornece serviços que você pode usar com segurança. A eficácia da nossa segurança é regularmente testada e verificada por auditores terceirizados 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 serviço da AWS que você usa. Você também é responsável por outros fatores, incluindo a confidencialidade de seus dados, os requisitos da sua organização e as 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 cumprir os objetivos de segurança e conformidade. Você também aprenderá como usar outros serviços da AWS que podem ajudar a monitorar e proteger seus recursos do Kinesis Data Streams.

## Tópicos

- [Proteção de dados no Amazon Kinesis Data Streams \(p. 217\)](#)
- [Controle do acesso aos recursos do Amazon Kinesis Data Streams usando o IAM \(p. 225\)](#)
- [Validação de conformidade do Amazon Kinesis Data Streams \(p. 228\)](#)
- [Resiliência no Amazon Kinesis Data Streams \(p. 229\)](#)
- [Segurança da infraestrutura no Kinesis Data Streams \(p. 230\)](#)
- [Práticas recomendadas de segurança do Kinesis Data Streams \(p. 230\)](#)

## Proteção de dados no Amazon Kinesis Data Streams

A criptografia no lado do servidor usando chaves do AWS Key Management Service (AWS KMS) facilita o atendimento de rigorosos requisitos de gerenciamento de seus dados criptografando os dados em repouso no Amazon Kinesis Data Streams.

## Tópicos

- [O que é a criptografia no lado do servidor para o Kinesis Data Streams? \(p. 218\)](#)
- [Custos, regiões e considerações sobre desempenho \(p. 218\)](#)
- [Como começo a usar a criptografia no lado do servidor? \(p. 219\)](#)
- [Criação e uso de chaves mestras do KMS geradas pelo usuário \(p. 220\)](#)
- [Permissões para usar as chaves mestras do KMS geradas pelo usuário \(p. 221\)](#)
- [Verificação e solução de problemas de permissões de chaves do KMS \(p. 222\)](#)
- [Usar o Amazon Kinesis Data Streams com VPC endpoints de interface \(p. 222\)](#)

## O que é a criptografia no lado do servidor para o Kinesis Data Streams?

A criptografia no lado do servidor é um recurso no Amazon Kinesis Data Streams que criptografa automaticamente os dados antes de estarem em repouso usando uma chave mestra de cliente (CMK) do AWS KMS especificada por você. Os dados são criptografados antes de serem gravados na camada de armazenamento do stream 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 stream do Kinesis não precisam gerenciar chaves mestras ou operações de criptografia. Se a retenção de dados estiver habilitada, os dados serão criptografados automaticamente conforme entram e saem do Kinesis Data Streams, de modo que os dados em repouso são criptografados. O AWS KMS fornece todas as chaves mestras usadas pelo recurso de criptografia no lado do servidor. O AWS KMS facilita o uso de uma CMK para o Kinesis que é gerenciada pelo AWS, uma CMK do AWS KMS especificada pelo usuário, ou uma chave mestra importada no serviço do 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.

## Custos, regiões e considerações sobre desempenho

Ao aplicar criptografia no lado do servidor, você está sujeito aos custos de uso da chave e da API do AWS KMS. Ao contrário das chaves mestras do KMS personalizadas, a chave mestra do cliente (CMK) do (Default) `aws/kinesis` é oferecida gratuitamente. No entanto, você ainda deverá pagar os custos de uso da API que o Amazon Kinesis Data Streams gera em seu nome.

Os custos de uso da API aplicam-se a cada CMK, incluindo as personalizadas. O Kinesis Data Streams chama o AWS KMS aproximadamente a cada cinco minutos quando está alternando a chave de dados. Em um mês de 30 dias, o custo total das chamadas à API do AWS KMS que são iniciadas por um streaming do Kinesis deve ser de apenas alguns dólares. Esse custo cresce com o número de credenciais de usuário que você usa nos seus produtores e consumidores de dados, pois cada credencial de usuário requer uma chamada de API exclusiva para o AWS KMS. Quando você usa uma função do IAM para autenticação, cada chamada de função assumida resulta em credenciais de usuário exclusivas. Para economizar gastos com o KMS, armazene em cache as credenciais de usuário que são retornadas pela chamada de função assumida.

