



Entwicklerhandbuch

# Amazon Kinesis Data Streams



# Amazon Kinesis Data Streams: Entwicklerhandbuch

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

Die Handelsmarken und Handelsaufmachung von Amazon dürfen nicht in einer Weise in Verbindung mit nicht von Amazon stammenden Produkten oder Services verwendet werden, durch die Kunden irregeführt werden könnten oder Amazon in schlechtem Licht dargestellt oder diskreditiert werden könnte. Alle anderen Handelsmarken, die nicht Eigentum von Amazon sind, gehören den jeweiligen Besitzern, die möglicherweise zu Amazon gehören oder nicht, mit Amazon verbunden sind oder von Amazon gesponsert werden.

---

# Table of Contents

Was ist Amazon Kinesis Data Streams? .....	1
Was kann ich mit Kinesis Data Streams machen? .....	1
Vorteile der Verwendung von Kinesis Data Streams .....	2
Verwandte Dienstleistungen .....	3
Terminologie und Konzepte .....	4
High-Level-Architektur .....	4
Terminologie der Kinesis Data Streams .....	5
Kinesis Data Stream .....	5
Datensatz .....	5
Kapazitätsmodus .....	5
Aufbewahrungszeitraum .....	5
Produzent .....	6
Konsument .....	6
Anwendung von Amazon Kinesis Data Streams .....	6
Shard .....	6
Partitionsschlüssel .....	7
Sequenznummer .....	7
Kinesis Client Library .....	7
Anwendungsname .....	8
Serverseitige Verschlüsselung .....	8
Kontingente und Einschränkungen .....	9
Limits für API .....	11
KDS-Limits für die API der Steuerebene .....	11
Limits für die KDS-API der Datenebene .....	16
Erhöhung der Kontingente .....	19
Einrichten .....	20
Registrieren bei AWS .....	20
Herunterladen von Bibliotheken und Tools .....	20
Konfigurieren der Entwicklungsumgebung .....	21
Erste Schritte .....	22
Installieren und Konfigurieren der AWS CLI .....	22
Installieren AWS CLI .....	22
Konfigurieren von AWS CLI .....	24
Ausführen von einfachen Kinesis-Daten-Stream-Operationen mit der AWS CLI .....	24

Schritt 1: Stream erstellen .....	25
Schritt 2: Senden eines Datensatzes .....	26
Schritt 3: Abrufen des Datensatzes .....	27
Schritt 4: Bereinigen .....	30
Beispiele .....	32
Tutorial: Verarbeiten von Wertpapier-Echtzeitdaten mit KPL und KCL 2.x .....	32
Voraussetzungen .....	33
Schritt 1: Erstellen eines Daten-Streams .....	34
Schritt 2: Erstellen von IAM-Richtlinie und -Benutzer .....	35
Schritt 3: Herunterladen und Erstellen des Codes .....	41
Schritt 4: Produzent implementieren .....	41
Schritt 5: Konsument implementieren .....	46
Schritt 6: (optional) Konsument erweitern .....	51
Schritt 7: Abschluss .....	52
Tutorial: Verarbeiten von Wertpapier-Echtzeitdaten mit KPL und KCL 1.x .....	54
Voraussetzungen .....	55
Schritt 1: Erstellen eines Daten-Streams .....	56
Schritt 2: Erstellen von IAM-Richtlinie und -Benutzer .....	58
Schritt 3: Implementierungscode herunterladen und erstellen .....	63
Schritt 4: Produzent implementieren .....	64
Schritt 5: Konsument implementieren .....	69
Schritt 6: (optional) Konsument erweitern .....	73
Schritt 7: Abschluss .....	74
Tutorial: Analysieren von Wertpapierdaten in Echtzeit mit Managed Service für Apache Flink ....	76
Voraussetzungen .....	77
Schritt 1: Einrichten eines Kontos .....	78
Schritt 2: Einrichten von AWS CLI .....	82
Schritt 3: Erstellen einer Anwendung .....	83
Anleitung: Verwenden von AWS Lambda mit Amazon Kinesis Data Streams .....	101
AWS-Streaming-Datenlösung .....	102
Erstellen und Verwalten von Streams .....	103
Auswahl des Datenstrom-Kapazitätsmodus .....	103
Was ist ein Datenstream-Kapazitätsmodus? .....	104
On-Demand-Modus .....	104
Modus bereitgestellter Kapazität .....	106
Zwischen Kapazitätsmodi wechseln .....	107

Erstellen eines Streams über die AWS-Management-Konsole .....	108
Erstellen eines Streams über die APIs .....	109
Kinesis-Data-Streams-Client erstellen .....	109
Erstellen des Streams .....	110
Aktualisieren eines Streams .....	112
.....	112
Aktualisieren eines Streams mit der API .....	113
Aktualisieren eines Streams mit der AWS CLI .....	113
Auflisten von Streams .....	114
Auflisten von Shards .....	115
ListShards API — Empfohlen .....	115
DescribeStream API — Veraltet .....	118
Löschen eines Streams .....	119
Resharding eines Streams .....	120
Strategien für das Resharding .....	121
Teilen eines Shards .....	122
Zusammenführen von zwei Shards .....	124
Nach dem Resharding .....	125
Ändern des Zeitraums der Datenaufbewahrung .....	128
Markieren Ihrer Streams .....	130
Grundlagen zu Tags .....	130
Kosten mithilfe von Tags verfolgen .....	131
Tag-Einschränkungen .....	131
Taggen von Streams mithilfe der Kinesis-Data-Streams-Konsole .....	132
Tagging von Streams mithilfe der AWS CLI .....	133
Taggen von Streams mithilfe der Kinesis-Data-Streams-API .....	133
Schreiben in Datenströme .....	134
Verwenden der KPL .....	135
Die Rolle der KPL .....	136
Vorteile der Verwendung der KPL .....	137
Wann die KPL nicht verwendet werden sollte .....	138
Installieren der KPL .....	138
Umstieg auf Amazon Trust Services (ATS)-Zertifikate für die Kinesis Producer Library .....	139
KPL-unterstützte Plattformen .....	139
Die wichtigsten Konzepte von KPL .....	140
Integrieren der KPL mit Producer-Code .....	143

Schreiben in den Kinesis Data Stream .....	145
Konfigurieren der KPL .....	147
Datenproduzent – Disaggregation .....	148
Verwenden der KPL mit Firehose .....	152
Verwenden der KPL mit der AWS Glue Schema Registry .....	152
KPL-Proxy-Konfiguration .....	153
Verwenden der API .....	154
Hinzufügen von Daten zu einem Stream .....	154
Interaktion mit Daten mithilfe des AWS Glue Schema-Registry .....	161
Verwenden des Agents .....	162
Voraussetzungen .....	162
Herunterladen und Installieren des Agenten .....	163
Konfigurieren und Starten des Agenten .....	164
Konfigurationseinstellungen für den Agenten .....	165
Überwachen mehrerer Dateiverzeichnisse und Schreiben in mehrere Streams .....	169
Verwenden des Agenten zur Datenvorverarbeitung .....	169
CLI-Befehle des Agenten .....	174
Häufig gestellte Fragen .....	175
Nutzung anderer AWS-Services .....	176
AWS Amplify .....	177
Amazon Aurora .....	177
Amazon CloudFront .....	177
Amazon CloudWatch Logs .....	178
Amazon Connect .....	178
AWS Database Migration Service .....	178
Amazon DynamoDB .....	179
Amazon EventBridge .....	179
AWS IoT Core .....	179
Amazon Relational Database Service .....	179
Amazon Pinpoint .....	180
Amazon Quantum Ledger Database .....	180
Integrationen von Drittanbietern verwenden .....	180
Apache Flink .....	181
Fluentd .....	181
Debezium .....	181
Oracle GoldenGate .....	181

Kafka Connect .....	181
Adobe Experience .....	181
Striim .....	182
Fehlerbehebung .....	182
Die Produzentenanwendung schreibt Daten langsamer als erwartet .....	182
Fehler aufgrund fehlender KMS-Masterschlüsselberechtigung .....	184
Häufig auftretende Probleme, Fragen und Ideen zur Problemlösung für Produzenten .....	184
Fortgeschrittene Themen .....	185
Begrenzungen für Wiederholungen und Quoten .....	185
Überlegungen zur Verwendung der KPL-Aggregation .....	186
Lesen aus Datenströmen .....	188
Verwendung von Data Viewer in der Kinesis-Konsole .....	190
Abfragen Ihrer Datenströme in der Kinesis-Konsole .....	191
Verwenden von AWS Lambda .....	191
Nutzung von Managed Service für Apache Flink .....	192
Verwenden von Firehose .....	192
Verwenden der Kinesis Client Library .....	192
Was ist die Kinesis Client Library? .....	193
Verfügbare KCL-Versionen .....	194
KCL-Konzepte .....	195
Mithilfe einer Leasetabelle können Sie die von der KCL-Konsumenten-anwendung verarbeiteten Shards verfolgen .....	197
Verarbeitung mehrerer Datenströme mit derselben Konsumenten-anwendung KCL 2.x für Java .....	210
Verwenden der Kinesis Client Library mit dem AWS Glue Schema Registry .....	214
Entwickeln benutzerdefinierter Verbraucher mit gemeinsam genutztem Durchsatz .....	214
Entwickeln benutzerdefinierter Verbraucher mit gemeinsam genutztem Durchsatz unter Verwendung von KCL .....	215
Entwickeln benutzerdefinierter Verbraucher mit gemeinsam genutztem Durchsatz unter Verwendung von AWS SDK for Java .....	254
Entwickeln benutzerdefinierter Verbraucher mit dediziertem Durchsatz (Erweitertes Rundsenden) .....	261
Entwicklung von Verbrauchern für erweitertes Rundsenden mit KCL 2.x .....	263
Entwicklung Verbrauchern für erweitertes Rundsenden mit der API für Kinesis Data Streams .....	269

Verwalten von Verbrauchern für erweitertes Rundsenden mit dem AWS Management Console .....	272
Migrieren von Verbrauchern von KCL 1.x zu KCL 2.x .....	273
Migrieren des Datensatzprozessors .....	274
Migrieren der Datensatzprozessor-Factory .....	279
Migrieren der Auftragnehmer .....	281
Konfiguration des Amazon-Kinesis-Clients .....	282
Entfernen der Leerlaufzeit .....	287
Entfernen von Client-Konfigurationen .....	288
Nutzung anderer AWS-Services .....	289
Nutzung von Amazon EMR .....	289
Amazon EventBridge Pipes verwenden .....	289
Verwenden von AWS Glue .....	290
Verwenden von Amazon Redshift .....	290
Integrationen von Drittanbietern verwenden .....	290
Apache Flink .....	291
Adobe Experience Platform .....	291
Apache Druid .....	291
Apache Spark .....	291
Databricks .....	292
Kafka Confluent Plattform .....	292
Kinesumer .....	292
Talend .....	292
Problembehandlung bei Verbrauchern von Kinesis Data Streams .....	292
Einige Datensätze in Kinesis Data Streams werden bei der Nutzung der Kinesis Client Library übersprungen .....	293
Datensätze, die zum selben Shard gehören, werden gleichzeitig von verschiedenen Datensatzprozessoren verarbeitet .....	293
Die Konsumentenanzahl liest Daten langsamer aus als erwartet .....	294
GetRecords Gibt leeres Datensatz-Array zurück, auch wenn sich Daten im Stream befinden .....	295
Der Shard-Iterator verliert unerwartet seine Gültigkeit .....	296
Die Verarbeitung der Konsumentendatensätze hängt hinterher .....	296
Fehler aufgrund fehlender KMS-Masterschlüsselberechtigung .....	297
Häufig gestellte Probleme, Fragen und Ideen zur Problemlösung für Konsumenten .....	297
Fortgeschrittene Themen .....	298



Verarbeitung mit geringer Latenz .....	298
Verwenden von AWS Lambda mit der Kinesis-Producer-Bibliothek .....	299
Resharding, Skalierung und Parallelverarbeitung .....	300
Umgang mit doppelten Datensätzen .....	301
Umgang mit Startup, Herunterfahren und Drosselung .....	304
Überwachung von Datenströmen .....	307
Überwachung des Services mit CloudWatch .....	307
Amazon Kinesis Data Streams – Metriken und Dimensionen .....	308
Zugreifen auf Amazon-CloudWatch-Metriken für Kinesis Data Streams .....	325
Überwachung des Agenten mit CloudWatch .....	326
Überwachung mit CloudWatch .....	326
Protokollieren von Amazon-Kinesis-Data-Streams-API-Aufrufen mithilfe von AWS CloudTrail ..	327
Informationen zu Kinesis Data Streams in CloudTrail .....	327
Beispiel: Einträge in der Protokolldatei für Kinesis Data Streams .....	329
Überwachung der KCL mit CloudWatch .....	333
Metriken und Namespaces .....	333
Metrikstufen und Dimensionen .....	333
Metrik-Konfiguration .....	335
Liste der Metriken .....	335
Überwachung der KPL mit CloudWatch .....	347
Metriken, Dimensionen und Namespaces .....	348
Metrikstufe und Granularität .....	348
Lokaler Zugriff und Amazon-CloudWatch-Upload .....	349
Liste der Metriken .....	350
Sicherheit .....	355
Datenschutz .....	356
Was bedeutet eine serverseitige Verschlüsselung in Kinesis Data Streams? .....	356
Kosten, Regionen und Leistungsanforderungen .....	358
Wie beginne ich mit der serverseitigen Verschlüsselung? .....	360
Erstellen und Verwenden benutzergenerierter KMS-Masterschlüssel .....	361
Berechtigungen für die Nutzung benutzergenerierter KMS-Masterschlüssel .....	361
Überprüfung und Fehlerbehebung bei KMS-Schlüssel Berechtigungen .....	363
Verwendung von -Schnittstellen-VPC-Endpunkten .....	363
Zugriffssteuerung .....	367
Richtliniensyntax .....	369
Aktionen für Kinesis Data Streams .....	369

Amazon-Ressourcennamen (ARNs) für Kinesis Data Streams .....	370
Beispielrichtlinien für Kinesis Data Streams .....	370
Freigabe Ihres Datenstroms für ein anderes Konto .....	373
Konfigurieren einer - AWS Lambda Funktion zum Lesen aus Kinesis Data Streams in einem anderen Konto .....	379
Freigabe des Zugriffs mithilfe von ressourcenbasierten Richtlinien .....	379
Compliance-Validierung .....	381
Ausfallsicherheit .....	382
Notfallwiederherstellung .....	383
Sicherheit der Infrastruktur .....	384
Bewährte Methoden für die Sicherheit .....	384
Implementieren des Zugriffs mit geringsten Berechtigungen .....	384
Verwenden von IAM-Rollen .....	385
Implementieren einer serverseitigen Verschlüsselung in abhängigen Ressourcen .....	385
Verwenden von CloudTrail zur Überwachung von API-Aufrufen .....	385
Dokumentverlauf .....	387
AWS-Glossar .....	390
.....	cccxc

# Was ist Amazon Kinesis Data Streams?

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

Kinesis Data Streams ist zusammen mit [Firehose](#) , [Kinesis Video Streams](#) und [Managed Service für Apache Flink](#) Teil der Kinesis-Streaming-Datenplattform.

Weitere Informationen zu AWS Big-Data-Lösungen finden Sie unter [Big Data auf AWS](#). Weitere Informationen zu AWS -Streaming-Datenlösungen finden Sie unter [Was sind Streaming-Daten?](#).

## Themen

- [Was kann ich mit Kinesis Data Streams machen?](#)
- [Vorteile der Verwendung von Kinesis Data Streams](#)
- [Verwandte Dienstleistungen](#)

# Was kann ich mit Kinesis Data Streams machen?

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

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

## Beschleunigte Protokoll- und Datenfeedaufnahme und -verarbeitung

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

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

## Echtzeitmetriken und -berichte

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

## Echtzeitdatenanalysen

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

## Komplexe Stream-Verarbeitung

Sie können aus Anwendungen und Datenströme von Kinesis Data Streams Directed Acyclic Graphs (DAGs) erstellen. Dazu gehört in der Regel das Übertragen von Daten aus mehreren Anwendungen von Kinesis Data Streams in einen anderen Stream für die nachgeordnete Verarbeitung durch eine andere Anwendung von Kinesis Data Streams.

# Vorteile der Verwendung von Kinesis Data Streams

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

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

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

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

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

## Verwandte Dienstleistungen

Weitere Informationen zur Verwendung von Amazon-EMR-Clustern zum direkten Lesen und Verarbeiten von Kinesis-Datenströmen finden Sie unter [Kinesis Connector](#).

# Terminologie und Konzepte von Amazon Kinesis Data Streams

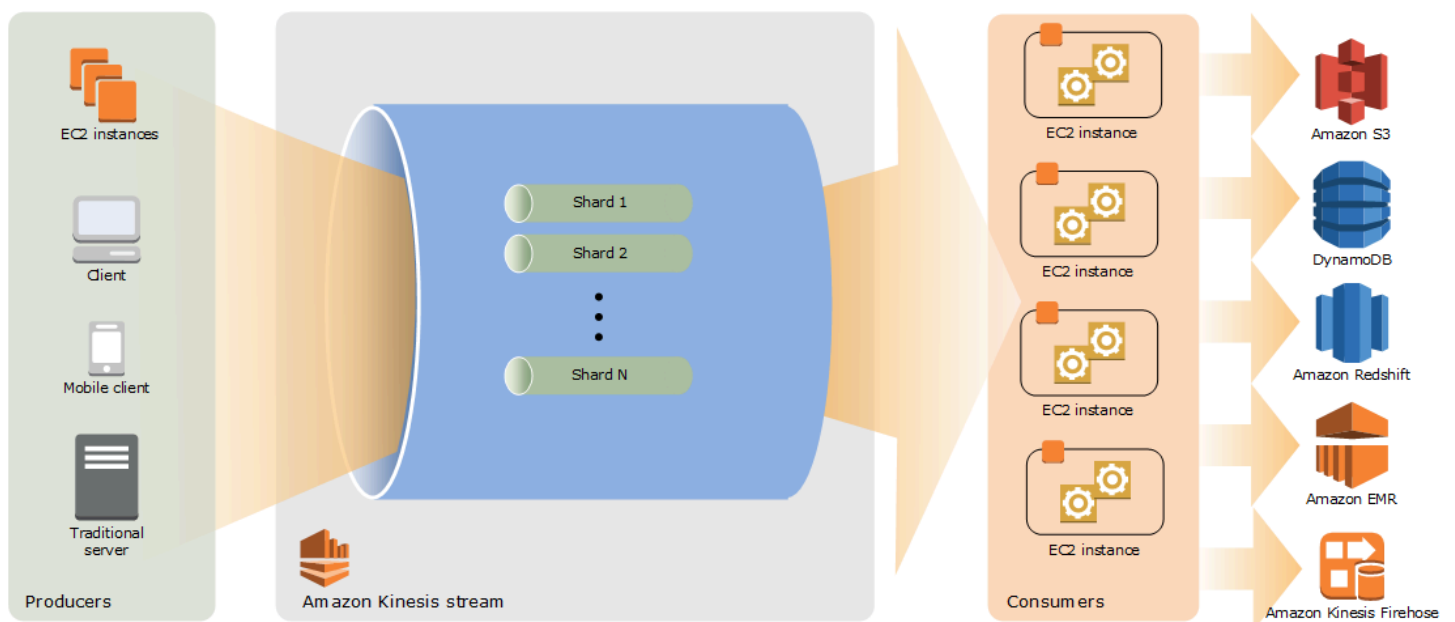
Wenn Sie mit Amazon Kinesis Data Streams arbeiten, werden Sie vom Verständnis der zugrunde liegenden Architektur und Terminologie profitieren.

Themen

- [High-Level-Architektur von Kinesis Data Streams](#)
- [Terminologie der Kinesis Data Streams](#)

## High-Level-Architektur von Kinesis Data Streams

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



# Terminologie der Kinesis Data Streams

## Kinesis Data Stream

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

## Datensatz

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

## Kapazitätsmodus

Ein Datenstrom-Kapazitätsmodus bestimmt, wie Kapazität verwaltet wird und wie Ihnen die Nutzung Ihres Datenstroms in Rechnung gestellt wird. Derzeit können Sie in Kinesis Data Streams zwischen einem On-Demand-Modus und einem bereitgestellten Modus für Ihre Datenströme wählen. Weitere Informationen finden Sie unter [Auswahl des Datenstrom-Kapazitätsmodus](#).

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

Mit einem bereitgestellten Modus müssen Sie die Anzahl der Shards für den Datenstrom angeben. Die Gesamtkapazität eines Datenstroms ist die Summe der Kapazitäten der einzelnen Shards. Sie können die Anzahl der Shards in einem Datenstrom nach Bedarf erhöhen oder verringern, und die Anzahl der Shards wird Ihnen nach einem Stundensatz in Rechnung gestellt. Weitere Informationen finden Sie unter [Modus bereitgestellter Kapazität](#).

## Aufbewahrungszeitraum

Der Aufbewahrungszeitraum gibt den Zeitraum an, in dem auf Datensätze zugegriffen werden kann, nachdem sie dem Stream hinzugefügt wurden. Die Standard-Aufbewahrungsdauer eines Streams beträgt nach Erstellung 24 Stunden. Sie können den Aufbewahrungszeitraum mit der Operation auf bis zu 8 760 Stunden (365 Tage) erhöhen [IncreaseStreamRetentionPeriod](#) und den

Aufbewahrungszeitraum mit der [DecreaseStreamRetentionPeriod](#) Operation auf mindestens 24 Stunden reduzieren. Für Streams mit einem Aufbewahrungszeitraum von mehr als 24 Stunden fallen zusätzliche Gebühren an. Weitere Informationen finden Sie unter [Preise für Amazon Kinesis Daten-Streams](#).

## Produzent

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

## Konsument

Verbraucher erhalten Datensätze von Amazon Kinesis Data Streams und verarbeiten sie. Diese Konsumenten werden als [Anwendung von Amazon Kinesis Data Streams](#) bezeichnet.

## Anwendung von Amazon Kinesis Data Streams

Eine Anwendung von Amazon Kinesis Data Streams ist ein Verbraucher eines Streams, der in der Regel auf einer Flotte von EC2-Instances ausgeführt wird.

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

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

## Shard

Ein Shard ist eine eindeutig identifizierbare Sequenz von Datensätzen in einem Stream. Ein Stream besteht aus einem oder mehreren Shards. Jeder Shard stellt eine feste Einheit der Kapazität bereit. Jeder Shard kann bis zu 5 Transaktionen pro Sekunde für Lesevorgänge unterstützen, bis zu einer maximalen Gesamtdatenleserate von 2 MB pro Sekunde und bis zu 1 000 Datensätze pro Sekunde für Schreibvorgänge, bis zu einer maximalen Gesamtdatenschreibrate von 1 MB pro Sekunde (einschließlich Partitionsschlüssel). Die Datenkapazität Ihres Streams bezieht sich auf die Anzahl



der Shards, die Sie für den Stream festlegen. Die Gesamtkapazität des Streams ist die Summe der Kapazitäten der einzelnen Shards.

Wenn sich Ihre Datenrate erhöht, können Sie die Anzahl der zu Ihrem Stream zugewiesenen Shards erhöhen oder verringern. Weitere Informationen finden Sie unter [Resharding eines Streams](#).

## Partitionsschlüssel

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

## Sequenznummer

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

### Note

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

## Kinesis Client Library

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

eine Amazon-DynamoDB-Tabelle zum Speichern von Kontrolldaten. Sie generiert eine Anwendung pro Tabelle, die die Daten verarbeitet.

Es gibt zwei Hauptversionen der Kinesis Client Library. Welche Sie verwenden, hängt vom Typ des zu erstellenden Konsumenten ab. Weitere Informationen finden Sie unter [Lesen von Daten aus Amazon Kinesis Data Streams](#).

## Anwendungsname

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

## Serverseitige Verschlüsselung

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

### Note

Zum Auslesen aus einem verschlüsselten Stream oder zum Schreiben in einen solchen benötigen Konsumenten Anwendungen eine Berechtigung für den Zugriff auf den Masterschlüssel. Informationen zum Gewähren von Berechtigungen für Produzenten- und Konsumenten Anwendungen finden Sie unter [the section called “Berechtigungen für die Nutzung benutzergenerierter KMS-Masterschlüssel”](#).

### Note

Für die serverseitige Verschlüsselung fallen AWS Key Management Service (AWS KMS)-Kosten an. Weitere Informationen finden Sie unter [AWS Key Management Service – Preise](#).

# Kontingente und Einschränkungen

Amazon Kinesis Data Streams hat die folgenden Stream- und Shard-Kontingente und -Limits.

Kontingent	On-Demand-Modus	Modus bereitgestellter Kapazität
Anzahl der Datenströme	Es gibt kein Kontingent für die Anzahl der Streams in Ihrem AWS-Konto. Standardmäßig können Sie im On-Demand-Kapazitätsmodus bis zu 50 Datenströme erstellen. Wenn Sie eine Erhöhung dieses Kontingents benötigen, erstellen Sie bitte ein <a href="#">Support-Ticket</a> .	Es gibt kein maximales Kontingent für die Anzahl der Streams im Bereitstellungsmodus innerhalb eines Kontos.
Anzahl der Shards	Es gibt keine Obergrenze. Die Anzahl der Shards hängt von der Menge der erfassten Daten und dem erforderlichen Durchsatz ab. Kinesis Data Streams skaliert die Anzahl der Shards automatisch als Reaktion auf Änderungen des Datenvolumens und des Datenverkehrs.	Es gibt keine Obergrenze. Das Standardkontingent für Shards beträgt 500 Shards pro AWS-Konto für die folgenden AWS-Regionen: USA Ost (Nord-Virginia), USA West (Oregon) und Europa (Irland). Für alle anderen Regionen beträgt das Standardkontingent 200 Shards pro AWS-Konto. Um eine Erhöhung der Kontingente für Shards pro Datenstrom anzufordern, siehe <a href="#">Anfordern einer Kontingenterhöhung</a> .
Datenstrom-Durchsatz	Standardmäßig haben neue Datenströme, die mit dem On-Demand-Kapazitätsmodus erstellt wurden, einen	Es gibt keine Obergrenze. Der maximale Durchsatz hängt von der Anzahl der für den Stream bereitges

Kontingent	On-Demand-Modus	Modus bereitgestellter Kapazität
	<p>Schreibdurchsatz von 4 Mbit/s und einen Lesedurchsatz von 8 MB/s. Mit steigendem Datenverkehr skalieren Datenströme mit dem On-Demand-Kapazitätsmodus auf bis zu 200 Mbit/s Schreib- und 400 Mbit/s Lesedurchsatz. Wenn Sie eine Erhöhung auf 2 Gbit/s Schreib- und 4 Gbit/s Lesekapazität benötigen, reichen Sie ein <a href="#">Support-Ticket</a> ein</p>	<p>teiltten Shards ab. Jeder Shard kann einen Schreibdurchsatz von bis zu 1 Mbit/s oder 1 000 Datensätze/Sekunde oder einen Lesedurchsatz von bis zu 2 Mbit/s oder 2 000 Datensätze/Sekunde unterstützen. Wenn Sie mehr Aufnahmekapazität benötigen, können Sie die Anzahl der Shards im Stream problemlos mit der AWS Management Console oder der <a href="#">UpdateShardCount</a>-API erhöhen.</p>
Daten-Nutzlastgröße	Die maximale Größe der Daten-Nutzlast eines Datensatzes vor der base64-encoding beträgt bis zu 1 MB.	
GetRecords -Transaktionsgröße	<p><a href="#">GetRecords</a> kann bis zu 10 MB Daten pro Aufruf von einem einzigen Shard und bis zu 10.000 Datensätze pro Aufruf abrufen. Jeder Aufruf von <code>GetRecords</code> gilt als eine Lese-Transaktion. Ein Shard kann bis zu fünf Lese-Transaktionen pro Sekunde unterstützen. Jede Lese-Transaktion kann bis zu 10.000 Datensätze mit einem Kontingent von 10 MB pro Transaktion liefern.</p>	
Datenleserate pro Shard	<p>Ein Shard kann eine Gesamtanlagevermögen von bis zu 2 MB pro Sekunde über <a href="#">GetRecords</a> unterstützen. Wenn ein Aufruf von <code>GetRecords</code> 10 MB zurückgibt, lösen nachfolgende Aufrufe innerhalb der nächsten 5 Sekunden eine Ausnahme aus.</p>	
Anzahl der registrierten Verbraucher pro Datenstrom	<p>Sie können bis zu 20 registrierte Verbraucher (Enhanced Fan-Out Limit) für jeden Datenstrom erstellen.</p>	

Kontingent	On-Demand-Modus	Modus bereitgestellter Kapazität
Zwischen Bereitstellungs- und On-Demand-Modus wechseln	Für jeden Datenstrom in Ihrem AWS-Konto können Sie innerhalb von 24 Stunden zweimal zwischen den Modi „On-Demand-Kapazität“ und „Bereitgestellte Kapazität“ wechseln.	

## Limits für API

Wie die meisten AWS-APIs sind Kinesis Data Streams API-Operationen durchsatzbegrenzt. Die folgenden Limits gelten pro AWS-Konto pro Region. Weitere Informationen zu Kinesis Data Streams-APIs finden Sie unter [Amazon-Kinesis-API-Referenz](#).

### KDS-Limits für die API der Steuerebene

Im folgenden Abschnitt werden die Limits für KDS-APIs der Steuerebene beschrieben. Mit KDS-APIs der Steuerungsebene können Sie Datenströme erstellen und verwalten. Diese Limits gelten pro AWS-Konto pro Region.

#### Limits für die API der Steuerebene

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

API	API-Aufruf-Limit	Pro Konto/Stream	Beschreibung
			<ul style="list-style-type: none"> <li>• Es befinden sich zu einem beliebigen Zeitpunkt mehr als fünf Streams im CREATING-Zustand.</li> <li>• Sie erstellen mehr Shards als für Ihr Konto zulässig sind.</li> </ul>
DecreaseStreamRetentionPeriod	5 TPS	Pro Stream	Der Mindestwert für den Aufbewahrungszeitraum eines Datenstroms beträgt 24 Stunden.
DeleteResourcePolicy	5 TPS	Pro Konto	Wenn Sie eine Erhöhung dieses Limits benötigen, erstellen Sie bitte ein <a href="#">Support-Ticket</a> .
DeleteStream	5 TPS	Pro Konto	
DeregisterStreamConsumer	5 TPS	Pro Stream	
DescribeLimits	1 TPS	Pro Konto	
DescribeStream	10 TPS	Pro Konto	
DescribeStreamConsumer	20 TPS	Pro Stream	

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

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



API	API-Aufruf-Limit	Pro Konto/Stream	Beschreibung
StartStreamEncryption		Pro Stream	Sie können einen neuen AWS KMS-Schlüssel für die serverseitige Verschlüsselung in einem umlaufenden 24-Stunden-Zeitraum 25 Mal erfolgreich anwenden.
StopStreamEncryption		Pro Stream	Sie können die serverseitige Verschlüsselung in einem fortlaufenden 24-Stunden-Zeitraum 25 Mal erfolgreich deaktivieren.
UpdateShardCount		Pro Stream	Gilt nur für bereitgestellte. Das Standardlimit für die Anzahl der Shards beträgt 10 000. Es gibt zusätzliche Beschränkungen für diese API. Weitere Informationen finden Sie unter <a href="#">UpdateShardCount</a> .

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

## Limits für die KDS-API der Datenebene

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

### Grenzwerte für die API der Datenebene

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

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

API	API-Aufruf-Limit	Nutzlast-Limit	Weitere Details
PutRecords		<p>Jede PutRecords-Anfrage kann bis zu 500 Datensätze unterstützen. Die maximale Größe jedes Datensatzes in der Anforderung beträgt 1 MB bis zu einem Limit von 5 MB für die gesamte Anforderung einschließlich Partitionsschlüssel. In. Jeder Shard unterstützt Schreibvorgänge von bis zu 1.000 Datensätzen pro Sekunde bis zu einem maximalen Datenschreibvolumen von 1 MB pro Sekunde.</p>	
SubscribeToShard	<p>Sie können pro Sekunde und registriertem Verbraucher pro Shard einen einzelnen Aufruf von SubscribeToShard ausführen.</p>		<p>Wenn Sie SubscribeToShard innerhalb von 5 Sekunden nach einem erfolgreichen Aufruf erneut mit demselben Verbraucher-ARN und derselben Shard-ID aufrufen, wird eine ResourceNotFoundException ausgelöst.</p>

## Erhöhung der Kontingente

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

Um eine Erhöhung des Servicekontingents anzufordern, befolgen Sie das unter [Kontingenterhöhung beantragen](#) beschriebenen Verfahren.

# Einrichten von Amazon Kinesis Data Streams

Bevor Sie Amazon Kinesis Data Streams zum ersten Mal verwenden, führen Sie die folgenden Aufgaben aus.

## Aufgaben

- [Registrieren bei AWS](#)
- [Herunterladen von Bibliotheken und Tools](#)
- [Konfigurieren der Entwicklungsumgebung](#)

## Registrieren bei AWS

Bei der Registrierung für Amazon Web Services (AWS) wird Ihr AWS-Konto automatisch für alle Dienste in AWS registriert, einschließlich Kinesis Data Streams. Berechnet werden Ihnen aber nur die Services, die Sie nutzen.

Wenn Sie bereits ein AWS-Konto haben, gehen Sie direkt zur nächsten Aufgabe. Wenn Sie kein AWS-Konto haben, führen Sie die folgenden Schritte zum Erstellen eines Kontos aus.

Für ein AWS-Konto registrieren Sie sich wie folgt:

1. Öffnen Sie <https://portal.aws.amazon.com/billing/signup>.
2. Folgen Sie den Online-Anweisungen.

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

Wenn Sie sich für ein AWS-Konto anmelden, wird ein Root-Benutzer des AWS-Kontos erstellt. Der Stammbenutzer hat Zugriff auf alle AWS-Services und Ressourcen des Kontos. Als bewährte Methode zur Gewährleistung der Sicherheit sollten Sie den [administrativen Zugriff einem administrativen Benutzer zuweisen](#) und nur den Root-Benutzer verwenden, um [Aufgaben auszuführen, die einen Root-Benutzerzugriff erfordern](#).

## Herunterladen von Bibliotheken und Tools

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

- Die [Amazon-Kinesis-API-Referenz](#) ist der grundlegende Satz von Vorgängen, die Kinesis Data Streams unterstützt. Weitere Informationen zum Durchführen grundlegender Operationen mit Java-Code finden Sie in den folgenden Ressourcen:
  - [Entwicklung von Produzenten mithilfe der API für Amazon Kinesis Data Streams mit der AWS SDK for Java](#)
  - [Entwickeln benutzerdefinierter Verbraucher mit gemeinsam genutztem Durchsatz unter Verwendung von AWS SDK for Java](#)
  - [Erstellen und Verwalten von Streams](#)
- Die AWS SDKs für [Go](#), [Java](#), [JavaScript](#), [.NET](#), [Node.js](#), [PHP](#), [Python](#) und [Ruby](#) umfassen Unterstützung für Kinesis Data Streams und passende Beispiele. Wenn Ihre Version von AWS SDK for Java keine Beispiele für Kinesis Data Streams enthält, können Sie diese von [GitHub](#) herunterladen.
- Die Kinesis Client Library (KCL) stellt ein einfach zu verwendendes Programmiermodell für die Datenverarbeitung zur Verfügung. Die KCL unterstützt Sie beim schnellen Start mit Kinesis Data Streams in Java, Node.js, .NET, Python und Ruby. Weitere Informationen finden Sie unter [Lesen von Daten aus Streams](#).
- Das [AWS Command Line Interface](#) unterstützt Kinesis Data Streams. Die AWS CLI ermöglicht die Steuerung mehrerer AWS-Services über die Befehlszeile und ihre Automatisierung mithilfe von Skripts.

## Konfigurieren der Entwicklungsumgebung

Damit Sie die KCL verwenden können, müssen Sie sicherstellen, dass Ihre Java-Entwicklungsumgebung die folgenden Anforderungen erfüllt:

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

Beachten Sie, dass das [AWS SDK for Java](#) Apache Commons und Jackson im Drittanbieter-Ordner enthält. Allerdings arbeitet das SDK for Java mit Java 1.6, während die Kinesis Client Library Java 1.7 erfordert.

# Erste Schritte mit Amazon Kinesis Data Streams

Die Informationen in diesem Abschnitt helfen Ihnen beim Einstieg in die Verwendung von Amazon Kinesis Data Streams. Wenn Sie Kinesis Data Streams zum ersten Mal verwenden, sollten Sie sich zunächst mit den Konzepten und der Terminologie in [Terminologie und Konzepte von Amazon Kinesis Data Streams](#) vertraut machen.

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

Themen

- [Installieren und Konfigurieren der AWS CLI](#)
- [Ausführen von einfachen Kinesis-Daten-Stream-Operationen mit der AWS CLI](#)

Für CLI-Zugriff benötigen Sie eine Zugriffsschlüssel-ID und einen geheimen Zugriffsschlüssel. Verwenden Sie möglichst temporäre Anmeldeinformationen anstelle langfristiger Zugriffsschlüssel. Temporäre Anmeldeinformationen bestehen aus einer Zugriffsschlüssel-ID, einem geheimen Zugriffsschlüssel und einem Sicherheits-Token, das angibt, wann die Anmeldeinformationen ablaufen. Weitere Informationen finden Sie unter [Verwenden von temporären Anmeldeinformationen mit AWS-Ressourcen](#) im IAM-Benutzerhandbuch.

Detaillierte Schritt-für-Schritt-Anweisungen zum Einrichten von IAM und Sicherheitsschlüsseln finden Sie unter [Erstellen eines IAM-Benutzers](#).

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

## Installieren und Konfigurieren der AWS CLI

### Installieren AWS CLI

Ausführliche Schritte zum Installieren der AWS CLI für die Betriebssysteme Windows, Linux, OS X und Unix finden Sie unter [Installieren der AWS-CLI](#).



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

```
aws help
```

Verwenden Sie den Service Kinesis Data Streams, sodass Sie mit dem folgenden Befehl die AWS CLI-Unterbefehle überprüfen können, die zu Kinesis Data Streams gehören:

```
aws kinesis help
```

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

#### AVAILABLE COMMANDS

- o add-tags-to-stream
- o create-stream
- o delete-stream
- o describe-stream
- o get-records
- o get-shard-iterator
- o help
- o list-streams
- o list-tags-for-stream
- o merge-shards
- o put-record
- o put-records
- o remove-tags-from-stream
- o split-shard
- o wait

Diese Befehlsliste entspricht der API von Kinesis Data Streams, die in der [Amazon Kinesis Service API Reference](#) beschrieben wird. Zum Beispiel entspricht der Befehl `create-stream` der API-Aktion `CreateStream`.

Die AWS CLI wurde jetzt erfolgreich installiert, jedoch nicht konfiguriert. Dies wird im nächsten Abschnitt veranschaulicht.

## Konfigurieren von AWS CLI

Für den allgemeinen Gebrauch ist der Befehl `aws configure` die schnellste Möglichkeit, eine AWS CLI-Installation einzurichten. Weitere Informationen finden Sie unter [Konfigurieren der AWS-CLI](#).

## Ausführen von einfachen Kinesis-Daten-Stream-Operationen mit der AWS CLI

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

### Note

Nach dem Erstellen eines Streams fallen für die Nutzung von Kinesis Data Streams mit Ihrem Konto Nominalgebühren an, da Kinesis Data Streams nicht für das kostenlose Kontingent für AWS infrage kommt. Wenn Sie mit diesem Tutorial fertig sind, sollten Sie Ihre AWS-Ressourcen löschen, damit keine weiteren Gebühren anfallen. Weitere Informationen finden Sie unter [Schritt 4: Bereinigen](#).

### Themen

- [Schritt 1: Stream erstellen](#)
- [Schritt 2: Senden eines Datensatzes](#)
- [Schritt 3: Abrufen des Datensatzes](#)
- [Schritt 4: Bereinigen](#)

## Schritt 1: Stream erstellen

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

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

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

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

Sie sollten eine Ausgabe ähnlich dem folgenden Beispiel erhalten:

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

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

```
{
  "StreamDescriptionSummary": {
```

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

Es gibt Informationen in dieser Ausgabe, die Sie für dieses Tutorial nicht benötigen. Die wichtigste Information ist momentan "StreamStatus": "ACTIVE". Dadurch erfahren Sie, dass der Stream einsatzbereit ist, und Sie erhalten die Informationen zu dem einzelnen Shard, den Sie angefordert haben. Sie können die Existenz Ihres neuen Streams auch mit dem Befehl `list-streams` überprüfen, wie in der folgenden Abbildung dargestellt:

```
aws kinesis list-streams
```

Ausgabe:

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

## Schritt 2: Senden eines Datensatzes

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

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

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

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

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

## Schritt 3: Abrufen des Datensatzes

### GetShardIterator

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

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

Wie Sie wissen, gibt es eine API von Kinesis Data Streams zu den `aws kinesis`-Befehlen. Wenn Sie also mehr zu den angezeigten Parametern erfahren möchten, informieren Sie sich im API-Verweisthema [GetShardIterator](#). Bei erfolgreicher Ausführung wird eine Ausgabe ähnlich dem folgenden Beispiel angezeigt (horizontal scrollen, um die gesamte Ausgabe zu sehen):

```
{
  "ShardIterator": "AAAAAAAAAAHSywljv0zEgPX4NyKdZ5wryMzP9yALs8NeKbUjp1IxtZs1Sp
+KEd9I6AJ9ZG4lNR1EMi+9Md/nHvtLyxpfhEzYvkTZ4D9DQVz/mBYWR060TZRKnW9gd
+efGN2aHFdkH1rJl4BL9Wyrk+ghYGG22D2T1Da2EyNSH1+LAbK33gQweTJADBdyMwlo5r6PqcP2dzhg="
}
```

Die lange Zeichenfolge scheinbar zufälliger Zeichen ist der Shard-Iterator (Ihrer lautet jedoch anders). Sie müssen den Shard-Iterator kopieren und in den Abrufbefehl einfügen, wie unten dargestellt. Shard-Iteratoren verfügen über eine Gültigkeitsdauer von 300 Sekunden. Dies sollte ausreichen, um den Shard-Iterator zu kopieren und in den nächsten Befehl einzufügen. Beachten Sie, dass Sie Zeilenumbrüche aus Ihrem Shard-Iterator entfernen müssen, bevor Sie ihn in den nächsten Befehl

einfügen. Wenn Sie die Fehlermeldung erhalten, dass der Shard-Iterator nicht mehr gültig ist, führen Sie einfach den Befehl `get-shard-iterator` erneut aus.

## GetRecords

Der Befehl `get-records` ruft Daten aus dem Stream ab und wird in einen Aufruf an [GetRecords](#) in der API von Kinesis Data Streams aufgelöst. Der Shard-Iterator gibt die Position im Shard an, ab der Datensätze sequenziell ausgelesen werden sollen: Wenn keine Datensätze in dem Teil des Shards verfügbar sind, auf die der Iterator zeigt, gibt `GetRecords` eine leere Liste zurück. Bitte beachten Sie, dass möglicherweise mehrere Aufrufe erforderlich sind, um einen Teil des Shards abzurufen, der Datensätze enthält.

Im folgenden Beispiel des Befehls `get-records` (horizontal scrollen, um den gesamten Befehl zu sehen):

```
aws kinesis get-records --shard-iterator
AAAAAASywljv0zEgPX4NyKdZ5wryMzP9yALs8NeKbUjp1IxtZs1Sp+KEd9I6AJ9ZG41NR1EMi
+9Md/nHvtLyxpfhEzYvkTZ4D9DQVz/mBYWR060TZRKnW9gd+efGN2aHFdkH1rJl4BL9Wyrk
+ghYG22D2T1Da2EyNSH1+LABk33gQweTJADBdyMwlo5r6PqcP2dzhg=
```

Wenn Sie dieses Tutorial über einen Unix-ähnlichen Befehlsprozessor wie Bash ausführen, können Sie das Abrufen des Shard-Iterators über einen verschachtelten Befehl automatisieren, wie folgt (horizontal scrollen, um den gesamten Befehl zu sehen):

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

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

Wenn Sie dieses Tutorial über ein System ausführen, das PowerShell unterstützt, können Sie das Abrufen des Shard-Iterators über einen verschachtelten Befehl automatisieren, wie folgt (horizontal scrollen, um den gesamten Befehl zu sehen):

```
aws kinesis get-records --shard-iterator ((aws kinesis get-shard-iterator --shard-id
shardId-000000000000 --shard-iterator-type TRIM_HORIZON --stream-name Foo).split('')
[4])
```

Das erfolgreiche Ergebnis des Befehls `get-records` fordert Datensätze aus Ihrem Stream für den Shard an, den Sie beim Abrufen des Shard-Iterators angegeben haben, wie im folgenden Beispiel (horizontal scrollen, um die gesamte Ausgabe zu sehen):

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

  "NextShardIterator": "AAAAAAAAAAED0W3ugseWPE4503kqN1yN1UaodY8unE0sYs1MUmC6lX9hlig5+t4RtZM0/
tALfiI4QGjunVgJvQsjxjh2aLyxaAaPr
+LaoENQ7eVs4EdYXgKyThTZGPcca2fVXYJWL3yafv9dsDwsYVedI66dbMZFC8rPMWc797zxQkv4pSKvP0ZvrUIudb8UkH3V
}
}
```

Beachten Sie, dass `get-records` oben als Anforderung beschrieben ist. Dies bedeutet, dass Sie null oder mehr Datensätze empfangen können, selbst wenn Datensätze in Ihrem Stream vorhanden sind, und dass die zurückgegebenen Datensätze möglicherweise nicht alle aktuell in Ihrem Stream vorhandenen Datensätze wiedergeben. Dies ist völlig normal, der Produktionscode ruft einfach in angemessenen Intervallen den Stream für Datensätze ab (die Abrufgeschwindigkeit hängt von Ihren spezifischen Anforderungen an das Anwendungsdesign ab).

Das Erste, was Sie an Ihrem Datensatz in diesem Teil des Tutorials wahrscheinlich feststellen werden, ist, dass die Daten unbrauchbar aussehen – wir haben nicht den Klartext `testdata` gesendet. Dies liegt an der Art und Weise, auf die `put-record` Base64-Codierung verwendet, um Ihnen das Senden von Binärdaten zu ermöglichen. Die Unterstützung von Kinesis Data Streams in der AWS CLI stellt jedoch keine Base64-Decodierung bereit, da eine Base64-Decodierung von rohem Binärinhalt, der in `stdout` ausgegeben wird, auf bestimmten Plattformen und Terminals zu unerwünschtem Verhalten und potenziellen Sicherheitsproblemen führen kann. Wenn Sie einen Base64-Decoder verwenden (z. B. <https://www.base64decode.org/>), um `dGVzdGRhdGE=` manuell zu decodieren, sehen Sie, dass es tatsächlich `testdata` ist. Dies ist ausreichend für dieses Tutorial, da die AWS CLI in der Praxis nur selten verwendet wird, um Daten zu konsumieren. Sie wird vielmehr zum Überwachen des Stream-Status und zum Abrufen von Informationen verwendet, wie zuvor gezeigt (`describe-stream` und `list-streams`). In zukünftigen Tutorials erfahren Sie, wie Sie Konsumenten Anwendungen mit Produktionsqualität mithilfe der Kinesis Client Library (KCL) erstellen, wobei Ihnen der Base64-Prozess abgenommen wird. Weitere Informationen zur KCL finden Sie unter [Entwickeln benutzerdefinierter Verbraucher mit gemeinsamem Durchsatz mithilfe von KCL](#).

Es ist nicht immer der Fall, dass `get-records` alle Datensätze im angegebenen Stream/Shard zurückgibt. Verwenden Sie in diesem Fall den `NextShardIterator` aus dem letzten Ergebnis,

um die nächste Datensatzgruppe abzurufen. Wenn mehr Daten in den Stream eingespeist würden (was der normalen Situation in Produktionsanwendungen entspricht), könnten Sie weiterhin jedes Mal mithilfe von `get-records` Daten abrufen. Wenn Sie `get-records` jedoch nicht über den nächsten Shard-Iterator innerhalb der 300 Sekunden Lebensdauer des zweiten Shard-Iterators aufrufen, erhalten Sie eine Fehlermeldung und Sie müssen den Befehl `get-shard-iterator` verwenden, um einen neuen Shard-Iterator abzurufen.

In dieser Ausgabe wird ebenfalls `MillisBehindLatest` bereitgestellt. Dabei handelt es sich um die Anzahl der Millisekunden, die die Antwort des [GetRecords](#)-Vorgangs vom vorderen Ende des Streams entfernt ist. Dies zeigt an, wie weit der Konsument hinter der aktuellen Zeit zurückliegt. Der Wert Null gibt an, dass die Datenverarbeitung aktuell ist und dass zurzeit keine neuen zu verarbeitenden Datensätze vorhanden sind. In diesem Tutorial sehen Sie möglicherweise eine recht große Zeit, wenn Sie sich die Zeit dafür genommen haben, stets mitzulesen. Das ist kein Problem, da die Datensätze standardmäßig 24 Stunden lang in einem Stream bleiben, damit Sie diese abrufen können. Dieser Zeitraum wird als Aufbewahrungszeitraum bezeichnet und ist bis zu 365 Tage konfigurierbar.

Beachten Sie, dass ein erfolgreiches `get-records`-Ergebnis stets einen `NextShardIterator` aufweist, selbst wenn sich aktuell keine weiteren Datensätze im Stream befinden. Hierbei handelt es sich um ein Abrufmodell, bei dem davon ausgegangen wird, dass ein Produzent zu einem bestimmten Zeitpunkt möglicherweise mehr Datensätze in den Stream einfügt. Auch wenn Sie Ihre eigene Abrufroutine schreiben können, wird Ihnen das Abrufen abgenommen, wenn Sie die zuvor erwähnte KCL zum Entwickeln von Konsumenten Anwendungen verwenden.

Wenn Sie `get-records` solange aufrufen, bis im Stream und im Shard, aus denen der Abruf erfolgt, keine Datensätze mehr vorhanden sind, wird eine Ausgabe mit leeren Datensätzen angezeigt, ähnlich dem folgenden Beispiel (horizontal scrollen, um die gesamte Ausgabe zu sehen):

```
{
  "Records": [],
  "NextShardIterator": "AAAAAAAAAAGCJ5jzQNjmdh06B/YDIDE56jmZmrmMA/r1WjoHXC/
kPJXc1rckt3TFL55dENfe5meNgdkyCRpUPGzJpMgYHaJ53C3nCAjQ6s7ZupjXeJGoUFs5oCuFwhP+Wu1/
EhyNeSs5DYXLSSC5XCcapmCAYGFjYER69QsdQjxMmBPE/hiybFDi5qtkT6/PsZNz6kFoqtDk="
}
```

## Schritt 4: Bereinigen

Zuletzt sollten Sie Ihren Stream löschen, um Ressourcen freizugeben und unbeabsichtigte Änderungen am Konto zu verhindern, wie zuvor erwähnt. Führen Sie diese Aktion in der Praxis jedes



Mal durch, wenn Sie einen Stream erstellt haben und diesen nicht mehr benötigen, da weiterhin Kosten pro Stream anfallen – gleich, ob Sie mit diesem Daten bereitstellen und abrufen oder nicht. Der Bereinigungsbefehl ist einfach:

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

Bei Erfolg erfolgt keine Ausgabe, sodass Sie gegebenenfalls `describe-stream` möchten, um den Fortschritt des Löschvorgangs zu überprüfen:

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

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

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

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

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

# Beispiel-Tutorials für Amazon Kinesis Data Streams

Die Beispiel-Tutorials in diesem Abschnitt wurden entwickelt, um Ihnen weitere Einblicke in die Konzepte und Funktionen von Amazon Kinesis Data Streams zu gewähren.

## Themen

- [Tutorial: Verarbeiten von Wertpapier-Echtzeitdaten mit KPL und KCL 2.x](#)
- [Tutorial: Verarbeiten von Wertpapier-Echtzeitdaten mit KPL und KCL 1.x](#)
- [Tutorial: Analysieren von Wertpapierdaten in Echtzeit mit Managed Service für Apache Flink](#)
- [Anleitung: Verwenden von AWS Lambda mit Amazon Kinesis Data Streams](#)
- [AWS-Streaming-Datenlösung für Amazon Kinesis](#)

## Tutorial: Verarbeiten von Wertpapier-Echtzeitdaten mit KPL und KCL 2.x

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

### Important

Nach dem Erstellen eines Streams fallen für die Nutzung von Kinesis Data Streams mit Ihrem Konto Nominalgebühren an, da Kinesis Data Streams nicht für das kostenlose Kontingent für AWS infrage kommt. Nach dem Start der Konsumentenanzwendung fallen nominale Gebühren für die Amazon-DynamoDB-Nutzung an. Die Konsumentenanzwendung verwendet DynamoDB zum Verfolgen des Verarbeitungsstatus. Wenn Sie mit dieser Anwendung fertig sind, sollten Sie Ihre AWS-Ressourcen löschen, damit keine weiteren Gebühren anfallen. Weitere Informationen finden Sie unter [Schritt 7: Abschluss](#).

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

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

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

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

## Aufgaben

- [Voraussetzungen](#)
- [Schritt 1: Erstellen eines Daten-Streams](#)
- [Schritt 2: Erstellen von IAM-Richtlinie und -Benutzer](#)
- [Schritt 3: Herunterladen und Erstellen des Codes](#)
- [Schritt 4: Produzent implementieren](#)
- [Schritt 5: Konsument implementieren](#)
- [Schritt 6: \(optional\) Konsument erweitern](#)
- [Schritt 7: Abschluss](#)

## Voraussetzungen

Sie müssen die folgenden Anforderungen erfüllen, um dieses Tutorial durchzuführen:

### Amazon-Web-Services-Konto

Bevor Sie beginnen, müssen Sie sich mit den in [Terminologie und Konzepte von Amazon Kinesis Data Streams](#) erörterten Konzepten vertraut machen, insbesondere mit Streams, Shards, Produzenten und Verbrauchern. Es ist zudem sinnvoll, die Schritte in der folgenden Anleitung durchzuführen: [Installieren und Konfigurieren der AWS CLI](#).

Sie benötigen ein AWS-Konto und einen Web-Browser für den Zugriff auf die AWS Management Console.

Verwenden Sie für den Konsolenzugriff Ihren IAM-Benutzernamen und Ihr Passwort, um sich über die IAM-Anmeldeseite bei [AWS Management Console](#) anzumelden. Informationen zu AWS-Sicherheitsanmeldedaten, einschließlich programmatischem Zugriff und Alternativen zu langfristigen Anmeldeinformationen, finden Sie unter [AWS-Sicherheitsanmeldedaten](#) im IAM-Benutzerhandbuch. Weitere Informationen zum Anmelden bei AWS-Konto finden Sie unter [Anmelden bei AWS](#) im Benutzerhandbuch von AWS-Anmeldung.

Weitere Informationen zu IAM und dem Einrichten von Sicherheitsschlüsseln finden Sie unter [Erstellen eines IAM-Benutzers](#).

## Systemsoftwareanforderungen

Im System, das Sie zum Ausführen der Anwendung verwenden, muss Java 7 oder höher installiert sein. Zum Herunterladen und Installieren des neuesten Java Development Kit (JDK) rufen Sie die [Website für die Java SE-Installation von Oracle](#) auf.

Wenn Sie über eine Java-IDE wie [Eclipse](#) verfügen, können Sie den Quellcode darin öffnen, bearbeiten, erstellen und ausführen.

Sie benötigen die neueste [AWS SDK for Java](#)-Version. Wenn Sie Eclipse als IDE (integrierte Entwicklungsumgebung) nutzen, können Sie stattdessen das [AWS-Toolkit for Eclipse](#) installieren.

Die Verbraucheranwendung benötigt Kinesis Client Library (KCL) Version 2.2.9 oder höher, die Sie aus GitHub oder unter <https://github.com/awslabs/amazon-kinesis-client/tree/master> herunterladen können.

## Nächste Schritte

### [Schritt 1: Erstellen eines Daten-Streams](#)

## Schritt 1: Erstellen eines Daten-Streams

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

So erstellen Sie einen Stream

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

2. Klicken Sie im Navigationsbereich auf Data Streams (Daten-Streams).
3. Erweitern Sie in der Navigationsleiste die Regionsauswahl und wählen Sie eine aus.
4. Wählen Sie Create Kinesis Stream (Kinesis-Stream erstellen).
5. Geben Sie einen Namen für den Daten-Stream ein (z. B. **StockTradeStream**).
6. Geben Sie **1** für die Anzahl der Shards ein und erweitern Sie Estimate the number of shards you'll need (Anzahl der benötigten Shards schätzen) nicht.
7. Wählen Sie Create Kinesis Stream (Kinesis-Stream erstellen).

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

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

## Nächste Schritte

### [Schritt 2: Erstellen von IAM-Richtlinie und -Benutzer](#)

## Schritt 2: Erstellen von IAM-Richtlinie und -Benutzer

Die bewährten Sicherheitsmethoden für AWS geben die Zuweisung fein abgestimmter Berechtigungen vor, um den Zugriff auf verschiedene Ressourcen zu steuern. AWS Identity and Access Management (IAM) können Sie Benutzer und Benutzerberechtigungen in AWS verwalten. Eine [IAM-Richtlinie](#) führt explizit zulässige Aktionen sowie Ressourcen auf, für die diese Aktionen relevant sind.

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

### Produzent

Aktionen	Ressource	Zweck
DescribeStream , DescribeStreamSumm	Kinesis Data Stream	Vor dem Versuch, Datensätze zu lesen, prüft der Verbraucher, ob der Stream vorhanden ist und ob er aktiv ist und ob Shards im Daten-

Aktionen	Ressource	Zweck
<code>DescribeStreamConsumer</code>		
<code>SubscribeToShard</code> , <code>RegisterStreamConsumer</code>	Kinesis Data Stream	Abonniert und registriert Verbraucher bei einem Shard.
<code>PutRecord</code> , <code>PutRecords</code>	Kinesis Data Stream	Schreibt Datensätze in Kinesis Data Streams.

### Konsument

Aktionen	Resource	Zweck
<code>DescribeStream</code>	Kinesis Data Stream	Vor dem Versuch, Datensätze zu lesen, prüft der Verbraucher, ob ein Shard vorhanden ist und ob er aktiv ist und ob Shards im Datenstrom vorhanden sind.
<code>GetRecords</code> , <code>GetShardIterator</code>	Kinesis Data Stream	Auslesen von Datensätzen aus einem Shard.
<code>CreateTable</code> , <code>DescribeTable</code> , <code>GetItem</code> , <code>PutItem</code> , <code>Scan</code> , <code>UpdateItem</code>	Amazon-DynamoDB-Tabelle.	Wenn der Verbraucher mit der Kinesis Client Library (KCL) entwickelt wird, benötigt er Berechtigungen für die DynamoDB-Tabelle, um den Verarbeitungszustand der Anwendung zu verfolgen.
<code>DeleteItem</code>	Amazon-DynamoDB-Tabelle.	Für den Fall, dass der Konsument Split/Merge-Operationen auf Kinesis Data Streams ausführt.
<code>PutMetricData</code>	Amazon CloudWatch Logs	Die KCL lädt auch Metriken in CloudWatch hoch. Diese sind für die Anwendung nützlich.

Im Rahmen dieses Tutorials erstellen Sie eine einzelne IAM-Richtlinie, die alle oben genannten Berechtigungen gewährt. Im Praxiseinsatz können Sie zwei Richtlinien erstellen, eine für Produzenten und eine für Verbraucher.

## So erstellen Sie eine IAM-Richtlinie

1. Suchen Sie den Amazon-Ressourcennamen (ARN) für den neuen Daten-Stream, den Sie im Schritt oben erstellt haben. Sie finden diesen ARN als Stream ARN (Stream-ARN) oben auf der Registerkarte Details. Das ARN-Format sieht folgendermaßen aus:

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

### region

Der AWS-Regionscode, beispielsweise `us-west-2`. Weitere Informationen finden Sie unter [Regionen und Verfügbarkeitskonzepte](#).

### Konto

Die AWS-Konto-ID, wie Sie in den [Kontoeinstellungen](#) angezeigt wird.

### Name

Der Name des Daten-Streams, den Sie im Schritt oben erstellt haben, also `StockTradeStream`.

2. Bestimmen Sie den ARN für die DynamoDB-Tabelle, die vom Verbraucher verwendet und von der ersten Verbraucher-Instance erstellt wird. Er muss das folgende Format aufweisen:

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

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

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

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

AWS-Service	Aktionen	ARN
Amazon DynamoDB	<code>CreateTable</code> , <code>DeleteItem</code> , <code>DescribeTable</code> , <code>GetItem</code> , <code>PutItem</code> , <code>Scan</code> , <code>UpdateItem</code>	Der ARN der DynamoDB-Tabelle, den Sie in Schritt 2 dieses Verfahrens erstellt haben.
Amazon CloudWatch	<code>PutMetricData</code>	*

Das Sternchen (\*), das bei der Angabe einer ARN verwendet wird, ist nicht erforderlich. Dies ist hier der Fall, da es in CloudWatch keine bestimmte Ressource gibt, auf der die `PutMetricData`-Aktion aufgerufen wird.

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

Das resultierende Richtliniendokument sollte folgendermaßen aussehen:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmnt123",
      "Effect": "Allow",
      "Action": [
        "kinesis:DescribeStream",
        "kinesis:PutRecord",
        "kinesis:PutRecords",
        "kinesis:GetShardIterator",
        "kinesis:GetRecords",
        "kinesis:ListShards",

```



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

## So erstellen Sie einen IAM-Benutzer

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

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

## So fügen Sie einem Benutzer eine IAM-Richtlinie hinzu

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

## Nächste Schritte

### [Schritt 3: Herunterladen und Erstellen des Codes](#)

## Schritt 3: Herunterladen und Erstellen des Codes

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

So laden Sie den Code herunter und erstellen ihn

1. Laden Sie den Quellcode aus dem GitHub-Repository <https://github.com/aws-samples/amazon-kinesis-learning> auf den Computer herunter.
2. Erstellen Sie in der IDE ein Projekt mit dem Quellcode. Behalten Sie die bereitgestellte Verzeichnisstruktur bei.
3. Fügen Sie dem Projekt die folgenden Bibliotheken hinzu:
  - Amazon Kinesis Client Library (KCL)
  - AWS-SDK
  - Apache HttpCore
  - Apache HttpClient
  - Apache Commons Lang
  - Apache Commons Logging
  - Guava (Google-Kernbibliotheken für Java)
  - Jackson Annotations
  - Jackson Core
  - Jackson Databind
  - Jackson Dataformat: CBOR
  - Joda Time
4. Je nach IDE wird das Projekt möglicherweise automatisch erstellt. Wenn nicht, erstellen Sie es mit den für Ihre IDE erforderlichen Schritten.

Wenn Sie diese Schritte erfolgreich abgeschlossen haben, können Sie zum nächsten Abschnitt wechseln, [the section called "Schritt 4: Produzent implementieren"](#).

Nächste Schritte

## Schritt 4: Produzent implementieren

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

Beachten Sie den [Quellcode](#) und prüfen Sie die folgenden Informationen.

### StockTrade-Klasse

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

### Stream-Datensatz

Ein Stream ist eine Sequenz von Datensätzen. Ein Datensatz ist die Serialisierung einer `StockTrade`-Instance im JSON-Format. Beispiele:

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

### StockTradeGenerator-Klasse

`StockTradeGenerator` enthält die Methode `getRandomTrade()`, die bei jedem Aufruf eine zufällig generierte Wertpapiertransaktion zurückgibt. Dieser Klasse wird für Sie implementiert.

### StockTradesWriter-Klasse

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

1. Namen des Daten-Streams und der Region als Eingabe lesen.
2. Region, Anmeldeinformationen und Client-Konfiguration mit `KinesisAsyncClientBuilder` festlegen.

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

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

```
private static void sendStockTrade(StockTrade trade, KinesisAsyncClient
kinesisClient,
    String streamName) {
    byte[] bytes = trade.toJsonAsBytes();
    // The bytes could be null if there is an issue with the JSON serialization
    by the Jackson JSON library.
    if (bytes == null) {
        LOG.warn("Could not get JSON bytes for stock trade");
        return;
    }

    LOG.info("Putting trade: " + trade.toString());
    PutRecordRequest request = PutRecordRequest.builder()
        .partitionKey(trade.getTickerSymbol()) // We use the ticker symbol
        as the partition key, explained in the Supplemental Information section below.
        .streamName(streamName)
        .data(SdkBytes.fromByteArray(bytes))
        .build();

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

Beachten Sie die folgende Code-Struktur:

- Die `PutRecord`-API nimmt ein Byte-Array entgegen. Sie müssen die Transaktion in das JSON-Format konvertieren. Diese einzelne Codezeile führt die Operation aus:

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

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

```
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();
```

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

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

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

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

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

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

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

Der Grund besteht darin, dass eine Kinesis Data Streams-PUT-Operation aufgrund eines Netzwerkfehlers fehlschlagen kann oder gedrosselt wird, weil die Durchsatzgrenze des Streams erreicht wird. Sie sollten die Wiederholungsrichtlinie für put-Operationen sorgfältig planen, um Datenverluste zu vermeiden, wie es bei einfachen Wiederholungen geschehen kann.

- Eine Statusprotokollierung ist hilfreich, wenn auch optional:

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

Der hier gezeigte Produzent verwendet die API-Funktionalität von Kinesis Data Streams für einzelne Datensätze `PutRecord`. In der Praxis ist es oft effizienter, die Eignung von `PutRecords` für mehrere Datensätze zu nutzen und mehrere Datensatzstapel gleichzeitig zu senden, wenn ein Produzent viele Datensätze erstellt. Weitere Informationen finden Sie unter [Schreiben von Daten in Amazon Kinesis Data Streams](#).

So führen Sie den Produzenten aus

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

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

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

Die Ausgabe sollte folgendermaßen oder ähnlich aussehen:

```
Feb 16, 2015 3:53:00 PM
com.amazonaws.services.kinesis.samples.stocktrades.writer.StockTradesWriter
  sendStockTrade
INFO: Putting trade: ID 8: SELL 996 shares of BUD for $124.18
Feb 16, 2015 3:53:00 PM
com.amazonaws.services.kinesis.samples.stocktrades.writer.StockTradesWriter
  sendStockTrade
INFO: Putting trade: ID 9: BUY 159 shares of GE for $20.85
Feb 16, 2015 3:53:01 PM
com.amazonaws.services.kinesis.samples.stocktrades.writer.StockTradesWriter
  sendStockTrade
INFO: Putting trade: ID 10: BUY 322 shares of WMT for $90.08
```

Ihre Wertpapierdaten werden nun von Kinesis Data Streams eingespeist.

## Nächste Schritte

### [Schritt 5: Konsument implementieren](#)

## Schritt 5: Konsument implementieren

Die Verbraucheranwendung in diesem Tutorial verarbeitet die Wertpapiertransaktionen im Daten-Stream kontinuierlich. Sie gibt dann für jede Minute die beliebtesten Aktien aus, die gekauft und verkauft wurden. Die Anwendung setzt auf Kinesis Client Library (KCL) auf, die viele der mühsamen Arbeiten einer Verbraucheranwendung übernimmt. Weitere Informationen finden Sie unter [Verwenden der Kinesis Client Library](#).

Überprüfen Sie die folgenden Informationen in Bezug auf den Quellcode.

### StockTradesProcessor-Klasse

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

- Namen von Anwendung, Daten-Stream und Region lesen, die als Argumente übergeben werden.



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

### StockTradeRecordProcessor-Klasse

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

Die Methoden `initialize` und `shutdownRequested` werden von KCL verwendet, um dem Datensatzverarbeiter mitzuteilen, wann er bereit sein muss, Datensätze zu empfangen, und wann der Empfang von Datensätzen gestoppt werden muss, damit alle anwendungsspezifischen Einrichtungs- und Beendigungsaufgaben ausgeführt werden können. `leaseLost` und `shardEnded` werden verwendet, um Logik zu implementieren, die beim Verlust eines Lease oder bei Erreichen des Shards-Endes im Rahmen der Shard-Verarbeitung auszuführen ist. In diesem Beispiel protokollieren wir einfach Meldungen dieser Ereignisse.

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

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

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

- Für jeden übergebenen Datensatz wird `processRecord` aufgerufen.

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

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

### StockStats-Klasse

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

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

Diese Klasse verfolgt die beliebtesten Wertpapiere, indem kontinuierlich die Anzahl der Handelstransaktionen für jedes Wertpapier sowie die maximale Anzahl gezählt werden. Sie aktualisiert diese Werte, sobald eine neue Handelstransaktion empfangen wird.

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

So implementieren Sie den Konsumenten

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

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

```

        return;
    }
    stockStats.addStockTrade(trade);

```

2. Implementieren Sie eine einfache `reportStats`-Methode. Sie können das Ausgabeformat an die jeweiligen Anforderungen anpassen.

```

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

```

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

```

stockStats = new StockStats();

```

4. Implementieren der folgenden Methoden, die für die `ShardRecordProcessor`-Schnittstelle benötigt werden

```

@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(RecordProcessorCheckpointter checkpointer) {
        log.info("Checkpointing shard " + kinesisShardId);
        try {
            checkpointer.checkpoint();
        } catch (ShutdownException se) {
            // Ignore checkpoint if the processor instance has been shutdown (fail
            over).
            log.info("Caught shutdown exception, skipping checkpoint.", se);
        } catch (ThrottlingException e) {
            // Skip checkpoint when throttled. In practice, consider a backoff and
            retry policy.
            log.error("Caught throttling exception, skipping checkpoint.", e);
        } catch (InvalidStateException e) {
            // This indicates an issue with the DynamoDB table (check for table,
            provisioned IOPS).
            log.error("Cannot save checkpoint to the DynamoDB table used by the Amazon
            Kinesis Client Library.", e);
        }
    }
}
```

So führen Sie den Konsumenten aus

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

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

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

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

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

## Nächste Schritte

### [Schritt 6: \(optional\) Konsument erweitern](#)

## Schritt 6: (optional) Konsument erweitern

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

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

So erweitern Sie den Konsumenten

1. Fügen Sie neue Instance-Variablen hinzu:

```
// Ticker symbol of the stock that had the largest quantity of shares sold
private String largestSellOrderStock;
// Quantity of shares for the largest sell order trade
private long largestSellOrderQuantity;
```

2. Fügen Sie folgenden Code zu `addStockTrade` hinzu:

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

```
    }  
}
```

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

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

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

```
***** Shard shardId-000000000001 stats for last 1 minute *****  
Most popular stock being bought: WMT, 27 buys.  
Most popular stock being sold: PTR, 14 sells.  
Largest sell order: 996 shares of BUD.  
*****
```

## Nächste Schritte

### [Schritt 7: Abschluss](#)

## Schritt 7: Abschluss

Da Sie für die Nutzung des Kinesis Data Streams zahlen, sollten Sie diesen ebenso wie die entsprechende Amazon-DynamoDB-Tabelle löschen, wenn sie nicht mehr benötigt werden. Für einen aktiven Stream fallen auch dann nominale Gebühren an, wenn Sie keine Datensätze senden

oder abrufen. Der Grund hierfür ist, dass ein aktiver Stream durch kontinuierlichen „Horchen“ darauf, ob neue Datensätze oder Anforderungen zum Abrufen von Datensätzen eingehen, Ressourcen verbraucht.

So löschen Sie den Stream und die Tabelle

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

## Übersicht

Für die Verarbeitung großer Datenmengen in nahezu Echtzeit muss weder ein magischer Code geschrieben noch eine umfangreiche Infrastruktur entwickelt werden. Das Ganze ist so einfach wie das Schreiben einer Logik für die Verarbeitung einer kleinen Datenmenge (wie das Schreiben von `processRecord(Record)`), die dann zu Skalierungszwecken mit Kinesis Data Streams kombiniert wird, damit auch große Mengen an Streaming-Daten verarbeitet werden können. Sie müssen sich keine Gedanken über die Skalierung der Verarbeitung machen, da Kinesis Data Streams das für Sie übernimmt. Sie müssen lediglich Ihre Streaming-Datensätze an Kinesis Data Streams senden und eine Logik für die Verarbeitung neu empfangener Datensätze schreiben.

Nachfolgend einige mögliche Erweiterungen für diese Anwendung.

### Shard-übergreifende Aggregation

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

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

## Skalierung

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

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

Da ein Stream kontinuierlich verarbeitet wird, kann seine Ausgabe an andere Ziele gesendet werden. AWS bietet [Konnektoren](#) zur Integration von Kinesis Data Streams mit anderen AWS-Services und Tools von Drittanbietern.

## Tutorial: Verarbeiten von Wertpapier-Echtzeitdaten mit KPL und KCL 1.x

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

### Important

Nach dem Erstellen eines Streams fallen für die Nutzung von Kinesis Data Streams mit Ihrem Konto Nominalggebühren an, da Kinesis Data Streams nicht für das kostenlose Kontingent für AWS infrage kommt. Nach dem Start der Konsumentenanzwendung fallen nominale Gebühren für die Amazon-DynamoDB-Nutzung an. Die Konsumentenanzwendung verwendet DynamoDB zum Verfolgen des Verarbeitungsstatus. Wenn Sie mit dieser Anwendung fertig sind, sollten Sie Ihre AWS-Ressourcen löschen, damit keine weiteren Gebühren anfallen. Weitere Informationen finden Sie unter [Schritt 7: Abschluss](#).

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



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

Sie können die in diesem Tutorial aufgeführten Schritte auf einem Desktop-PC oder Laptop durchführen und sowohl den Produzenten- als auch den Konsumentencode auf demselben Rechner oder einer Plattform ausführen, die die angegebenen Anforderungen erfüllt, beispielsweise Amazon Elastic Compute Cloud (Amazon EC2).

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

## Aufgaben

- [Voraussetzungen](#)
- [Schritt 1: Erstellen eines Daten-Streams](#)
- [Schritt 2: Erstellen von IAM-Richtlinie und -Benutzer](#)
- [Schritt 3: Implementierungscode herunterladen und erstellen](#)
- [Schritt 4: Produzent implementieren](#)
- [Schritt 5: Konsument implementieren](#)
- [Schritt 6: \(optional\) Konsument erweitern](#)
- [Schritt 7: Abschluss](#)

## Voraussetzungen

Für [Tutorial: Verarbeiten von Wertpapier-Echtzeitdaten mit KPL und KCL 1.x](#) gelten die folgenden Anforderungen.

### Amazon-Web-Services-Konto

Bevor Sie beginnen, müssen Sie sich mit den in [Terminologie und Konzepte von Amazon Kinesis Data Streams](#) behandelten Konzepten vertraut machen, insbesondere mit Streams, Shards, Produzenten und Konsumenten. Das Durcharbeiten von [Installieren und Konfigurieren der AWS CLI](#) ist ebenfalls hilfreich.

Sie benötigen ein AWS-Konto und einen Webbrowser für den Zugriff auf die AWS Management Console.

Verwenden Sie für den Konsolenzugriff Ihren IAM-Benutzernamen und Ihr Passwort, um sich über die IAM-Anmeldeseite bei [AWS Management Console](#) anzumelden. Informationen zu AWS-Sicherheitsanmeldedaten, einschließlich programmatischem Zugriff und Alternativen zu langfristigen Anmeldeinformationen, finden Sie unter [AWS-Sicherheitsanmeldedaten](#) im IAM-Benutzerhandbuch. Weitere Informationen zum Anmelden bei AWS-Konto finden Sie unter [Anmelden bei AWS](#) im Benutzerhandbuch von AWS-Anmeldung.

Weitere Informationen zu IAM und dem Einrichten von Sicherheitsschlüsseln finden Sie unter [Erstellen eines IAM-Benutzers](#).

## Systemsoftwareanforderungen

Auf dem System, das die Anwendung ausführt, muss Java 7 oder höher installiert sein. Zum Herunterladen und Installieren des neuesten Java Development Kit (JDK) rufen Sie die [Website für die Java SE-Installation von Oracle](#) auf.

Wenn Sie über eine Java IDE wie [Eclipse](#) verfügen, können Sie den Quellcode öffnen, bearbeiten, erstellen und ausführen.

Sie benötigen die neueste [AWS SDK for Java](#)-Version. Wenn Sie Eclipse als IDE (integrierte Entwicklungsumgebung) nutzen, können Sie stattdessen das [AWS-Toolkit for Eclipse](#) installieren.

Die Konsumentenanzwendung erfordert die Kinesis Client Library (KCL), Version 1.2.1 oder höher, die Sie von GitHub unter [Kinesis Client Library \(Java\)](#) herunterladen können.

## Nächste Schritte

### [Schritt 1: Erstellen eines Daten-Streams](#)

## Schritt 1: Erstellen eines Daten-Streams

Im ersten Schritt von [Tutorial: Verarbeiten von Wertpapier-Echtzeitdaten mit KPL und KCL 1.x](#) erstellen Sie den Stream, der in den weiteren Schritten genutzt wird.

So erstellen Sie einen Stream

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

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

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

## Weitere Informationen zu Shards

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

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

### Durchschnittliche Datensatzgröße

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

### Max. geschriebene Datensätze

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

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

### Anzahl der Konsumenten Anwendungen

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

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

## Nächste Schritte

### [Schritt 2: Erstellen von IAM-Richtlinie und -Benutzer](#)

## Schritt 2: Erstellen von IAM-Richtlinie und -Benutzer

Die bewährten Sicherheitsmethoden für AWS geben die Zuweisung fein abgestimmter Berechtigungen vor, um den Zugriff auf verschiedene Ressourcen zu steuern. AWS Identity and Access Management (IAM) können Sie Benutzer und Benutzerberechtigungen in AWS verwalten. Eine [IAM-Richtlinie](#) führt explizit zulässige Aktionen sowie Ressourcen auf, für die diese Aktionen relevant sind.

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

### Produzent

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

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

## Konsument

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

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

## So erstellen Sie eine IAM-Richtlinie

1. Suchen Sie den Amazon-Ressourcennamen (ARN) für den neuen Stream. Sie finden diesen ARN als Stream ARN (Stream-ARN) oben auf der Registerkarte Details. Das ARN-Format sieht folgendermaßen aus:

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

### region

Der Regionscode, beispielsweise `us-west-2`. Weitere Informationen finden Sie unter [Regionen und Verfügbarkeitskonzepte](#).

### Konto

Die AWS-Konto-ID, wie Sie in den [Kontoeinstellungen](#) angezeigt wird.

### Name

Der Name des Streams von [Schritt 1: Erstellen eines Daten-Streams](#), hier `StockTradeStream`.

2. Bestimmen Sie den ARN für die DynamoDB-Tabelle, die vom Konsumenten verwendet wird (und von der ersten Konsumenten-Instance erstellt wird). Er muss das folgende Format aufweisen:

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

Region und Konto stammen vom selben Ort wie im vorherigen Schritt. Hier ist der Name jedoch der Name der Tabelle, die von der Verbraucheranwendung erstellt und verwendet wird. Die vom Konsumenten verwendete KCL nutzt den Anwendungsnamen als Tabellennamen. Verwenden Sie `StockTradesProcessor`, dies ist der Anwendungsname, der später genutzt wird.

3. Wählen Sie in der IAM-Konsole unter Richtlinien (<https://console.aws.amazon.com/iam/home#policies>) die Option Richtlinie erstellen aus. Wenn Sie zum ersten Mal mit IAM-Richtlinien arbeiten, wählen Sie Erste Schritte, Richtlinie erstellen aus.
4. Wählen Sie Select (Auswählen) neben Policy Generator (Richtliniengenerator) aus.
5. Wählen Sie Amazon Kinesis als AWS-Service aus.
6. Legen Sie `DescribeStream`, `GetShardIterator`, `GetRecords`, `PutRecord` und `PutRecords` als zulässige Aktionen fest.
7. Geben Sie den in Schritt 1 erstellten ARN ein.

## 8. Verwenden Sie Add Statement (Statement hinzufügen) für die folgenden Elemente:

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

Das Sternchen (\*), das bei der Angabe einer ARN verwendet wird, ist nicht erforderlich. Dies ist hier der Fall, da es in CloudWatch keine bestimmte Ressource gibt, auf der die PutMetricData-Aktion aufgerufen wird.

## 9. Wählen Sie Next Step (Weiter) aus.

## 10. Ändern Sie Policy Name (Richtliniennamen) in StockTradeStreamPolicy, prüfen Sie den Code und wählen sie Create Policy (Richtlinie erstellen) aus.

Das erstellte Richtliniendokument sollte etwa wie folgt aussehen:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmnt123",
      "Effect": "Allow",
      "Action": [
        "kinesis:DescribeStream",
        "kinesis:PutRecord",
        "kinesis:PutRecords",
        "kinesis:GetShardIterator",
        "kinesis:GetRecords",
        "kinesis:ListShards",
        "kinesis:DescribeStreamSummary",
        "kinesis:RegisterStreamConsumer"
      ],
      "Resource": [
```

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

## So erstellen Sie einen IAM-Benutzer

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



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

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

So fügen Sie einem Benutzer eine IAM-Richtlinie hinzu

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

## Nächste Schritte

### [Schritt 3: Implementierungscode herunterladen und erstellen](#)

## Schritt 3: Implementierungscode herunterladen und erstellen

Für [the section called "Tutorial: Verarbeiten von Wertpapier-Echtzeitdaten mit KPL und KCL 1.x"](#) wird Skeleton-Code bereitgestellt. Er enthält eine Stub-Implementierung für die Übernahme des Wertpapierdaten-Streams (Produzent) und die Verarbeitung der Daten (Verbraucher). Im folgenden Verfahren wird beschrieben, wie die Implementierungen abgeschlossen werden.