Veja a seguir a descrição dos custos por recurso:

### Chaves

- A CMK do Kinesis que é gerenciada pela AWS (alias = `aws/kinesis`) é gratuita.
- As chaves do KMS geradas pelo usuário estão sujeitas aos custos de chave do KMS. Para mais informações, consulte [definição de preço do AWS Key Management Service](#).

## Uso da API do KMS

Para cada streaming criptografado, o serviço do Kinesis chama o serviço do AWS KMS aproximadamente a cada cinco minutos para criar uma nova chave de criptografia de dados. Em um mês de 30 dias, cada fluxo criptografado gera aproximadamente 8.640 solicitações à API do KMS. As solicitações à API para gerar novas chaves de criptografia de dados estão sujeitas aos custos de uso do AWS KMS. Para mais informações, consulte [Definição de preço do AWS Key Management Service: uso](#).



## Disponibilidade da criptografia no lado do servidor por região

A criptografia no lado do servidor de streams do Kinesis está disponível nas regiões a seguir.

Nome da região	Região
Leste dos EUA (Ohio)	us-east-2
Leste dos EUA (Norte da Virgínia)	us-east-1
Oeste dos EUA (Oregon)	us-west-2
Oeste dos EUA (Norte da Califórnia)	us-west-1
AWS GovCloud (US-West)	us-gov-west-1
Canadá (Central)	ca-central-1
Europa (Irlanda)	eu-west-1
Europa (Londres)	eu-west-2
Europa (Frankfurt)	eu-central-1
Região Ásia-Pacífico (Tóquio)	ap-northeast-1
Região da Ásia-Pacífico (Seul)	ap-northeast-2
Ásia-Pacífico (Cingapura)	ap-southeast-1
Ásia Pacífico (Mumbai)	ap-south-1
Ásia-Pacífico (Sydney)	ap-southeast-2
América do Sul (São Paulo)	sa-east-1

## 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 começo a usar a criptografia no lado do servidor?

A maneira mais fácil de começar a usar a criptografia no lado do servidor é usar o Console de gerenciamento da AWS e a chave do serviço do KMS do Amazon Kinesis, `aws/kinesis`.

O procedimento a seguir demonstra como ativar a criptografia no lado do servidor para um stream do Kinesis.

Para ativar a criptografia no lado do servidor para um stream do Kinesis

1. Faça login no Console de gerenciamento da AWS e abra o [Amazon Kinesis Data Streams console](#).

2. Crie ou selecione um streaming do Kinesis no Console de gerenciamento da AWS.
3. Selecione a guia Details (Detalhes).
4. Em Server-side encryption (Criptografia no lado do servidor), selecione Edit (Editar).

### Server-side encryption

Enable server-side encryption to encrypt sensitive data in the Kinesis stream with an AWS KMS master key. [Learn more](#)

Server-side encryption Disabled

5. A não ser que você queira usar uma chave mestra do KMS gerada pelo usuário, selecione a chave mestra do KMS (Default) aws/kinesis ((Padrão) aws/kinesis). Essa é a chave mestra do KMS gerada pelo serviço Kinesis. Selecione Enabled (Habilitado) e, em seguida, selecione Save (Salvar).

#### Note

A chave mestra padrão do serviço do Kinesis é gratuita, porém, as chamadas à API feitas pelo Kinesis para o serviço do AWS KMS estão sujeitas a custos de uso do KMS.

### Server-side encryption

Enable server-side encryption to encrypt sensitive data in the Kinesis stream with an AWS KMS master key. [Learn more](#)

Server-side encryption ☐ Disabled ☒ Enabled

KMS master key (Default) aws/kinesis

6. O fluxo passa por um estado pending (pendente). Assim que o fluxo voltar a um estado active (ativo) com a criptografia habilitada, todos os dados recebidos gravados no fluxo serão criptografados usando a chave mestra selecionada do KMS.
7. Para desabilitar a criptografia do lado do servidor, selecione Disabled (Desabilitada) em Server-side encryption (Criptografia do lado do servidor) no Console de gerenciamento da AWS e, em seguida, selecione Save (Salvar).

## Criação e uso de chaves mestras do KMS geradas pelo usuário

Esta seção descreve como criar e usar suas próprias chaves mestras do KMS em vez de usar a chave mestra administrada pelo Amazon Kinesis.