## So laden und erstellen Sie den Implementierungscode

1. Laden Sie den [Quellcode](#) auf den Computer herunter.
2. Erstellen Sie mit dem Quellcode unter Einhaltung der bereitgestellten Verzeichnisstruktur ein Projekt in der bevorzugten IDE.
3. Fügen Sie dem Projekt die folgenden Bibliotheken hinzu:
  - Amazon Kinesis Client Library (KCL)
  - AWS-SDK
  - Apache HttpCore
  - Apache HttpClient
  - Apache Commons Lang
  - Apache Commons Logging
  - Guava (Google-Kernbibliotheken für Java)
  - Jackson Annotations
  - Jackson Core
  - Jackson Databind
  - Jackson Dataformat: CBOR
  - Joda Time
4. Je nach IDE wird das Projekt möglicherweise automatisch erstellt. Wenn nicht, erstellen Sie es mit den für Ihre IDE erforderlichen Schritten.

Wenn Sie diese Schritte erfolgreich abgeschlossen haben, können Sie zum nächsten Abschnitt wechseln, [the section called “Schritt 4: Produzent implementieren”](#). Wenn der Build Fehler erzeugt, müssen Sie diese untersuchen und beheben, bevor Sie fortfahren.

## Nächste Schritte

### Schritt 4: Produzent implementieren

Die Anwendung im [Tutorial: Verarbeiten von Wertpapier-Echtzeitdaten mit KPL und KCL 1.x](#) verwendet das reale Szenario einer Überwachung des Wertpapierhandels. Im Folgenden wird kurz erläutert, wie dieses Szenario zum Produzenten und der unterstützenden Codestruktur zugeordnet wird.

Überprüfen Sie die folgenden Informationen in Bezug auf den Quellcode.

## StockTrade-Klasse

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

## Stream-Datensatz

Ein Stream ist eine Sequenz von Datensätzen. Ein Datensatz ist die Serialisierung einer `StockTrade`-Instance im JSON-Format. Beispiele:

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

## StockTradeGenerator-Klasse

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

## StockTradesWriter-Klasse

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

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

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

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

```
private static void sendStockTrade(StockTrade trade, AmazonKinesis kinesisClient,
String streamName) {
    byte[] bytes = trade.toJsonAsBytes();
    // The bytes could be null if there is an issue with the JSON serialization by
the Jackson JSON library.
    if (bytes == null) {
        LOG.warn("Could not get JSON bytes for stock trade");
        return;
    }

    LOG.info("Putting trade: " + trade.toString());
    PutRecordRequest putRecord = new PutRecordRequest();
    putRecord.setStreamName(streamName);
    // We use the ticker symbol as the partition key, explained in the Supplemental
Information section below.
    putRecord.setPartitionKey(trade.getTickerSymbol());
    putRecord.setData(ByteBuffer.wrap(bytes));

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

Beachten Sie die folgende Code-Struktur:

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

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

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

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

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

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

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

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

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

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

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

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

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

Der Grund besteht darin, dass eine Kinesis-Data-Streams-put-Operation aufgrund eines Netzwerkfehlers fehlschlagen kann oder gedrosselt wird, weil die Durchsatzgrenze des Streams erreicht wird. Sie sollten sich Ihre Wiederholungsrichtlinie für put-Operationen sorgfältig überlegen, um Datenverluste zu vermeiden, beispielsweise durch eine einfache Wiederholung.

- Eine Statusprotokollierung ist hilfreich, wenn auch optional:

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

Der hier gezeigte Produzent verwendet die API-Funktionalität von Kinesis Data Streams für einzelne Datensätze `PutRecord`. In der Praxis ist es oft effizienter, die Eignung von `PutRecords` für mehrere Datensätze zu nutzen und mehrere Datensatzstapel gleichzeitig zu senden, wenn ein Produzent viele Datensätze erstellt. Weitere Informationen finden Sie unter [Hinzufügen von Daten zu einem Stream](#).

So führen Sie den Produzenten aus

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

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

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

Die Ausgabe sollte folgendermaßen oder ähnlich aussehen:

```
Feb 16, 2015 3:53:00 PM
com.amazonaws.services.kinesis.samples.stocktrades.writer.StockTradesWriter
  sendStockTrade
INFO: Putting trade: ID 8: SELL 996 shares of BUD for $124.18
Feb 16, 2015 3:53:00 PM
com.amazonaws.services.kinesis.samples.stocktrades.writer.StockTradesWriter
  sendStockTrade
INFO: Putting trade: ID 9: BUY 159 shares of GE for $20.85
Feb 16, 2015 3:53:01 PM
com.amazonaws.services.kinesis.samples.stocktrades.writer.StockTradesWriter
  sendStockTrade
INFO: Putting trade: ID 10: BUY 322 shares of WMT for $90.08
```

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

## Nächste Schritte

### [Schritt 5: Konsument implementieren](#)

## Schritt 5: Konsument implementieren

Die Konsumentenanzwendung in [Tutorial: Verarbeiten von Wertpapier-Echtzeitdaten mit KPL und KCL 1.x](#) verarbeitet kontinuierlich den in [erstellten Stream](#) mit Wertpapiertransaktionen. Sie gibt dann für jede Minute die beliebtesten Aktien aus, die gekauft und verkauft wurden. Die Anwendung setzt auf Kinesis Client Library (KCL) auf, die viele der mühsamen Arbeiten einer Verbraucheranzwendung übernimmt. Weitere Informationen finden Sie unter [Entwicklung von KCL 1.x-Verbrauchern](#).

Überprüfen Sie die folgenden Informationen in Bezug auf den Quellcode.

### StockTradesProcessor-Klasse

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

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

### StockTradeRecordProcessor-Klasse

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

Wie an den Namen zu erkennen ist, werden `initialize` und `shutdown` von der Kinesis Client Library verwendet, um dem Datensatzprozessor mitzuteilen, wann er mit dem Empfang von Datensätzen beginnen bzw. wann er diesen stoppen soll, sodass er entsprechende anwendungsspezifische Einrichtungs- und Beendigungsaufgaben ausführen kann. Der Code hierfür wird für Sie bereitgestellt. Die wesentliche Verarbeitung erfolgt mit der `processRecords`

Methode, die wiederum `processRecord` für die einzelnen Datensätze nutzt. Die letztgenannte Methode wird als nahezu leeres Code-Skelett bereitgestellt, das im nächsten Schritt näher erläutert und von Ihnen implementiert wird.

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

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

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

Diese Methode verwendet für das Checkpointing und die Berichterstellung ein Intervall von 60 Sekunden. Weitere Informationen zum Checkpointing finden Sie unter [Weitere Informationen zum Konsumenten](#).

## StockStats-Klasse

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

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

Diese Klasse verfolgt die beliebtesten Wertpapiere, indem kontinuierlich die Anzahl der Handelstransaktionen für jedes Wertpapier sowie die maximale Anzahl gezählt werden. Sie aktualisiert diese Werte, sobald eine neue Handelstransaktion empfangen wird.

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



## So implementieren Sie den Konsumenten

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

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

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

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

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

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

## So führen Sie den Konsumenten aus

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

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

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

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

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

## Weitere Informationen zum Konsumenten

Wenn Sie mit den Vorteilen der Kinesis Client Library vertraut sind, die unter [Entwicklung von KCL 1.x-Verbrauchern](#) und anderswo erörtert werden, fragen Sie sich möglicherweise, warum Sie sie hier nutzen sollten. Obwohl Sie für die Verarbeitung nur einen einzelnen Shard-Stream und eine einzelne Konsumenten-Instance benötigen, ist es immer noch einfacher, den Konsumenten unter Verwendung der KCL zu implementieren. Vergleichen Sie die Implementierungsschritte im Produzentenabschnitt mit denen für den Konsumenten und Sie werden feststellen, dass die Implementierung eines Konsumenten vergleichsweise einfach ist. Dies liegt hauptsächlich an den von der KCL bereitgestellten Services.

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

Der Begriff Checkpointing bezeichnet das Aufzeichnen der Stelle im Stream, bis zu der die Datensätze bislang verbraucht und verarbeitet wurden, damit im Falle eines Absturzes der Anwendung das Lesen des Streams an dieser Stelle fortgesetzt werden kann. Das Checkpointing sowie zugehörige Entwurfsmuster und bewährte Methoden sind nicht Gegenstand dieses Kapitels. Es ist jedoch etwas, das Ihnen in Produktionsumgebungen begegnen könnte.

Wie Sie in erfahren haben, verwenden die put-Operationen in der API für Kinesis Data Streams einen Partitionsschlüssel als Eingabe. Kinesis Data Streams verwendet einen Partitionsschlüssel zum

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

In dieser Anwendung findet keine intensive Verarbeitung der Datensätze durch den Konsumenten statt. Deshalb können Sie einen Shard verwenden und die Verarbeitung im selben Thread wie der KCL-Thread durchführen. In der Praxis sollten Sie allerdings zunächst die Anzahl der Shards skalieren. Gelegentlich kann es vorkommen, dass Sie die Verarbeitung an einen anderen Thread übergeben oder einen Thread-Pool nutzen möchten, wenn eine intensive Datensatzverarbeitung ansteht. Dadurch kann die KCL die neuen Datensätze schneller abrufen, während die anderen Threads parallel dazu die Datensätze verarbeiten. Das Multithread-Design ist sehr komplex und bedarf ausgeklügelter Techniken. Deshalb ist die Erhöhung Ihrer Shard-Anzahl in der Regel die einfachste und effektivste Möglichkeit einer Skalierung.

## Nächste Schritte

### [Schritt 6: \(optional\) Konsument erweitern](#)

## Schritt 6: (optional) Konsument erweitern

Möglicherweise ist die Anwendung [Tutorial: Verarbeiten von Wertpapier-Echtzeitdaten mit KPL und KCL 1.x](#) bereits ausreichend für Ihre Zwecke. Dieser optionale Abschnitt zeigt, wie Sie den Konsumentencode erweitern können, um einem komplexeren Szenario gerecht zu werden.

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

So erweitern Sie den Konsumenten

1. Fügen Sie neue Instance-Variablen hinzu:

```
// Ticker symbol of the stock that had the largest quantity of shares sold
private String largestSellOrderStock;
// Quantity of shares for the largest sell order trade
private long largestSellOrderQuantity;
```

## 2. Fügen Sie folgenden Code zu `addStockTrade` hinzu:

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

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

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

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

```
***** Shard shardId-000000000001 stats for last 1 minute *****
Most popular stock being bought: WMT, 27 buys.
Most popular stock being sold: PTR, 14 sells.
Largest sell order: 996 shares of BUD.
*****
```

## Nächste Schritte

### [Schritt 7: Abschluss](#)

## Schritt 7: Abschluss

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

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

So löschen Sie den Stream und die Tabelle

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

## Übersicht

Für die Verarbeitung großer Datenmengen in nahezu Echtzeit muss weder ein magischer Code geschrieben noch eine umfangreiche Infrastruktur entwickelt werden. Das Ganze ist so einfach wie das Schreiben einer Logik für die Verarbeitung einer kleinen Datenmenge (wie das Schreiben von `processRecord(Record)`), die dann zu Skalierzwecken mit Kinesis Data Streams kombiniert wird, damit auch große Mengen an Streaming-Daten verarbeitet werden können. Sie müssen sich keine Gedanken über die Skalierung der Verarbeitung machen, da Kinesis Data Streams das für Sie übernimmt. Sie müssen lediglich Ihre Streaming-Datensätze an Kinesis Data Streams senden und eine Logik für die Verarbeitung neu empfangener Datensätze schreiben.

Nachfolgend einige mögliche Erweiterungen für diese Anwendung.

### Shard-übergreifende Aggregation

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

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

## Skalierung

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

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

Da ein Stream kontinuierlich verarbeitet wird, kann seine Ausgabe an andere Ziele gesendet werden. AWS bietet [Konnektoren](#) zur Integration von Kinesis Data Streams mit anderen AWS-Services und Tools von Drittanbietern.

## Nächste Schritte

- Weitere Informationen zur Verwendung der API-Operationen von Kinesis Data Streams finden Sie unter [Entwicklung von Produzenten mithilfe der API für Amazon Kinesis Data Streams mit der AWS SDK for Java](#), [Entwickeln benutzerdefinierter Verbraucher mit gemeinsam genutztem Durchsatz unter Verwendung von AWS SDK for Java](#), und [Erstellen und Verwalten von Streams](#).
- Weitere Informationen zur Kinesis Client Library finden Sie unter [Entwicklung von KCL 1.x-Verbrauchern](#).
- Weitere Informationen zur Optimierung Ihrer Anwendung finden Sie unter [Fortgeschrittene Themen](#).

## Tutorial: Analysieren von Wertpapierdaten in Echtzeit mit Managed Service für Apache Flink

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

Mit Managed Service für Apache Flink für Flink-Anwendungen können Sie Java oder Scala verwenden, um Streaming-Daten zu verarbeiten und zu analysieren. Der Service ermöglicht die

Erstellung und Ausführung von Java- oder Scala-Code für Streaming-Quellen zum Durchführen von Zeitreihenanalysen, Füllen von Echtzeit-Dashboards und Erstellen von Echtzeitmetriken.

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

#### Important

Nachdem Sie zwei Datenströme und eine Anwendung erstellt haben, fallen für die Nutzung von Kinesis Data Streams und Managed Service für Apache Flink mit Ihrem Konto Nominalgebühren an, da sie nicht für das kostenlose Kontingent für AWS infrage kommen. Wenn Sie mit dieser Anwendung fertig sind, sollten Sie Ihre AWS-Ressourcen löschen, damit keine weiteren Gebühren anfallen.

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

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

#### Aufgaben

- [Voraussetzungen für das Fertigstellen der Übungen](#)
- [Schritt 1: Einrichten eines AWS-Kontos und Erstellen eines Administratorbenutzers](#)
- [Schritt 2: Einrichten der AWS Command Line Interface \(AWS CLI\)](#)
- [Schritt 3: Erstellen und Ausführen eines Managed Service für Apache Flink für die Flink-Anwendung](#)

## Voraussetzungen für das Fertigstellen der Übungen

Zur Durchführung der Schritte in dieser Anleitung benötigen Sie Folgendes:

- [Java Development Kit \(JDK\), Version 8](#). Legen Sie die JAVA\_HOME Umgebungsvariable so fest, dass sie auf Ihren JDK-Installationspeicherort weist.
- Wir empfehlen die Verwendung einer Entwicklungsumgebung (wie [Eclipse Java Neon](#) oder [IntelliJ Idea](#)), um Ihre Anwendung zu entwickeln und zu kompilieren.
- [Git-Client](#). Installieren Sie den Git-Client, wenn Sie dies noch nicht getan haben.
- [Apache Maven-Compiler-Plugin](#). Maven muss sich in Ihrem Arbeitspfad befinden. Zum Testen Ihrer Apache Maven-Installation geben Sie Folgendes ein:

```
$ mvn -version
```

Um zu beginnen, gehen Sie zu [Schritt 1: Einrichten eines AWS-Kontos und Erstellen eines Administratorbenutzers](#).

## Schritt 1: Einrichten eines AWS-Kontos und Erstellen eines Administratorbenutzers

Führen Sie die folgenden Aufgaben aus, bevor Sie Amazon Managed Service für Apache Flink für Flink-Anwendungen zum ersten Mal verwenden:

1. [Registrieren bei AWS](#)
2. [Erstellen eines IAM-Benutzers](#)

### Registrieren bei AWS

Bei der Registrierung für Amazon Web Services (AWS) wird Ihr AWS-Konto automatisch für alle Dienste in AWS registriert, einschließlich Amazon Managed Service für Apache Flink. Berechnet werden Ihnen aber nur die Services, die Sie nutzen.

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

Wenn Sie bereits ein AWS-Konto haben, wechseln Sie zur nächsten Aufgabe. Wenn Sie noch kein AWS-Konto haben, führen Sie die folgenden Schritte aus, um ein Konto zu erstellen.



## So erstellen Sie ein AWS-Konto

1. Öffnen Sie <https://portal.aws.amazon.com/billing/signup>.
2. Folgen Sie den Online-Anweisungen.

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

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

Notieren Sie Ihre AWS-Konto-ID. Sie benötigen sie im nächsten Schritt.

## Erstellen eines IAM-Benutzers

Services in AWS, wie z. B. Amazon Managed Service für Apache Flink, erfordern beim Zugriff die Eingabe von Anmeldeinformationen. Der Service kann feststellen, ob Sie über die Berechtigung für den Zugriff auf die Ressourcen im Besitz dieses Service verfügen. Die AWS Management Console erfordert, dass Sie Ihr Passwort eingeben.

Sie können für Ihr AWS-Konto Zugriffsschlüssel erstellen, um auf die AWS Command Line Interface (AWS CLI) oder die API zuzugreifen. Wir raten Ihnen jedoch davon ab, mittels der Anmeldeinformationen für Ihr AWS-Konto auf AWS zuzugreifen. Stattdessen empfehlen wir, AWS Identity and Access Management (IAM) zu verwenden. Erstellen Sie einen IAM-Benutzer und fügen Sie den Benutzer zu einer IAM-Gruppe mit Administrator-Berechtigungen hinzu. Anschließend gewähren Sie dem von Ihnen erstellten IAM-Benutzer administrative Berechtigungen. Danach können Sie mithilfe einer speziellen URL und der Anmeldeinformationen des IAM-Benutzers auf AWS zugreifen.

Wenn Sie sich zwar bei AWS angemeldet, aber für sich selbst keinen IAM-Benutzer erstellt haben, können Sie mithilfe der IAM-Konsole einen Benutzer erstellen.

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

So erstellen Sie eine Gruppe für Administratoren:

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

Ihre neue Gruppe wird unter Group Name aufgeführt.

Zum Erstellen eines IAM-Benutzers für sich selbst, fügen Sie ihn der Administratorengruppe hinzu und erstellen Sie ein Passwort

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


So melden Sie sich als neuer IAM-Benutzer an

1. Melden Sie sich von AWS Management Console ab.
2. Verwenden Sie das folgende URL-Format zum Anmelden bei der Konsole:

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

Die `aws_account_number` ist Ihre AWS-Konto-ID ohne Bindestriche. Wenn Ihre AWS-Konto-ID beispielsweise 1234-5678-9012 lautet, ersetzen Sie `aws_account_number` mit **123456789012**. Weitere Informationen darüber, wie Sie Ihre Kontonummer finden, finden Sie unter [Ihre AWS-Konto-ID und der dazugehörige Alias](#) im IAM-Benutzerhandbuch.

3. Geben Sie den IAM-Benutzernamen und das von Ihnen soeben erstellte Passwort ein. Nachdem Sie sich angemeldet haben, wird in der Navigationsleiste `your_user_name @ your_aws_account_id` angezeigt.

 Note

Wenn Sie nicht möchten, dass die URL für Ihre Anmeldeseite Ihre AWS-Konto-ID enthält, können Sie einen Konto-Alias erstellen.

So erstellen oder entfernen Sie einen Konto-Alias

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

Nach dem Erstellen eines Konto-Alias verwenden Sie die folgende URL, um sich anzumelden:

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

Um den Anmeldelink der IAM-Benutzer Ihres Kontos zu verifizieren, öffnen Sie die IAM-Konsole und prüfen dies im Dashboard unter IAM users sign-in link.

Weitere Informationen zu IAM finden Sie unter:

- [AWS Identity and Access Management \(IAM\)](#)
- [Erste Schritte](#)
- [IAM Benutzerhandbuch](#)

## Nächster Schritt

### [Schritt 2: Einrichten der AWS Command Line Interface \(AWS CLI\)](#)

## Schritt 2: Einrichten der AWS Command Line Interface (AWS CLI)

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

### Note

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

### Note

Wenn Sie die AWS CLI bereits installiert haben, müssen Sie möglicherweise ein Upgrade durchführen, um die neueste Funktionalität zu erhalten. Weitere Informationen finden Sie unter [Installieren der AWS-Befehlszeilenschnittstelle](#) im AWS Command Line Interface-Benutzerhandbuch. Zum Überprüfen der Version der AWS CLI führen Sie den folgenden Befehl aus:

```
aws --version
```

Die Übungen in diesem Tutorial erfordern die folgende AWS CLI-Version oder höher:

```
aws-cli/1.16.63
```

Um das AWS CLI einzurichten

1. Herunterladen und Konfigurieren von AWS CLI. Eine Anleitung finden Sie unter den folgenden Themen im AWS Command Line Interface-Benutzerhandbuch:
  - [Installieren des AWS Command Line Interface](#)
  - [Konfigurieren von AWS CLI](#)

2. Fügen Sie ein benanntes Profil für den Administratorbenutzer in der AWS CLI-Konfigurationsdatei hinzu. Verwenden Sie dieses Profil beim Ausführen von AWS CLI-Befehlen. Weitere Informationen zu benannten Profilen finden Sie unter [Benannte Profile](#) im AWS Command Line Interface Benutzerhandbuch.

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

Eine Liste der verfügbaren AWS-Regionen finden Sie unter [AWSRegionen und Endpunkte](#) in Allgemeine Amazon Web Services-Referenz.

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

```
aws help
```

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

## Nächster Schritt

### [Schritt 3: Erstellen und Ausführen eines Managed Service für Apache Flink für die Flink-Anwendung](#)

## Schritt 3: Erstellen und Ausführen eines Managed Service für Apache Flink für die Flink-Anwendung

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

Dieser Abschnitt enthält die folgenden Schritte:

- [Erstellen von zwei Amazon Kinesis Data Streams](#)
- [Schreiben Sie Beispieldatensätze in den Eingabe-Stream](#)
- [Herunterladen und Überprüfen des Apache Flink-Streaming-Java-Codes](#)
- [Kompilieren des Anwendungscodes](#)

- [Hochladen des Apache Flink-Streaming-Java-Codes](#)
- [Erstellen und führen Sie die Anwendung Managed Service für Apache Flink aus](#)

## Erstellen von zwei Amazon Kinesis Data Streams

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

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

So erstellen Sie die Daten-Streams (AWS CLI)

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

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

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

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

## Schreiben Sie Beispieldatensätze in den Eingabe-Stream

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

**Note**

Dieser Abschnitt erfordert [AWS SDK for Python \(Boto\)](#).

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

```
import datetime
import json
import random
import boto3

STREAM_NAME = "ExampleInputStream"

def get_data():
    return {
        "EVENT_TIME": datetime.datetime.now().isoformat(),
        "TICKER": random.choice(["AAPL", "AMZN", "MSFT", "INTC", "TBV"]),
        "PRICE": round(random.random() * 100, 2),
    }

def generate(stream_name, kinesis_client):
    while True:
        data = get_data()
        print(data)
        kinesis_client.put_record(
            StreamName=stream_name, Data=json.dumps(data),
            PartitionKey="partitionkey"
        )

if __name__ == "__main__":
    generate(STREAM_NAME, boto3.client("kinesis"))
```

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

```
$ python stock.py
```

## Herunterladen und Überprüfen des Apache Flink-Streaming-Java-Codes

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

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

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

2. Navigieren Sie zum GettingStarted Verzeichnis .

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

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

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

## Kompilieren des Anwendungscodes

In diesem Abschnitt verwenden Sie den Apache Maven-Compiler zum Erstellen des Java-Codes für die Anwendung. Weitere Informationen zum Installieren von Apache Maven und des Java Development Kit (JDK) finden Sie unter [Voraussetzungen für das Fertigstellen der Übungen](#).

Ihre Java-Anwendung erfordert die folgenden Komponenten:

- Eine [Projektobjektmodell \(pom.xml\)](#)-Datei. Diese Datei enthält Informationen über die Konfiguration und Abhängigkeiten der Anwendung, einschließlich der Bibliotheken des Managed Service für Apache Flink für Flink-Anwendungen.
- Eine `main`-Methode, die die Logik der Anwendung enthält.



**Note**

Zur Nutzung des Kinesis-Konnektors für die folgende Anwendung müssen Sie den Quellcode für den Konnektor herunterladen und ihn erstellen. Einzelheiten dazu finden Sie in der [Apache-Flink-Dokumentation](#).

So erstellen und kompilieren Sie den Anwendungscode

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

```
package com.amazonaws.services.kinesisanalytics;

import com.amazonaws.services.kinesisanalytics.runtime.KinesisAnalyticsRuntime;
import org.apache.flink.api.common.serialization.SimpleStringSchema;
import org.apache.flink.streaming.api.datastream.DataStream;
import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
import org.apache.flink.streaming.connectors.kinesis.FlinkKinesisConsumer;
import org.apache.flink.streaming.connectors.kinesis.FlinkKinesisProducer;
import
    org.apache.flink.streaming.connectors.kinesis.config.ConsumerConfigConstants;

import java.io.IOException;
import java.util.Map;
import java.util.Properties;

public class StreamingJob {

    private static final String region = "us-east-1";
    private static final String inputStreamName = "ExampleInputStream";
    private static final String outputStreamName = "ExampleOutputStream";

    private static DataStream<String>
    createSourceFromStaticConfig(StreamExecutionEnvironment env) {
```

```
        Properties inputProperties = new Properties();
        inputProperties.setProperty(ConsumerConfigConstants.AWS_REGION, region);

inputProperties.setProperty(ConsumerConfigConstants.STREAM_INITIAL_POSITION,
"LATEST");

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

    private static DataStream<String>
createSourceFromApplicationProperties(StreamExecutionEnvironment env)
        throws IOException {
        Map<String, Properties> applicationProperties =
KinesisAnalyticsRuntime.getApplicationProperties();
        return env.addSource(new FlinkKinesisConsumer<>(inputStreamName, new
SimpleStringSchema(),
                applicationProperties.get("ConsumerConfigProperties")));
    }

    private static FlinkKinesisProducer<String> createSinkFromStaticConfig() {
        Properties outputProperties = new Properties();
        outputProperties.setProperty(ConsumerConfigConstants.AWS_REGION, region);
        outputProperties.setProperty("AggregationEnabled", "false");

        FlinkKinesisProducer<String> sink = new FlinkKinesisProducer<>(new
SimpleStringSchema(), outputProperties);
        sink.setDefaultStream(outputStreamName);
        sink.setDefaultPartition("0");
        return sink;
    }

    private static FlinkKinesisProducer<String>
createSinkFromApplicationProperties() throws IOException {
        Map<String, Properties> applicationProperties =
KinesisAnalyticsRuntime.getApplicationProperties();
        FlinkKinesisProducer<String> sink = new FlinkKinesisProducer<>(new
SimpleStringSchema(),
                applicationProperties.get("ProducerConfigProperties"));

        sink.setDefaultStream(outputStreamName);
        sink.setDefaultPartition("0");
        return sink;
    }
}
```

```
public static void main(String[] args) throws Exception {
    // set up the streaming execution environment
    final StreamExecutionEnvironment env =
StreamExecutionEnvironment.getExecutionEnvironment();

    /*
     * if you would like to use runtime configuration properties, uncomment the
     * lines below
     * DataStream<String> input = createSourceFromApplicationProperties(env);
     */

    DataStream<String> input = createSourceFromStaticConfig(env);

    /*
     * if you would like to use runtime configuration properties, uncomment the
     * lines below
     * input.addSink(createSinkFromApplicationProperties())
     */

    input.addSink(createSinkFromStaticConfig());

    env.execute("Flink Streaming Java API Skeleton");
}
}
```

Beachten Sie die folgenden Informationen zum vorherigen Codebeispiel:

- Diese Datei enthält die `main`-Methode, die die Funktionalität der Anwendung definiert.
  - Ihre Anwendung erstellt Quell- und Senkenkonnektoren für den Zugriff auf externe Ressourcen, indem ein `StreamExecutionEnvironment`-Objekt verwendet wird.
  - Die Anwendung erstellt Quell- und Senkenkonnektoren mit statischen Eigenschaften. Zum Verwenden dynamischer Anwendungseigenschaften verwenden Sie die Methoden `createSourceFromApplicationProperties` und `createSinkFromApplicationProperties`, um die Konnektoren zu erstellen. Diese Methoden lesen die Eigenschaften der Anwendung zum Konfigurieren der Konnektoren.
3. Zum Verwenden Ihres Anwendungscodes kompilieren und packen Sie ihn in eine JAR-Datei. Sie können Ihren Code auf zwei Arten kompilieren und packen:

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

```
mvn package
```

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

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

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

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

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

## Hochladen des Apache Flink-Streaming-Java-Codes

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

So laden Sie den Anwendungscode hoch

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

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

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

## Erstellen und führen Sie die Anwendung Managed Service für Apache Flink aus

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

### Note

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

## Themen

- [Erstellen und Ausführen der Anwendung \(Konsole\)](#)
- [Erstellen und Ausführen der Anwendung \(AWS CLI\)](#)

## Erstellen und Ausführen der Anwendung (Konsole)

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

### Erstellen Sie die Anwendung

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

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

#### Note

Beim Erstellen einer Anwendung von Managed Service für Apache Flink für Flink mit der Konsole haben Sie die Möglichkeit, eine IAM-Rolle und -Richtlinie für Ihre Anwendung erstellen zu lassen. Ihre Anwendung verwendet diese Rolle und Richtlinie für den Zugriff auf ihre abhängigen Ressourcen. Diese IAM-Ressourcen werden unter Verwendung Ihres Anwendungsnamens und der Region wie folgt benannt:

- Richtlinie: `kinesis-analytics-service-MyApplication-us-west-2`
- Rolle: `kinesis-analytics-MyApplication-us-west-2`

## Bearbeiten der IAM-Richtlinie

Bearbeiten Sie die IAM-Richtlinie zum Hinzufügen von Berechtigungen für den Zugriff auf die Kinesis-Datenströme.

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

4. Fügen Sie den markierten Abschnitt der folgenden Beispielrichtlinie der Richtlinie hinzu. Ersetzen Sie die beispielhaften Konto-IDs (*012345678901*) mit Ihrer Konto-ID.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ReadCode",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:GetObjectVersion"
      ],
      "Resource": [
        "arn:aws:s3:::ka-app-code-username/java-getting-started-1.0.jar"
      ]
    },
    {
      "Sid": "ListCloudwatchLogGroups",
      "Effect": "Allow",
      "Action": [
        "logs:DescribeLogGroups"
      ],
      "Resource": [
        "arn:aws:logs:us-west-2:012345678901:log-group:*"
      ]
    },
    {
      "Sid": "ListCloudwatchLogStreams",
      "Effect": "Allow",
      "Action": [
        "logs:DescribeLogStreams"
      ],
      "Resource": [
        "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:*"
      ]
    },
    {
      "Sid": "PutCloudwatchLogs",
      "Effect": "Allow",
      "Action": [
        "logs:PutLogEvents"
      ]
    }
  ]
}
```

```

    ],
    "Resource": [
        "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:kinesis-analytics-log-stream"
    ]
  },
  {
    "Sid": "ReadInputStream",
    "Effect": "Allow",
    "Action": "kinesis:*",
    "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
  },
  {
    "Sid": "WriteOutputStream",
    "Effect": "Allow",
    "Action": "kinesis:*",
    "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleOutputStream"
  }
]
}

```

## Konfigurieren der Anwendung

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



Schlüssel	Wert
<b>flink.inputstream.initpos</b>	<b>LATEST</b>
<b>aws:region</b>	<b>us-west-2</b>
<b>AggregationEnabled</b>	<b>false</b>

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

#### Note

Wenn Sie die CloudWatch Protokollierung aktivieren, erstellt Managed Service für Apache Flink eine Protokollgruppe und einen Protokollstream für Sie. Die Namen dieser Ressourcen lauten wie folgt:

- Protokollgruppe: /aws/kinesis-analytics/MyApplication
- Protokollstream: kinesis-analytics-log-stream

### Ausführen der Anwendung

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

### Stoppen der Anwendung

Wählen Sie auf der MyApplication Seite Stoppen aus. Bestätigen Sie die Aktion.

### Aktualisieren der Anwendung

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

der JAR-Anwendungsdatei aktualisieren. Außerdem können Sie die JAR-Anwendungsdatei erneut aus dem Amazon-S3-Bucket laden, wenn Sie den Anwendungscode aktualisieren müssen.

Wählen Sie auf der MyApplication Seite Konfigurieren aus. Aktualisieren Sie die Anwendungseinstellungen und klicken Sie auf Aktualisieren.

### Erstellen und Ausführen der Anwendung (AWS CLI)

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

### Erstellen einer Berechtigungsrichtlinie

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

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

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "S3",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:GetObjectVersion"
      ],
      "Resource": [
        "arn:aws:s3:::ka-app-code-username",
        "arn:aws:s3:::ka-app-code-username/*"
      ]
    },
    {
```

```
        "Sid": "ReadInputStream",
        "Effect": "Allow",
        "Action": "kinesis:*",
        "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
    },
    {
        "Sid": "WriteOutputStream",
        "Effect": "Allow",
        "Action": "kinesis:*",
        "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleOutputStream"
    }
]
}
```

step-by-step Anweisungen zum Erstellen einer Berechtigungsrichtlinie finden Sie unter [Tutorial: Erstellen und Anfügen Ihrer ersten vom Kunden verwalteten Richtlinie](#) im IAM-Benutzerhandbuch.

#### Note

Um auf andere AWS-Services zuzugreifen, können Sie AWS SDK for Java verwenden. Managed Service für Apache Flink setzt die vom SDK benötigten Anmeldeinformationen automatisch auf die der IAM-Rolle für die Dienstauführung, die mit Ihrer Anwendung verknüpft ist. Es sind keine weiteren Schritte erforderlich.

## Erstellen einer IAM-Rolle

In diesem Abschnitt erstellen Sie eine IAM-Rolle, die die Anwendung von Managed Service für Apache Flink für Flink annehmen kann, um einen Quell-Stream zu lesen und in den Senken-Stream zu schreiben.

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

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

## So erstellen Sie eine IAM-Rolle

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

Wählen Sie Weiter: Berechtigungen aus.

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

Jetzt haben Sie eine neue IAM-Rolle mit dem Namen `KA-stream-rw-role` erstellt. Im nächsten Schritt aktualisieren Sie die Vertrauens- und Berechtigungsrichtlinien für die Rolle.

6. Fügen Sie die Berechtigungsrichtlinie der Rolle an.

### Note

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

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

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

step-by-step Anweisungen zum Erstellen einer Rolle finden Sie unter [Erstellen einer IAM-Rolle \(Konsole\)](#) im IAM-Benutzerhandbuch.

Erstellen Sie die Anwendung Managed Service für Apache Flink

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

```
{
  "ApplicationName": "test",
  "ApplicationDescription": "my java test app",
  "RuntimeEnvironment": "FLINK-1_6",
  "ServiceExecutionRole": "arn:aws:iam::012345678901:role/KA-stream-rw-role",
  "ApplicationConfiguration": {
    "ApplicationCodeConfiguration": {
      "CodeContent": {
        "S3ContentLocation": {
          "BucketARN": "arn:aws:s3:::ka-app-code-username",
          "FileKey": "java-getting-started-1.0.jar"
        }
      },
      "CodeContentType": "ZIPFILE"
    },
    "EnvironmentProperties": {
      "PropertyGroups": [
        {
          "PropertyGroupId": "ProducerConfigProperties",
          "PropertyMap" : {
            "flink.stream.initpos" : "LATEST",
            "aws.region" : "us-west-2",
            "AggregationEnabled" : "false"
          }
        },
        {
          "PropertyGroupId": "ConsumerConfigProperties",
          "PropertyMap" : {
            "aws.region" : "us-west-2"
          }
        }
      ]
    }
  }
}
```