### Criação de chaves mestras do KMS geradas pelo usuário

Para obter instruções sobre como criar suas próprias chaves mestras, consulte [Criação de chaves](#) no AWS Key Management Service Developer Guide. Depois de criar chaves para sua conta, o serviço do Kinesis Data Streams retorna essas chaves na lista KMS master key (Chave mestra do KMS).

### Uso de chaves mestras do KMS geradas pelo usuário

Assim que as permissões corretas forem aplicadas aos seus consumidores, produtores e administradores, você poderá usar chaves mestras personalizadas do KMS em sua própria conta da AWS ou em outra

conta da AWS. Todas as chaves mestras do KMS em sua conta aparecem na lista KMS Master Key (Chave mestra do KMS) no Console de gerenciamento da AWS.

Para usar chaves mestras do KMS personalizadas localizadas em outra conta, você precisa de permissões para usar essas chaves. Você também deve especificar o ARN da chave mestra do KMS na caixa de entrada ARN no Console de gerenciamento da AWS.

## Permissões para usar as chaves mestras do KMS geradas pelo usuário

Antes que você possa usar a criptografia no lado do servidor com uma chave mestra do KMS gerada pelo usuário, você deve configurar as políticas de chaves do AWS KMS para permitir a criptografia de streams e a criptografia e a descryptografia de registros de stream. Para obter exemplos e mais informações sobre as permissões do AWS KMS, consulte [Permissões da API do KMS do AWS: referência de ações e recursos](#).

### Note

O uso da chave padrão do serviço para criptografia não requer a aplicação de permissões personalizadas do IAM.

Antes de usar as chaves mestras do KMS geradas pelo usuário, verifique se seus produtores e consumidores de streaming do Kinesis (entidades principais do IAM) são usuários na política de chaves mestras do KMS. 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 chaves do KMS usando políticas do IAM. Para mais informações, consulte [Uso de políticas do IAM com o AWS KMS](#).

## Exemplo de permissões de produtor

Seus produtores de streaming 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 de consumidor

Seus consumidores de streaming 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"
  }
]
```

O Amazon Kinesis Data Analytics e o AWS Lambda usam funções para consumir streamings do Kinesis. Adicione a permissão `kms:Decrypt` às funções que esses consumidores usam.

## Permissões de administrador de stream

Os administradores de streaming do Kinesis devem ter autorização para chamar `kms:List*` e `kms:DescribeKey*`.

## Verificação e solução de problemas de permissões de chaves do KMS

Depois de habilitar a criptografia em um streaming do Kinesis, recomendamos que você monitore o sucesso de suas chamadas à `putRecord`, `putRecords` e `getRecords` usando as seguintes métricas do Amazon CloudWatch:

- `PutRecord.Success`
- `PutRecords.Success`
- `GetRecords.Success`

Para obter mais informações, consulte [Monitorar o Amazon Kinesis Data Streams \(p. 188\)](#)

## Usar o Amazon Kinesis Data Streams com VPC endpoints de interface

Você pode usar um VPC endpoint de interface para manter o tráfego entre sua Amazon VPC e o Kinesis Data Streams saindo da rede da Amazon. VPC endpoints de interface não exigem um gateway de Internet, dispositivo NAT, conexão VPN ou conexão do AWS Direct Connect. Os VPC endpoints de interface são desenvolvidos pelo AWS PrivateLink, uma tecnologia da AWS que permite a comunicação privada entre os serviços da AWS usando uma interface de rede elástica com IPs privados em sua Amazon VPC. Para obter mais informações, consulte [Amazon Virtual Private Cloud](#) e [VPC endpoints de interface \(AWS PrivateLink\)](#).

### Tópicos

- [Usar VPC endpoints de interface para o Kinesis Data Streams \(p. 223\)](#)
- [Controlar o acesso a endpoints de VPCE para o Kinesis Data Streams \(p. 223\)](#)

- [Disponibilidade de políticas de VPC endpoint para o Kinesis Data Streams \(p. 224\)](#)

## Usar VPC endpoints de interface para o 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 de entrada e de saída do Kinesis Data Streams para os recursos da Amazon VPC comece a fluir por meio do endpoint da VPC da interface. Para obter mais informações, consulte [Criação de um endpoint de interface](#).

A Kinesis Producer Library (KPL) e a Kinesis Consumer Library (KCL) chamam serviços da AWS, como o Amazon CloudWatch e o Amazon DynamoDB, usando endpoints públicos ou VPC endpoints de interface privados, dependendo do que já estiver sendo usado. Por exemplo, se o aplicativo da KCL está sendo executado em uma VPC com interface do DynamoDB e VPC endpoints habilitados, as chamadas entre o DynamoDB e o aplicativo da KCL estão transcorrendo por meio do VPC endpoint de interface.

## Controlar o acesso a endpoints de VPCE para o Kinesis Data Streams

As políticas de VPC endpoint permitem que você controle o acesso anexando uma política a um VPC endpoint ou usando campos adicionais em uma política anexada a um usuário, um grupo ou uma função do IAM para restringir o acesso a ocorrer apenas por meio do VPC endpoint especificado. Essas políticas podem ser usadas para restringir o acesso a fluxos específicos a um VPC endpoint especificado quando usadas em conjunto com as políticas do IAM para conceder acesso somente a ações de fluxo de dados do Kinesis por meio do VPC endpoint especificado.

Veja a seguir exemplos de políticas de endpoint para acessar fluxos de dados do Kinesis.

- Exemplo de política de VPC: acesso somente leitura — esse exemplo de política pode ser anexado a um VPC endpoint. (Para obter mais informações, consulte [Como controlar o acesso aos recursos da Amazon VPC](#)). Ele restringe as ações a somente listar e descrever um fluxo de dados do Kinesis por meio do VPC endpoint ao qual está anexado.

```
{
  "Statement": [
    {
      "Sid": "ReadOnly",
      "Principal": "*",
      "Action": [
        "kinesis:List*",
        "kinesis:Describe*"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

- Exemplo de política de VPC: restringir o acesso a um fluxo 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 anexado.

```
{
  "Statement": [
    {
      "Sid": "AccessToSpecificDataStream",
      "Principal": "*",
      "Action": "kinesis:*",

```

```
    "Effect": "Allow",
    "Resource": "arn:aws:kinesis:us-east-1:123456789012:stream/MyStream"
  }
]
```

- Exemplo de política do IAM: restringir o acesso a um fluxo específico apenas de determinado VPC endpoint — esse exemplo de política pode ser anexado a um usuário, uma função ou um grupo do IAM. Ele restringe o acesso a um fluxo de dados especificado do Kinesis para ocorrer somente em determinado VPC endpoint.

```
{
  "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

VPC endpoints de interface do Kinesis Data Streams com políticas são compatíveis com as seguintes regiões:

- Oeste dos EUA (Oregon)
- Europa (Paris)
- Europa (Irlanda)
- Leste dos EUA (Norte da Virgínia)
- Europa (Estocolmo)
- Ásia Pacífico (Mumbai)
- Leste dos EUA (Ohio)
- Europa (Frankfurt)
- América do Sul (São Paulo)
- Ásia-Pacífico (Seul)
- Europa (Londres)
- Ásia-Pacífico (Tóquio)
- Oeste dos EUA (Norte da Califórnia)
- Ásia-Pacífico (Cingapura)
- Ásia-Pacífico (Sydney)
- Canadá (Central)
- China (Pequim)
- China (Ningxia)
- Ásia-Pacífico (Hong Kong)
- Oriente Médio (Bahrein)

- Europa (Milão)
- África (Cidade do Cabo)

## Controle do acesso aos recursos do Amazon Kinesis Data Streams usando o IAM

O AWS Identity and Access Management (IAM) permite que você:

- Crie usuários e grupos na conta da AWS
- Atribua credenciais de segurança exclusivas a cada usuário em sua conta da AWS
- Controle as permissões de cada usuário para executar tarefas usando recursos da AWS
- Permita que os usuários em outra conta da AWS compartilhem seus recursos da AWS
- Crie funções para sua conta da AWS e defina os usuários ou os serviços que podem assumi-las
- Use identidades existentes em sua empresa a fim de conceder permissões para executar tarefas usando recursos da AWS

Ao usar o IAM com o Kinesis Data Streams, você pode controlar se os usuários de sua organização podem executar uma tarefa usando ações específicas da API do Kinesis Data Streams e se podem usar recursos específicos da AWS.

Se você estiver desenvolvendo um aplicativo usando a Kinesis Client Library (KCL), a política precisa incluir permissões para o Amazon DynamoDB e o Amazon CloudWatch. A KCL usa o DynamoDB para rastrear informações de estado do aplicativo, e o CloudWatch para enviar métricas da KCL ao CloudWatch em seu nome. Para obter mais informações sobre o KCL, consulte [Desenvolver consumidores do KCL 1.x](#) (p. 131).

Para obter mais informações sobre IAM, consulte o seguinte:

- [AWS Identity and Access Management \(IAM\)](#)
- [Conceitos básicos](#)
- [Guia do usuário do IAM](#)

Para obter mais informações sobre o IAM e o Amazon DynamoDB, consulte [Uso do IAM para controlar o acesso aos recursos do Amazon DynamoDB](#) no Guia do desenvolvedor do Amazon DynamoDB.

Para obter mais informações sobre o IAM e o Amazon CloudWatch, consulte [Controle de acesso do usuário à sua conta da AWS](#) no Guia do usuário do Amazon CloudWatch.

### Tópicos

- [Sintaxe da política](#) (p. 225)
- [Ações do Kinesis Data Streams](#) (p. 226)
- [Nomes de recurso da Amazon \(ARNs\) para o Kinesis Data Streams](#) (p. 226)
- [Exemplo de políticas para o Kinesis Data Streams](#) (p. 227)