```
}
  }
]
}
}
```

2. Führen Sie die [CreateApplication](#)-Aktion mit der vorherigen Anforderung zum Erstellen der Anwendung aus:

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

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

### Starten der Anwendung

In diesem Abschnitt verwenden Sie die [StartApplication](#)-Aktion, um die Anwendung zu starten.

So starten Sie die Anwendung

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

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

2. Führen Sie die [StartApplication](#)-Aktion mit der vorherigen Anforderung zum Starten der Anwendung aus:

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

Die Anwendung wird jetzt ausgeführt. Sie können die Metriken von Managed Service für Apache Flink in der Amazon- CloudWatch Konsole überprüfen, um zu überprüfen, ob die Anwendung funktioniert.

## Stoppen der Anwendung

In diesem Abschnitt verwenden Sie die [StopApplication](#)-Aktion, um die Anwendung zu stoppen.

So stoppen Sie die Anwendung

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

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

2. Führen Sie die [StopApplication](#)-Aktion mit der folgenden Anforderung zum Stoppen der Anwendung aus:

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

Die Anwendung wird nun gestoppt.

## Anleitung: Verwenden von AWS Lambda mit Amazon Kinesis Data Streams

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

Eine detaillierte Schritt-für-Schritt-Anleitung finden Sie unter [Anleitung: Verwenden von AWS Lambda mit Amazon Kinesis](#).

### Note

In diesem Tutorial wird davon ausgegangen, dass Sie über Kenntnisse zu den grundlegenden Lambda-Operationen und der Lambda-Konsole verfügen. Sofern noch nicht

geschehen, befolgen Sie die Anweisungen unter [Erste Schritte mit AWS Lambda](#), um Ihre erste Lambda-Funktion zu erstellen.

## AWS-Streaming-Datenlösung für Amazon Kinesis

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

Jede Lösung enthält die folgenden Komponenten:

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

Die Lösung finden Sie hier: [Streaming-Datenlösung für Amazon Kinesis](#)



# Erstellen und Verwalten von Streams

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

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

## Themen

- [Auswahl des Datenstrom-Kapazitätsmodus](#)
- [Erstellen eines Streams über die AWS-Management-Konsole](#)
- [Erstellen eines Streams über die APIs](#)
- [Aktualisieren eines Streams](#)
- [Auflisten von Streams](#)
- [Auflisten von Shards](#)
- [Löschen eines Streams](#)
- [Resharding eines Streams](#)
- [Ändern des Zeitraums der Datenaufbewahrung](#)
- [Tagging von Streams in Amazon Kinesis Data Streams](#)

## Auswahl des Datenstrom-Kapazitätsmodus

### Themen

- [Was ist ein Datenstream-Kapazitätsmodus?](#)
- [On-Demand-Modus](#)
- [Modus bereitgestellter Kapazität](#)
- [Zwischen Kapazitätsmodi wechseln](#)

## Was ist ein Datenstream-Kapazitätsmodus?

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

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

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

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

### On-Demand-Modus

Datenströme im On-Demand-Modus erfordern keine Kapazitätsplanung und werden automatisch skaliert, sodass sie einen Schreib- und Lesedurchsatz von mehreren Gigabyte pro Minute verarbeiten können. Der On-Demand-Modus vereinfacht die Aufnahme und Speicherung großer Datenmengen bei niedriger Latenz, da er die Bereitstellung und Verwaltung von Servern, Speicher oder Durchsatz überflüssig macht. Sie können Milliarden von Datensätzen pro Tag aufnehmen, ohne dass der Betrieb beeinträchtigt wird.

Der On-Demand-Modus ist ideal, um den Anforderungen eines stark variablen und unvorhersehbaren Anwendungsverkehrs gerecht zu werden. Sie müssen diese Workloads nicht mehr für

Spitzenkapazitäten bereitstellen, was bei geringer Auslastung zu höheren Kosten führen kann. Der On-Demand-Modus eignet sich für Workloads mit unvorhersehbaren und stark variablen Datenverkehrsmustern.

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

Sie können einen neuen Datenstrom im On-Demand-Modus erstellen, indem Sie die Kinesis-Data-Streams-Konsole, -APIs oder CLI-Befehle verwenden.

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

Die aggregierte Lesekapazität eines Datenstroms im On-Demand-Modus steigt proportional zum Schreibdurchsatz. Auf diese Weise wird sichergestellt, dass Verbraucheranwendungen immer über einen ausreichenden Lesedurchsatz verfügen, um eingehende Daten in Echtzeit zu verarbeiten. Der Schreibdurchsatz ist mindestens doppelt so hoch wie beim Lesen von Daten mit der `GetRecords`-API. Wir empfehlen, dass Sie eine Verbraucheranwendung mit der `GetRecord`-API verwenden, damit sie genügend Spielraum hat, um aufzuholen, wenn die Anwendung sich von einer Ausfallzeit erholen muss. Es wird empfohlen, die Funktion Erweitertes Rundsenden von Kinesis Data Streams für Szenarien zu verwenden, in denen mehr als eine Verbraucheranwendung hinzugefügt werden muss. Erweitertes Rundsenden unterstützt das Hinzufügen von bis zu 20 Verbraucheranwendungen zu einem Datenstrom mithilfe der `SubscribeToShard`-API, wobei jede Verbraucheranwendung über einen eigenen Durchsatz verfügt.

## Behandlung von Ausnahmen beim Lese- und Schreibdurchsatz

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

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

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

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

## Modus bereitgestellter Kapazität

Im Bereitstellungsmodus können Sie, nachdem Sie den Datenstream erstellt haben, Ihre Shard-Kapazität mithilfe der oder der AWS Management Console API dynamisch nach oben oder unten skalieren. [UpdateShardCount](#) Sie können Aktualisierungen vornehmen, während eine Produzenten- oder Konsumenten-anwendung von Kinesis Data Streams Daten in den Stream schreibt oder aus dem Stream liest.

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

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

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

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

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

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

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

## Zwischen Kapazitätsmodi wechseln

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

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

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

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

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

## Erstellen eines Streams über die AWS-Management-Konsole

Sie können einen Stream mithilfe der Konsole von Kinesis Data Streams, der API von Kinesis Data Streams oder der AWS Command Line Interface (AWS CLI) erstellen.

So erstellen Sie einen Daten-Stream mit der Konsole

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

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

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

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

So erstellen Sie einen Stream mit der Kinesis-Data-Streams-API

- Weitere Informationen zum Erstellen eines Streams mit der API von Kinesis Data Streams finden Sie unter [Erstellen eines Streams über die APIs](#).

So erstellen Sie einen Stream mit der AWS CLI

- Weitere Informationen zum Erstellen eines Streams mit der AWS CLI finden Sie unter dem Befehl [create-stream](#).

## Erstellen eines Streams über die APIs

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

### Kinesis-Data-Streams-Client erstellen

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

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

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

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

Weitere Informationen finden Sie unter [Regionen und Endpunkte der Kinesis Data Streams](#) im Allgemeine AWS-Referenz.

## Erstellen des Streams

Nachdem Sie Ihren Kinesis-Data-Streams-Client erstellt haben, können Sie mithilfe der Konsole von Kinesis Data Streams oder programmgesteuert einen Stream erstellen, mit dem Sie anschließend arbeiten. Um einen Stream programmgesteuert zu erstellen, instanziiieren Sie ein `CreateStreamRequest`-Objekt und legen Sie einen Namen für den Stream sowie (wenn Sie den Bereitstellungsmodus verwenden möchten) die Anzahl der für den Stream zu verwendenden Shards fest.

- On-Demand-Modus:

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

- Bereitgestellt:

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

Der Streamname bezeichnet den Stream. Der Name ist auf das von der Anwendung verwendete AWS-Konto beschränkt. Er ist auch nach Region beschränkt. Das bedeutet: Zwei Streams in zwei verschiedenen AWS-Konten können denselben Namen haben und zwei Streams im selben AWS-Konto, jedoch in zwei verschiedenen Regionen, können denselben Namen haben, nicht jedoch zwei Streams auf demselben Konto und in derselben Region.

Der Durchsatz des Streams bezieht sich auf die Anzahl der Shards; für einen höheren bereitgestellten Durchsatz sind mehr Shards erforderlich. Mehr Shards erhöhen auch die Kosten, die AWS für den Stream berechnet. Weitere Informationen zum Berechnen einer ausreichenden Anzahl von Shards für Ihre Anwendung finden Sie unter [Auswahl des Datenstrom-Kapazitätsmodus](#).

Wenn das `createStreamRequest`-Objekt konfiguriert ist, erstellen Sie einen Stream, indem Sie die `createStream`-Methode auf dem Client aufrufen. Warten Sie nach dem Aufrufen von



`createStream`, bis der Stream den Status `ACTIVE` erreicht hat, bevor Sie eine Operation auf dem Stream ausführen. Rufen Sie die `describeStream`-Methode auf, um den Status des Streams zu überprüfen. Allerdings löst `describeStream` eine Ausnahme aus, wenn der Stream nicht existiert. Aus diesem Grund müssen Sie den Aufruf `describeStream` in einen `try/catch`-Block einschließen.

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

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

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

# Aktualisieren eines Streams

Sie können die Details eines Streams mithilfe der Konsole von Kinesis Data Streams, der API von Kinesis Data Streams oder der AWS CLI aktualisieren.

## Note

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

So aktualisieren Sie einen Daten-Stream mit der Konsole

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

## Important

Für jeden Datenstrom in Ihrem AWS-Konto können Sie innerhalb von 24 Stunden zweimal zwischen den Modi „On-Demand“ und „Bereitgestellt“ wechseln.

5. Um für einen Datenstrom im Bereitstellungsmodus die Anzahl der Shards zu bearbeiten, wählen Sie auf der Registerkarte Konfiguration die Option Bereitgestellte Shards bearbeiten aus und geben Sie dann eine neue Shard-Anzahl ein.
6. Wählen Sie zum Aktivieren der serverseitigen Verschlüsselung von Datensätzen die Option Edit (Bearbeiten) im Abschnitt Server-side encryption (Serverseitige Verschlüsselung) aus. Wählen Sie einen KMS-Schlüssel aus, der als Masterschlüssel für die Verschlüsselung verwendet werden soll, oder nutzen Sie den Standard-Masterschlüssel aws/kinesis, der von Kinesis verwaltet wird. Wenn Sie die Verschlüsselung für einen Stream aktivieren und einen eigenen AWS KMS-Masterschlüssel verwenden, müssen Sie sicherstellen, dass die Produzenten- und

Konsumenten Anwendungen Zugriff auf den von Ihnen verwendeten AWS KMS-Masterschlüssel haben. Informationen zum Zuweisen von Berechtigungen zu Anwendungen für den Zugriff auf einen benutzergenerierten AWS KMS-Schlüssel finden Sie unter [the section called “Berechtigungen für die Nutzung benutzergenerierter KMS-Masterschlüssel”](#).

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

## Aktualisieren eines Streams mit der API

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

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

## Aktualisieren eines Streams mit der AWS CLI

Weitere Informationen zur Aktualisierung eines Streams mit der AWS CLI finden Sie in der [Kinesis-CLI-Referenz](#).

## Auflisten von Streams

Wie im vorherigen Abschnitt beschrieben, sind Streams auf das mit den zum Instanzieren des Kinesis-Data-Streams-Clients verwendeten AWS-Anmeldeinformationen verknüpfte AWS-Konto beschränkt, sowie auf die für den Client festgelegte Region. In einem AWS-Konto könnten viele Streams gleichzeitig aktiv sein. Sie können Ihre Streams in der Konsole von Kinesis Data Streams oder programmgesteuert auflisten. Der Code in diesem Abschnitt zeigt, wie alle Streams für Ihr AWS-Konto aufgelistet werden.

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

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

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

Dieser Code ruft die `getHasMoreStreams`-Methode für `listStreamsRequest` auf, um zu überprüfen, ob zusätzlich zu den im ersten Aufruf von `listStreams` zurückgegebenen Streams

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

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

- CREATING
- ACTIVE
- UPDATING
- DELETING

Sie können den Status eines Streams mit der `describeStream`-Methode überprüfen, wie im vorherigen Abschnitt [Erstellen eines Streams über die APIs](#) dargestellt.

## Auflisten von Shards

Ein Datenstrom kann einen oder mehrere Shards aufweisen. Es gibt zwei Methoden zum Auflisten (oder Abrufen) von Shards aus einem Datenstrom.

Themen

- [ListShards API — Empfohlen](#)
- [DescribeStream API — Veraltet](#)

## ListShards API — Empfohlen

Die empfohlene Methode zum Auflisten oder Abrufen der Shards aus einem Datenstrom ist die Verwendung der [ListShards](#)API. Das folgende Beispiel zeigt, wie Sie eine Liste der Shards in einem Datenstrom erhalten. Eine vollständige Beschreibung der in diesem Beispiel verwendeten Hauptoperation und aller Parameter, die Sie für die Operation festlegen können, finden Sie unter [ListShards](#)

```
import software.amazon.awssdk.services.kinesis.KinesisAsyncClient;
```

```
import software.amazon.awssdk.services.kinesis.model.ListShardsRequest;
import software.amazon.awssdk.services.kinesis.model.ListShardsResponse;

import java.util.concurrent.TimeUnit;

public class ShardSample {

    public static void main(String[] args) {

        KinesisAsyncClient client = KinesisAsyncClient.builder().build();

        ListShardsRequest request = ListShardsRequest
            .builder().streamName("myFirstStream")
            .build();

        try {
            ListShardsResponse response = client.listShards(request).get(5000,
                TimeUnit.MILLISECONDS);
            System.out.println(response.toString());
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Zum Ausführen des vorherigen Codebeispiels können Sie eine POM-Datei wie die folgende verwenden.

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

    <groupId>kinesis.data.streams.samples</groupId>
    <artifactId>shards</artifactId>
    <version>1.0-SNAPSHOT</version>
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
```

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

Mit der `ListShards` API können Sie den [ShardFilter](#) Parameter verwenden, um die Antwort der API herauszufiltern. Sie können jeweils nur einen Filter angeben.

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

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

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

#### Important

APIs `DescribeStreamSummary` und `ListShard` bieten eine skalierbarere Möglichkeit, Informationen über Ihre Datenströme abzurufen. Insbesondere können die Kontingente

für die DescribeStream API zu einer Drosselung führen. Weitere Informationen finden Sie unter [Kontingente und Einschränkungen](#). Beachten Sie außerdem, dass Kontingente DescribeStream von allen Anwendungen gemeinsam genutzt werden, die mit allen Datenströmen in Ihrem AWS-Konto interagieren. Die Kontingente für die ListShards API sind dagegen spezifisch für einen einzelnen Datenstrom. Mit der ListShards API erhalten Sie also nicht nur höhere TPS-Werte, sondern die Aktion lässt sich auch besser skalieren, je mehr Datenströme Sie erstellen.

Wir empfehlen Ihnen, all Ihre Produzenten und Verbraucher, die die DescribeStream API aufrufen, zu migrieren, um stattdessen die DescribeStreamSummary und die ListShard APIs aufzurufen. Um diese Hersteller und Verbraucher zu identifizieren, empfehlen wir, Athena zum Analysieren von CloudTrail Protokollen zu verwenden, während Benutzeragenten für KPL und KCL in den API-Aufrufen erfasst werden.

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

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

## DescribeStream API — Veraltet

### Important

Die folgenden Informationen beschreiben eine derzeit veraltete Methode zum Abrufen von Shards aus einem Datenstrom über die API. DescribeStream Es wird derzeit dringend



empfohlen, die `ListShards`-API zum Abrufen der Shards zu verwenden, aus denen der Datenstrom besteht.

Mit dem von der `describeStream`-Methode zurückgegebenen Antwortobjekt können Sie Informationen zu den Shards abrufen, die der Stream enthält. Zum Abrufen der Shards rufen Sie die `getShards`-Methode auf diesem Objekt auf. Diese Methode gibt möglicherweise nicht alle Shards aus dem Stream in einem einzigen Aufruf zurück. Im folgenden Code überprüfen Sie die `getHasMoreShards`-Methode auf `getStreamDescription`, um zu ermitteln, ob weitere Shards vorhanden sind, die nicht zurückgegeben wurden. Wenn dies der Fall ist, d. h. wenn diese Methode `true` zurückgibt, rufen wir `getShards` weiterhin in Schleifen auf, sodass wir alle neuen Stapel von zurückgegebenen Shards zu unserer Liste von Shards hinzufügen. Die Schleife wird beendet, wenn `getHasMoreShards` den Wert `false` zurückgibt, d. h. dass alle Shards zurückgegeben wurden. Beachten Sie, dass `getShards` keine Shards zurückgibt, die sich im Zustand `EXPIRED` befinden. Weitere Informationen zu Shard-Zuständen, einschließlich des Status `EXPIRED` finden Sie unter [Routing von Daten, dauerhafte Speicherung von Daten und Shard-Status nach einem Resharding](#).

```
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 );
```

## Löschen eines Streams

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

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

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

## Resharding eines Streams

### Important

Sie können Ihren Stream mithilfe der API erneut teilen. [UpdateShardCount](#) Andernfalls können Sie auch weiterhin Teilungen und Zusammenführungen ausführen, wie hier beschrieben.

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

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

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

Gleichermaßen verringern Zusammenführungen die Anzahl der Shards in Ihrem Stream und damit die Datenkapazität – und Kosten – des Streams.

Das Resharding wird üblicherweise von einer administrativen Anwendung ausgeführt. Diese unterscheiden sich von den Produzentenanwendungen (zum Senden) und den Konsumentenanwendungen (zum Abrufen). Eine solche Verwaltungsanwendung überwacht die Gesamtleistung des Streams auf der Grundlage von Metriken, die von Amazon bereitgestellt werden, CloudWatch oder auf der Grundlage von Metriken, die von Herstellern und Verbrauchern gesammelt wurden. Die administrative Anwendung benötigt auch eine breitere Palette von IAM-Berechtigungen als die Konsumenten oder Produzenten, da diese in der Regel keinen Zugriff auf die APIs für das Resharding benötigen. Weitere Informationen zu IAM-Berechtigungen für Kinesis Data Streams finden Sie unter [Steuern des Zugriffs auf Ressourcen von Amazon Kinesis Data Streams mithilfe von IAM](#).

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

Themen

- [Strategien für das Resharding](#)
- [Teilen eines Shards](#)
- [Zusammenführen von zwei Shards](#)
- [Nach dem Resharding](#)

## Strategien für das Resharding

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

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

Sie können darüber hinaus Metriken verwenden, um zu ermitteln, welche Ihre heißen oder kalten Shards sind, d. h. Shards, die viel mehr bzw. viel weniger Daten empfangen als erwartet. Sie könnten dann selektiv die heißen Shards teilen, um die Kapazität für die Hash-Schlüssel zu erhöhen, die

auf solche Shards abzielen. Gleichmaßen könnten Sie kalte Shards zusammenführen, um deren ungenutzte Kapazitäten besser zu nutzen.

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

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

Kinesis Data Streams verwendet [MD5](#) zum Verarbeiten des Hash-Schlüssels aus dem Partitionsschlüssel. Da Sie den Partitionsschlüssel für den Datensatz festlegen, könnten Sie MD5 verwenden, um den Hash-Schlüsselwert für diesen Datensätze zu verarbeiten und zu protokollieren.

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

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

Mit den Shard-IDs und den Hash-Schlüsselwerten können Sie bestimmen, welche Shards und Hash-Schlüssel am meisten bzw. am wenigsten Datenverkehr empfangen. Anschließend können Sie mithilfe von Resharding mehr oder weniger Kapazitäten bereitstellen, wie für diese Schlüssel angemessen.

## Teilen eines Shards

Um einen Shard in Amazon Kinesis Data Streams zu teilen, müssen Sie festlegen, wie Hash-Schlüsselwerte aus dem übergeordneten Shard auf die untergeordneten Shards umverteilt werden sollten. Wenn Sie einen Datensatz zu einem Stream hinzufügen, wird dieser basierend auf einem Hash-Schlüsselwert einem Shard zugeordnet. Der Hash-Schlüsselwert ist der [MD5](#)-Hashwert des Partitionsschlüssels, den Sie zu dem Zeitpunkt, an dem Sie den Datensatz zum Stream hinzufügen, für den Datensatz festlegen. Datensätze mit demselben Partitionsschlüssel haben auch denselben Hash-Schlüsselwert.

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

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

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

Der folgende Code zeigt einen Shard -Teilungsvorgang, bei dem die Hash-Schlüssel gleichmäßig zwischen den untergeordneten Shards verteilt werden, wobei der übergeordnete Shard im Wesentlichen in zwei Hälften geteilt wird. Dies ist nur eine Möglichkeit, den übergeordneten Shard zu teilen. Sie könnten beispielsweise den Shard so teilen, dass das untere Drittel des Schlüssels aus dem übergeordneten Shard an einen untergeordneten Shard und die zwei oberen Drittel der Schlüssel an einen anderen untergeordneten Shard übergehen. Bei vielen Anwendungen ist jedoch das Teilen von Shards in zwei Hälften ein effektiver Ansatz.

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

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

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

```
BigInteger startingHashKey = new
    BigInteger(shard.getHashKeyRange().getStartingHashKey());
BigInteger endingHashKey = new
    BigInteger(shard.getHashKeyRange().getEndingHashKey());
```

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

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

Der erste Schritt nach diesem Verfahren wird unter [Warten, bis ein Stream wieder aktiviert wird](#) veranschaulicht.

## Zusammenführen von zwei Shards

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

### Shard-Nachbarschaft

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

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

Die Reihe aller OPEN-Shards in einem Stream umfasst – als Gruppe – immer den gesamten Bereich von MD5-Hash-Schlüsselwerten. Weitere Informationen zu Shard-Status – wie beispielsweise CLOSED – finden Sie unter [Routing von Daten, dauerhafte Speicherung von Daten und Shard-Status nach einem Resharding](#).

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

```
if( null == shard.getSequenceNumberRange().getEndingSequenceNumber() )
{
```

```
// Shard is OPEN, so it is a possible candidate to be merged.
}
```

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

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

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

### Code für den Zusammenführungsvorgang

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

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

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

Der erste Schritt nach diesem Verfahren wird unter [Warten, bis ein Stream wieder aktiviert wird](#) veranschaulicht.

## Nach dem Resharding

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

### Themen

- [Warten, bis ein Stream wieder aktiviert wird](#)
- [Routing von Daten, dauerhafte Speicherung von Daten und Shard-Status nach einem Resharding](#)

## Warten, bis ein Stream wieder aktiviert wird

Nachdem Sie einen Resharding-Vorgang aufgerufen haben, entweder `splitShard` oder `mergeShards`, müssen Sie warten, bis der Stream wieder aktiviert wird. Der zu verwendende Code entspricht dem Code beim Warten, bis ein Stream nach dem [Erstellen eines Streams](#) aktiviert wird.

Dieser Code sieht wie folgt aus:

```
DescribeStreamRequest describeStreamRequest = new DescribeStreamRequest();
describeStreamRequest.setStreamName( myStreamName );

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

    try {
        DescribeStreamResult describeStreamResponse =
client.describeStream( describeStreamRequest );
        String streamStatus =
describeStreamResponse.getStreamDescription().getStreamStatus();
        if ( streamStatus.equals( "ACTIVE" ) ) {
            break;
        }
        //
        // sleep for one second
        //
        try {
            Thread.sleep( 1000 );
        }
        catch ( Exception e ) {}
    }
    catch ( ResourceNotFoundException e ) {}
}
if ( System.currentTimeMillis() >= endTime )
{
```



```
throw new RuntimeException( "Stream " + myStreamName + " never went active" );
}
```

## Routing von Daten, dauerhafte Speicherung von Daten und Shard-Status nach einem Resharding

Kinesis Data Streams ist ein Service für das Echtzeit-Daten-Streaming, d. h. Ihre Anwendungen sollten voraussetzen, dass Daten kontinuierlich durch die Shards in Ihrem Stream geleitet werden. Bei einem Resharding werden Datensätze, die an die übergeordneten Shards geleitet wurden, basierend auf den Hash-Schlüsselwerten, die den Partitionsschlüsseln zu den Datensätzen zugeordnet werden, zu den untergeordneten Shards umgeleitet. Datensätze, die sich vor dem Resharding in den übergeordneten Shards befunden haben, verbleiben jedoch in diesen Shards. Mit anderen Worten, die übergeordneten Shards gehen bei einem Resharding nicht verloren. Sie bleiben zusammen mit den Daten erhalten, die sie vor dem Resharding enthielten. Die Datensätze in den übergeordneten Shards können mithilfe der Operationen [getShardIterator](#) und [getRecords](#) in der API für Kinesis Data Streams und über die Kinesis Client Library aufgerufen werden.

### Note

Datensätze können ab dem Zeitpunkt, an dem Sie zum Stream hinzugefügt wurden, bis zum aktuellen Aufbewahrungszeitraum aufgerufen werden. Dies gilt unabhängig von Änderungen an den Shards in dem Stream in diesem Zeitraum. Weitere Informationen zum Aufbewahrungszeitraum eines Streams finden Sie unter [Ändern des Zeitraums der Datenaufbewahrung](#).

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

- **OPEN:** Vor einem Resharding-Vorgang befindet sich ein übergeordneter Shard im Status OPEN. Dies bedeutet, dass Datensätze sowohl zum Shard hinzugefügt als auch vom Shard abgerufen werden können.
- **CLOSED:** Nach einem Reshard-Vorgang geht der übergeordnete Shard in den Status CLOSED über. Das bedeutet, dass Datensätze nicht mehr zum Shard hinzugefügt werden. Die Datensätze, die zu diesem Shard hinzugefügt worden wären, werden nun stattdessen zu einem untergeordneten Shard hinzugefügt. Datensätze können jedoch weiterhin für einen begrenzten Zeitraum von dem Shard abgerufen werden.

- **EXPIRED:** Wenn der Aufbewahrungszeitraum des Streams abgelaufen ist, sind auch alle Datensätze im übergeordneten Shard abgelaufen und nicht mehr zugänglich. Zu diesem Zeitpunkt geht der Shard selbst in den Status EXPIRED über. Aufrufe an `getStreamDescription().getShards` zum Aufzählen der Shards im Stream berücksichtigen keine Shards mit dem Status EXPIRED in der Liste der zurückgegebenen Shards. Weitere Informationen zum Aufbewahrungszeitraum eines Streams finden Sie unter [Ändern des Zeitraums der Datenaufbewahrung](#).

Wenn sich der Stream nach einem Resharding-Vorgang wieder im Status ACTIVE befindet, können Sie sofort mit dem Auslesen von Daten aus den untergeordneten Shards beginnen. Die nach dem Resharding verbleibenden übergeordneten Shards könnten jedoch weiterhin Daten enthalten, die noch nicht ausgelesen wurden und die vor dem Resharding zum Stream hinzugefügt wurden. Wenn Sie Daten aus den untergeordneten Shards auslesen, bevor alle Daten aus den übergeordneten Shards ausgelesen wurden, lesen Sie die Daten für einen bestimmten Hash-Schlüssel möglicherweise nicht in der durch die Sequenznummern der Datensätze festgelegten Reihenfolge. Unter der Annahme, dass die Reihenfolge der Daten wichtig ist, sollten Sie daher nach einem Resharding Daten aus den übergeordneten Shards immer so lange weiter auslesen, bis sie ausgeschöpft sind. Erst dann sollten Sie mit dem Auslesen von Daten aus den untergeordneten Shards beginnen. Wenn `getRecordsResult.getNextShardIterator` den Wert `null` zurückgibt, bedeutet dies, dass alle Daten im übergeordneten Shard ausgelesen wurden. Wenn Sie Daten mithilfe der Kinesis Client Library lesen, stellt die Bibliothek sicher, dass Sie die Daten auch bei einem Resharding in der richtigen Reihenfolge empfangen.

## Ändern des Zeitraums der Datenaufbewahrung

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

Sie können den Aufbewahrungszeitraum über die Kinesis Data Streams Streams-Konsole oder mithilfe der [DecreaseStreamRetentionPeriod](#) Operationen [IncreaseStreamRetentionPeriod](#) und den Vorgängen aktualisieren. Mit der Konsole von Kinesis Data Streams können Sie den Aufbewahrungszeitraum mehrerer Datenströme gleichzeitig bearbeiten. Sie können die

Aufbewahrungsdauer mithilfe der [IncreaseStreamRetentionPeriod](#) Operation oder der Kinesis Data Streams Streams-Konsole auf maximal 8760 Stunden (365 Tage) erhöhen. Sie können die Aufbewahrungsdauer mithilfe der [DecreaseStreamRetentionPeriod](#) Operation oder der Kinesis Data Streams Streams-Konsole auf mindestens 24 Stunden reduzieren. Die Anforderungssyntax für beide Operationen enthält den Stream-Namen und den Aufbewahrungszeitraum in Stunden. Zuletzt können Sie den aktuellen Aufbewahrungszeitraum eines Streams überprüfen, indem Sie die [DescribeStream](#) Operation aufrufen.

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

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

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

Kinesis Data Streams macht bei einer Verringerung des Aufbewahrungszeitraums beinahe sofort die Datensätze unzugänglich, die vor dem neuen Aufbewahrungszeitraum liegen. Deshalb sollten Sie beim Aufrufen der [DecreaseStreamRetentionPeriod](#) Operation äußerst vorsichtig vorgehen.

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

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

# Tagging von Streams in Amazon Kinesis Data Streams

Sie können den Streams, die Sie in Amazon Kinesis Data Streams erstellen, eigene Metadaten in Form von Tags zuweisen. Ein Tag ist ein Schlüssel-Wert-Paar, das Sie für einen Stream definieren. Die Verwendung von Tags ist ein einfacher, aber effizienter Weg, um AWS-Ressourcen zu verwalten und Daten, einschließlich Fakturierungsdaten, zu organisieren.

## Inhalt

- [Grundlagen zu Tags](#)
- [Kosten mithilfe von Tags verfolgen](#)
- [Tag-Einschränkungen](#)
- [Taggen von Streams mithilfe der Kinesis-Data-Streams-Konsole](#)
- [Tagging von Streams mithilfe der AWS CLI](#)
- [Taggen von Streams mithilfe der Kinesis-Data-Streams-API](#)

## Grundlagen zu Tags

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

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

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

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

- Umgebung: Produktion

## Kosten mithilfe von Tags verfolgen

Sie können Tags verwenden, um Ihre AWS-Kosten zu kategorisieren und zu verfolgen. Wenn Sie Tags auf AWS-Ressourcen anwenden, einschließlich Streams, enthält der AWS-Kostenzuordnungsbericht mit Tags aggregierte Nutzungs- und Kostendaten. Sie können Tags anwenden, die geschäftliche Kategorien (wie Kostenstellen, Anwendungsnamen oder Eigentümer) darstellen, um die Kosten für mehrere Services zu organisieren. Weitere Informationen finden Sie unter [Verwenden von Kostenzuordnungs-Tags für benutzerdefinierte Fakturierungsberichte](#) im AWS Billing-Benutzerhandbuch.

## Tag-Einschränkungen

Für Tags gelten die folgenden Einschränkungen.

### Grundlegende Einschränkungen

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

### Einschränkungen für Tag-Schlüssel

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

### Einschränkungen für den Tag-Wert

- Tag-Werte müssen zwischen 0 und 255 Unicode-Zeichen lang sein.

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

## Taggen von Streams mithilfe der Kinesis-Data-Streams-Konsole

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

### Anzeigen der Tags für einen Stream

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

### Hinzufügen eines Tags zu einem Stream

1. Öffnen Sie die Kinesis-Data-Streams-Konsole. Erweitern Sie in der Navigationsleiste die Regionsauswahl und wählen Sie eine Region aus.
2. Wählen Sie auf der Seite Stream List (Stream-Liste) einen Stream aus.
3. Klicken Sie auf der Seite Stream Details (Stream-Details) auf die Registerkarte Tags.
4. Geben Sie den Tag-Schlüssel in das Feld Key (Schlüssel) ein oder geben Sie optional einen Tag-Wert im Feld Value (Wert) an und klicken Sie dann auf Add Tag (Tag hinzufügen).

Wenn die Schaltfläche Add Tag (Tag hinzufügen) nicht aktiviert ist, entspricht entweder der angegebene Tag-Schlüssel oder Tag-Wert nicht den Tag-Einschränkungen. Weitere Informationen finden Sie unter [Tag-Einschränkungen](#).

5. Um Ihr neues Tag in der Liste auf der Registerkarte Tags anzuzeigen, klicken Sie auf das Aktualisierungssymbol.

### Entfernen eines Tags von einem Stream

1. Öffnen Sie die Kinesis-Data-Streams-Konsole. Erweitern Sie in der Navigationsleiste die Regionsauswahl und wählen Sie eine Region aus.
2. Wählen Sie auf der Seite „Stream List“ einen Stream aus.
3. Klicken Sie auf der Seite „Stream Details (Stream-Details)“ auf die Registerkarte Tags und anschließend auf der Symbol Remove (Entfernen) für den Tag.

4. Klicken Sie im Dialogfeld Delete Tag (Tag löschen) auf Yes, Delete (Ja, löschen).

## Tagging von Streams mithilfe der AWS CLI

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

### [add-tags-to-stream](#)

Fügt Tags für den angegebenen Stream hinzu oder aktualisiert diese.

### [list-tags-for-stream](#)

Listet die Tags für den angegebenen Stream auf.

### [remove-tags-from-stream](#)

Entfernt Tags von dem angegebenen Stream.

## Taggen von Streams mithilfe der Kinesis-Data-Streams-API

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

### [AddTagsToStream](#)

Fügt Tags für den angegebenen Stream hinzu oder aktualisiert diese.

### [ListTagsForStream](#)

Listet die Tags für den angegebenen Stream auf.

### [RemoveTagsFromStream](#)

Entfernt Tags von dem angegebenen Stream.

# Schreiben von Daten in Amazon Kinesis Data Streams

Ein Produzent ist eine Anwendung, die Daten in Amazon Kinesis Data Streams schreibt. Sie können Produzenten für Kinesis Data Streams unter Verwendung der AWS SDK for Java und der Kinesis Producer Library erstellen.

Wenn Sie Kinesis Data Streams zum ersten Mal verwenden, sollten Sie sich zunächst mit den Konzepten und der Terminologie in [Was ist Amazon Kinesis Data Streams?](#) und [Erste Schritte mit Amazon Kinesis Data Streams](#) vertraut machen.

## Important

Kinesis Data Streams unterstützt Änderungen des Zeitraums der Datensatzaufbewahrung für einen Datenstrom. Weitere Informationen finden Sie unter [Ändern des Zeitraums der Datenaufbewahrung](#).

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

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

## Inhalt

- [Entwickeln von Produzenten mit der Amazon Kinesis Producer Library](#)
- [Entwicklung von Produzenten mithilfe der API für Amazon Kinesis Data Streams mit der AWS SDK for Java](#)
- [Schreiben in Amazon Kinesis Data Streams mit Kinesis Agent](#)
- [Schreiben in Kinesis Data Streams mithilfe anderer AWS-Services](#)
- [Integrationen von Drittanbietern verwenden](#)



- [Fehlerbehebung bei Amazon Kinesis Data Streams Producers](#)
- [Fortgeschrittene Themen für Kinesis-Data-Streams-Produzenten](#)

## Entwickeln von Produzenten mit der Amazon Kinesis Producer Library

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

Sie können die KPL mit Amazon überwachen CloudWatch. Weitere Informationen finden Sie unter [Überwachen der Kinesis Producer Library mit Amazon CloudWatch](#).

### Inhalt

- [Die Rolle der KPL](#)
- [Vorteile der Verwendung der KPL](#)
- [Wann die KPL nicht verwendet werden sollte](#)
- [Installieren der KPL](#)
- [Umstieg auf Amazon Trust Services \(ATS\)-Zertifikate für die Kinesis Producer Library](#)
- [KPL-unterstützte Plattformen](#)
- [Die wichtigsten Konzepte von KPL](#)
- [Integrieren der KPL mit Producer-Code](#)
- [Schreiben in den Kinesis Data Streams mit der KPL](#)
- [Konfiguration der Kinesis Producer Library](#)
- [Datenproduzent – Disaggregation](#)
- [Verwenden der KPL mit Firehose](#)
- [Verwenden der KPL mit der AWS Glue Schema Registry](#)
- [KPL-Proxy-Konfiguration](#)

**Note**

Es wird empfohlen, ein Upgrade auf die neueste KPL-Version durchzuführen. KPL wird regelmäßig mit neueren Versionen aktualisiert, die die neuesten Abhängigkeits- und Sicherheitspatches, Bugfixes und abwärtskompatible neue Features enthalten. Weitere Informationen finden Sie unter <https://github.com/awslabs/amazon-kinesis-producer/releases/>.

## Die Rolle der KPL

Die KPL ist eine hoch konfigurierbare -Bibliothek easy-to-use, mit der Sie in einen Kinesis-Datenstrom schreiben können. Sie dient als Vermittler zwischen dem Code Ihrer Produzentenanwendung und den API-Aktionen von Kinesis Data Streams. Die KPL führt die folgenden primären Aufgaben durch:

- Schreiben in einen oder mehrere Kinesis-Datenströme mit einem automatischen und konfigurierbaren Wiederholungsmechanismus
- Sammeln von Datensätzen und Verwenden von `PutRecords`, um pro Anforderung mehrere Datensätze für mehrere Shards zu schreiben
- Aggregieren von Benutzerdatensätzen zur Erhöhung der Nutzlast und Verbesserung des Durchsatzes
- Nahtlose Integration in die [Kinesis Client Library](#) (KCL), für eine Disaggregation gestapelter Datensätze auf Konsumentenseite
- Sendet Amazon- CloudWatch Metriken in Ihrem Namen, um Einblick in die Leistung des Produzenten zu gewähren

Beachten Sie, dass sich die KPL von der API für Kinesis Data Streams unterscheidet, die in den [AWS SDKs](#) enthalten ist. Die API für Kinesis Data Streams unterstützt Sie bei der Verwaltung vieler Aspekte von Kinesis Data Streams (einschließlich Stream-Erstellung, Resharding sowie Hinzufügen und Abrufen von Datensätzen), während die KPL eine Abstraktionsebene speziell für die Datenübernahme bereitstellt. Informationen zur Kinesis-Data-Streams-API finden Sie in der [Amazon-Kinesis-API-Referenz](#).

## Vorteile der Verwendung der KPL

In der folgenden Liste sind einige der größten Vorteile der Verwendung der KPL für die Entwicklung von Produzenten von Kinesis Data Streams aufgeführt.

Die KPL kann für synchrone oder asynchrone Anwendungsfälle eingesetzt werden. Wir empfehlen, die höhere Leistung der asynchronen Schnittstelle zu verwenden, sofern keine besonderen Gründe für die Nutzung des synchronen Verhaltens vorliegen. Weitere Informationen zu diesen beiden Anwendungsfällen sowie einen Beispiel-Code finden Sie unter [Schreiben in den Kinesis Data Streams mit der KPL](#).

### Leistungsvorteile

Mit der KPL können Sie leistungsstarke Produzenten erstellen. Angenommen, Ihre Amazon-EC2-Instances dienen als Proxy zum Erfassen von 100-Byte-Ereignissen von mehreren Hundert oder Tausend energiesparenden Geräten und zum Schreiben von Datensätzen in einen Kinesis-Datenstrom. Diese EC2 Instances müssen jeweils Tausende von Ereignissen pro Sekunde in Ihren Datenstream schreiben. Um den nötigen Durchsatz zu erreichen, müssen Produzenten eine komplexe Logik, beispielsweise Stapelverarbeitung oder Multithreading, implementieren und auf Konsumentenseite eine Wiederholung der Logik sowie die Disaggregation der Datensätze sicherstellen. Die KPL führt all diese Aufgaben für Sie durch.

### Anwenderfreundlichkeit auf Konsumentenseite

Entwickler auf Konsumentenseite, die die KCL in Java nutzen, können die KPL ohne zusätzlichen Aufwand integrieren. Wenn die KCL einen aggregierten Datensatz von Kinesis Data Streams abrufen, der aus mehreren KPL-Benutzerdatensätzen besteht, ruft sie automatisch die KPL auf, um die einzelnen Benutzerdatensätze vor der Rückgabe an den Benutzer zu extrahieren.

Für Entwickler auf Verbraucherseite, die anstelle der KCL direkt die API-Operation `GetRecords` nutzen, steht eine KPL-Java-Bibliothek zum Extrahieren der einzelnen Benutzerdatensätze vor der Rückgabe an den Benutzer zur Verfügung.

### Überwachen von Produzenten

Sie können Ihre Produzenten von Kinesis Data Streams mit Amazon und der KPL erfassen, überwachen CloudWatch und analysieren. Die KPL gibt in CloudWatch Ihrem Namen Durchsatz, Fehler und andere Metriken an aus und kann für die Überwachung auf Stream-, Shard- oder Produzentenebene konfiguriert werden.

## Asynchrone Architektur

Da die KPL Datensätze möglicherweise vor dem Senden an Kinesis Data Streams puffert, wird die Anwendung des Aufrufers nicht zum Blockieren und Warten auf die Bestätigung, dass der Datensatz beim Server angekommen ist, gezwungen, ehe die Ausführung fortgesetzt werden kann. Der Aufruf, einen Datensatz an die KPL zu übergeben, wird stets sofort zurückgegeben. Es wird nicht darauf gewartet, dass der Datensatz gesendet bzw. eine Antwort vom Server empfangen wird. Stattdessen wird ein Future-Objekt erstellt, das zu einem späteren Zeitpunkt das Ergebnis der Sendung des Datensatzes an Kinesis Data Streams empfängt. Dies ist dasselbe Verhalten wie asynchrone Clients im AWS SDK.

## Wann die KPL nicht verwendet werden sollte

Die KPL kann eine weitere Verarbeitungsverzögerung von `RecordMaxBufferedTime` innerhalb der Bibliothek verursachen (vom Benutzer konfigurierbar). Größere Werte von `RecordMaxBufferedTime` führen zu einer schnelleren Verpackung und einer besseren Leistung. Anwendungen, die diese zusätzliche Verzögerung nicht tolerieren können, müssen möglicherweise das AWS -SDK direkt verwenden. Weitere Informationen zur Verwendung des AWS SDK mit Kinesis Data Streams finden Sie unter [Entwicklung von Produzenten mithilfe der API für Amazon Kinesis Data Streams mit der AWS SDK for Java](#). Weitere Informationen zu `RecordMaxBufferedTime` und anderen vom Benutzer konfigurierbaren Eigenschaften der KPL finden Sie unter [Konfiguration der Kinesis Producer Library](#).

## Installieren der KPL

Amazon stellt vorgefertigte Binärdateien der C++ Kinesis Producer Library (KPL) für macOS, Windows und die neuesten Linux-Distributionen zur Verfügung (Informationen zu unterstützten Plattformen finden Sie im nächsten Abschnitt). Diese Binärdateien sind Teil der JAR-Dateien von Java und werden automatisch aufgerufen und verwendet, wenn Sie das Paket mit Maven installieren. Mit den folgenden Maven-Links können Sie nach den neusten Versionen der KPL und KCL suchen:

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

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

Benutzer älterer Linux-Distributionen können die KPL anhand der Build-Anweisungen erstellen, die zusammen mit der Quelle auf bereitgestellt werden GitHub. Informationen zum Herunterladen der KPL von finden Sie GitHubin der [Kinesis Producer Library](#).

## Umstieg auf Amazon Trust Services (ATS)-Zertifikate für die Kinesis Producer Library

Am 9. Februar 2018 um 9:00 AM PST hat Amazon Kinesis Data Streams ATS-Zertifikate installiert. Um weiterhin Datensätze in Kinesis Data Streams unter Verwendung der Kinesis Producer Library (KPL) schreiben zu können, müssen Sie die Installation der KPL auf [Version 0.12.6](#) oder höher aktualisieren. Diese Änderung wirkt sich auf alle AWS Regionen aus.

Informationen zum Wechsel zu ATS finden Sie unter [Vorbereitung auf den Wechsel AWS von zu seiner eigenen Zertifizierungsstelle](#).

Falls Sie Probleme haben und technischen Support benötigen, [erstellen Sie einen Fall](#) im AWS Support Center.

## KPL-unterstützte Plattformen

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

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

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

### Important

Windows Server 2008 und höher wird für alle KPL-Versionen bis Version 0.14.0 unterstützt. Die Windows-Plattform wird ab KPL Version 0.14.0 oder höher NICHT unterstützt.

Beachten Sie, dass es sich bei der KPL um eine 64-Bit-Version handelt.

## Quellcode

Für den Fall, dass die Binärdateien, die bei der KPL-Installation bereitgestellt werden, für Ihre Umgebung nicht ausreichen, wird der Kern der KPL als C++-Modul bereitgestellt. Der Quellcode für das C++-Modul und die Java-Schnittstelle werden unter der Amazon Public License veröffentlicht und sind auf GitHub in der [Kinesis Producer Library](#) verfügbar. Obwohl die KPL auf allen Plattformen verwendet werden kann, für die ein neuer, standardmäßiger C++-Compiler und eine Java-Laufzeitumgebung verfügbar sind, unterstützt Amazon offiziell keine Plattformen, die nicht auf der Liste der unterstützten Plattformen enthalten sind.

## Die wichtigsten Konzepte von KPL

Die folgenden Abschnitte enthalten Konzepte und Begriffe, die Sie kennen sollten, um von der Kinesis Producer Library (KPL) zu profitieren.

### Themen

- [Datensätze](#)
- [Stapelverarbeitung](#)
- [Aggregation](#)
- [Sammlung](#)

## Datensätze

In dieser Anleitung wird zwischen KPL-Benutzerdatensätzen und Datensätzen von Kinesis Data Streams unterschieden. Wenn von einem Datensatz ohne weitere Angabe die Rede ist, ist ein KPL-Benutzerdatensatz gemeint. Wenn wir uns auf einen Datensatz von Kinesis Data Streams beziehen, sagen wir ausdrücklich Datensatz von Kinesis Data Streams.

Ein KPL-Benutzerdatensatz ist ein Daten-Blob mit einer bestimmten Bedeutung für den Benutzer. Zu den Beispielen zählen ein JSON-Blob, der ein Ereignis auf der Benutzeroberfläche einer Website darstellt oder der Protokolleintrag eines Webservers.

Ein Datensatz von Kinesis Data Streams ist eine Instance der Record-Datenstruktur, die von der API des Kinesis-Data-Streams-Dienstes definiert wird. Er enthält eine Sequenznummer, einen Partitionsschlüssel und einen Daten-Blob.

## Stapelverarbeitung

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

In diesem Kontext steht „Element“ für einen Datensatz, der durch die Aktion an Kinesis Data Streams gesendet wird. Ohne Stapelverarbeitung würden Sie jeden Datensatz in einen separaten Datensatz von Kinesis Data Streams platzieren und eine HTTP-Anforderung erstellen, um ihn an Kinesis Data Streams zu senden. Mit Stapelverarbeitung können mehrere Datensätze in eine HTTP-Anforderung eingeschlossen werden.

Die KPL unterstützt zwei Arten der Stapelverarbeitung:

- Aggregation – Speichern mehrerer Datensätze in einem einzelnen Datensatz von Kinesis Data Streams.
- Sammlung – Verwenden der API-Operation `PutRecords` zum Senden mehrerer Datensätze aus Kinesis Data Streams an einen oder mehrere Shards in Ihrem Kinesis-Datenstrom.

Die zwei Arten der KPL-Stapelverarbeitung sind so konzipiert, dass sie gleichzeitig und unabhängig voneinander aktiviert oder deaktiviert werden können. Standardmäßig sind beide aktiviert.

### Aggregation

Aggregation bezieht sich auf die Speicherung mehrerer Datensätze in einem Datensatz von Kinesis Data Streams. Die Aggregation ermöglicht Kunden, die Anzahl der Datensätze zu erhöhen, die pro API-Aufruf gesendet werden. Dadurch steigt der Durchsatz des Produzenten deutlich.

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

Stellen Sie sich vor, es gibt einen Shard in der Region `us-east-1`, der derzeit mit einer konstanten Rate von 1.000 Datensätzen pro Sekunde ausgeführt wird. Jeder Datensatz besteht aus 512 Bytes. Mit der KPL-Aggregation können Sie 1 000 Datensätze in nur 10 Datensätze von Kinesis Data Streams verpacken und reduzieren so die Anfragen pro Sekunde (RPS) auf 10 (jeweils 50 KB).

## Sammlung

Mit Sammlung ist die Stapelverarbeitung mehrerer Datensätze von Kinesis Data Streams und das Senden von diesen in einer einzelnen HTTP-Anforderung mit einem Aufruf der API-Operation `PutRecords` gemeint. Dies steht im Gegensatz zum Senden einzelner Datensätze von Kinesis Data Streams in separaten HTTP-Anforderungen.

Durch eine Sammlung wird der Durchsatz erhöht, da der Aufwand für das Erstellen vieler separater HTTP-Anforderungen entfällt. `PutRecords` wurde speziell für diesen Zweck entwickelt.

Eine Sammlung unterscheidet sich von der Aggregation dahingehend, dass mit Gruppen von Datensätzen von Kinesis Data Streams gearbeitet wird. Die gesammelten Datensätze von Kinesis Data Streams können trotzdem mehrere Datensätze eines Benutzers enthalten. Die Beziehung lässt sich wie folgt visualisieren:

```

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

```



## Integrieren der KPL mit Producer-Code

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

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

Im Prozess sind möglicherweise Aufgaben enthalten, die nichts mit Kinesis Data Streams zu tun haben. Diese können auch bei einem Absturz der KPL weiterhin durchgeführt werden. Die KPL kann von Ihrem übergeordneten Benutzerprozess auch neu gestartet werden, sodass eine volle Betriebsfähigkeit wiederhergestellt wird (die Funktion ist in den offiziellen Wrappern enthalten).

Ein Beispiel hierfür ist ein Webserver, der Metriken an Kinesis Data Streams sendet. Der Server kann auch noch Seiten bereitstellen, wenn die Komponente Kinesis Data Streams nicht mehr betriebsfähig ist. Ein Absturz des gesamten Servers aufgrund eines Fehlers in der KPL würde einen unnötigen Ausfall verursachen.

2) Es können beliebige Clients unterstützt werden

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

### Empfohlene Nutzungsmatrix

Die folgende Nutzungs-Matrix führt die empfohlenen Einstellungen für verschiedene Benutzer auf und informiert Sie darüber, ob und wie Sie die KPL einsetzen sollten. Beachten Sie, dass bei aktivierter Aggregation auch eine Disaggregation verwendet werden muss, um die Datensätze auf der Konsumentenseite zu extrahieren.

Sprache auf Produzentenseite	Sprache auf Konsumentenseite	KCL-Version	Prüfpunkt-Logik	Können Sie die KPL verwenden?	Einschränkungen
Alles außer Java	*	*	*	Nein	N/A
Java	Java	Direkte Java-SDK-Verwendung	N/A	Ja	Bei aktivierter Aggregation muss nach

Sprache auf Produzentenseite	Sprache auf Konsumentenseite	KCL-Version	Prüfpunkt-Logik	Können Sie die KPL verwenden?	Einschränkungen
					GetRecords -Aufrufen die bereitgestellte Disaggregationsbibliothek verwendet werden.
Java	Alles außer Java	Direkte SDK-Verwendung	N/A	Ja	Aggregation muss deaktiviert werden.
Java	Java	1.3.x	N/A	Ja	Aggregation muss deaktiviert werden.
Java	Java	1.4.x	Prüfpunkt-Aufruf ohne Argumente	Ja	None

Sprache auf Produzentenseite	Sprache auf Konsumentenseite	KCL-Version	Prüfpunkt-Logik	Können Sie die KPL verwenden?	Einschränkungen
Java	Java	1.4.x	Prüfpunkt-Aufruf mit expliziter Sequenznummer	Ja	Entweder muss die Aggregation deaktiviert oder der Code so geändert werden, dass erweiterte Sequenznummern für das Setzen von Prüfpunkten verwendet werden.
Java	Alles außer Java	1.3.x + mehrsprachiger Daemon + sprachspezifischer Wrapper	N/A	Ja	Aggregation muss deaktiviert werden.

## Schreiben in den Kinesis Data Streams mit der KPL

In den folgenden Abschnitten wird ein Beispiel-Code in Entwicklung gezeigt (vom reinen Produzenten bis hin zum komplexen asynchronen Code).

### Reiner Produzentencode

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

```
// KinesisProducer gets credentials automatically like
// DefaultAWSCredentialsProviderChain.
// It also gets region automatically from the EC2 metadata service.
KinesisProducer kinesis = new KinesisProducer();
// Put some records
for (int i = 0; i < 100; ++i) {
    ByteBuffer data = ByteBuffer.wrap("myData".getBytes("UTF-8"));
    // doesn't block
    kinesis.addUserRecord("myStream", "myPartitionKey", data);
}
// Do other stuff ...
```

## Synchrone Reaktion auf Ergebnisse

Im vorherigen Beispiel wird vom Code nicht geprüft, ob die KPL-Benutzerdatensätze erfolgreich verarbeitet wurden. Die KPL führt alle Wiederholungen durch, die notwendig sind, um Ausfälle zu kompensieren. Wenn Sie jedoch die Ergebnisse prüfen möchten, können Sie sich diese, wie im folgenden Beispiel gezeigt, mit den Future-Objekten ansehen, die von `addUserRecord` zurückgegeben werden (vorheriges Beispiel aus Kontextgründen erneut aufgeführt).

```
KinesisProducer kinesis = new KinesisProducer();

// Put some records and save the Futures
List<Future<UserRecordResult>> putFutures = new
    LinkedList<Future<UserRecordResult>>();
for (int i = 0; i < 100; i++) {
    ByteBuffer data = ByteBuffer.wrap("myData".getBytes("UTF-8"));
    // doesn't block
    putFutures.add(
        kinesis.addUserRecord("myStream", "myPartitionKey", data));
}

// Wait for puts to finish and check the results
for (Future<UserRecordResult> f : putFutures) {
    UserRecordResult result = f.get(); // this does block
    if (result.isSuccessful()) {
        System.out.println("Put record into shard " +
            result.getShardId());
    } else {
        for (Attempt attempt : result.getAttempts()) {
            // Analyze and respond to the failure
        }
    }
}
```

```
}  
}
```

## Asynchrone Reaktion auf Ergebnisse

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

```
KinesisProducer kinesis = new KinesisProducer();  
  
FutureCallback<UserRecordResult> myCallback = new FutureCallback<UserRecordResult>() {  
  
    @Override public void onFailure(Throwable t) {  
        /* Analyze and respond to the failure */  
    };  
  
    @Override public void onSuccess(UserRecordResult result) {  
        /* Respond to the success */  
    };  
};  
  
for (int i = 0; i < 100; ++i) {  
    ByteBuffer data = ByteBuffer.wrap("myData".getBytes("UTF-8"));  
    ListenableFuture<UserRecordResult> f = kinesis.addUserRecord("myStream",  
"myPartitionKey", data);  
    // If the Future is complete by the time we call addCallback, the callback will be  
    invoked immediately.  
    Futures.addCallback(f, myCallback);  
}
```

## Konfiguration der Kinesis Producer Library

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

```
KinesisProducerConfiguration config = new KinesisProducerConfiguration()  
    .setRecordMaxBufferedTime(3000)  
    .setMaxConnections(1)  
    .setRequestTimeout(60000)
```

```
.setRegion("us-west-1");  
  
final KinesisProducer kinesisProducer = new KinesisProducer(config);
```

Sie können zudem eine Konfiguration aus einer Eigenschaftendatei laden:

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

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

Die Eigenschaftendatei sollte Parameter unter Verwendung ihrer Namen in angeben PascalCase. Die Namen stimmen mit denen überein, die in den SET-Methoden der `KinesisProducerConfiguration`-Klasse verwendet werden. Beispielsweise:

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

Weitere Informationen zu Nutzungsregeln und Wertlimits für Konfigurationsparameter finden Sie in der [Beispieldatei mit Konfigurationseigenschaften auf GitHub](#).

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

## Datenproduzent – Disaggregation

Ab Version 1.4.0 unterstützt die KCL eine automatische Disaggregation von KPL-Benutzerdatensätzen. Code für Konsumenten Anwendungen, der mit früheren KCL-Versionen geschrieben wurde, wird nach der Aktualisierung der KCL ohne Änderungen kompiliert. Wenn jedoch eine KPL-Aggregation auf Konsumentenseite verwendet wird, ist beim Checkpointing Folgendes zu beachten: Alle untergeordneten Datensätze innerhalb eines aggregierten Datensatzes haben dieselbe Sequenznummer. Somit müssen zusätzliche Daten mit dem Prüfpunkt gespeichert werden, wenn Sie zwischen untergeordneten Datensätze unterscheiden müssen. Diese zusätzlichen Daten werden als Teilsequenznummer bezeichnet.

## Migrieren von früheren KCL-Versionen

Sie müssen Ihre vorhandenen Aufrufe zum Setzen eines Prüfpunkts in Verbindung mit einer Aggregation nicht ändern. Sie können nach wie vor alle Datensätze erfolgreich abrufen, die in Kinesis Data Streams gespeichert sind. Die KCL stellt nun zwei neue Prüfpunkt-Operationen zur Verfügung, um bestimmte, unten beschriebene Anwendungsfälle zu unterstützen.

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

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

Das Aufrufen der neuen KCL-Prüfpunkt-Operation `checkpoint(Record record)` entspricht semantisch dem Checkpointing der angegebenen Sequenznummer von `Record` zusammen mit der impliziten Teilsequenznummer 0 (null). Handelt es sich beim `Record`-Aufruf um einen `UserRecord`, wird ein Checkpointing der Sequenznummer und Teilsequenznummer von `UserRecord` durchgeführt.

Das Aufrufen der neuen KCL-Prüfpunkt-Operation `checkpoint(String sequenceNumber, Long subSequenceNumber)` führt explizit zu einem Checkpointing der angegebenen Sequenznummer zusammen mit der angegebenen Teilsequenznummer.

In all diesen Fällen kann, nachdem der Prüfpunkt in der Amazon-DynamoDB-Prüfpunktabelle gespeichert wurde, die KCL das Abrufen der Datensätze fehlerfrei wieder aufnehmen, auch wenn die Anwendung abstürzt und neu gestartet wird. Sind mehr Datensätze in der Sequenz enthalten, beginnt das Abrufen mit dem Datensatz, dem die nächste Teilsequenznummer zugeordnet wurde, innerhalb des Datensatzes mit der Sequenznummer, für die zuletzt ein Prüfpunkt gesetzt wurde. Enthält der letzte Prüfpunkt die allerletzte Teilsequenznummer des vorherigen Sequenznummerndatensatzes, beginnt das Abrufen mit dem Datensatz, dem die nächst folgende Sequenznummer zugeordnet ist.

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

## KCL-Erweiterungen für die KPL-Disaggregation

Wie bereits erwähnt kann die KPL-Disaggregation ein Checkpointing der Teilsequenz umfassen. Zur Unterstützung des Checkpointings einer Teilsequenz wurde eine `UserRecord`-Klasse zur KCL hinzugefügt:

```
public class UserRecord extends Record {
    public long getSubSequenceNumber() {
        /* ... */
    }
    @Override
    public int hashCode() {
        /* contract-satisfying implementation */
    }
    @Override
    public boolean equals(Object obj) {
        /* contract-satisfying implementation */
    }
}
```

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

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

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

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

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

Neue Formularcode:

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



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

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

Die Umwandlung von `Record` in `UserRecord` ist stets erfolgreich, da bei der Implementierung stets `UserRecord` im Hintergrund verwendet wird. Wenn Sie keine arithmetischen Operationen für die Sequenznummern durchführen müssen, ist dieser Ansatz nicht zu empfehlen.

Während der Verarbeitung der KPL-Benutzerdatensätze schreibt die KCL die Teilsequenznummer für jede Zeile als zusätzliches Feld in die Amazon DynamoDB. Frühere Versionen der KCL nutzten `AFTER_SEQUENCE_NUMBER` zum Abrufen von Datensätzen bei der Wiederaufnahme von Prüfpunkten. Die aktuelle KCL mit KPL-Support verwendet stattdessen `AT_SEQUENCE_NUMBER`. Wenn der Datensatz bei der Sequenznummer abgerufen wird, bei der ein Prüfpunkt gesetzt wurde, wird die Teilsequenznummer, für die ein Checkpointing durchgeführt wurde, geprüft und untergeordnete Datensätze gegebenenfalls ausgelassen (möglicherweise alle, wenn beim letzten Datensatz ein Prüfpunkt gesetzt wurde). Nochmals: Nicht aggregierte Datensätze können als aggregierte Datensätze mit einem einzelnen untergeordneten Datensatz betrachtet werden, sodass derselbe Algorithmus sowohl für aggregierte als auch für nicht aggregierte Datensätze funktioniert.

## Direktes Verwenden von `GetRecords`

Sie können sich auch gegen die Verwendung der KCL entscheiden und stattdessen direkt die API-Operation `GetRecords` aufrufen, um Datensätze aus Kinesis Data Streams abzurufen. Zum Entpacken dieser abgerufenen Datensätze in Ihre ursprünglichen KPL-Benutzerdatensätze rufen Sie eine der folgenden statischen Operationen in `UserRecord.java` auf:

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

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

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

Jede dieser Operationen führt eine Disaggregation der vorhandenen Liste der Datensätze aus Kinesis Data Streams in eine Liste mit KPL-Benutzerdatensätzen durch. Alle KPL-Benutzerdatensätze, deren expliziter Hash- oder Partitionsschlüssel außerhalb des Bereichs von `startingHashKey` (einschließlich) liegt, sowie `endingHashKey` (einschließlich) werden aus der zurückgegebenen Datensatzliste entfernt.

## Verwenden der KPL mit Firehose

Wenn Sie die Kinesis Producer Library (KPL) verwenden, um Daten an einen Kinesis Data Stream zu schreiben, können Sie die Aggregation dazu verwenden, die an diesen Kinesis Data Stream geschriebenen Datensätze zu kombinieren. Wenn Sie dann diesen Datenstrom als Quelle für Ihren Firehose-Bereitstellungsdatenstrom verwenden, deaggregiert Firehose die Datensätze, bevor sie an das Ziel geliefert werden. Wenn Sie Ihren Bereitstellungsdatenstrom für die Transformation der Daten konfigurieren, deaggregiert Firehose die Datensätze, bevor sie an übermitteln werden AWS Lambda. Weitere Informationen finden Sie unter [Schreiben zu Amazon Firehose mithilfe von Kinesis Data Streams](#).

## Verwenden der KPL mit der AWS Glue Schema Registry

Sie können Ihre Kinesis-Datenströme in die - AWS Glue-Schemaregistrierung integrieren. Mit dem AWS Glue Schema Registry können Sie Schemata zentral erkennen, steuern und weiterentwickeln und gleichzeitig sicherstellen, dass die erstellten Daten kontinuierlich durch ein registriertes Schema validiert werden. Ein Schema definiert die Struktur und das Format eines Datensatzes. Ein Schema ist eine versionierte Spezifikation für zuverlässige Datenveröffentlichung, -nutzung oder -speicherung. Mit der AWS Glue Schema Registry können Sie die end-to-end Datenqualität und Datenverwaltung in Ihren Streaming-Anwendungen verbessern. Weitere Informationen finden Sie unter [AWS Glue Schema Registry](#). Alternativ können Sie diese Integration über die Bibliotheken KPL und Kinesis Client Library (KCL) in Java einrichten.

### Important

Derzeit wird die Integration der Schemaregistrierung von Kinesis Data Streams und AWS Glue nur für die Kinesis-Datenströme unterstützt, die in Java implementierte KPL-Produzenten verwenden. Mehrsprachige Unterstützung wird nicht bereitgestellt.

Ausführliche Anweisungen zum Einrichten der Integration von Kinesis Data Streams mit Schema Registry mithilfe der KPL finden Sie im Abschnitt „Interaktion mit Daten mithilfe der KPL/KCL-

Bibliotheken“ unter [Anwendungsfall: Integration von Amazon Kinesis Data Streams mit der AWS Glue Schema Registry](#).

## KPL-Proxy-Konfiguration

Für Anwendungen, die keine direkte Verbindung zum Internet herstellen können, unterstützen alle AWS SDK-Clients die Verwendung von HTTP- oder HTTPS-Proxys. In einer typischen Unternehmensumgebung muss der gesamte ausgehende Netzwerkverkehr über Proxyserver geleitet werden. Wenn Ihre Anwendung die Kinesis Producer Library (KPL) verwendet, um Daten zu sammeln und an AWS in einer Umgebung zu senden, die Proxyserver verwendet, benötigt Ihre Anwendung eine KPL-Proxy-Konfiguration. KPL ist eine High-Level-Bibliothek, die auf dem AWS Kinesis SDK basiert. Sie ist in einen systemeigenen Prozess und einen Wrapper aufgeteilt. Der native Prozess führt alle Aufgaben der Verarbeitung und des Sendens von Datensätzen aus, während der Wrapper den nativen Prozess verwaltet und mit ihm kommuniziert. Weitere Informationen finden Sie unter [Implementieren effizienter und zuverlässiger Produzenten mit der Amazon Kinesis Producer Library](#).

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

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

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

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

# Entwicklung von Produzenten mithilfe der API für Amazon Kinesis Data Streams mit der AWS SDK for Java

Sie können Produzenten entwickeln, die die API von Amazon Kinesis Data Streams mit dem AWS SDK für Java verwenden. Wenn Sie Kinesis Data Streams zum ersten Mal verwenden, sollten Sie sich zunächst mit den Konzepten und der Terminologie in [Was ist Amazon Kinesis Data Streams?](#) und [Erste Schritte mit Amazon Kinesis Data Streams](#) vertraut machen.

In den Beispielen wird die [API für Kinesis Data Streams](#) behandelt und das [AWS-SDK for Java](#) eingesetzt, um Daten zu einem Stream hinzuzufügen. Für die meisten Anwendungsfälle sollten Sie jedoch die KPL-Bibliothek von Kinesis Data Streams verwenden. Weitere Informationen finden Sie unter [Entwickeln von Produzenten mit der Amazon Kinesis Producer Library](#).

Anhand des Java-Beispiel-Codes in diesem Abschnitt, der nach Operationstyp logisch unterteilt ist, wird gezeigt, wie Sie grundlegende API-Vorgänge mit Kinesis Data Streams durchführen. Diese Beispiele stellen keinen produktionsbereiten Code dar, d. h. es werden nicht alle möglichen Ausnahmen geprüft und es werden nicht alle möglichen Sicherheits- oder Leistungsüberlegungen berücksichtigt. Sie können zudem die [API für Kinesis Data Streams](#) mit anderen Programmiersprachen aufrufen. Weitere Informationen über verfügbare AWS-SDKs finden Sie unter [Entwickeln Sie Ihre Apps mit Amazon Web Services](#).

Für jede Aufgabe gibt es Voraussetzungen. So können Sie beispielsweise erst dann Daten zu einem Stream hinzufügen, wenn Sie einen erstellt haben. Deshalb müssen Sie einen Client anlegen. Weitere Informationen finden Sie unter [Erstellen und Verwalten von Streams](#).

## Themen

- [Hinzufügen von Daten zu einem Stream](#)
- [Interaktion mit Daten mithilfe des AWS Glue Schema-Registry](#)

## Hinzufügen von Daten zu einem Stream

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

Die API für Kinesis Data Streams unterstützt das Hinzufügen von Daten zu einem Stream mittels zweier Operationen: [PutRecords](#) und [PutRecord](#). Die Operation `PutRecords` sendet per HTTP-Anforderung mehrere Datensätze an Ihren Stream und die Operation `PutRecord` sendet immer nur einen Datensatz, dabei ist für jeden Datensatz eine separate HTTP-Anforderung erforderlich. Für die meisten Anwendungen sollten Sie `PutRecords` bevorzugen, da diese Operation pro Datenproduzent einen höheren Durchsatz erzielt. Weitere Informationen zu diesen Operationen finden Sie unten in separaten Unterabschnitten.

## Themen

- [Hinzufügen mehrerer Datensätze mit PutRecords](#)
- [Hinzufügen eines einzelnen Datensatzes mit PutRecord](#)

Beachten Sie Folgendes: Wenn Ihre Quellanwendung Daten mit der API für Kinesis Data Streams zum Stream hinzufügt, gibt es vermutlich eine oder mehrere Konsumenten Anwendungen, die gleichzeitig Daten aus dem Stream verarbeiten. Weitere Informationen dazu, wie Konsumenten Daten über die API für Kinesis Data Streams abrufen, erhalten Sie unter [Abrufen von Daten aus einem Stream](#).

### Important

[Ändern des Zeitraums der Datenaufbewahrung](#)

## Hinzufügen mehrerer Datensätze mit PutRecords

Die [PutRecords](#)-Operation sendet in einer einzelnen Anforderung mehrere Datensätze an Kinesis Data Streams. Mit `PutRecords` erzielen Produzenten einen höheren Durchsatz, wenn sie Daten an ihren Kinesis-Datenstrom senden. Jede `PutRecords`-Anfrage kann bis zu 500 Datensätze unterstützen. Die maximale Größe jedes Datensatzes in der Anforderung beträgt 1 MB bis zu einem Limit von 5 MB für die gesamte Anforderung einschließlich Partitionsschlüsseln. Wie bei der unten beschriebenen einzelnen `PutRecord`-Operation nutzt `PutRecords` Sequenznummern und Partitionsschlüssel. Der `PutRecord`-Parameter `SequenceNumberForOrdering` ist jedoch nicht in einem `PutRecords`-Aufruf enthalten. Die `PutRecords`-Operation versucht, alle Datensätze in der natürlichen Reihenfolge der Anforderung zu verarbeiten.

Jeder Datensatz hat eine eindeutige Sequenznummer. Die Sequenznummer wird von Kinesis Data Streams zugewiesen, nachdem Sie `client.putRecords` aufgerufen haben, um die Datensätze

zum Stream hinzuzufügen. Sequenznummern für denselben Partitionsschlüssel steigen in der Regel im Laufe der Zeit; je länger die Zeitdauer zwischen PutRecords-Anforderungen ist, desto größer wird die Sequenznummer.

### Note

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

Eine PutRecords-Anforderung kann Datensätze mit unterschiedlichen Partitionsschlüsseln enthalten. Der Anforderung gilt für einen Stream. Jede Anforderung kann eine beliebige Kombination aus Partitionsschlüsseln und Datenschlüsseln enthalten, sofern die Grenzwerte der Anforderung nicht überschritten werden. Anforderungen mit vielen verschiedenen Partitionsschlüsseln an Streams mit vielen verschiedenen Shards sind im Allgemeinen schneller als Anforderungen mit wenigen Partitionsschlüsseln an Streams mit nur wenigen Shards. Die Anzahl der Partitionsschlüssel sollte viel größer sein als die Anzahl der Shards, um Latenzzeiten zu verkürzen und einen maximalen Durchsatz zu erzielen.

### PutRecords – Beispiel

Der folgende Code erstellt 100 Datensätze mit sequenziellen Partitionsschlüsseln und übergibt diese an einen Stream namens `DataStream`.

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

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

AmazonKinesis kinesisClient = clientBuilder.build();

PutRecordsRequest putRecordsRequest = new PutRecordsRequest();
putRecordsRequest.setStreamName(streamName);
List <PutRecordsRequestEntry> putRecordsRequestEntryList = new ArrayList<>();
for (int i = 0; i < 100; i++) {
    PutRecordsRequestEntry putRecordsRequestEntry = new
PutRecordsRequestEntry();
```

```
putRecordsRequestEntry.setData(ByteBuffer.wrap(String.valueOf(i).getBytes()));
    putRecordsRequestEntry.setPartitionKey(String.format("partitionKey-%d",
i));
    putRecordsRequestEntryList.add(putRecordsRequestEntry);
}

putRecordsRequest.setRecords(putRecordsRequestEntryList);
PutRecordsResult putRecordsResult =
kinesisClient.putRecords(putRecordsRequest);
System.out.println("Put Result" + putRecordsResult);
```

Die PutRecords-Antwort umfasst ein Array von Records-Antworten. Jeder Eintrag im Antwort-Array korreliert direkt mit einem Eintrag im Anforderungs-Array und zwar in natürlicher Reihenfolge von oben nach unten wie in der Anforderung und der Antwort. Das Records-Antwort-Array enthält stets die gleiche Anzahl an Datensätzen wie das Anforderungs-Array.

### Fehlerbehandlung bei der Verwendung von PutRecords

Standardmäßig führt ein Fehler bei einzelnen Datensätzen innerhalb einer Anforderung nicht zur Beendigung der Verarbeitung nachfolgender Datensätze in einer PutRecords-Anforderung. Das bedeutet, dass ein Records-Antwort-Array sowohl erfolgreich verarbeitete als auch nicht erfolgreich verarbeitete Datensätze enthält. Sie müssen nicht erfolgreich verarbeitete Datensätze erkennen und in einen nachfolgenden Aufruf aufnehmen.

Erfolgreich verarbeitete Datensätze enthalten SequenceNumber- und ShardID-Werte, nicht erfolgreiche verarbeitete Datensätze enthalten ErrorCode- und ErrorMessage-Werte. Der ErrorCode-Parameter gibt den Fehlertyp an. Folgende Werte sind möglich: ProvisionedThroughputExceededException oder InternalFailure. ErrorMessage enthält weitere Informationen zur ProvisionedThroughputExceededException-Ausnahme, einschließlich Konto-ID, Stream-Name und Shard-ID des gedrosselten Datensatzes. Im folgenden Beispiel umfasst eine PutRecords-Anforderung drei Datensätze. Der zweite Datensatz schlägt fehl und wird in der Antwort angegeben.

### Example PutRecords-Anforderungssyntax

```
{
  "Records": [
    {
      "Data": "XzXkYXRhPl8w",
      "PartitionKey": "partitionKey1"
```

```

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

```

### Example PutRecords-Antwortsyntax

```

{
  "FailedRecordCount": 1,
  "Records": [
    {
      "SequenceNumber": "21269319989900637946712965403778482371",
      "ShardId": "shardId-000000000001"
    },
    {
      "ErrorCode": "ProvisionedThroughputExceededException",
      "ErrorMessage": "Rate exceeded for shard shardId-000000000001 in stream exampleStreamName under account 111111111111."
    },
    {
      "SequenceNumber": "21269319989999637946712965403778482985",
      "ShardId": "shardId-000000000002"
    }
  ]
}

```

Datensätze, die nicht erfolgreich verarbeitet wurden, können in nachfolgende PutRecords-Anforderungen aufgenommen werden. Überprüfen Sie zuerst den Parameter `FailedRecordCount` im `putRecordsResult`, um zu bestätigen, ob Datensätze in der Anforderung fehlgeschlagen sind. Wenn dies der Fall ist, darf jeder `putRecordsEntry`, dessen `errorCode` nicht `null` ist, nicht in der nächsten Anforderung hinzugefügt werden. Ein Beispiel für diese Art von Handler finden Sie in folgendem Code:



## Example PutRecords-Ausfall-Handler

```
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);
}
```

## Hinzufügen eines einzelnen Datensatzes mit PutRecord

Jeder Aufruf von [PutRecord](#) wird auf einem einzelnen Datensatz ausgeführt. Bevorzugen Sie die PutRecords-Operation, die in [Hinzufügen mehrerer Datensätze mit PutRecords](#) beschrieben ist, es sei denn, Ihre Anwendung muss stets einzelne Datensätze per Anforderung senden oder es liegen andere Gründe vor, die gegen den Einsatz von PutRecords sprechen.

Jeder Datensatz hat eine eindeutige Sequenznummer. Die Sequenznummer wird von Kinesis Data Streams zugewiesen, nachdem Sie `client.putRecord` aufgerufen haben, um den Datensatz zum

Stream hinzuzufügen. Sequenznummern für denselben Partitionsschlüssel steigen in der Regel im Laufe der Zeit; je länger die Zeitdauer zwischen `PutRecord`-Anforderungen ist, desto größer wird die Sequenznummer.

Wenn nacheinander viele `PUT`-Operationen durchgeführt werden, steigen die zurückgegebenen Sequenznummern nicht zwangsläufig an, da Kinesis Data Streams diese als gleichzeitig interpretiert. Um steigende Sequenznummern für denselben Partitionsschlüssel sicherzustellen, sollten Sie den `SequenceNumberForOrdering`-Parameter so verwenden, wie es im [PutRecord – Beispiel](#)-Codebeispiel gezeigt wird.

Unabhängig davon, ob Sie `SequenceNumberForOrdering` verwenden, werden Datensätze, die Kinesis Data Streams über einen `GetRecords`-Aufruf empfängt, streng nach Sequenznummern sortiert.

#### Note

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

Ein Partitionsschlüssel wird verwendet, um Daten in einem Stream zu gruppieren. Ein Datensatz wird einem Shard innerhalb des Streams basierend auf dessen Partitionsschlüssel zugewiesen. Insbesondere Kinesis Data Streams verwendet den Partitionsschlüssel als Eingabe für eine Hash-Funktion, die den Partitionsschlüssel (und die zugehörigen Daten) einem bestimmten Shard zuordnet.

Aufgrund dieses Hashing-Mechanismus werden alle Datensätze mit demselben Partitionsschlüssel zum selben Shard im Stream zugeordnet. Wenn jedoch die Anzahl der Partitionsschlüssel die Anzahl der Shards übersteigt, enthalten einige Shards zwangsläufig Datensätze mit unterschiedlichen Partitionsschlüsseln. Vom Design-Standpunkt aus sollte die Anzahl der Shards (die über die `setShardCount`-Methode von `CreateStreamRequest` angegeben wird) wesentlich geringer sein als die Anzahl der eindeutigen Partitionsschlüssel und das Datenvolumen für einen einzelnen Partitionsschlüssel sollte wesentlich geringer sein, als die Shard-Kapazität, um sicherzustellen, dass alle Shards gut ausgelastet sind.

#### PutRecord – Beispiel

Der folgende Code erstellt 10 Datensätze, die auf zwei Partitionsschlüssel verteilt werden, und fügt diese einem Stream namens `myStreamName` hinzu.

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

Beim zuvor beschriebenen Codebeispiel wird `setSequenceNumberForOrdering` eingesetzt, um eine aufsteigende Reihenfolge innerhalb der einzelnen Partitionsschlüssel zu gewährleisten. Für eine effektive Verwendung des Parameters setzen Sie die `SequenceNumberForOrdering` des aktuellen Datensatzes (Datensatz *n*) auf die Sequenznummer des vorhergehenden Datensatzes (Datensatz *n-1*). Rufen Sie `getSequenceNumber` auf dem Ergebnis von `putRecord` auf, um die Sequenznummer des Datensatzes zu erhalten, der zum Stream hinzugefügt wurde.

Der `SequenceNumberForOrdering`-Parameter stellt strikt aufsteigende Sequenznummern für denselben Partitionsschlüssel sicher. `SequenceNumberForOrdering` stellt keine Datensatzsortierung bei mehreren Partitionsschlüsseln bereit.

## Interaktion mit Daten mithilfe des AWS Glue Schema-Registry

Sie können Ihre Kinesis-Datenströme in das AWS Glue Schema Registry integrieren. Mit dem AWS Glue Schema Registry können Sie Schemata zentral erkennen, steuern und weiterentwickeln und gleichzeitig sicherstellen, dass die erstellten Daten kontinuierlich durch ein registriertes Schema validiert werden. Ein Schema definiert die Struktur und das Format eines Datensatzes. Ein Schema ist eine versionierte Spezifikation für zuverlässige Datenveröffentlichung, -nutzung oder -speicherung. Mit dem AWS Glue Schema Registry können Sie die durchgängige Datenqualität und Datenverwaltung in Ihren Streaming-Anwendungen verbessern. Weitere Informationen finden Sie unter [AWS Glue Schema Registry](#). Eine der Möglichkeiten, diese Integration einzurichten, sind die APIs `PutRecords` und `PutRecord` von Kinesis Data Streams, die im AWS-Java SDK verfügbar sind.

Detaillierte Anweisungen zur Einrichtung der Integration von Kinesis Data Streams mit Schema Registry mithilfe der APIs `PutRecords` und `PutRecord` Kinesis Data Streams finden Sie im Abschnitt

„Interaktion mit Daten mithilfe der APIs für Kinesis Data Streams“ unter [Anwendungsfall: Integration von Amazon Kinesis Data Streams mit der AWS Glue Schema Registry](#).

## Schreiben in Amazon Kinesis Data Streams mit Kinesis Agent

Kinesis Agent ist eine eigenständige Java-Anwendung, mit der Sie Daten einfach erfassen und an Kinesis Data Streams senden können. Der Agent überwacht kontinuierlich eine Reihe von Dateien und sendet neue Daten an Ihren Stream. Der Agent übernimmt die Dateirotation, das Checkpointing und die Wiederholung bei Fehlern. Er sorgt für eine zuverlässige, zeitgerechte und einfache Bereitstellung Ihrer Daten. Es gibt auch Amazon CloudWatch-Metriken aus, die Ihnen helfen, den Streaming-Prozess besser zu überwachen und Fehler zu beheben.

Standardmäßig werden Datensätze aus den einzelnen Dateien anhand des Zeilenumbruchzeichens ('`\n`') analysiert. Der Agent kann jedoch auch für die Analyse mehrzeiliger Datensätze konfiguriert werden (siehe [Konfigurationseinstellungen für den Agenten](#)).

Sie können den Agenten in Linux-Serverumgebungen installieren, beispielsweise auf Webservern, Protokollservern und Datenbankservern. Nach der Installation des Agenten konfigurieren Sie diesen, indem Sie die zu überwachenden Dateien sowie den Stream für die Daten angeben. Nachdem der Agent konfiguriert wurde, sammelt er permanent Daten aus den Dateien und sendet sie zuverlässig an den Stream.

### Themen

- [Voraussetzungen](#)
- [Herunterladen und Installieren des Agenten](#)
- [Konfigurieren und Starten des Agenten](#)
- [Konfigurationseinstellungen für den Agenten](#)
- [Überwachen mehrerer Dateiverzeichnisse und Schreiben in mehrere Streams](#)
- [Verwenden des Agenten zur Datenvorverarbeitung](#)
- [CLI-Befehle des Agenten](#)
- [Häufig gestellte Fragen](#)

## Voraussetzungen

- Sie benötigen entweder ein Amazon Linux AMI-Betriebssystem, Version 2015.09 oder höher, oder Version 7 oder höher von Red Hat Enterprise Linux.

- Starten Sie Ihre EC2 Instance, wenn Sie Amazon EC2 verwenden, um den Agenten auszuführen.
- Verwalten Sie Ihre AWS Anmeldeinformationen mit einer der folgenden Methoden:
  - Geben Sie eine IAM-Rolle an, wenn Sie Ihre EC2 Instance starten.
  - Geben Sie AWS Anmeldeinformationen an, wenn Sie den Agenten konfigurieren (siehe [awsAccessKeyId](#) und [awsSecretAccessSchlüssel](#)).
  - Bearbeiten Sie `/etc/sysconfig/aws-kinesis-agent`, um Ihre Region und AWS Zugriffsschlüssel anzugeben.
  - Wenn sich Ihre EC2-Instance in einem anderen AWS Konto befindet, erstellen Sie eine IAM-Rolle, um Zugriff auf den Kinesis Data Streams-Service zu gewähren, und geben Sie diese Rolle an, wenn Sie den Agenten konfigurieren (siehe [assumeRoleARN](#) und [assumeRoleExternalId](#)). Verwenden Sie eine der vorherigen Methoden, um die AWS Anmeldeinformationen eines Benutzers in dem anderen Konto anzugeben, der berechtigt ist, diese Rolle zu übernehmen.
- Die von Ihnen angegebene IAM-Rolle oder AWS Anmeldeinformationen müssen über die Berechtigung zum Ausführen der Kinesis-Data-Streams-[PutRecords](#) Operation verfügen, damit der Agent Daten an Ihren Stream senden kann. Wenn Sie die CloudWatch Überwachung für den Agenten aktivieren, ist auch die Berechtigung zum Ausführen des CloudWatch [PutMetricData](#) Vorgangs erforderlich. Weitere Informationen finden Sie unter [Steuern des Zugriffs auf Ressourcen von Amazon Kinesis Data Streams mithilfe von IAM](#), [Überwachung von Kinesis Data Streams Agent Health für Amazon CloudWatch](#) und [CloudWatch Zugriffskontrolle](#).

## Herunterladen und Installieren des Agenten

Stellen Sie zunächst eine Verbindung mit Ihrer Instance her. Weitere Informationen finden Sie unter [Verbinden mit Ihrer Instance](#) im Amazon-EC2-Benutzerhandbuch für Linux-Instances. Weitere Informationen zur [Behebung von Problemen beim Herstellen einer Verbindung mit Ihrer Instance](#) finden Sie im Amazon-EC2-Benutzerhandbuch für Linux-Instances.

So richten Sie den Agenten mithilfe der Amazon Linux AMI ein

Verwenden Sie den folgenden Befehl zum Herunterladen und Installieren des Agenten:

```
sudo yum install -y aws-kinesis-agent
```

So richten Sie den Agenten mit Red Hat Enterprise Linux ein

Verwenden Sie den folgenden Befehl zum Herunterladen und Installieren des Agenten:

```
sudo yum install -y https://s3.amazonaws.com/streaming-data-agent/aws-kinesis-agent-latest.amzn2.noarch.rpm
```

So richten Sie den Agenten mit ein GitHub

1. Laden Sie den Agenten von [awslabs/amazon-kinesis-agent](https://github.com/awslabs/amazon-kinesis-agent) herunter.
2. Installieren Sie den Agenten, indem Sie zum Download-Verzeichnis navigieren und den folgenden Befehl ausführen:

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

So richten Sie den Agenten in einem Docker-Container ein

Kinesis Agent kann auch in einem Container über die [amazonlinux](https://github.com/awslabs/amazonlinux)-Containerbasis ausgeführt werden. Verwenden Sie die folgende Docker-Datei und führen Sie dann `docker build` aus.

```
FROM amazonlinux

RUN yum install -y aws-kinesis-agent which findutils
COPY agent.json /etc/aws-kinesis/agent.json

CMD ["start-aws-kinesis-agent"]
```

## Konfigurieren und Starten des Agenten

So konfigurieren und starten Sie den Agenten

1. Öffnen und bearbeiten Sie die Konfigurationsdatei (als Superuser, wenn Sie standardmäßige Dateizugriffsberechtigungen nutzen): `/etc/aws-kinesis/agent.json`

Geben Sie in der Konfigurationsdatei die Dateien ("`filePattern`") an, aus denen der Agent Daten sammelt, sowie den Namen des Streams ("`kinesisStream`", an den der Agent die Daten sendet. Beachten Sie, dass der Dateiname ein Muster ist und der Agent Dateierotationen erkennt. Sie können nur einmal pro Sekunde Dateien rotieren oder neue Dateien erstellen. Der Agent verwendet den Zeitstempel der Dateierstellung, um die Dateien zu ermitteln, die gesammelt und zum Stream hinzugefügt werden. Wenn häufiger als einmal pro Sekunde Dateien rotiert oder neue Dateien erstellt werden, kann der Agent nicht mehr ordnungsgemäß zwischen den Dateien unterscheiden.

```
{
  "flows": [
    {
      "filePattern": "/tmp/app.log*",
      "kinesisStream": "yourkinesisstream"
    }
  ]
}
```

2. Starten Sie den Agenten manuell:

```
sudo service aws-kinesis-agent start
```

3. (Optional) Konfigurieren Sie den Agenten so, dass er beim Startup des Systems gestartet wird:

```
sudo chkconfig aws-kinesis-agent on
```

Der Agent wird jetzt als Systemdienst im Hintergrund ausgeführt. Er überwacht kontinuierlich die angegebenen Dateien und sendet Daten an den angegebenen Stream. Die Agentenaktivität wird in `/var/log/aws-kinesis-agent/aws-kinesis-agent.log` protokolliert.

## Konfigurationseinstellungen für den Agenten

Der Agent unterstützt die beiden obligatorischen Konfigurationseinstellungen `filePattern` und `kinesisStream` sowie optionale Konfigurationseinstellungen für zusätzliche Funktionen. Sie können sowohl die obligatorische als auch die optionale Konfiguration in `/etc/aws-kinesis/agent.json` angeben.

Wenn Sie die Konfigurationsdatei ändern, müssen Sie den Agenten mit den folgenden Befehlen anhalten und starten:

```
sudo service aws-kinesis-agent stop
sudo service aws-kinesis-agent start
```

Alternativ können Sie auch den folgenden Befehl nutzen:

```
sudo service aws-kinesis-agent restart
```

Im Folgenden finden Sie die allgemeinen Konfigurationseinstellungen.

Konfigurationseinstellung	Beschreibung
<code>assumeRoleARN</code>	Der ARN der Rolle, die vom Benutzer übernommen werden muss. Weitere Informationen finden Sie unter <a href="#">Delegieren des Zugriffs in allen - AWS Konten mithilfe von IAM-Rollen</a> im IAM-Benutzerhandbuch.
<code>assumeRoleExternalId</code>	Eine optionale Kennung, die festlegt, wer die Rolle übernehmen kann. Weitere Informationen finden Sie unter <a href="#">Verwendung einer externen ID</a> im IAM-Benutzerhandbuch.
<code>awsAccessKeyId</code>	AWS -Zugriffsschlüssel-ID, die die Standardanmeldeinformationen überschreibt. Diese Einstellung hat Vorrang vor allen anderen Anbietern von Anmeldeinformationen.
<code>awsSecretAccessKey</code>	AWS Geheimer Schlüssel, der die Standardanmeldeinformationen überschreibt. Diese Einstellung hat Vorrang vor allen anderen Anbietern von Anmeldeinformationen.
<code>cloudwatch.emitMetrics</code>	Ermöglicht es dem Agenten, Metriken an auszugeben CloudWatch , falls festgelegt (true).  Standard: true
<code>cloudwatch.endpoint</code>	Der regionale Endpunkt für CloudWatch.  Standard: <code>monitoring.us-east-1.amazonaws.com</code>
<code>kinesis.endpoint</code>	Der regionale Endpunkt für Kinesis Data Streams.  Standard: <code>kinesis.us-east-1.amazonaws.com</code>

Im Folgenden finden Sie die Konfigurationseinstellungen für den Ablauf.



Konfigurationseinstellung	Beschreibung
dataProcessingOptions	Die Liste der Verarbeitungsoptionen, die auf jeden analysierten Datensatz angewendet werden, ehe ein Datensatz an den Stream gesendet wird. Die Verarbeitungsoptionen werden in der angegebenen Reihenfolge ausgeführt. Weitere Informationen finden Sie unter <a href="#">Verwenden des Agenten zur Datenvorverarbeitung</a> .
kinesisStream	[Erforderlich] Der Name des Streams.
filePattern	[Erforderlich] Das Verzeichnis und das Dateimuster, die übereinstimmen müssen, um vom Agenten aufgenommen zu werden. Für alle Dateien, die mit diesem Muster übereinstimmen, müssen Leseberechtigungen für <code>aws-kinesis-agent-user</code> vergeben werden. Für das die Dateien enthaltende Verzeichnis müssen Lese- und Ausführberechtigungen für <code>aws-kinesis-agent-user</code> vergeben werden.
initialPosition	Die Position, an der mit der Analyse der Datei begonnen wurde. Gültige Werte sind <code>START_OF_FILE</code> und <code>END_OF_FILE</code> .  Standard: <code>END_OF_FILE</code>
maxBufferAgeMillis	Die maximale Zeit (in Millisekunden), während der durch den Agenten Daten gepuffert werden, bevor sie an den Stream gesendet werden.  Wertebereich: 1.000 bis 900.000 (1 Sekunde bis 15 Minuten)  Standard: 60.000 (1 Minute)
maxBufferSizeBytes	Die maximale Größe (in Bytes), bis zu der durch den Agenten Daten gepuffert werden, bevor sie an den Stream gesendet werden.  Wertebereich: 1 bis 4.194.304 (4 MB)  Standard: 4.194.304 (4 MB)
maxBufferSizeRecords	Die maximale Anzahl der Datensätze, für die der Agent Daten puffert, ehe diese an den Stream gesendet werden.

Konfigurationseinstellung	Beschreibung
	<p>Wertebereich: 1 bis 500</p> <p>Standard: 500</p>
<code>minTimeBetweenFilePollsMillis</code>	<p>Das Zeitintervall (in Millisekunden), in dem der Agent die überwachten Dateien auf neue Daten abfragt und analysiert.</p> <p>Wertebereich: 1 oder höher</p> <p>Standard: 100</p>
<code>multilineStartPattern</code>	<p>Das Muster für die Identifizierung des Datensatzbeginns. Ein Datensatz besteht aus einer Zeile, die mit dem angegebenen Muster übereinstimmt, und allen folgenden Zeilen, die nicht dem Muster entsprechen. Gültige Werte sind reguläre Ausdrücke. Standardmäßig wird jede neue Zeile in den Protokolldateien als einziger Datensatz analysiert.</p>
<code>partitionKeyOption</code>	<p>Die Methode zum Erstellen des Partitionsschlüssels. Gültige Werte sind <code>RANDOM</code> (zufällig generierte Ganzzahl) und <code>DETERMINISTIC</code> (ein aus den Daten berechneter Hash-Wert).</p> <p>Standard: <code>RANDOM</code></p>
<code>skipHeaderLines</code>	<p>Die Anzahl der Zeilen, die der Agent überspringt, ehe mit der Analyse der überwachten Dateien begonnen wird.</p> <p>Wertebereich: 0 oder höher</p> <p>Standard: 0 (null)</p>
<code>truncatedRecord Terminator</code>	<p>Die Zeichenfolge, mit der der Agent einen analysierten Datensatz kürzt, wenn die Datensatzgröße das zulässige Datensatz-Limit von Kinesis Data Streams überschreitet. (1,000 KB)</p> <p>Standard: <code>'\n'</code> (Zeilenumbruch)</p>

## Überwachen mehrerer Dateiverzeichnisse und Schreiben in mehrere Streams

Wenn Sie mehrere Ablaufkonfigurationseinstellungen angeben, können Sie den Agenten so konfigurieren, dass er mehrere Dateiverzeichnisse überwacht und Daten an verschiedene Streams sendet. Im folgenden Konfigurationsbeispiel überwacht der Agent zwei Dateiverzeichnisse und sendet Daten an einen Kinesis-Stream bzw. einen Firehose-Bereitstellungs-Stream. Beachten Sie, dass Sie unterschiedliche Endpunkte für Kinesis Data Streams und Firehose angeben können, sodass sich Ihr Kinesis-Stream und Firehose-Bereitstellungs-Stream nicht in derselben Region befinden müssen.

```
{
  "cloudwatch.emitMetrics": true,
  "kinesis.endpoint": "https://your/kinesis/endpoint",
  "firehose.endpoint": "https://your/firehose/endpoint",
  "flows": [
    {
      "filePattern": "/tmp/app1.log*",
      "kinesisStream": "yourkinesisstream"
    },
    {
      "filePattern": "/tmp/app2.log*",
      "deliveryStream": "yourfirehosedeliverystream"
    }
  ]
}
```

Ausführlichere Informationen zur Verwendung des Agenten mit Firehose finden Sie unter [Schreiben in Amazon Data Firehose mit Kinesis Agent](#).

## Verwenden des Agenten zur Datenvorverarbeitung

Der Agent kann die Datensätze vorverarbeiten, die aus den überwachten Dateien analysiert wurden, ehe diese an Ihren Stream gesendet werden. Sie können dieses Feature aktivieren, indem Sie Ihrem Dateifluss die Konfigurationseinstellung `dataProcessingOptions` hinzufügen. Sie können eine oder mehrere Verarbeitungsoptionen hinzufügen. Diese werden in der angegebenen Reihenfolge ausgeführt.

Der Agent unterstützt die folgenden aufgelisteten Verarbeitungsoptionen. Der Agent ist ein Open-Source-Tool, sodass Sie dessen Verarbeitungsoptionen optimieren und erweitern können. Sie können den Agenten von [Kinesis Agent](#) herunterladen.

## Verarbeitungsoptionen

### SINGLELINE

Konvertiert einen mehrzeiligen Datensatz in einen einzeiligen Datensatz, indem Zeilenumbruchzeichen sowie vorangestellte und folgende Leerzeichen entfernt werden.

```
{
  "optionName": "SINGLELINE"
}
```

### CSVTOJSON

Konvertiert einen Datensatz aus dem durch Trennzeichen getrennten Format in einen Datensatz im JSON-Format.

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

#### customFieldNames

[Erforderlich] Die Feldnamen, die als Schlüssel in den einzelnen JSON-Schlüssel-Wert-Paaren verwendet werden. Wenn Sie beispielsweise [ "f1", "f2" ] angeben, wird der Datensatz „v1, v2“ in { "f1": "v1", "f2": "v2" } konvertiert.

#### delimiter

Die Zeichenfolge, die als Trennzeichen im Datensatz verwendet wird. Standardmäßig wird ein Komma (,) verwendet.

### LOGTOJSON

Konvertiert einen Datensatz aus einem Protokollformat in einen Datensatz im JSON-Format. Folgende Protokollformate werden unterstützt: Apache Common Log, Apache Combined Log, Apache Error Log und RFC3164 Syslog.

```
{
  "optionName": "LOGTOJSON",
  "logFormat": "logformat",
  "matchPattern": "yourregexpattern",
  "customFieldNames": [ "field1", "field2", ... ]
}
```

## logFormat

[Erforderlich] Das Format des Protokolleintrags. Folgende Werte sind möglich:

- COMMONAPACHELOG – Das Apache-Common-Log-Format. Jeder Protokolleintrag weist standardmäßig das folgende Muster auf: „`%{host} %{ident} %{authuser} [%{datetime}] \"%{request}\" %{response} %{bytes}`“.
- COMBINEDAPACHELOG – Das Apache-Combined-Log-Format. Jeder Protokolleintrag weist standardmäßig das folgende Muster auf: „`%{host} %{ident} %{authuser} [%{datetime}] \"%{request}\" %{response} %{bytes} %{referrer} %{agent}`“.
- APACHEERRORLOG – Das Apache-Error-Log-Format. Jeder Protokolleintrag weist standardmäßig das folgende Muster auf: „`[%{timestamp}] [%{module}: %{severity}] [pid %{processid}:tid %{threadid}] [client: %{client}] %{message}`“.
- SYSLOG – Das RFC3164-Syslog-Format. Jeder Protokolleintrag weist standardmäßig das folgende Muster auf: „`%{timestamp} %{hostname} %{program}[%{processid}]: %{message}`“.

## matchPattern

Das reguläre Ausdrucksmuster, mit dem Werte aus den Protokolleinträgen extrahiert werden. Diese Einstellung wird verwendet, wenn Ihr Protokolleintrag nicht eines der vordefinierten Protokollformate aufweist. Bei dieser Einstellung müssen Sie auch `customFieldNames` angeben.

## customFieldNames

Die benutzerdefinierten Feldnamen, die als Schlüssel in den einzelnen JSON-Schlüssel-Wert-Paaren verwendet werden. Mit dieser Einstellung können Sie Feldnamen für Werte definieren, die aus `matchPattern` extrahiert wurden, oder die Standardfeldnamen von vordefinierten Protokollformaten überschreiben.

## Example : LOGTOJSON-Konfiguration

Nachfolgend ein Beispiel einer LOGTOJSON-Konfiguration für einen Apache Common Log-Eintrag, der in ein JSON-Format konvertiert wurde:

```
{
  "optionName": "LOGTOJSON",
  "logFormat": "COMMONAPACHELOG"
}
```

Vor der Konvertierung:

```
64.242.88.10 - - [07/Mar/2004:16:10:02 -0800] "GET /mailman/listinfo/hsdivision
HTTP/1.1" 200 6291
```

Nach der Konvertierung:

```
{"host":"64.242.88.10","ident":null,"authuser":null,"datetime":"07/
Mar/2004:16:10:02 -0800","request":"GET /mailman/listinfo/hsdivision
HTTP/1.1","response":"200","bytes":"6291"}
```

## Example : LOGTOJSON-Konfiguration mit benutzerdefinierten Feldern

Im Folgenden ein weiteres Beispiel einer LOGTOJSON-Konfiguration:

```
{
  "optionName": "LOGTOJSON",
  "logFormat": "COMMONAPACHELOG",
  "customFieldNames": ["f1", "f2", "f3", "f4", "f5", "f6", "f7"]
}
```

Durch diese Konfigurationseinstellung wird der Apache Common Log-Eintrag aus dem vorherigen Beispiel wie folgt in ein JSON-Format konvertiert:

```
{"f1":"64.242.88.10","f2":null,"f3":null,"f4":"07/Mar/2004:16:10:02 -0800","f5":"GET /
mailman/listinfo/hsdivision HTTP/1.1","f6":"200","f7":"6291"}
```

## Example : Konvertieren eines Apache Common Log-Eintrags

Bei der folgenden Ablaufkonfiguration wird ein Apache Common Log-Eintrag in einen einzelnen Datensatz im JSON-Format umgewandelt:

```

{
  "flows": [
    {
      "filePattern": "/tmp/app.log*",
      "kinesisStream": "my-stream",
      "dataProcessingOptions": [
        {
          "optionName": "LOGTOJSON",
          "logFormat": "COMMONAPACHELOG"
        }
      ]
    }
  ]
}

```

### Example : Konvertieren mehrzeiliger Datensätze

Bei der folgenden Ablaufkonfiguration werden mehrzeilige Datensätze analysiert, deren erste Zeile mit „[SEQUENCE=“ beginnt. Jeder Datensatz wird in einen einzeiligen Datensatz konvertiert. Anschließend werden Werte aus dem Datensatz basierend auf einem Tabulatortrennzeichen extrahiert. Die extrahierten Werte werden zu angegebenen `customFieldNames`-Werten zugeordnet und ergeben so einen einzeiligen Datensatz im JSON-Format.

```

{
  "flows": [
    {
      "filePattern": "/tmp/app.log*",
      "kinesisStream": "my-stream",
      "multiLineStartPattern": "\\[SEQUENCE=",
      "dataProcessingOptions": [
        {
          "optionName": "SINGLELINE"
        },
        {
          "optionName": "CSVTOJSON",
          "customFieldNames": [ "field1", "field2", "field3" ],
          "delimiter": "\\t"
        }
      ]
    }
  ]
}

```

## Example : LOGTOJSON-Konfiguration mit Übereinstimmungsmuster

Nachfolgend ein Beispiel einer LOGTOJSON-Konfiguration für einen Apache Common Log-Eintrag, der in das JSON-Format konvertiert wurde. Das letzte Feld (Bytes) wurde ausgelassen:

```
{
  "optionName": "LOGTOJSON",
  "logFormat": "COMMONAPACHELOG",
  "matchPattern": "^(([\\d.]+) (\\S+) (\\S+) \\[[([\\w:/]+\\s[+\\-]\\d{4})\\]\\] \\\"(.+?)\\\" (\\d{3})\"",
  "customFieldNames": ["host", "ident", "authuser", "datetime", "request",
    "response"]
}
```

Vor der Konvertierung:

```
123.45.67.89 - - [27/Oct/2000:09:27:09 -0400] "GET /java/javaResources.html HTTP/1.0"
200
```

Nach der Konvertierung:

```
{"host":"123.45.67.89","ident":null,"authuser":null,"datetime":"27/Oct/2000:09:27:09
-0400","request":"GET /java/javaResources.html HTTP/1.0","response":"200"}
```

## CLI-Befehle des Agenten

Automatisches Startup des Agenten beim Systemstart:

```
sudo chkconfig aws-kinesis-agent on
```

Prüfen des Status des Agenten:

```
sudo service aws-kinesis-agent status
```

Beenden des Agenten:

```
sudo service aws-kinesis-agent stop
```

Auslesen der Protokolldatei des Agenten von diesem Speicherort:



```
/var/log/aws-kinesis-agent/aws-kinesis-agent.log
```

Deinstallieren des Agenten:

```
sudo yum remove aws-kinesis-agent
```

## Häufig gestellte Fragen

Gibt es einen Kinesis Agent für Windows?

[Kinesis Agent für Windows](#) ist eine andere Software als Kinesis Agent für Linux-Plattformen.

Warum verlangsamt sich Kinesis Agent und/oder **RecordSendErrors** nimmt zu?

Dies ist normalerweise auf die Drosselung durch Kinesis zurückzuführen. Überprüfen Sie die `-WriteProvisionedThroughputExceededMetrik` für Kinesis Data Streams oder die `-ThrottledRecordsMetrik` für Firehose-Bereitstellungs-Streams. Jede Erhöhung dieser Metriken von 0 zeigt an, dass die Stream-Grenzwerte erhöht werden müssen. Weitere Informationen finden Sie unter [Kinesis-Datenstrom-Grenzwerte](#) und [Amazon Firehose Delivery Streams](#).

Sobald Sie die Drosselung ausgeschlossen haben, überprüfen Sie, ob der Kinesis Agent so konfiguriert ist, dass er eine große Menge kleiner Dateien durchsucht. Es gibt eine Verzögerung, wenn der Kinesis Agent eine neue Datei überwacht, daher sollte der Kinesis-Agent eine kleine Menge größerer Dateien überwachen. Versuchen Sie, Ihre Protokolldateien in größeren Dateien zusammenzufassen.

Warum erhalte ich **java.lang.OutOfMemoryError** -Ausnahmen?

Kinesis Agent verfügt nicht über genügend Arbeitsspeicher, um seinen aktuellen Workload zu bewältigen. Versuchen Sie, `JAVA_START_HEAP` und `JAVA_MAX_HEAP` in `/usr/bin/start-aws-kinesis-agent` zu erhöhen und den Agenten neu zu starten.

Warum erhalte ich **IllegalStateException : connection pool shut down**-Ausnahmen?

Kinesis Agent verfügt nicht über genügend Verbindungen, um seinen aktuellen Workload zu bewältigen. Versuchen Sie, `maxConnections` und `maxSendingThreads` in den allgemeinen

Konfigurationseinstellungen des Agenten unter `/etc/aws-kinesis/agent.json` zu erhöhen. Der Standardwert für diese Felder ist das 12-fache der verfügbaren Laufzeitprozessoren. Weitere Informationen zu erweiterten Agentenkonfigurationseinstellungen finden Sie unter [AgentConfiguration.java](#).

Wie kann ich ein anderes Problem mit Kinesis Agent beheben?

DEBUG-Level-Protokolle können in `/etc/aws-kinesis/log4j.xml` aktiviert werden.

Wie sollte ich Kinesis Agent konfigurieren?

Je kleiner das `maxBufferSizeBytes`, desto häufiger sendet der Kinesis Agent Daten. Dies kann nützlich sein, da es die Lieferzeit von Datensätzen verkürzt, aber es erhöht auch die Anfragen pro Sekunde an Kinesis.

Warum sendet Kinesis Agent doppelte Datensätze?

Dies ist auf eine Fehlkonfiguration bei der Dateiüberwachung zurückzuführen. Stellen Sie sicher, dass jedes `fileFlow`'s `filePattern` nur einer Datei entspricht. Dies kann auch auftreten, wenn der verwendete `logrotate`-Modus im `copytruncate`-Modus ist. Versuchen Sie, den Modus auf den Standard- oder Erstellungsmodus zu ändern, um Duplikate zu vermeiden. Weitere Informationen zum Umgang mit doppelten Datensätzen finden Sie unter [Umgang mit doppelten Datensätzen](#).

## Schreiben in Kinesis Data Streams mithilfe anderer AWS-Services

Im Folgenden finden Sie eine Liste anderer AWS-Dienste, die direkt in Kinesis Data Streams integriert werden können, um Daten in Kinesis Data Streams zu schreiben:

Themen

- [AWS Amplify](#)
- [Amazon Aurora](#)
- [Amazon CloudFront](#)
- [Amazon CloudWatch Logs](#)
- [Amazon Connect](#)
- [AWS Database Migration Service](#)
- [Amazon DynamoDB](#)

- [Amazon EventBridge](#)
- [AWS IoT Core](#)
- [Amazon Relational Database Service](#)
- [Amazon Pinpoint](#)
- [Amazon Quantum Ledger Database](#)

## AWS Amplify

Sie können Amazon Kinesis Data Streams verwenden, um Daten aus Ihren mit AWS Amplify erstellten mobilen Anwendungen für die Echtzeitverarbeitung zu streamen. Sie können dann Echtzeit-Dashboards erstellen, Ausnahmen erfassen, Warnungen generieren, Empfehlungen erzeugen und in Echtzeit andere geschäftliche oder betriebliche Entscheidungen treffen. Sie können zudem Daten mühelos an andere Services wie Amazon Simple Storage Service, Amazon DynamoDB und Amazon Redshift übertragen.

Weitere Informationen finden Sie unter [Verwenden von Amazon Kinesis](#) im AWS Amplify Developer Center.

## Amazon Aurora

Sie können Amazon Kinesis Data Streams verwenden, um Aktivitäten auf Ihren Amazon Aurora-DB-Clustern zu überwachen. Mithilfe von Database Activity Streams überträgt Ihr Aurora-DB-Cluster Aktivitäten in Echtzeit an einen Amazon Kinesis Data Streams. Anschließend können Sie Anwendungen für das Compliance-Management entwickeln, die diese Aktivitäten nutzen, prüfen und Warnmeldungen generieren. Sie können auch Amazon Firehose verwenden, um die Daten zu speichern.

Weitere Informationen finden Sie unter [Datenbankaktivitätsstreams](#) im Entwicklerhandbuch zu Amazon Aurora.

## Amazon CloudFront

Sie können Amazon Kinesis Data Streams mit CloudFront-Echtzeitprotokollen verwenden und Informationen über Anforderungen an eine Verteilung in Echtzeit abrufen. Sie können dann Ihren eigenen [Kinesis-Datenstrom-Verbraucher](#) erstellen oder Amazon Firehose verwenden, um die Protokolldaten an Amazon Simple Storage Service (Amazon S3), Amazon Redshift, Amazon

OpenSearch Service (OpenSearch Service) oder einen Drittanbieter-Protokollverarbeitungsservice zu senden.

Weitere Informationen finden Sie unter [Echtzeitprotokolle](#) im Amazon CloudFront-Entwicklerhandbuch.

## Amazon CloudWatch Logs

Sie können CloudWatch-Abonnements verwenden, um den Zugriff auf einen Echtzeit-Feed für Protokollereignisse von Amazon CloudWatch Logs zu erhalten. Diese werden dann einem Amazon Kinesis Data Streams für die Verarbeitung, Analyse und das Laden in andere Systeme bereitgestellt.

Weitere Informationen finden Sie unter [Protokolldaten-Verarbeitung in Echtzeit mit Abonnements](#) im Benutzerhandbuch für Amazon CloudWatch Logs.

## Amazon Connect

Sie können Kinesis Data Streams verwenden, um Kontaktdatensätze und Agentenereignisse in Echtzeit aus Ihrer Amazon-Connect-Instance zu exportieren. Sie können auch Datenstreaming von Amazon Connect Customer Profiles aktivieren, um automatisch Updates für einen Kinesis Data Streams über die Erstellung neuer Profile oder Änderungen an bestehenden Profilen zu erhalten.

Anschließend können Sie Verbraucheranwendungen erstellen, um die Daten in Echtzeit zu verarbeiten und zu analysieren. Mithilfe von Kontaktdatensätzen und Kundenprofilen können Sie beispielsweise Ihre Quellsystemdaten wie CRMs und Tools zur Marketingautomatisierung stets auf dem neuesten Stand halten. Mithilfe der Ereignisdaten der Agenten können Sie Dashboards erstellen, die Agenteninformationen und Ereignisse anzeigen und benutzerdefinierte Benachrichtigungen über bestimmte Agentenaktivitäten auslösen.

Weitere Informationen finden Sie unter [Datenstreaming für Ihre Instance](#), [Echtzeit-Export einrichten](#) und [Agenten-Event-Streams](#) im Amazon-Connect-Administratorhandbuch.

## AWS Database Migration Service

Sie können mit AWS Database Migration Service Daten zu einem Amazon Kinesis Data Streams migrieren. Anschließend können Sie Verbraucheranwendungen erstellen, die die Datensätze in Echtzeit verarbeiten. Sie können zudem Daten mühelos an andere Services wie Amazon Simple Storage Service, Amazon DynamoDB und Amazon Redshift übertragen.

Weitere Informationen finden Sie unter [Nutzung von Kinesis Data Streams](#) im Benutzerhandbuch für AWS Database Migration Service.

## Amazon DynamoDB

Sie können Amazon Kinesis Data Streams verwenden, um Änderungen an Amazon DynamoDB zu erfassen. Kinesis Data Streams erfasst Änderungen auf Elementebene in jeder DynamoDB-Tabelle und repliziert sie in einen Kinesis Data Streams. Ihre Verbraucheranwendungen können auf diesen Stream zugreifen, um Änderungen auf Artelebene in Echtzeit anzuzeigen und diese Änderungen anschließend weiterzuleiten oder auf der Grundlage des Inhalts Maßnahmen zu ergreifen.

Weitere Informationen finden Sie unter [wie Kinesis Data Streams mit DynamoDB funktionieren](#) im Amazon-DynamoDB-Entwicklerhandbuch.

## Amazon EventBridge

Mit Kinesis Data Streams können Sie den AWS-API-Aufruf [ereignisse](#) in EventBridge an einen Stream senden, Verbraucheranwendungen erstellen und große Datenmengen verarbeiten. Sie können Kinesis Data Streams auch als Ziel in EventBridge Pipes verwenden und Datensätze nach optionaler Filterung und Anreicherung als Stream aus einer der verfügbaren Quellen bereitstellen.

Weitere Informationen finden Sie unter [Senden von Ereignissen an einen Amazon-Kinesis-Stream](#) und [EventBridge Pipes](#) im Benutzerhandbuch zu Amazon EventBridge.

## AWS IoT Core

Mithilfe von AWS-IoT-Regelaktionen können Sie Daten in Echtzeit aus MQTT-Nachrichten in AWS IoT Core schreiben. Anschließend können Sie Anwendungen erstellen, die die Daten verarbeiten, ihren Inhalt analysieren und Warnmeldungen generieren und sie an Analyseanwendungen oder andere AWS-Services weiterleiten.

Weitere Informationen finden Sie unter [Kinesis Data Streams](#) im AWS IoT Core-Entwicklerhandbuch.

## Amazon Relational Database Service

Sie können Amazon Kinesis Data Streams verwenden, um Aktivitäten auf Ihren Amazon-RDS-Instances zu überwachen. Mithilfe von Database Activity Streams überträgt Amazon RDS Aktivitäten in Echtzeit an einen Amazon Kinesis Data Streams. Anschließend können Sie Anwendungen für das Compliance-Management entwickeln, die diese Aktivitäten nutzen, prüfen und Warnmeldungen generieren. Sie können auch Amazon Firehose verwenden, um die Daten zu speichern.

Weitere Informationen finden Sie unter [Datenbankaktivitätsstreams](#) im Entwicklerhandbuch zu Amazon RDS.

## Amazon Pinpoint

Sie können Amazon Pinpoint so einrichten, dass Ereignisdaten an Amazon Kinesis Data Streams gesendet werden. Amazon Pinpoint kann Ereignisdaten für Kampagnen, Reisen und transaktionale E-Mail- und SMS-Nachrichten senden. Sie können die Daten dann in Analyseanwendungen aufnehmen oder Ihre eigenen Verbraucheranwendungen erstellen, die auf der Grundlage des Inhalts der Ereignisse Maßnahmen ergreifen.

Weitere Informationen finden Sie unter [Streaming-Events](#) im Entwicklerhandbuch zu Amazon Pinpoint.

## Amazon Quantum Ledger Database

Sie können einen Stream in QLDB erstellen, in dem jede Dokumentversion erfasst wird, die für Ihr Journal festgeschrieben ist, und mit dem diese Daten in Echtzeit an Amazon Kinesis Data Streams geliefert werden. Ein QLDB-Stream ist ein kontinuierlicher Datenfluss aus dem Ledger-Journal zu einer Kinesis-Datenstromressource. Anschließend können Sie die Kinesis-Streaming-Plattform oder die Kinesis-Client-Bibliothek verwenden, um Ihren Stream zu nutzen, die Datensätze zu verarbeiten und den Dateninhalt zu analysieren. Ein QLDB-Stream schreibt Ihre Daten in drei Datensatztypen in Kinesis Data Streams: `control`, `block summary` und `revision details`.

Weitere Informationen finden Sie unter [Streams](#) im Amazon-QLDB-Entwicklerhandbuch.

## Integrationen von Drittanbietern verwenden

Sie können Daten mit einer der folgenden Drittanbieteroptionen, die in Kinesis Data Streams integriert sind, in Kinesis Data Streams schreiben:

### Themen

- [Apache Flink](#)
- [Fluentd](#)
- [Debezium](#)
- [Oracle GoldenGate](#)
- [Kafka Connect](#)

- [Adobe Experience](#)
- [Striim](#)

## Apache Flink

Apache Flink ist ein Framework und eine verteilte Verarbeitungs-Engine für statusbehaftete Berechnungen über unbegrenzte und begrenzte Datenströme. Weitere Informationen zum Schreiben von Apache Flink in Kinesis Data Streams finden Sie unter [Amazon Kinesis Data Streams Connector](#).

## Fluentd

Fluentd ist ein Open-Source-Datensammler für eine einheitliche Protokollierungsebene. Weitere Informationen zum Schreiben von Fluentd in Kinesis Data Streams. Weitere Informationen finden Sie unter [Stream-Verarbeitung mit Kinesis](#).

## Debezium

Debezium ist eine verteilte Open-Source-Plattform für die Erfassung von Änderungsdaten. Weitere Informationen zum Schreiben von Debezium in Kinesis Data Streams finden Sie unter [Streaming von MySQL-Datenänderungen zu Amazon Kinesis](#).

## Oracle GoldenGate

Oracle GoldenGate ist ein Softwareprodukt, mit dem Sie Daten von einer Datenbank in eine andere Datenbank replizieren, filtern und transformieren können. Weitere Informationen zum Schreiben von Oracle GoldenGate in Kinesis Data Streams finden Sie unter [Datenreplikation zu Kinesis Data Streams mit Oracle GoldenGate](#).

## Kafka Connect

Kafka Connect ist ein Tool für skalierbares und zuverlässiges Streamen von Daten zwischen Apache Kafka und anderen Systemen. Weitere Informationen zum Schreiben von Apache Kafka in Kinesis Data Streams finden Sie unter [der Kinesis-Kafka-Konnektor](#).

## Adobe Experience

Die Adobe Experience Platform ermöglicht es Unternehmen, Kundendaten aus jedem System zu zentralisieren und zu standardisieren. Anschließend werden Datenwissenschaft und Machine

Learning angewendet, um das Design und die Bereitstellung umfassender, personalisierter Erlebnisse erheblich zu verbessern. Weitere Informationen zum Schreiben von Daten von der Adobe Experience Platform in Kinesis Data Streams finden Sie unter [So erstellen Sie eine Amazon-Kinesis-Verbindung](#).

## Striim

Striim ist eine vollständige, durchgängige In-Memory-Plattform zum Sammeln, Filtern, Transformieren, Anreichern, Aggregieren, Analysieren und Bereitstellen von Daten in Echtzeit. Weitere Informationen zum Schreiben von Daten von Striim in Kinesis Data Streams finden Sie im [Kinesis Writer](#).

## Fehlerbehebung bei Amazon Kinesis Data Streams Producers

In den folgenden Abschnitten finden Sie Lösungen für einige Probleme, die möglicherweise bei der Arbeit mit Produzenten von Amazon Kinesis Data Streams auftreten.

- [Die Produzentenanwendung schreibt Daten langsamer als erwartet](#)
- [Fehler aufgrund fehlender KMS-Masterschlüsselberechtigung](#)
- [Häufig auftretende Probleme, Fragen und Ideen zur Problemlösung für Produzenten](#)

### Die Produzentenanwendung schreibt Daten langsamer als erwartet

Die häufigsten Gründe für einen langsameren Schreibdurchsatz sind:

- [Überschreitung der Service Limits](#)
- [Produzentenoptimierung](#)

### Überschreitung der Service Limits

Wenn Sie wissen möchten, ob die Service Limits überschritten werden, prüfen Sie, ob Ihr Datenproduzent Ausnahmen im Service auslöst, und validieren Sie, welche API-Operationen gedrosselt werden. Beachten Sie, dass es je nach Aufruf verschiedene Limits gibt, siehe [Kontingente und Einschränkungen](#). Beispielsweise gelten zusätzlich zu den bekannten Limits für Schreib- und Lesevorgänge auf Shard-Ebene folgenden Limits auf Stream-Ebene:

- [CreateStream](#)
- [DeleteStream](#)



- [ListStreams](#)
- [GetShardIterator](#)
- [MergeShards](#)
- [DescribeStream](#)
- [DescribeStreamSummary](#)

Die Operationen `CreateStream`, `DeleteStream`, `ListStreams`, `GetShardIterator` und `MergeShards` dürfen maximal 5 mal pro Sekunde aufgerufen werden. Die `DescribeStream`-Operation ist auf 10 Aufrufe pro Sekunde begrenzt. Die `DescribeStreamSummary`-Operation ist auf 20 Aufrufe pro Sekunde begrenzt.

Wenn diese Aufrufe nicht das Problem sind, stellen Sie sicher, dass Sie einen Partitionsschlüssel ausgewählt haben, mit dem Sie die PUT-Operationen gleichmäßig auf alle Shards verteilen können, und dass Sie keinen Partitionsschlüssel haben, der die Service Limits ausschöpft, wenn der Rest dies nicht tut. Dazu müssen Sie den Spitzendurchsatz messen und die Anzahl der Shards im Stream berücksichtigen. Weitere Informationen zum Verwalten von Streams finden Sie unter [Erstellen und Verwalten von Streams](#).

#### Tip

Bitte beachten, dass Sie bei der Berechnung der Durchsatzdrosselung auf den nächsten Byte aufrunden müssen, wenn Sie die Einzeldatensatz-Operation [PutRecord](#) verwenden. Bei der Mehrfachdatensatz-Operation [PutRecords](#) wird die kumulative Anzahl der Datensätze in den einzelnen Aufrufen gerundet. Eine `PutRecords`-Anforderung mit 600 Datensätzen und einer Größe von 1,1 KB wird beispielsweise nicht gedrosselt.

## Produzentenoptimierung

Bevor Sie mit der Optimierung Ihres Produzenten beginnen, müssen Sie einige wesentliche Aufgaben abschließen. Bestimmen Sie zunächst den gewünschten Spitzenwert hinsichtlich Datensatzgröße und Datensätze pro Sekunde. Schließen Sie dann die Stream-Kapazität als Begrenzungsfaktor aus ([Überschreitung der Service Limits](#)). Nutzen Sie die folgenden Tipps zur Fehlerbehebung und Optimierungsanleitungen für die beiden häufigsten Produzententypen, wenn Sie die Stream-Kapazität als Begrenzungsfaktor ausgeschlossen haben.

### Großer Produzent

Ein großer Produzent wird in der Regel von einem lokalen Server oder einer Amazon-EC2-Instance ausgeführt. Für Kunden, die einen höheren Durchsatz eines großen Produzenten benötigen, ist in der Regel die Latenz pro Datensatz entscheidend. Nachfolgend einige Strategien für den Umgang mit Latenzen: Wenn der Kunde Datensätze puffern oder in Mikrostackeln verpacken kann, nutzen Sie die [Kinesis Producer Library](#) (diese verfügt über eine erweiterte Aggregationslogik), die Mehrfachdatensatz-Operation [PutRecords](#) oder aggregieren Sie Datensätze in eine größere Datei, ehe Sie die Einzeldatensatz-Operation [PutRecord](#) durchführen. Wenn Sie keine Stapel bilden und keine Pufferung verwenden können, nutzen Sie mehrere Threads, um gleichzeitig in Service Kinesis Data Streams schreiben zu können. Das AWS SDK for Java und andere SDKs enthalten asynchrone Clients, die dies mit einer kleinen Codeanweisung übernehmen können.