## Sintaxe da política

A política do IAM é um documento JSON 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:

- **Effect:** o efeito pode ser **Allow** ou **Deny**. Por padrão, os usuários do IAM não têm permissão para usar recursos e ações da API. Por isso, todas as solicitações são negadas. Uma permissão explícita substitui o padrão. Uma negação explícita substitui todas as permissões.
- **Action:** a ação é a ação de API específica para a qual você está concedendo ou negando permissão.
- **Resource:** o recurso afetado pela ação. Para especificar um recurso na declaração, você precisa usar o Amazon Resource Name (ARN – Nome de recurso da Amazon).
- **Condition:** condições são opcionais. Elas podem ser usadas para controlar quando as políticas entrarão em vigor.

Conforme cria e gerencia políticas do IAM, você pode querer usar o [Gerador de políticas do IAM](#) e o [Simulador de políticas do IAM](#).

## Ações do Kinesis Data Streams

Em uma declaração de política do IAM, você pode especificar qualquer ação de API de qualquer serviço que dê suporte ao IAM. Para o Kinesis Data Streams, use o seguinte prefixo com o nome da ação da API `kinesis:`. Por exemplo: `kinesis:CreateStream`, `kinesis:ListStreams` e `kinesis:DescribeStream`.

Para especificar várias ações em uma única declaração, separe-as com vírgulas, conforme o seguinte:

```
"Action": ["kinesis:action1", "kinesis:action2"]
```

Você também pode especificar várias ações usando caracteres curinga. 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 asterisco (\*) conforme o seguinte:

```
"Action": "kinesis:*"
```

Para obter a lista completa de ações da API do Kinesis Data Streams, consulte o [Amazon Kinesis API Reference](#).

## Nomes de recurso da Amazon (ARNs) para o Kinesis Data Streams

Cada declaração de política do IAM se aplica aos recursos que você especifica usando os ARNs.



Use o seguinte formato de recursos do ARN para Kinesis data streams:

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

Por exemplo:

```
"Resource": arn:aws:kinesis:*:111122223333:stream/my-stream
```

## Exemplo de políticas para o Kinesis Data Streams

As políticas de exemplo a seguir demonstram como você pode controlar o acesso do usuário aos Kinesis data streams.

Example 1: Permitir que os usuários obtenham dados de um stream

Esta política permite que um usuário ou grupo execute as operações `DescribeStream`, `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:DescribeStream"
      ],
      "Resource": [
        "arn:aws:kinesis:us-east-1:111122223333:stream/stream1"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "kinesis:ListStreams"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

Example 2: Permitir que os usuários adicionem dados a qualquer stream na conta

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: Permitir qualquer ação do Kinesis Data Streams em um determinado stream

Esta política permite que um usuário ou grupo use qualquer operação do Kinesis Data Streams no stream 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: Permitir qualquer ação do Kinesis Data Streams em qualquer stream

Esta política permite que um usuário ou grupo use qualquer operação do Kinesis Data Streams em qualquer stream 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/*"
      ]
    }
  ]
}
```

## Validação de conformidade do Amazon Kinesis Data Streams

Audidores terceirizados avaliam a segurança e a conformidade do Amazon Kinesis Data Streams como parte de vários programas de conformidade da AWS. Isso inclui SOC, PCI, FedRAMP, HIPAA e outros.