### Kleiner Produzent

Eine kleiner Produzent ist in der Regel eine mobile App, ein IoT-Gerät oder ein Webclient. Bei einer mobilen App sollten Sie die `PutRecords`-Operation oder den Kinesis Recorder in den AWS Mobil SDKs nutzen. Weitere Informationen finden Sie im AWS Mobile SDK for Android-Leitfaden zu den ersten Schritten und im AWS Mobile SDK for iOS-Handbuch für erste Schritte. Mobile Anwendungen müssen Verbindungsunterbrechungen bewältigen und benötigen eine Art Stapeleingabe, beispielsweise `PutRecords`. Wenn Sie aus irgendeinem Grund keine Stapel bilden können, lesen Sie sich die oben stehenden Informationen zu großen Produzenten durch. Ist der Produzent ein Browser, ist die Menge der generierten Dateien in der Regel sehr klein. Allerdings platzieren Sie `PUT`-Operationen im kritischen Pfad der Anwendung. Dies wird von uns nicht empfohlen.

## Fehler aufgrund fehlender KMS-Masterschlüsselberechtigung

Dieser Fehler tritt auf, wenn eine Produzentenanwendung ohne Berechtigung für den KMS-Masterschlüssel Daten in einen verschlüsselten Stream schreibt. Informationen zum Zuweisen von Berechtigungen zu einer Anwendung für den Zugriff auf einen KMS-Schlüssel finden Sie unter [Verwenden von Schlüsselrichtlinien in AWS KMS](#) und [Verwenden von IAM-Richtlinien mit AWS KMS](#).

## Häufig auftretende Probleme, Fragen und Ideen zur Problemlösung für Produzenten

- [Warum gibt mein Kinesis-Datenstrom einen 500 Internal Server Error zurück?](#)
- [Wie behebe ich Timeout-Fehler beim Schreiben von Flink in Kinesis Data Streams?](#)
- [Wie behebe ich Drosselungsfehler in Kinesis Data Streams?](#)

- [Warum drosselt mein Kinesis-Datenstrom?](#)
- [Wie kann ich Datensätze mithilfe der KPL in einen Kinesis-Datenstrom einfügen?](#)

## Fortgeschrittene Themen für Kinesis-Data-Streams-Produzenten

In diesem Abschnitt wird erläutert, wie Sie Ihre Produzenten von Amazon Kinesis Data Streams optimieren.

### Themen

- [Begrenzungen für KPL-Wiederholungen und Quoten](#)
- [Überlegungen zur Verwendung der KPL-Aggregation](#)

## Begrenzungen für KPL-Wiederholungen und Quoten

Wenn Sie Benutzerdatensätze von Kinesis Producer Library (KPL) mit der KPL-Operation `addUserRecord()` hinzufügen, erhält ein Datensatz einen Zeitstempel und wird einem Puffer hinzugefügt, wobei der Konfigurationsparameter `RecordMaxBufferedTime` eine Frist festsetzt. Diese Kombination aus Zeitstempel und Frist legt die Pufferpriorität fest. Datensätze werden aufgrund der folgenden Kriterien aus dem Puffer entfernt:

- Puffer-Priorität
- Aggregierungskonfiguration
- Sammlungskonfiguration

Die Parameter der Aggregierungskonfiguration und der Sammlungskonfiguration, die das Pufferverhalten beeinflussen, sind:

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

Die gelöschten Datensätze werden dann mithilfe eines Aufrufs der API-Operation `PutRecords` von Kinesis Data Streams als Datensätze von Amazon Kinesis Data Streams an Ihren Kinesis-

Datenstrom gesendet. Die Operation `PutRecords` sendet Anfragen an Ihre Stream, die gelegentlich vollständige oder teilweise Fehlschläge zeigen. Fehlgeschlagene Datensätze werden automatisch wieder dem KPL-Puffer hinzugefügt. Die neue Frist wird auf der Grundlage des kleineren dieser beiden Werte festgelegt:

- Die Hälfte der aktuellen `RecordMaxBufferedTime`-Konfiguration
- Die Aufbewahrungszeit des Datensatzes

Mit dieser Strategie können wiederholte KPL-Benutzerdatensätze in nachfolgenden API-Aufrufe von Kinesis Data Streams aufgenommen werden, um den Durchsatz zu verbessern und die Komplexität zu verringern. Gleichzeitig wird die Aufbewahrungszeit des Datensatzes von Kinesis Data Streams durchgesetzt. Es gibt keinen Backoff-Algorithmus, was dies zu einer relativ aggressiven Wiederholungsstrategie macht. Spamming aufgrund übermäßiger Wiederholungen wird durch die Quotengrenze bewirkt, die im nächsten Abschnitt erläutert wird.

## Quotenbegrenzung

Das KPL enthält eine Quotenbegrenzungsfunktion, die den pro Shard von einem einzelnen Producer gesendeten Durchsatz begrenzt. Die Quotenbegrenzung wird durch einen Token-Bucket-Algorithmus mit separaten Buckets für Datensätze von Kinesis Data Streams und Bytes implementiert. Jeder erfolgreiche Schreibvorgang in einen Kinesis-Datenstrom fügt jedem Bucket ein Token (oder mehrere Token) hinzu, bis ein bestimmter Grenzwert erreicht ist. Dieser Grenzwert ist konfigurierbar, ist aber standardmäßig um 50 % höher als das tatsächliche Shard-Limit eingestellt, um die Shard-Sättigung durch einen einzelnen Produzenten zu ermöglichen.

Sie können diesen Grenzwert herabsetzen, um Spamming durch übermäßig viele Wiederholungsversuche zu reduzieren. Die bewährte Methode besteht jedoch darin, dass jeder Produzent aggressiv einen maximalen Durchsatz durch Wiederholungen zu erzielen versucht und mit der daraus resultierenden, als übermäßig angesehenen Drosselung so umzugehen, dass die Kapazität des Streams erweitert und eine geeignete Partitionsschlüsselstrategie implementiert wird.

## Überlegungen zur Verwendung der KPL-Aggregation

Während das Folgenummernschema der resultierenden Datensätze von Amazon Kinesis Data Streams unverändert bleibt, führt die Aggregation zur Indizierung der Benutzerdatensätze von Kinesis Producer Library (KPL) in einem aggregierten Datensatz von Kinesis Data Streams mit dem Beginn 0 (Null). Solange Sie sich bei der Identifizierung Ihrer KPL-Benutzerdatensätze nicht ausschließlich auf die Folgenummern stützen, kann Ihr Code dies ignorieren, da die Aggregation

(der KPL-Benutzerdatensätze in einen Datensatz von Kinesis Data Streams) und die anschließende Deaggregation (eines Datensatzes von Kinesis Data Streams in die KPL-Benutzerdatensätze) dies automatisch für Sie übernimmt. Dies gilt unabhängig davon, ob Ihr Consumer das KCL oder das AWS-SDK verwendet. Zur Verwendung dieser Aggregierungsfunktion müssen Sie den Java-Teil des KCL in Ihren Build ziehen, wenn Ihr Consumer mit der im AWS-SDK bereitgestellten API geschrieben wurde.

Wenn Sie beabsichtigen, Folgenummern als einzige Kennungen Ihrer KPL-Benutzerdatensätze zu verwenden, empfehlen wir die Verwendung der vertragsgemäßen Operationen `public int hashCode()` und `public boolean equals(Object obj)` in `Record` und `UserRecord` zur Aktivierung des Vergleichs Ihrer KPL-Benutzerdatensätze. Wenn Sie zudem die Teilsequenznummer des KPL;-Benutzerdatensatzes untersuchen möchten, können Sie eine Typumwandlung in eine `UserRecord`-Instance vornehmen und deren Teilsequenznummer abrufen.

Weitere Informationen finden Sie unter [Datenproduzent – Disaggregation](#).

# Lesen von Daten aus Amazon Kinesis Data Streams

Ein Konsument ist eine Anwendung, die alle Daten aus einem Kinesis-Datenstrom verarbeitet. Wenn ein Konsument erweiterte Rundsendungen verwendet, erhält er ein eigenes Kontingent von 2 MB/s für den Lesedurchsatz, sodass mehrere Konsumenten parallel Daten aus demselben Stream lesen können, ohne den Lesedurchsatz von anderen Konsumenten abziehen zu müssen. Informationen zum Verwenden der Shard-Funktionen für erweiterte Rundsendungen erhalten Sie unter [Entwickeln benutzerdefinierter Verbraucher mit dediziertem Durchsatz \(Erweitertes Rundsenden\)](#).

Standardmäßig stellen Shards in einem Stream 2 MB/s Lesedurchsatz pro Shard bereit. Dieser Durchsatz gilt insgesamt für alle Konsumenten, die aus dem gegebenen Shard lesen. Der Standardwert von 2 MB/s Lesedurchsatz pro Shard ist also ein fester Wert, auch wenn mehrere Konsumenten aus dem betreffenden Shard lesen. Informationen zur Nutzung dieses Standarddurchsatzes von Shards finden Sie unter [Entwickeln benutzerdefinierter Verbraucher mit gemeinsam genutztem Durchsatz](#).

Die folgende Tabelle vergleicht den Standard-Durchsatz mit den erweiterten Rundsendungen. Die Verzögerung der Nachrichtenverbreitung ist definiert als die Zeit in Millisekunden, die eine Nutzlast benötigt, die mit den Payload-Dispatching-APIs (wie PutRecord und PutRecords) gesendet wird, um die Verbraucheranwendung über die nutzlastverbrauchenden APIs (wie GetRecords und ) zu erreichen SubscribeToShard.

Merkmale	Nicht registrierte Verbraucher ohne erweitertes Rundsenden	Registrierte Verbraucher mit erweitertem Rundsenden
Shard-Lesedurchsatz	Fester Gesamtwert von 2 MB/s pro Shard. Wenn mehrere Konsumenten aus demselben Shard lesen, teilen sie sich diesen Durchsatz. Die Summe des Lesedurchsatzes der einzelnen Konsumenten eines Shards beträgt maximal 2 MB/s.	Der Wert erhöht sich nur, wenn Konsumenten für erweiterte Rundsendungen registriert werden. Jeder für erweiterte Rundsendungen registrierte Konsument erhält einen eigenen Lesedurchsatz von bis zu 2 MB/s pro Shard, den er nicht mit anderen Konsumenten teilen muss.

Merkmale	Nicht registrierte Verbraucher ohne erweitertes Rundsenden	Registrierte Verbraucher mit erweitertem Rundsenden
Verzögerung der Nachrichtenverbreitung	Ein Durchschnitt von etwa 200 ms, wenn Sie einen Verbraucher haben, der aus dem Stream liest. Dieser Durchschnitt steigt bis zu 1000 ms an, wenn Sie fünf Verbraucher haben.	In der Regel durchschnittlich 70 ms, unabhängig davon, ob Sie einen oder fünf Verbraucher haben.
Kosten	N/A	Es gibt Datenabrufkosten und Konsumenten-Shard-Stundensätze. Weitere Informationen finden Sie unter <a href="#">Preise für Amazon Kinesis Daten-Streams</a> .
Datensatzbereitstellungsmodell	Ziehen Sie das Modell über HTTP mit <a href="#">GetRecords</a> .	Kinesis Data Streams überträgt die Datensätze mithilfe von über HTTP/2 an Sie <a href="#">SubscribeToShard</a> .

## Themen

- [Verwendung von Data Viewer in der Kinesis-Konsole](#)
- [Abfragen Ihrer Datenströme in der Kinesis-Konsole](#)
- [Entwickeln von Konsumenten mit AWS Lambda](#)
- [Entwicklung von Verbrauchern, die Amazon Managed Service für Apache Flink nutzen](#)
- [Entwickeln von Konsumenten mit Amazon Data Firehose](#)
- [Verwenden der Kinesis Client Library](#)
- [Entwickeln benutzerdefinierter Verbraucher mit gemeinsam genutztem Durchsatz](#)
- [Entwickeln benutzerdefinierter Verbraucher mit dediziertem Durchsatz \(Erweitertes Rundsenden\)](#)
- [Migrieren von Verbrauchern von KCL 1.x zu KCL 2.x](#)
- [Nutzung anderer AWS-Services zum Lesen von Daten aus Kinesis Data Streams](#)
- [Integrationen von Drittanbietern verwenden](#)
- [Problembehandlung bei Verbrauchern von Kinesis Data Streams](#)
- [Fortgeschrittene Themen für Amazon Kinesis Data Streams-Verbraucher](#)

## Verwendung von Data Viewer in der Kinesis-Konsole

Mit dem Data Viewer in der Kinesis Management Console können Sie Datensätze innerhalb des angegebenen Shards Ihres Datenstroms anzeigen, ohne eine Verbraucheranwendung entwickeln zu müssen. Gehen Sie folgendermaßen vor, um den Data Viewer zu verwenden:

1. Melden Sie sich bei der an AWS Management Console und öffnen Sie die Kinesis-Konsole unter <https://console.aws.amazon.com/kinesis>.
2. Wählen Sie den aktiven Datenstrom aus, dessen Datensätze Sie mit dem Data Viewer anzeigen möchten, und wählen Sie dann die Registerkarte Data Viewer.
3. Wählen Sie auf der Registerkarte Data Viewer für den ausgewählten aktiven Datenstrom den Shard aus, dessen Datensätze Sie anzeigen möchten, wählen Sie die Startposition aus und klicken Sie dann auf Datensätze abrufen. Sie können die Startposition auf einen der folgenden Werte stellen:
  - Bei Sequenznummer: Zeigt Datensätze von der Position an, die durch die im Feld Sequenznummer angegebene Sequenznummer gekennzeichnet ist.
  - Nach Sequenznummer: Zeigt Datensätze direkt nach der Position an, die durch die im Feld Sequenznummer angegebene Sequenznummer gekennzeichnet ist.
  - Bei Zeitstempel: Zeigt Datensätze von der Position an, die durch den im Zeitstempelfeld angegebenen Zeitstempel gekennzeichnet ist.
  - Horizont kürzen: Zeigt Datensätze am letzten nicht getrimmten Datensatz im Shard an, der der älteste Datensatz im Shard ist.
  - Aktuell: Zeigt Datensätze direkt nach dem neuesten Datensatz im Shard an, sodass Sie immer die neuesten Daten im Shard lesen.

Die generierten Datensätze, die mit der angegebenen Shard-ID und Startposition übereinstimmen, werden dann in einer Datensatztable in der Konsole angezeigt. Es werden maximal 50 Datensätze gleichzeitig angezeigt. Um die nächste Gruppe von Datensätzen anzuzeigen, klicken Sie auf die Schaltfläche Weiter.

4. Klicken Sie auf einen einzelnen Datensatz, um die Nutzdaten dieses Datensatzes im Rohdaten- oder JSON-Format in einem separaten Fenster anzuzeigen.

Beachten Sie, dass beim Klicken auf Datensätze abrufen oder Weiter in Data Viewer die GetRecords API aufgerufen wird. Dies gilt für das GetRecords API-Limit von 5 Transaktionen pro Sekunde.



## Abfragen Ihrer Datenströme in der Kinesis-Konsole

Auf der Registerkarte Data Analytics in der Kinesis Data Streams Console können Sie Ihre Datenströme mithilfe von SQL abfragen. Gehen Sie folgendermaßen vor, um diese Funktion zu verwenden:

1. Melden Sie sich bei der an AWS Management Console und öffnen Sie die Kinesis-Konsole unter <https://console.aws.amazon.com/kinesis>.
2. Wählen Sie den aktiven Datenstrom aus, den Sie mit SQL abfragen möchten, und wählen Sie dann die Registerkarte Datenanalysen.
3. Auf der Registerkarte Datenanalysen können Sie die Stream-Inspektion und -Visualisierung mit einem Managed Apache Flink Studio-Notebook durchführen. Mit Apache Zeppelin können Sie Ad-hoc-SQL-Abfragen durchführen, um Ihren Datenstrom zu überprüfen und Ergebnisse in Sekundenschnelle anzuzeigen. Wählen Sie auf der Registerkarte Datenanalysen die Option Ich stimme zu und dann Notebook erstellen aus, um ein Notebook zu erstellen.
4. Nachdem das Notebook erstellt wurde, wählen Sie In Apache Zeppelin öffnen aus. Dadurch wird Ihr Notebook auf einer neuen Registerkarte geöffnet. Ein Notebook ist eine interaktive Schnittstelle, über die Sie Ihre SQL-Abfragen einreichen können. Wählen Sie die Notiz aus, die den Namen Ihres Streams enthält.
5. Sie sehen eine Notiz mit einer SELECT Beispielabfrage zur Ausgabe der Daten im bereits ausgeführten Stream. Auf diese Weise können Sie das Schema für Ihren Datenstrom anzeigen.
6. Um andere Abfragen wie rollierende oder gleitende Fenster auszuprobieren, wählen Sie auf der Registerkarte Datenanalyse die Option Beispielabfragen anzeigen aus. Kopieren Sie die Abfrage, ändern Sie sie entsprechend Ihrem Datenstromschema und führen Sie sie dann in einem neuen Absatz in Ihrer Zeppelin-Notiz aus.

## Entwickeln von Konsumenten mit AWS Lambda

Sie können eine - AWS Lambda Funktion verwenden, um Datensätze in einem Datenstrom zu verarbeiten. AWS Lambda ist ein Datenverarbeitungsservice, mit dem Sie Code ausführen können, ohne Server bereitstellen oder verwalten zu müssen. Es führt Ihren Code nur bei Bedarf aus und skaliert automatisch – von einigen Anfragen pro Tag bis zu Tausenden pro Sekunde. Sie zahlen nur für die tatsächlich aufgewendete Zeit. Es werden keine Gebühren berechnet, solange Ihr Code nicht ausgeführt wird. Mit können AWS Lambda Sie Code für praktisch jede Art von Anwendung oder Backend-Service ausführen, und all das ohne Administration. Der Service

führt Ihren Code auf einer hoch verfügbaren Datenverarbeitungsinfrastruktur aus und erledigt die gesamte Administration der Datenverarbeitungsressourcen, einschließlich der Server- und Betriebssystemwartung, Kapazitätsbereitstellung und automatischen Skalierung sowie Code-Überwachung und -Protokollierung. Weitere Informationen finden Sie unter [Verwenden von AWS Lambda mit Amazon Kinesis](#).

Informationen zur Fehlerbehebung finden Sie unter [Warum kann der Kinesis-Data-Streams-Trigger meine Lambda-Funktion nicht aufrufen?](#)

## Entwicklung von Verbrauchern, die Amazon Managed Service für Apache Flink nutzen

Sie können eine Anwendung von Amazon Managed Service für Apache Flink verwenden, um Daten in einem Kinesis-Stream mit SQL, Java oder Scala zu verarbeiten und zu analysieren. Anwendungen des Managed Service für Apache Flink kann Daten mithilfe von Referenzquellen anreichern, Daten im Laufe der Zeit aggregieren oder Machine Learning verwenden, um Datenanomalien zu finden. Anschließend können Sie die Analyseergebnisse in einen anderen Kinesis-Stream, einen Firehose-Bereitstellungs-Stream oder eine Lambda-Funktion schreiben. Weitere Informationen finden Sie im [Entwicklerhandbuch für Managed Service für Apache Flink für SQL-Anwendungen](#) oder im [Entwicklerhandbuch für Managed Service für Apache Flink für Flink-Anwendungen](#).

## Entwickeln von Konsumenten mit Amazon Data Firehose

Sie können einen Firehose verwenden, um Datensätze aus einem Kinesis-Stream zu lesen und zu verarbeiten. Firehose ist ein vollständig verwalteter Service für die Bereitstellung von Echtzeit-Streaming-Daten an Ziele wie Amazon S3, Amazon Redshift, Amazon OpenSearch Service und Splunk. Firehose unterstützt auch alle benutzerdefinierten HTTP-Endpunkte oder HTTP-Endpunkte, die unterstützten Drittanbietern gehören, einschließlich Datadog, MongoDB und New Relic. Sie können Firehose auch so konfigurieren, dass Ihre Datensätze transformiert und das Datensatzformat konvertiert wird, bevor Ihre Daten an das Ziel übermittelt werden. Weitere Informationen finden Sie unter [Schreiben in Firehose mit Kinesis Data Streams](#).

## Verwenden der Kinesis Client Library

Eine der Methoden zur Entwicklung benutzerdefinierter Verbraucheranwendungen, die Daten aus KDS-Datenströmen verarbeiten können, ist die Verwendung der Kinesis Client Library (KCL).

## Themen

- [Was ist die Kinesis Client Library?](#)
- [Verfügbare KCL-Versionen](#)
- [KCL-Konzepte](#)
- [Mithilfe einer Leasetabelle können Sie die von der KCL-Konsumentenanzwendung verarbeiteten Shards verfolgen](#)
- [Verarbeitung mehrerer Datenströme mit derselben Konsumentenanzwendung KCL 2.x für Java](#)
- [Verwenden der Kinesis Client Library mit dem AWS Glue Schema Registry](#)

### Note

Sowohl für KCL 1.x als auch für KCL 2.x wird empfohlen, je nach Nutzungsszenario ein Upgrade auf die neueste KCL-1.x-Version oder KCL-2.x-Version durchzuführen. Sowohl KCL 1.x als auch KCL 2.x werden regelmäßig mit neueren Versionen aktualisiert, die die neuesten Abhängigkeits- und Sicherheitspatches, Bugfixes und abwärtskompatible neue Features enthalten. Weitere Informationen finden Sie unter <https://github.com/awslabs/amazon-kinesis-client/veroeffentlicht>.

## Was ist die Kinesis Client Library?

KCL unterstützt Sie bei der Nutzung und Verarbeitung von Daten aus einem Kinesis-Datenstrom, indem sie sich um viele der komplexen Aufgaben kümmert, die mit verteilter Datenverarbeitung verbunden sind. Dazu gehören der Lastenausgleich über mehrere Anwendungs-Instances hinweg, die Reaktion auf Ausfälle von Verbraucheranzwendungs-Instances, die Überprüfung verarbeiteter Datensätze und die Reaktion auf Resharding. Die KCL kümmert sich um all diese Unteraufgaben, sodass Sie sich darauf konzentrieren können, Ihre benutzerdefinierte Logik für die Verarbeitung von Datensätzen zu schreiben.

Die KCL unterscheidet sich von den Kinesis-Data-Streams-APIs, die in den AWS-SDKs verfügbar sind. Die Kinesis-Data-Streams-APIs helfen Ihnen bei der Verwaltung vieler Aspekte von Kinesis Data Streams, einschließlich Stream-Erstellung, Resharding sowie Hinzufügen und Abrufen von Datensätzen. Die KCL bietet eine Abstraktionsebene für all diese Unteraufgaben, sodass Sie sich auf die benutzerdefinierte Datenverarbeitungslogik Ihrer Verbraucheranzwendung konzentrieren können. Informationen zur Kinesis-Data-Streams-API finden Sie in der [Amazon-Kinesis-API-Referenz](#).

### Important

Die KCL ist eine Java-Bibliothek. Unterstützung für andere Sprachen als Java wird über eine mehrsprachige Schnittstelle bereitgestellt, die als bezeichnet wird MultiLangDaemon. Dieser Daemon basiert auf Java und wird im Hintergrund ausgeführt, wenn Sie eine andere KCL-Sprache als Java verwenden. Wenn Sie beispielsweise die KCL für Python installieren und Ihre Konsumentenanzahl vollständig in Python schreiben, benötigen Sie aufgrund der trotzdem Java auf Ihrem System MultiLangDaemon. Darüber hinaus MultiLangDaemon verfügt über einige Standardeinstellungen, die Sie möglicherweise für Ihren Anwendungsfall anpassen müssen, z. B. die AWS Region, mit der es eine Verbindung herstellt. Weitere Informationen zu finden Sie GitHubunter KCL MultiLangDaemon -Projekt . [MultiLangDaemon](#)

Die KCL dient als Vermittler zwischen Ihrer Datensatzverarbeitungslogik und Kinesis Data Streams. Die KCL führt die folgenden Aufgaben aus:

- Stellt eine Verbindung mit dem Datenstrom her
- Listet die Shards innerhalb des Datenstroms auf
- Nutzt Leases, um Shard-Verbindungen mit seinen Workern zu koordinieren
- Instanziiert einen Datensatzverarbeiter für jeden Shard, der verwaltet wird
- Ruft Datensätze aus dem Datenstrom ab
- Überträgt per Push Datensätze an den entsprechenden Datensatzverarbeiter
- Verwendet Checkpoints für verarbeitete Datensätze
- Gleicht zwischen Shard-Worker-Zuordnungen (Leases) aus, wenn sich die Anzahl der Worker-Instances ändert oder wenn der Datenstrom erneut geteilt wird (Shards werden aufgeteilt oder zusammengeführt)

## Verfügbare KCL-Versionen

Derzeit können Sie eine der folgenden unterstützten Versionen von KCL verwenden, um Ihre benutzerdefinierten Anwendungen für Privatanwender zu erstellen:

- KCL 1.x

Weitere Informationen finden Sie unter [Entwicklung von KCL 1.x-Verbrauchern](#).

- KCL 2.x

Weitere Informationen finden Sie unter [Entwicklung von KCL 2.x-Verbrauchern](#).

Sie können entweder KCL 1.x oder KCL 2.x verwenden, um Verbrauchieranwendungen zu erstellen, die einen gemeinsamen Durchsatz verwenden. Weitere Informationen finden Sie unter [Entwickeln benutzerdefinierter Verbraucher mit gemeinsam genutztem Durchsatz unter Verwendung von KCL](#).

Um Verbrauchieranwendungen zu erstellen, die einen dedizierten Durchsatz verwenden (Enhanced Fan-Out-Consumer), können Sie nur KCL 2.x verwenden. Weitere Informationen finden Sie unter [Entwickeln benutzerdefinierter Verbraucher mit dediziertem Durchsatz \(Erweitertes Rundsenden\)](#).

Informationen zu den Unterschieden zwischen KCL 1.x und KCL 2.x sowie Anweisungen zur Migration von KCL 1.x zu KCL 2.x finden Sie unter [Migrieren von Verbrauchern von KCL 1.x zu KCL 2.x](#).

## KCL-Konzepte

- KCL-Konsumentenanzwendung – eine Anwendung, die unter Verwendung von KCL kundenspezifisch entwickelt wurde und zum Lesen und Verarbeiten von Datensätzen aus Datenströmen bestimmt ist.
- Konsumentenanzwendungs-Instance – KCL-Konsumentenanzwendungen werden in der Regel verteilt, wobei eine oder mehrere Anwendungs-Instances gleichzeitig ausgeführt werden, um Fehler zu koordinieren und einen dynamischen Lastenausgleich bei der Verarbeitung von Datensätzen vorzunehmen.
- Worker – eine übergeordnete Klasse, die eine KCL-Konsumentenanzwendungs-Instance verwendet, um mit der Datenverarbeitung zu beginnen.

### Important

Jede KCL-Konsumentenanzwendungs-Instance hat einen Worker.

Der Worker initialisiert und überwacht verschiedene Aufgaben, darunter das Synchronisieren von Shard- und Leasing-Informationen, das Verfolgen von Shard-Zuweisungen und das Verarbeiten von Daten aus den Shards. Ein Worker stellt KCL die Konfigurationsinformationen für die Konsumentenanzwendung zur Verfügung, z. B. den Namen des Datenstroms, dessen Datensätze diese KCL-Konsumentenanzwendung verarbeiten wird, und die AWS-Anmeldeinformationen, die für den Zugriff auf diesen Datenstrom erforderlich sind. Der Worker startet außerdem die

spezifische Instance der KCL-Konsumentenanzwendung, um Datensätze aus dem Datenstrom an die Datensatzprozessoren zu übermitteln.

**⚠ Important**

In KCL 1.x heißt diese Klasse Worker. Weitere Informationen (dies sind die Java-KCL-Repositoryys) finden Sie unter <https://github.com/aws-labs/amazon-kinesis-client/blob/v1.x/src/main/java/com/amazonaws/services/kinesis/clientlibrary/lib/worker/Worker.java> . In KCL 2.x heißt diese Klasse Scheduler. Der Zweck von Scheduler in KCL 2.x ist identisch mit dem Zweck von Worker in KCL 1.x. Weitere Informationen zur Scheduler-Klasse in KCL 2.x finden Sie unter <https://github.com/aws-labs/amazon-kinesis-client/blob/master/amazon-kinesis-client/src/main/java/software/amazon/kinesis/coordinator/Scheduler.java> .

- Lease – Daten, die die Bindung zwischen einem Worker und einem Shard definieren. Verteilte KCL-Konsumentenanzwendungen nutzen Leases, um die Verarbeitung von Datensätzen auf eine ganze Flotte von Workern zu verteilen. Zu einem bestimmten Zeitpunkt ist jeder Shard von Datensätzen durch einen Lease, der durch die leaseKey-Variable identifiziert wird, an einen bestimmten Worker gebunden.

Standardmäßig kann ein Worker einen oder mehrere Leases (abhängig vom Wert der maxLeasesForWorker-Variable) gleichzeitig abschließen.

**⚠ Important**

Jeder Worker versucht, alle verfügbaren Leases für alle verfügbaren Shards in einem Datenstrom zu halten. Aber es kann nur jeweils ein Worker jeden Lease zu einem bestimmten Zeitpunkt erfolgreich halten.

Wenn Sie beispielsweise eine Konsumentenanzwendungs-Instance A mit Worker A haben, die einen Datenstrom mit 4 Shards verarbeitet, kann Worker A Leases für die Shards 1, 2, 3 und 4 gleichzeitig halten. Wenn Sie jedoch zwei Konsumentenanzwendungs-Instances haben: A und B mit Worker A und Worker B und diese Instances einen Datenstrom mit 4 Shards verarbeiten, können Worker A und Worker B nicht gleichzeitig den Lease für Shard 1 halten. Ein Worker hält den Lease für einen bestimmten Shard, bis er bereit ist, die Verarbeitung der Datensätze dieses Shards zu beenden, oder bis er ausfällt. Wenn ein Worker den Lease beendet, nimmt ein anderer Worker den Lease auf und hält ihn.

Weitere Informationen (dies sind die Java-KCL-Repositoryys) finden Sie unter <https://github.com/aws-labs/amazon-kinesis-client/blob/v1.x/src/main/java/com/amazonaws/services/kinesis/leases/impl/Lease.java> für KCL 1.x und <https://github.com/aws-labs/amazon-kinesis-client/blob/master/amazon-kinesis-client/src/main/java/software/amazon/kinesis/leases/Lease.java> für KCL 2.x.

- Leasetabelle – Eine einzigartige Amazon-DynamoDB-Tabelle, die verwendet wird, um die Shards in einem KDS-Datenstrom zu verfolgen, die von den Workern der KCL-Konsumenten-anwendung geleast und verarbeitet werden. Die Leasetabelle muss (innerhalb eines Workers und für alle Worker) mit den neuesten Shard-Informationen aus dem Datenstrom synchronisiert bleiben, während die KCL-Konsumenten-anwendung ausgeführt wird. Weitere Informationen finden Sie unter [Mithilfe einer Leasetabelle können Sie die von der KCL-Konsumenten-anwendung verarbeiteten Shards verfolgen](#).
- Datensatzprozessor – Die Logik, die definiert, wie Ihre KCL-Konsumenten-anwendung die Daten verarbeitet, die sie aus den Datenströmen erhält. Zur Laufzeit instanziiert eine KCL-Konsumenten-anwendungs-Instance einen Worker, und dieser Worker instanziiert einen Datensatzprozessor für jeden Shard, für den er einen Lease hat.

## Mithilfe einer Leasetabelle können Sie die von der KCL-Konsumenten-anwendung verarbeiteten Shards verfolgen

### Themen

- [Was ist eine Leasetabelle](#)
- [Durchsatz](#)
- [Wie wird eine Leasetabelle mit den Shards in einem KDS-Datenstrom synchronisiert](#)

## Was ist eine Leasetabelle

Für jede Anwendung mit Amazon Kinesis Data Streams verwendet KCL eine einzigartige Leasetabelle (gespeichert in einer Amazon-DynamoDB-Tabelle), die verwendet wird, um die Shards in einem KDS-Datenstrom zu verfolgen, die von den Workern der KCL-Konsumenten-anwendung geleast und verarbeitet werden.

**⚠ Important**

KCL verwendet den Namen der Konsumentenanzwendung, um den Namen der Leasetabelle zu erstellen, die diese Konsumentenanzwendung verwendet. Daher muss der Name jeder Konsumentenanzwendung eindeutig sein.

Sie können die Leasetabelle mit der [Amazon-DynamoDB-Konsole](#) anzeigen, während die Konsumentenanzwendung ausgeführt wird.

Wenn die Leasetabelle für Ihre KCL-Konsumentenanzwendung beim Start der Anwendung nicht vorhanden ist, erstellt einer der Worker die Leasetabelle für diese Anwendung.

**⚠ Important**

Ihr Konto wird neben den Kosten für Kinesis Data Streams mit den Kosten belastet, die für die DynamoDB-Tabelle anfallen.

Jede Zeile in der Leasetabelle stellt einen Shard dar, der von den Workern Ihrer Konsumentenanzwendung verarbeitet wird. Wenn Ihre KCL-Konsumentenanzwendung nur einen Datenstrom verarbeitet, dann ist `leaseKey`, der Hash-Schlüssel für die Leasetabelle, die Shard-ID. Wenn Sie [Verarbeitung mehrerer Datenströme mit derselben Konsumentenanzwendung KCL 2.x für Java](#) sind, dann sieht die Struktur des `leaseKey` wie folgt aus: `account-id:StreamName:streamCreationTimestamp:ShardId`. Beispiel: `111111111:multiStreamTest-1:12345:shardId-000000000336`

Neben der Shard-ID enthält jede Zeile noch folgende Daten:

- `checkpoint`: Die letzte Prüfpunkt-Sequenznummer des Shards. Dieser Wert ist für alle Shards im Datenstrom eindeutig.
- `checkpointSubSequenceZahl`: Bei Verwendung der Aggregationsfunktion der Kinesis Producer Library ist dies eine Erweiterung des Checkpoints, der einzelne Benutzerdatensätze innerhalb des Kinesis-Datensatzes verfolgt.
- `leaseCounter`: Wird für ein Lease Versioning verwendet, damit Auftragnehmer erkennen, wenn ihr Lease von einem anderen Auftragnehmer übernommen wurde.
- `leaseKey`: Eine eindeutige Kennung für einen Lease. Jeder Lease gilt für einen bestimmten Shard im Datenstrom und wird immer nur von einem Worker gehalten.



- `leaseOwner`: Der Auftragnehmer, dem der Lease gehört.
- `ownerSwitchesSinceCheckpoint`: Wie oft dieser Lease die Mitarbeiter seit dem letzten Schreiben eines Checkpoints geändert hat.
- `parentShardId`: Wird verwendet, um sicherzustellen, dass der übergeordnete Shard vollständig verarbeitet wird, bevor die Verarbeitung auf den untergeordneten Shards beginnt. So wird sichergestellt, dass Datensätze in der gleichen Reihenfolge verarbeitet werden, in der sie in den Stream eingespeist wurden.
- `hashrange`: Wird von `PeriodicShardSyncManager` verwendet, um regelmäßige Synchronisierungen durchzuführen, um fehlende Shards in der Leasetabelle zu finden und bei Bedarf Leases für sie zu erstellen.

#### Note

Diese Daten sind in der Leasetabelle für jeden Shard enthalten, der mit KCL 1.14 und KCL 2.3 beginnt. Weitere Hinweise zu `PeriodicShardSyncManager` und die periodische Synchronisierung zwischen Leases und Shards finden Sie unter [Wie wird eine Leasetabelle mit den Shards in einem KDS-Datenstrom synchronisiert.](#)

- `childshards`: Wird von `LeaseCleanupManager` verwendet, um den Verarbeitungsstatus des untergeordneten Shards zu überprüfen und zu entscheiden, ob der übergeordnete Shard aus der Leasetabelle gelöscht werden kann.

#### Note

Diese Daten sind in der Leasetabelle für jeden Shard enthalten, der mit KCL 1.14 und KCL 2.3 beginnt.

- `shardID`: Die ID des Shards.

#### Note

Diese Daten sind nur dann in der Leasetabelle enthalten, wenn Sie [Verarbeitung mehrerer Datenströme mit derselben Konsumentenanzwendung KCL 2.x für Java](#) sind. Dies wird nur in KCL 2.x für Java unterstützt, beginnend mit KCL 2.3 für Java und darüber hinaus.

- `StreamName` Die Kennung des Datenstroms im folgenden Format: `account-id:StreamName:streamCreationTimestamp`.

**Note**

Diese Daten sind nur dann in der Leasetabelle enthalten, wenn Sie [Verarbeitung mehrerer Datenströme mit derselben Konsumentenanzwendung KCL 2.x für Java](#) sind. Dies wird nur in KCL 2.x für Java unterstützt, beginnend mit KCL 2.3 für Java und darüber hinaus.

## Durchsatz

Wenn Ihre Amazon Kinesis Data Streams-Anwendung Ausnahmen für den bereitgestellten Durchsatz erhält, sollten Sie den bereitgestellten Durchsatz für die DynamoDB-Tabelle erhöhen. Die KCL erstellt die Tabelle mit einem bereitgestellten Durchsatz von 10 Lese- und 10 Schreibvorgängen pro Sekunde, dies reicht aber möglicherweise für Ihre Anwendung nicht aus. Beispiel: Wenn Ihre Amazon Kinesis Data Streams-Anwendung häufig Prüfpunkte setzt oder einen Stream verarbeitet, der aus vielen Shards besteht, müssen Sie den Durchsatz möglicherweise erhöhen.

Weitere Informationen zum bereitgestellten Durchsatz in DynamoDB finden Sie unter [Lese-/Schreibkapazitätsmodus](#) und [Arbeiten mit Tabellen und Daten](#) im Amazon-DynamoDB-Entwicklerhandbuch.

## Wie wird eine Leasetabelle mit den Shards in einem KDS-Datenstrom synchronisiert

Worker in KCL-Konsumentenanzwendungen verwenden Leases, um Shards aus einem bestimmten Datenstrom zu verarbeiten. Die Informationen darüber, welcher Worker zu einem bestimmten Zeitpunkt welchen Shard least, werden in einer Leasetabelle gespeichert. Die Leasetabelle muss mit den neuesten Shard-Informationen aus dem Datenstrom synchronisiert bleiben, während die KCL-Konsumentenanzwendung ausgeführt wird. KCL synchronisiert die Leasetabelle mit den Shard-Informationen, die während des Bootstrappings der Konsumentenanzwendung (entweder wenn die Konsumentenanzwendung initialisiert oder neu gestartet wird) vom Dienst Kinesis Data Streams erfasst werden und auch immer dann, wenn ein Shard, der gerade verarbeitet wird, ein Ende erreicht (Resharding), erreicht. Mit anderen Worten, die Worker oder eine KCL-Konsumentenanzwendung werden mit dem Datenstrom synchronisiert, den sie beim ersten Bootstrap der Konsumentenanzwendung verarbeiten, und immer dann wenn die Konsumentenanzwendung auf ein Reshard-Ereignis des Datenstroms trifft.