Para obter uma lista de serviços da AWS no escopo de programas de conformidade específicos, consulte [Serviços da AWS no escopo por programa de conformidade](#). Para obter informações gerais, consulte [Programas de conformidade da AWS](#).

Você pode fazer download de relatórios de auditoria de terceiros usando o AWS Artifact. Para obter mais informações, consulte [Download de relatórios no AWS Artifact](#).

Sua responsabilidade de conformidade ao usar o Kinesis Data Streams é determinada pela confidencialidade dos seus dados, pelos objetivos de conformidade da sua empresa e pelos regulamentos e leis aplicáveis. Se o seu uso do Kinesis Data Streams estiver sujeito à conformidade com padrões como HIPAA, PCI ou FedRAMP, a AWS fornecerá recursos para ajudar:

- [Guias Quick Start de segurança e conformidade](#) – esses guias de implantação discutem as considerações de arquitetura e fornecem etapas para implantação de ambientes de linha de base focados em conformidade e segurança na AWS.
- [Whitepaper Arquitetura para segurança e conformidade com HIPAA](#) – esse whitepaper descreve como as empresas podem usar a AWS para criar aplicativos em conformidade com a HIPAA.
- [Recursos de conformidade da AWS](#) – esta coleção de manuais e guias pode ser aplicada ao seu setor e local.
- [AWS Config](#) – esse serviço da AWS avalia até que ponto suas configurações de recursos atendem adequadamente a práticas internas e a diretrizes e regulamentações do setor.
- [AWS Security Hub](#) – esse serviço da AWS fornece uma visão abrangente do estado da segurança na AWS que ajuda a verificar a conformidade com os padrões do setor de segurança e as práticas recomendadas.

## Resiliência no Amazon Kinesis Data Streams

A infraestrutura global da AWS é criada com base em regiões e zonas de disponibilidade da AWS. As regiões da AWS fornecem várias zonas de disponibilidade separadas e isoladas fisicamente, que são conectadas com baixa latência, altas taxas de transferência e redes altamente redundantes. Com as zonas de disponibilidade, você pode projetar e operar aplicativos e bancos de dados que executam 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 data center tradicionais.

Para obter mais informações sobre regiões e zonas de disponibilidade da AWS, consulte [Infraestrutura global da AWS](#).

Além da infraestrutura global da AWS, o Kinesis Data Streams oferece vários recursos para ajudar a oferecer suporte às suas necessidades de resiliência de dados e backup.

## Recuperação de desastres no Amazon Kinesis Data Streams

A falha pode ocorrer nos seguintes níveis quando você usa um Amazon Kinesis Data Streams application para processar dados de um stream:

- Um processador de registros pode falhar
- Um operador pode falhar ou a instância do aplicativo que instanciou o operador pode falhar
- Uma instância do EC2 que hospeda uma ou mais instâncias do aplicativo pode falhar

### Falha do processador de registros

O operador invoca os métodos de processador de registros usando tarefas [ExecutorService](#) do Java. 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 \(p. 187\)](#).

## Falha de operador ou aplicativo

Se um operador — ou uma instância do Amazon Kinesis Data Streams application — falhar, você deverá detectar e lidar com 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 operador ou o aplicativo falha, a falha não é detectada e há outras instâncias do aplicativo sendo executadas em outras instâncias do EC2, os operadores nessas 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 instâncias do EC2 aumenta de forma correspondente.

O cenário descrito aqui assume que, embora o operador ou aplicativo tenha falhado, a instância do EC2 que o hospeda ainda está em execução e, portanto, não é reiniciada por um grupo do Auto Scaling.

## Falha de instância do Amazon EC2

Recomendamos que você execute as instâncias do EC2 referentes ao seu aplicativo em um grupo do Auto Scaling. Dessa forma, se uma das instâncias do EC2 falhar, o grupo do Auto Scaling iniciará automaticamente uma nova instância para substituí-la. Você deve configurar as instâncias para iniciar seu Amazon Kinesis Data Streams application na inicialização.

# Segurança da infraestrutura no Kinesis Data Streams

Como serviço gerenciado, o Amazon Kinesis Data Streams é protegido pelos procedimentos de segurança da rede global da AWS que estão descritos no whitepaper [Amazon Web Services: visão geral dos processos de segurança](#).

Você usa chamadas de API publicadas pela AWS para acessar o Kinesis Data Streams por meio da rede. Os clientes devem oferecer suporte a Transport Layer Security (TLS) 1.0 ou posterior. Recomendamos TLS 1.2 ou posterior. Os clientes também devem ter suporte a pacotes de criptografia com sigilo de encaminhamento perfeito (PFS) como Ephemeral Diffie-Hellman (DHE) ou Ephemeral Elliptic Curve Diffie-Hellman (ECDHE). A maioria dos sistemas modernos como Java 7 e versões posteriores oferece suporte a esses modos.

Além disso, as solicitações devem ser assinadas usando um ID da chave de acesso e uma chave de acesso secreta associada a uma entidade principal do IAM. Ou você pode usar o [AWS Security Token Service](#) (AWS STS) para gerar credenciais de segurança temporárias para assinar solicitações.

## Práticas recomendadas de segurança do Kinesis Data Streams

O Amazon Kinesis Data Streams fornece vários recursos de segurança a serem considerados à medida que você desenvolve e implementa suas próprias políticas de segurança. As melhores práticas a seguir

são diretrizes gerais e não representam uma solução completa de segurança. Como essas melhores práticas podem não ser adequadas ou suficientes no seu ambiente, trate-as como considerações úteis em vez de requisitos.

## Implemente o privilégio de acesso mínimo

Ao conceder permissões, você decide quem recebe quais permissões a 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.

## Usar funções do IAM

Aplicativos de clientes e produtores devem ter credenciais válidas para acessar fluxos de dados do Kinesis. Você não deve armazenar credenciais da AWS diretamente em um aplicativo de cliente ou em um bucket do Amazon S3. Essas são credenciais de longo prazo que não são alternadas automaticamente e que poderão causar um impacto significativo nos negócios se forem comprometidas.

Em vez disso, você deve usar uma função do IAM para gerenciar credenciais temporárias para que seus aplicativos de clientes e produtores possam acessar 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 do usuário do IAM:

- [Funções do IAM](#)
- [Cenários comuns para funções: usuários, aplicativos e serviços](#)

## Implemente a criptografia do lado do servidor em recursos dependentes

Dados em repouso e dados em trânsito podem ser criptografados no Kinesis Data Streams. Para obter mais informações, consulte [Proteção de dados no Amazon Kinesis Data Streams](#) (p. 217).

## Use o CloudTrail para monitorar chamadas de API

O Kinesis Data Streams é integrado ao AWS CloudTrail, um serviço que fornece um registro das ações realizadas por um usuário, uma função ou um serviço da AWS no Kinesis Data Streams.

Com o uso das informações coletadas pelo CloudTrail, é possível determinar a solicitação que foi feita ao Kinesis Data Streams, o endereço IP a partir do qual foi feita a solicitação, quem fez a solicitação e quando ela foi feita, além de outros detalhes.

Para obter mais informações, consulte [the section called “Registro em log de chamadas de API do Amazon Kinesis Data Streams com o AWS CloudTrail”](#) (p. 199).

# Histórico do documento

A tabela a seguir descreve as alterações importantes feitas na documentação do Amazon Kinesis Data Streams.

Alteração	Descrição	Alterado em
Adição de informações sobre como controlar o acesso a endpoints de VPCE para o Kinesis Data Streams.	Para obter mais informações, consulte <a href="#">the section called “Controlar o acesso a endpoints de VPCE para o Kinesis Data Streams”</a> (p. 223).	23 de agosto de 2019
Nova documentação para consumidores que usam distribuição avançada.	Para obter mais informações, consulte <a href="#">the section called “Desenvolver consumidores personalizados com taxa de transferência dedicada (distribuição avançada)”</a> (p. 161).	2 de agosto de 2018
Resumo de limites de serviço atualizado.	Adição do <a href="#">Cotas e limites do Kinesis Data Streams</a> (p. 6).	6 de junho de 2018
Novo conteúdo de criptografia do lado do servidor	Adição do <a href="#">Proteção de dados no Amazon Kinesis Data Streams</a> (p. 217).	7 de julho de 2017
Novo conteúdo para aprimorar as métricas do CloudWatch.	Atualizado <a href="#">Monitorar o Amazon Kinesis Data Streams</a> (p. 188).	19 de abril de 2016
Novo conteúdo para aprimorar o agente do Kinesis.	Atualizado <a href="#">Gravação no Amazon Kinesis Data Streams por meio do agente do Kinesis</a> (p. 105).	11 de abril de 2016
Novo conteúdo para o uso de agentes do Kinesis.	Adição do <a href="#">Gravação no Amazon Kinesis Data Streams por meio do agente do Kinesis</a> (p. 105).	2 de outubro de 2015
Atualização do conteúdo da KPL para a versão 0.10.0.	Adição do <a href="#">Desenvolver produtores usando a Kinesis Producer Library da Amazon</a> (p. 89).	15 de julho de 2015
Atualização do tópico sobre métricas da KCL com métricas configuráveis.	Adição do <a href="#">Monitoramento da Kinesis Client Library com o Amazon CloudWatch</a> (p. 203).	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 do desenvolvedor da KPL.	Adição do <a href="#">Desenvolver produtores usando a Kinesis Producer Library da Amazon</a> (p. 89).	02 de junho de 2015

Alteração	Descrição	Alterado em
Novo tópico sobre métricas da KCL.	Adição do <a href="#">Monitoramento da Kinesis Client Library com o Amazon CloudWatch</a> (p. 203).	19 de maio de 2015
Compatibilidade com KCL .NET	Adição do <a href="#">Desenvolver um consumidor de Kinesis Client Library em .NET</a> (p. 140).	1 de maio de 2015
Compatibilidade com KCL Node.js	Adição do <a href="#">Desenvolver um consumidor de Kinesis Client Library em Node.js</a> (p. 137).	26 de março de 2015
Compatibilidade com KCL Ruby	Adição de links para a biblioteca KCL Ruby.	12 de janeiro de 2015
Nova API de PutRecords	Adição de informações sobre a nova API de PutRecords a <a href="#">the section called “Adicionar vários registros com PutRecords”</a> (p. 101).	15 de dezembro de 2014
Compatibilidade com atribuição de tags	Adição do <a href="#">Atribuir tags nos streams no Amazon Kinesis Data Streams</a> (p. 84).	11 de setembro de 2014
Nova métrica de CloudWatch	Adição da métrica <code>GetRecords.IteratorAgeMilliseconds</code> a <a href="#">Dimensões e métricas do Amazon Kinesis Data Streams</a> (p. 189).	3 de setembro de 2014
Novo capítulo de monitoramento	Adicionadas <a href="#">Monitorar o Amazon Kinesis Data Streams</a> (p. 188) e <a href="#">Monitoramento do serviço do Amazon Kinesis Data Streams com o Amazon CloudWatch</a> (p. 188).	30 de julho de 2014
Limite padrão de estilhaço	Atualização do <a href="#">Cotas e limites do Kinesis Data Streams</a> (p. 6): o limite padrão de estilhaço foi elevado de 5 para 10.	25 de fevereiro de 2014
Limite padrão de estilhaço	Atualização do <a href="#">Cotas e limites do Kinesis Data Streams</a> (p. 6): o limite padrão de estilhaço foi elevado de 2 para 5.	28 de janeiro de 2014
Atualizações de versão de API	Atualizações para a versão 2013-12-02 da API do Kinesis Data Streams.	12 de dezembro de 2013
Versão inicial	Versão inicial do Amazon Kinesis Developer Guide.	14 de novembro de 2013

# AWS Glossary

For the latest AWS terminology, see the [AWS Glossary](#) in the AWS General Reference.



Se fornecermos uma tradução da versão em inglês do guia, a versão em inglês prevalecerá caso haja qualquer conflito entre as versões. A tradução é fornecida com o uso de tradução por máquina.