## Themen

- [Synchronisation in KCL 1.0–1.13 und KCL 2.0–2.2](#)

- [Synchronisation in KCL 2.x, ab KCL 2.3 und höher](#)
- [Synchronisation in KCL 1.x, ab KCL 1.14 und höher](#)

## Synchronisation in KCL 1.0–1.13 und KCL 2.0–2.2

In KCL 1.0–1.13 und KCL 2.0–2.2 synchronisiert KCL beim Bootstrapping der Konsumentenanzwendung und auch bei jedem Reshard-Ereignis des Datenstroms die Leasetabelle mit den Shard-Informationen, die vom Dienst Kinesis Data Streams abgerufen wurden, indem die `ListShards` oder die `DescribeStream-Discovery`-APIs aufgerufen werden. In allen oben aufgeführten KCL-Versionen führt jeder Worker einer KCL-Konsumentenanzwendung die folgenden Schritte durch, um den Lease-/Shard-Synchronisierungsprozess während des Bootstrappings der Konsumentenanzwendung und bei jedem Stream-Reshard-Ereignis durchzuführen:

- Ruft alle Shards für Daten des Streams ab, der gerade verarbeitet wird
- Ruft alle Shard-Leases aus der Leasetabelle ab
- Filtert jeden offenen Shard heraus, für den es in der Leasetabelle keinen Lease gibt
- Iteriert über alle gefundenen offenen Shards und für jeden offenen Shard ohne offenes übergeordnetes Element:
  - Durchläuft den Hierarchiebaum über den Pfad der Vorgänger, um festzustellen, ob der Shard ein Nachfolger ist. Ein Shard wird als untergeordnetes Element betrachtet, wenn gerade ein Vorgänger-Shard verarbeitet wird (der Leaseeintrag für den Vorgänger-Shard ist in der Leasetabelle vorhanden) oder wenn ein Vorgänger-Shard verarbeitet werden soll (wenn die Anfangsposition beispielsweise `TRIM_HORIZON` oder `AT_TIMESTAMP` ist)
  - Handelt es sich bei dem offenen Shard im Kontext um einen Nachfolger-Shard, überprüft KCL den Shard anhand seiner Ausgangsposition und erstellt, falls erforderlich, Leaseverträge für die übergeordneten Shard

## Synchronisation in KCL 2.x, ab KCL 2.3 und höher

Beginnend mit den neuesten unterstützten Versionen von KCL 2.x (KCL 2.3) und höher unterstützt die Bibliothek nun die folgenden Änderungen am Synchronisationsprozess. Diese Änderungen an der Lease-/Shard-Synchronisierung reduzieren die Anzahl der API-Aufrufe, die von KCL-Konsumentenanzwendungen an den Service Kinesis Data Streams getätigt werden, erheblich und optimieren das Lease-Management in Ihrer KCL-Konsumentenanzwendung.

- Wenn beim Bootstrapping der Anwendung die Leasetabelle leer ist, verwendet KCL die Filteroption der `ListShard`-API (den optionalen Anforderungsparameter von `ShardFilter`), um Leases nur für einen Snapshot von Shards abzurufen und zu erstellen, die zu dem durch den `ShardFilter`-Parameter angegebenen Zeitpunkt geöffnet waren. Mit dem `ShardFilter`-Parameter können Sie die Antwort der `ListShards`-API herausfiltern. Die einzige erforderliche Eigenschaft des `ShardFilter`-Parameters ist `Type`. KCL verwendet die `Type`-Filtereigenschaft und die folgenden gültigen Werte, um offene Shards, für die möglicherweise neue Leases erforderlich sind, zu identifizieren und eine Momentaufnahme zurückzugeben:
  - `AT_TRIM_HORIZON` – Die Antwort umfasst alle Shards, die am `TRIM_HORIZON` geöffnet waren.
  - `AT_LATEST` – Die Antwort enthält nur die aktuell geöffneten Shards des Datenstroms.
  - `AT_TIMESTAMP` – Die Antwort umfasst alle Shards, deren Startzeitstempel kleiner oder gleich dem angegebenen Zeitstempel sind und deren Endzeitstempel größer oder gleich dem angegebenen Zeitstempel ist oder solche, die noch offen sind.

`ShardFilter` wird verwendet, wenn Leases für eine leere Leasetabelle erstellt werden, um Leases für einen Snapshot von Shards zu initialisieren, die unter `RetrievalConfig#initialPositionInStreamExtended` angegeben sind.

Mehr über `ShardFilter` erfahren Sie unter [https://docs.aws.amazon.com/kinesis/latest/APIReference/API\\_ShardFilter.html](https://docs.aws.amazon.com/kinesis/latest/APIReference/API_ShardFilter.html).

- Anstatt dass alle Worker die Leasing-/Shard-Synchronisierung durchführen, um die Leasetabelle mit den neuesten Shards im Datenstrom auf dem neuesten Stand zu halten, führt ein einziger gewählter Worker-Leader die Lease/Shard-Synchronisierung durch.
- KCL 2.3 verwendet den `ChildShards`-Rückgabeparameter der `GetRecords`- und der `SubscribeToShard`-APIs, um die Leasing-/Shard-Synchronisierung durchzuführen, die bei `SHARD_END` für geschlossene Shards stattfindet, sodass ein KCL-Worker nur Leases für die untergeordneten Shards des Shards erstellen kann, dessen Verarbeitung er abgeschlossen hat. Bei gemeinsam genutzten Konsumenten Anwendungen verwendet diese Optimierung der Lease/Shard-Synchronisierung den `ChildShards`-Parameter der `GetRecords`-API. Bei Konsumenten Anwendungen mit dediziertem Durchsatz (Enhanced Fan-out) wird für diese Optimierung der Lease/Shard-Synchronisierung der `ChildShards`-Parameter der `SubscribeToShard`-API verwendet. Weitere Informationen finden Sie unter [GetRecords](#), [SubscribeToShards](#) und [ChildShard](#).
- Mit den oben genannten Änderungen ändert sich das Verhalten von KCL von dem Modell, dass alle Worker über alle vorhandenen Shards lernen, hin zu dem Modell, dass Worker nur noch über die untergeordneten Shards der Shards lernen, die jeder Worker besitzt. Daher führt KCL jetzt

zusätzlich zu der Synchronisation, die beim Bootstrapping von Konsumenten Anwendungen und bei Reshard-Ereignissen stattfindet, auch zusätzliche regelmäßige Shard-/Lease-Scans durch, um potenzielle Lücken in der Leasetabelle zu identifizieren (mit anderen Worten, um mehr über alle neuen Shards zu erfahren), um sicherzustellen, dass der gesamte Hash-Bereich des Datenstroms verarbeitet wird, und um bei Bedarf Leases für sie zu erstellen. `PeriodicShardSyncManager` ist die Komponente, die für die regelmäßige Ausführung von Lease-/Shard-Scans verantwortlich ist.

Weitere Informationen zu `PeriodicShardSyncManager` in KCL 2.3 finden Sie unter <https://github.com/aws-labs/amazon-kinesis-client/blob/master/amazon-kinesis-client/src/main/java/software/amazon/kinesis/leases/LeaseManagementConfig.java#L201-L213>.

In KCL 2.3 stehen neue Konfigurationsoptionen, um `PeriodicShardSyncManager` in `LeaseManagementConfig` zu konfigurieren:

Name	Standardwert	Beschreibung
leasesRecoveryAuditorExecutionFrequencyMillis	120 000 (2 Minuten)	Häufigkeit (in Millionen), mit der der Auditor in der Leasetabelle nach teilweise n Leases sucht. Wenn der Auditor eine Lücke in den Leases für einen Stream feststellt, würde er die Shard-Synchronisierung auf der Grundlage von leasesRecoveryAuditorInconsistencyConfidenceThreshold auslösen.

Name	Standardwert	Beschreibung
leasesRecoveryAuditorInconsistencyConfidenceThreshold	3	Konfidenzschwellenwert für die regelmäßige Auditortätigkeit, um festzustellen, ob die Leases für einen Datenstrom in der Leasetabelle inkonsistent sind. Wenn der Auditor für diesen Datenstrom mehrmals hintereinander dieselben Inkonsistenzen feststellt, würde er eine Shard-Synchronisierung auslösen.

Neue CloudWatch Metriken werden jetzt auch ausgegeben, um den Zustand des zu überwachen `PeriodicShardSyncManager`. Weitere Informationen finden Sie unter [PeriodicShardSyncManager](#).

- Einschließlich einer Optimierung auf `HierarchicalShardSyncer`, um nur Leases für eine Shard-Ebene zu erstellen.

Synchronisation in KCL 1.x, ab KCL 1.14 und höher

Beginnend mit den neuesten unterstützten Versionen von KCL 1.x (KCL 1.14) und höher unterstützt die Bibliothek nun die folgenden Änderungen am Synchronisationsprozess. Diese Änderungen

an der Lease-/Shard-Synchronisierung reduzieren die Anzahl der API-Aufrufe, die von KCL-Konsumenten Anwendungen an den Service Kinesis Data Streams getätigt werden, erheblich und optimieren das Lease-Management in Ihrer KCL-Konsumenten Anwendung.

- Wenn beim Bootstrapping der Anwendung die Leasetabelle leer ist, verwendet KCL die Filteroption der `ListShard`-API (den optionalen Anforderungsparameter von `ShardFilter`), um Leases nur für einen Snapshot von Shards abzurufen und zu erstellen, die zu dem durch den `ShardFilter`-Parameter angegebenen Zeitpunkt geöffnet waren. Mit dem `ShardFilter`-Parameter können Sie die Antwort der `ListShards`-API herausfiltern. Die einzige erforderliche Eigenschaft des `ShardFilter`-Parameters ist `Type`. KCL verwendet die `Type`-Filtereigenschaft und die folgenden gültigen Werte, um offene Shards, für die möglicherweise neue Leases erforderlich sind, zu identifizieren und eine Momentaufnahme zurückzugeben:
  - `AT_TRIM_HORIZON` – Die Antwort umfasst alle Shards, die am `TRIM_HORIZON` geöffnet waren.
  - `AT_LATEST` – Die Antwort enthält nur die aktuell geöffneten Shards des Datenstroms.
  - `AT_TIMESTAMP` – Die Antwort umfasst alle Shards, deren Startzeitstempel kleiner oder gleich dem angegebenen Zeitstempel sind und deren Endzeitstempel größer oder gleich dem angegebenen Zeitstempel ist oder solche, die noch offen sind.

`ShardFilter` wird verwendet, wenn Leases für eine leere Leasetabelle erstellt werden, um Leases für einen Snapshot von Shards zu initialisieren, die unter `KinesisClientLibConfiguration#initialPositionInStreamExtended` angegeben sind.

Mehr über `ShardFilter` erfahren Sie unter [https://docs.aws.amazon.com/kinesis/latest/APIReference/API\\_ShardFilter.html](https://docs.aws.amazon.com/kinesis/latest/APIReference/API_ShardFilter.html).

- Anstatt dass alle Worker die Leasing-/Shard-Synchronisierung durchführen, um die Leasetabelle mit den neuesten Shards im Datenstrom auf dem neuesten Stand zu halten, führt ein einziger gewählter Worker-Leader die Lease/Shard-Synchronisierung durch.
- KCL 1.14 verwendet den `ChildShards`-Rückgabeparameter der `GetRecords`- und der `SubscribeToShard`-APIs, um die Leasing-/Shard-Synchronisierung durchzuführen, die bei `SHARD_END` für geschlossene Shards stattfindet, sodass ein KCL-Worker nur Leases für die untergeordneten Shards des Shards erstellen kann, dessen Verarbeitung er abgeschlossen hat. Weitere Informationen erhalten Sie unter [GetRecords](#) und [ChildShard](#).
- Mit den oben genannten Änderungen ändert sich das Verhalten von KCL von dem Modell, dass alle Worker über alle vorhandenen Shards lernen, hin zu dem Modell, dass Worker nur noch über die untergeordneten Shards der Shards lernen, die jeder Worker besitzt. Daher führt KCL jetzt



zusätzlich zu der Synchronisation, die beim Bootstrapping von Konsumenten Anwendungen und bei Reshard-Ereignissen stattfindet, auch zusätzliche regelmäßige Shard-/Lease-Scans durch, um potenzielle Lücken in der Leasetabelle zu identifizieren (mit anderen Worten, um mehr über alle neuen Shards zu erfahren), um sicherzustellen, dass der gesamte Hash-Bereich des Datenstroms verarbeitet wird, und um bei Bedarf Leases für sie zu erstellen. `PeriodicShardSyncManager` ist die Komponente, die für die regelmäßige Ausführung von Lease-/Shard-Scans verantwortlich ist.

Wenn `KinesisClientLibConfiguration#shardSyncStrategyType` auf `ShardSyncStrategyType.SHARD_END` gesetzt ist, wird `PeriodicShardSyncLeasesRecoveryAuditorInconsistencyConfidenceThreshold` verwendet, um den Schwellenwert für die Anzahl der aufeinanderfolgenden Scans mit Lücken in der Leasetabelle zu bestimmen, nach dem eine Shard-Synchronisierung erzwungen werden soll. Wenn `KinesisClientLibConfiguration#shardSyncStrategyType` auf `ShardSyncStrategyType.PERIODIC` gesetzt ist, wird `LeasesRecoveryAuditorInconsistencyConfidenceThreshold` ignoriert.

Weitere Informationen zu `PeriodicShardSyncManager` in KCL 1.14 finden Sie unter <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>.

In KCL 1.14 ist eine neue Konfigurationsoption verfügbar, um `PeriodicShardSyncManager` in `LeaseManagementConfig` zu konfigurieren:

Name	Standardwert	Beschreibung
leasesRec overyAuditorInconsistencyConfidenceThreshold	3	Konfidenzschwelle für die regelmäßige Auditortätigkeit, um festzustellen, ob die Leases für einen Datenstrom in der Leasetabelle inkonsistent sind. Wenn der Auditor für diesen Datenstrom mehrmals hintereinander dieselben Inkonsistenzen feststellt, würde er eine Shard-Synchronisierung auslösen.

Neue CloudWatch Metriken werden jetzt auch ausgegeben, um den Zustand des zu überwachen `PeriodicShardSyncManager`. Weitere Informationen finden Sie unter [PeriodicShardSyncManager](#).

- KCL 1.14 unterstützt jetzt auch die verzögerte Lease-Bereinigung. Leases werden asynchron von `LeaseCleanupManager` bei Erreichen von `SHARD_END` gelöscht, wenn ein Shard entweder die Aufbewahrungsfrist des Datenstroms überschritten hat oder wenn er aufgrund eines Resharding-Vorgangs geschlossen wurde.

Es stehen neue Konfigurationsoptionen zur Verfügung, um `LeaseCleanupManager` zu konfigurieren.

Name	Standardwert	Beschreibung
<code>leaseCleanupIntervalMillis</code>	1 Minute	Intervall, in dem der Lease-Cleanup-Thread ausgeführt werden soll.
<code>completedLeaseCleanupIntervalMillis</code>	5 Minuten	Intervall, in dem überprüft werden soll, ob ein Lease abgeschlossen ist oder nicht.
<code>garbageLeaseCleanupIntervalMillis</code>	30 Minuten	Intervall, in dem geprüft werden soll, ob ein Lease Datenmüll ist (d. h. über die Aufbewahrungsfrist des Datenstroms hinaus abgeschnitten wurde) oder nicht.

- Einschließlich einer Optimierung auf `KinesisShardSyncer`, um nur Leases für eine Shard-Ebene zu erstellen.

## Verarbeitung mehrerer Datenströme mit derselben Konsumentenanzwendung KCL 2.x für Java

In diesem Abschnitt werden die folgenden Änderungen in KCL 2.x für Java beschrieben, mit denen Sie KCL-Konsumentenanzwendungen erstellen können, die mehr als einen Datenstrom gleichzeitig verarbeiten können.

### Important

Die Multistream-Verarbeitung wird nur in KCL 2.x für Java unterstützt, beginnend mit KCL 2.3 für Java und darüber hinaus.

Die Multistream-Verarbeitung wird NICHT für andere Sprachen unterstützt, in denen KCL 2.x implementiert werden kann.

Die Multistream-Verarbeitung wird in keiner Version von KCL 1.x unterstützt.

- MultistreamTracker -Schnittstelle

Um eine Konsumentenanzwendung zu erstellen, die mehrere Streams gleichzeitig verarbeiten kann, müssen Sie eine neue Schnittstelle namens `implementieren` [MultistreamTracker](#). Diese Schnittstelle enthält die `streamConfigList`-Methode, die die Liste der Datenströme und ihrer Konfigurationen zurückgibt, die von der KCL-Konsumentenanzwendung verarbeitet werden sollen. Beachten Sie, dass die Datenströme, die verarbeitet werden, während der Laufzeit der Konsumentenanzwendung geändert werden können. `streamConfigList` wird regelmäßig von der KCL aufgerufen, um mehr über die Änderungen der zu verarbeitenden Datenströme zu erfahren.

Die `-streamConfigList`Methode füllt die [StreamConfig](#) Liste aus.

```
package software.amazon.kinesis.common;

import lombok.Data;
import lombok.experimental.Accessors;

@Data
@Accessors(fluent = true)
public class StreamConfig {
    private final StreamIdentifier streamIdentifier;
    private final InitialPositionInStreamExtended initialPositionInStreamExtended;
```

```
private String consumerArn;  
}
```

Beachten Sie, dass es sich bei den Feldern `StreamIdentifier` und `InitialPositionInStreamExtended` um Pflichtfelder handelt, während `consumerArn` optional ist. Sie müssen den `consumerArn` nur angeben, wenn Sie KCL 2.x verwenden, um eine erweiterte Fan-Out-Konsumentenanzwendung zu implementieren.

Weitere Informationen zu `StreamIdentifier` finden Sie unter <https://github.com/aws/aws-kinesis-client/blob/0c5042dadf794fe988438436252a5a8fe70b6b0b/amazon-kinesis-client/src/main/java/software/amazon/kinesis/common/StreamIdentifier.java#L29>. Sie können eine `Multistream-Instance` für die `StreamIdentifier` aus dem serialisierten `Stream-Identifier` erstellen. Der serialisierte `Stream-Identifier` sollte das folgende Format haben: `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` beinhaltet auch eine Strategie zum Löschen von Leases alter Streams in der Leasetabelle (`formerStreamsLeasesDeletionStrategy`). Beachten Sie, dass die Strategie während der Laufzeit der Konsumentenanzwendung NICHT geändert werden kann. Weitere Informationen finden Sie unter <https://github.com/aws/aws-kinesis-client/blob/0c5042dadf794fe988438436252a5a8fe70b6b0b/amazon-kinesis-client/src/main/java/software/amazon/kinesis/processor/FormerStreamsLeasesDeletionStrategy.java>

- [ConfigsBuilder](#) ist eine anwendungsweite Klasse, mit der Sie alle KCL-2.x-Konfigurationseinstellungen angeben können, die beim Erstellen Ihrer KCL-Konsumentenanzwendung verwendet werden sollen. Die `-ConfigsBuilder`-Klasse unterstützt jetzt die `-MultiStreamTracker`-Schnittstelle. Sie können entweder mit dem Namen des einen Datenstroms initialisieren `ConfigsBuilder`, aus dem Datensätze verwendet werden sollen:

```

/**
 * Constructor to initialize ConfigsBuilder with StreamName
 * @param streamName
 * @param applicationName
 * @param kinesisClient
 * @param dynamoDBClient
 * @param cloudWatchClient
 * @param workerIdentifier
 * @param shardRecordProcessorFactory
 */
public ConfigsBuilder(@NonNull String streamName, @NonNull String
applicationName,
    @NonNull KinesisAsyncClient kinesisClient, @NonNull DynamoDbAsyncClient
dynamoDBClient,
    @NonNull CloudWatchAsyncClient cloudWatchClient, @NonNull String
workerIdentifier,
    @NonNull ShardRecordProcessorFactory shardRecordProcessorFactory) {
    this.appStreamTracker = Either.right(streamName);
    this.applicationName = applicationName;
    this.kinesisClient = kinesisClient;
    this.dynamoDBClient = dynamoDBClient;
    this.cloudWatchClient = cloudWatchClient;
    this.workerIdentifier = workerIdentifier;
    this.shardRecordProcessorFactory = shardRecordProcessorFactory;
}

```

Oder Sie können `ConfigsBuilder` mit `initialisierenMultiStreamTracker` initialisieren, wenn Sie eine KCL-Konsumentenanzwendung implementieren möchten, die mehrere Streams gleichzeitig verarbeitet.

```

* Constructor to initialize ConfigsBuilder with MultiStreamTracker
 * @param multiStreamTracker
 * @param applicationName
 * @param kinesisClient
 * @param dynamoDBClient

```

```
* @param cloudWatchClient
* @param workerIdentifier
* @param shardRecordProcessorFactory
*/
public ConfigsBuilder(@NonNull MultiStreamTracker multiStreamTracker, @NonNull
String applicationName,
    @NonNull KinesisAsyncClient kinesisClient, @NonNull DynamoDbAsyncClient
dynamoDBClient,
    @NonNull CloudWatchAsyncClient cloudWatchClient, @NonNull String
workerIdentifier,
    @NonNull ShardRecordProcessorFactory shardRecordProcessorFactory) {
    this.appStreamTracker = Either.left(multiStreamTracker);
    this.applicationName = applicationName;
    this.kinesisClient = kinesisClient;
    this.dynamoDBClient = dynamoDBClient;
    this.cloudWatchClient = cloudWatchClient;
    this.workerIdentifier = workerIdentifier;
    this.shardRecordProcessorFactory = shardRecordProcessorFactory;
}
```

- Da die Multistream-Unterstützung für Ihre KCL-Konsumenten-anwendung implementiert ist, enthält jede Zeile der Leasetabelle der Anwendung jetzt die Shard-ID und den Stream-Namen der mehreren Datenströme, die diese Anwendung verarbeitet.
- Wenn die Multistream-Unterstützung für Ihre KCL-Konsumenten-anwendung implementiert ist, hat der leaseKey die folgende Struktur: account-id:StreamName:streamCreationTimestamp:ShardId. Beispiel: 11111111:multiStreamTest-1:12345:shardId-000000000336

#### Important

Wenn Ihre vorhandene KCL-Konsumenten-anwendung so konfiguriert ist, dass sie nur einen Datenstrom verarbeitet, ist der leaseKey (der Hash-Schlüssel für die Leasetabelle) die Shard-ID. Wenn Sie diese vorhandene KCL-Konsumenten-anwendung für die Verarbeitung mehrerer Datenströme neu konfigurieren, wird Ihre Leasetabelle beschädigt, da bei Multistream-Unterstützung die leaseKey-Struktur wie folgt aussehen muss: account-id:StreamName:StreamCreationTimestamp:ShardId.

## Verwenden der Kinesis Client Library mit dem AWS Glue Schema Registry

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

### Important

Derzeit wird die Integration von Kinesis Data Streams und dem AWS Glue Schema Registry nur für Kinesis-Datenströme unterstützt, die in Java implementierte KCL-2.3-Konsumenten verwenden. Mehrsprachige Unterstützung wird nicht bereitgestellt. KCL-1.0-Konsumenten werden nicht unterstützt. KCL-2.x-Konsumenten vor KCL 2.3 werden nicht unterstützt.

Detaillierte Anweisungen zur Einrichtung der Integration von Kinesis Data Streams mit Schema Registry mithilfe der KCL finden Sie im Abschnitt „Interaktion mit Daten mithilfe der KPL/KCL-Bibliotheken“ unter [Anwendungsfall: Integration von Amazon Kinesis Data Streams mit dem AWS Glue Schema Registry](#).

## Entwickeln benutzerdefinierter Verbraucher mit gemeinsam genutztem Durchsatz

Wenn Sie keinen spezifischen Durchsatz beim Empfangen von Daten von Kinesis Data Streams und keine Verbreitungswerte für Leseoperationen von unter 200 ms benötigen, können Sie Konsumenten Anwendungen wie in den folgenden Themen beschrieben erstellen.

### Themen

- [Entwickeln benutzerdefinierter Verbraucher mit gemeinsam genutztem Durchsatz unter Verwendung von KCL](#)
- [Entwickeln benutzerdefinierter Verbraucher mit gemeinsam genutztem Durchsatz unter Verwendung von AWS SDK for Java](#)



Für Informationen zum Erstellen von Verbrauchern, die Datensätze aus Kinesis-Datenströmen mit dediziertem Durchsatz empfangen können, siehe [Entwickeln benutzerdefinierter Verbraucher mit dediziertem Durchsatz \(Erweitertes Rundsenden\)](#).

## Entwickeln benutzerdefinierter Verbraucher mit gemeinsam genutztem Durchsatz unter Verwendung von KCL

Eine der Methoden zur Entwicklung einer benutzerdefinierten Verbraucheranwendung mit gemeinsamem Durchsatz ist die Verwendung der Kinesis Client Library (KCL).

### Themen

- [Entwicklung von KCL 1.x-Verbrauchern](#)
- [Entwicklung von KCL 2.x-Verbrauchern](#)

### Entwicklung von KCL 1.x-Verbrauchern

Sie können eine Verbraucheranwendung für Amazon Kinesis Data Streams entwickeln, indem Sie die Kinesis Client Library (KCL) verwenden. Weitere Informationen zu KCL finden Sie unter [Was ist die Kinesis Client Library?](#) .

### Inhalt

- [Entwickeln eines Kinesis Client Library-Verbrauchers in Java](#)
- [Entwickeln eines Kinesis-Client-Library-Verbrauchers in Node.js](#)
- [Entwickeln eines Kinesis Client Library-Verbrauchers in .NET](#)
- [Entwickeln eines Kinesis Client Library-Verbrauchers in Python](#)
- [Entwickeln eines Kinesis Client Library-Verbrauchers in Ruby](#)

### Entwickeln eines Kinesis Client Library-Verbrauchers in Java

Sie können die Kinesis Client Library (KCL) verwenden, um Anwendungen zu erstellen, die Daten aus Ihren Kinesis-Datenströmen verarbeiten. Die Kinesis Client Library ist in mehreren Sprachen verfügbar. In diesem Thema wird Java behandelt. Informationen zum Anzeigen der Javadoc-Referenz finden Sie im [AWS Javadoc-Thema für Klasse AmazonKinesisClient](#).

Um die Java KCL von herunterzuladen GitHub, gehen Sie zur [Kinesis Client Library \(Java\)](#). Um die Java KCL auf Apache Maven zu finden, navigieren Sie zur Seite für die [KCL-Suchergebnisse](#). Um

Beispielcode für eine Java-KCL-Konsumentenanzwendung von herunterzuladen GitHub, gehen Sie zur Seite für ein [KCL-für-Java-Beispielprojekt](#) auf GitHub.

Die Beispielanzwendung verwendet [Apache Commons Logging](#). Sie können die Konfiguration der Protokollierung in der statischen Methode `configure` ändern, die in der Datei `AmazonKinesisApplicationSample.java` definiert ist. Weitere Informationen zur Verwendung von Apache Commons Logging mit Log4j und AWS-Java-Anwendungen finden Sie unter [Protokollieren mit Log4j](#) im AWS SDK for Java-Entwicklerhandbuch.

Sie müssen die folgenden Aufgaben durchführen, wenn Sie eine KCL-Konsumentenanzwendung in Java implementieren:

### Aufgaben

- [Implementieren der I-RecordProcessor Methoden](#)
- [Implementieren einer Class Factory für die IRecordProcessor -Schnittstelle](#)
- [Erstellen von Auftragnehmern](#)
- [Ändern der Konfigurationseigenschaften](#)
- [Migration zu Version 2 der Datensatzverarbeiterschnittstelle](#)

### Implementieren der I-RecordProcessor Methoden

Die KCL unterstützt zurzeit zwei Versionen der `IRecordProcessor`-Schnittstelle: die ursprüngliche Schnittstelle, die mit der ersten Version der KCL verfügbar war, und Version 2, die ab KCL Version 1.5.0 verfügbar ist. Beide Schnittstellen werden vollständig unterstützt. Ihre Wahl hängt von den speziellen Anforderungen Ihres Anwendungsfalls ab. Um mehr über Unterschiede zu erfahren, betrachten Sie die lokal entwickelten Javadocs oder den Quellcode. In den folgenden Abschnitten wird die Mindestimplementierung für die ersten Schritte beschrieben.

### Ich -RecordProcessor Versionen

- [Ursprüngliche Schnittstelle \(Version 1\)](#)
- [Aktualisierte Schnittstelle \(Version 2\)](#)

### Ursprüngliche Schnittstelle (Version 1)

Die ursprüngliche `IRecordProcessor` Schnittstelle (package `com.amazonaws.services.kinesis.clientlibrary.interfaces`) stellt die folgenden Datensatzverarbeitermethoden bereit, die Ihr Konsument implementieren muss. Das Beispiel

stellt Implementierungen bereit, die Sie als Ausgangspunkt verwenden können (siehe `AmazonKinesisApplicationSampleRecordProcessor.java`).

```
public void initialize(String shardId)
public void processRecords(List<Record> records, IRecordProcessorCheckpointter
    checkpointter)
public void shutdown(IRecordProcessorCheckpointter checkpointter, ShutdownReason reason)
```

## initialize

Die KCL ruft die Methode `initialize` auf, wenn der Datensatzverarbeiter instanziiert wird, und übergibt eine spezifische Shard-ID als Parameter. Dieser Datensatzverarbeiter verarbeitet nur diese Shard und in der Regel ist dies auch umgekehrt der Fall (diese Shard wird nur durch diesen Datensatzverarbeiter verarbeitet). Ihr Konsument sollte jedoch die Möglichkeit berücksichtigen, dass ein Datensatz mehr als einmal verarbeitet werden könnte. Kinesis Data Streams besitzt eine Semantik nach dem Grundsatz mindestens einmal. Das bedeutet, dass jeder Datensatz aus einer Shard mindestens einmal von einem Worker in Ihrem Konsumenten verarbeitet wird. Weitere Informationen zu Fällen, in denen eine bestimmte Shard möglicherweise durch mehr als einen Auftragnehmer verarbeitet wird, finden Sie unter [Resharding, Skalierung und Parallelverarbeitung](#).

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

## processRecords

Die KCL ruft die Methode `processRecords` auf und übergibt eine Liste der Datensätze aus der Shard, die von der Methode `initialize(shardId)` angegeben wird. Der Datensatzverarbeiter verarbeitet die Daten in diesen Datensätzen entsprechend der Semantik des Konsumenten. Beispielsweise kann der Auftragnehmer eine Transformation für die Daten ausführen und das Ergebnis dann in einem Amazon Simple Storage Service (Amazon S3)-Bucket speichern.

```
public void processRecords(List<Record> records, IRecordProcessorCheckpointter
    checkpointter)
```

Zusätzlich zu den Daten selbst enthält der Datensatz auch eine Sequenznummer und einen Partitionsschlüssel. Der Auftragnehmer kann diese Werte beim Verarbeiten der Daten verwenden. Beispielsweise könnte der Auftragnehmer basierend auf dem Wert des Partitionsschlüssels den S3-Bucket wählen, in dem die Daten gespeichert werden sollen. Die Klasse `Record` stellt die folgenden Methoden bereit, die Zugriff auf die Daten des Datensatzes, die Sequenznummer und den Partitionsschlüssel bieten.

```
record.getData()  
record.getSequenceNumber()  
record.getPartitionKey()
```

In diesem Beispiel weist die private Methode `processRecordsWithRetries` Code auf, der zeigt, wie ein Auftragnehmer auf die Daten des Datensatzes, die Sequenznummer und den Partitionsschlüssel zugreifen kann.

Kinesis Data Streams erfordert, dass der Datensatzverarbeiter die Datensätze nachverfolgt, die bereits in einer Shard verarbeitet wurden. Die KCL übernimmt diese Nachverfolgung für Sie, indem ein Checkpointer (`IRecordProcessorCheckpoint`) an `processRecords` übergeben wird. Der Datensatzverarbeiter ruft die Methode `checkpoint` auf dieser Schnittstelle auf, um die KCL über die Fortschritte zu informieren, die sie beim Verarbeiten der Datensätze in der Shard gemacht hat. Wenn der Auftragnehmer fehlschlägt, verwendet die KCL diese Informationen, um die Verarbeitung der Shard mit dem letzten bekannten Datensatz neu zu starten.

Im Fall einer Teilungs- oder Zusammenführungsoperation beginnt die KCL erst dann mit der Verarbeitung der neuen Shards, wenn die Verarbeiter für die ursprünglichen Shards `checkpoint` aufgerufen haben, um zu signalisieren, dass die Verarbeitung der ursprünglichen Shards vollständig abgeschlossen ist.

Wenn Sie keinen Parameter übergeben, nimmt die KCL an, dass der Aufruf von `checkpoint` bedeutet, dass alle Datensätze bis zum letzten Datensatz, der an den Datensatzverarbeiter übergeben wurde, verarbeitet wurden. Daher sollte der Datensatzverarbeiter die Methode `checkpoint` erst aufrufen, wenn er alle Datensätze in der Liste, die ihm übergeben wurden, verarbeitet hat. Datensatzverarbeiter müssen `checkpoint` nicht bei jedem Aufruf von `processRecords` aufrufen. Ein Prozessor könnte beispielsweise `checkpoint` bei jedem dritten Aufruf von `processRecords` aufrufen. Sie können optional die exakte Sequenznummer eines Datensatzes als Parameter für `checkpoint` angeben. In diesem Fall nimmt die KCL an, dass alle Datensätze nur bis zu diesem Datensatz verarbeitet wurden.

Im Beispiel zeigt die private Methode `checkpoint`, wie `IRecordProcessorCheckpoint.checkpoint` mithilfe der entsprechenden Ausnahmebehandlung und Wiederholungslogik aufgerufen wird.

Die KCL ist bei der Behandlung von Ausnahmen, die während der Verarbeitung der Datensätze auftreten, von `processRecords` abhängig. Wenn `processRecords` eine Ausnahme aufwirft, überspringt die KCL die Datensätze, die vor der Ausnahme übergeben wurden. Das heißt, diese

Datensätze werden nicht erneut an den Datensatzprozessor gesendet, der die Ausnahme ausgelöst hat, oder an einen anderen Datensatzprozessor im Verbraucher.

## shutdown

Die KCL ruft die Methode `shutdown` entweder auf, wenn die Verarbeitung beendet wird (Grund für das Herunterfahren ist `TERMINATE`) oder wenn der Auftragnehmer nicht mehr reagiert (Grund für das Herunterfahren ist `ZOMBIE`).

```
public void shutdown(IRecordProcessorCheckpoint checkpoint, ShutdownReason reason)
```

Die Verarbeitung endet, wenn der Datensatzverarbeiter keine weiteren Datensätze aus der Shard erhält, entweder weil die Shard geteilt oder zusammengeführt wurde oder weil der Stream gelöscht wurde.

Die KCL übergibt auch eine `IRecordProcessorCheckpoint`-Schnittstelle an `shutdown`. Wenn der Grund für das Herunterfahren `TERMINATE` ist, sollte der Datensatzverarbeiter alle Datensätze fertigstellen und dann die Methode `checkpoint` in seiner Schnittstelle aufrufen.

## Aktualisierte Schnittstelle (Version 2)

Die aktualisierte `IRecordProcessor` Schnittstelle (`package com.amazonaws.services.kinesis.clientlibrary.interfaces.v2`) stellt die folgenden Datensatzverarbeitermethoden bereit, die Ihr Konsument implementieren muss:

```
void initialize(InitializationInput initializationInput)
void processRecords(ProcessRecordsInput processRecordsInput)
void shutdown(ShutdownInput shutdownInput)
```

Sie können auf alle Argumente aus der ursprünglichen Version der Schnittstelle über Get-Methoden für die Container-Objekte zugreifen. Um die Liste der Datensätze in `processRecords()` abzurufen, können Sie `processRecordsInput.getRecords()` verwenden.

Ab Version 2 dieser Schnittstelle (KCL 1.5.0 und höher) sind zusätzlich zu den Eingaben durch die ursprüngliche Schnittstelle die folgenden neuen Eingaben verfügbar:

## Startsequenznummer

Im `InitializationInput`-Objekt, das an die Operation `initialize()` übergeben wird, die Startsequenznummer, aus der Datensätze für die Datenverarbeiter-Instance bereitgestellt würden. Dies ist die Sequenznummer, die zuletzt durch die Datensatzverarbeiter-Instance überprüft wurde,

die dieselbe Shard zuvor verarbeitet hat. Sie wird für den Fall angegeben, dass Ihre Anwendung diese Informationen benötigt.

### Ausstehende Checkpoint-Sequenznummer

Im `InitializationInput`-Objekt, das an die Operation `initialize()` übergeben wird, die ausstehende Checkpoint-Sequenznummer (wenn vorhanden), die nicht übergeben werden konnte, bevor die vorherige Datensatzverarbeiter-Instance angehalten wurde.

### Implementieren einer Class Factory für die `IRecordProcessor`-Schnittstelle

Sie müssen darüber hinaus eine Factory für die Klasse implementieren, die die Datensatzverarbeitermethoden implementiert. Wenn der Konsument den Auftragnehmer instanziiert, übergibt er dieser Factory eine Referenz.

Im Beispiel wird die Factory-Klasse in der Datei `AmazonKinesisApplicationSampleRecordProcessorFactory.java` mithilfe der ursprünglichen Datensatzverarbeiter-Schnittstelle implementiert. Wenn Sie möchten, dass die Class Factory Datensatzverarbeiter mit Version 2 erstellt, verwenden Sie den Paketnamen `com.amazonaws.services.kinesis.clientlibrary.interfaces.v2`.

```
public class SampleRecordProcessorFactory implements IRecordProcessorFactory {
    /**
     * Constructor.
     */
    public SampleRecordProcessorFactory() {
        super();
    }
    /**
     * {@inheritDoc}
     */
    @Override
    public IRecordProcessor createProcessor() {
        return new SampleRecordProcessor();
    }
}
```

### Erstellen von Auftragnehmern

Wie in [Implementieren der I-RecordProcessor Methoden](#) beschrieben, gibt es zwei Versionen der KCL-Datensatzverarbeiterschnittstelle zur Auswahl. Die Version hat Auswirkungen auf die Art,

wie Sie einen Worker erstellen. Die ursprüngliche Datensatzverarbeiterschnittstelle verwendet die folgende Codestruktur, um einen Auftragnehmer zu erstellen:

```
final KinesisClientLibConfiguration config = new KinesisClientLibConfiguration(...)
final IRecordProcessorFactory recordProcessorFactory = new RecordProcessorFactory();
final Worker worker = new Worker(recordProcessorFactory, config);
```

Mit Version 2 der Datensatzverarbeiterschnittstelle können Sie `Worker.Builder` verwenden, um einen Auftragnehmer zu erstellen, ohne sich Gedanken über den Konstruktor und die Reihenfolge der Argumente zu machen. Die aktualisierte Datensatzverarbeiterschnittstelle verwendet die folgende Codestruktur, um einen Auftragnehmer zu erstellen:

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

## Ändern der Konfigurationseigenschaften

Das Beispiel zeigt Standardwerte für Konfigurationseigenschaften. Diese Konfigurationsdaten für den Auftragnehmer werden anschließend in einem `KinesisClientLibConfiguration`-Objekt konsolidiert. Dieses Objekt und eine Referenz auf die Class Factory für `IRecordProcessor` werden an den Aufruf übergeben, der den Auftragnehmer instanziiert. Sie können diese Eigenschaften mithilfe einer Java-Eigenschaftendatei (siehe `AmazonKinesisApplicationSample.java`) durch eigene Werte überschreiben.

## Anwendungsname

Die KCL erfordert einen Anwendungsnamen, der unter Ihren Anwendungen sowie den Amazon-DynamoDB-Tabellen in derselben Region eindeutig ist. Sie verwendet den Wert der Anwendungsnamenkonfiguration auf folgende Arten:

- Für mit diesem Anwendungsnamen verknüpfte Auftragnehmer wird angenommen, dass sie gemeinsam im gleichen Stream arbeiten. Diese Auftragnehmer können auf mehrere Instances verteilt sein. Wenn Sie eine zusätzliche Instance desselben Anwendungscodes ausführen, jedoch mit einem anderen Anwendungsnamen, behandelt die KCL die zweite Instance als eine völlig getrennte Anwendung, die ebenfalls im selben Stream arbeitet.

- Die KCL erstellt eine DynamoDB-Tabelle mit dem Namen der Anwendung und verwendet die Tabelle für die Verwaltung von Statusinformationen für die Anwendung (wie Checkpoints und Auftragnehmer-Shard-Zuweisungen). Jede Anwendung verfügt über eine eigene DynamoDB-Tabelle. Weitere Informationen finden Sie unter [Mithilfe einer Leasetabelle können Sie die von der KCL-Konsumentenanzwendung verarbeiteten Shards verfolgen](#).

## Einrichten von Anmeldeinformationen

Sie müssen Ihre AWS-Anmeldeinformationen für einen der Anmeldeinformationsanbieter in der Anmeldeinformationsanbieter-Standardkette bereitstellen. Wenn Sie beispielsweise Ihren Konsumenten auf einer EC2-Instance ausführen, empfehlen wir, die Instance mit einer IAM-Rolle zu starten. AWS-Anmeldeinformationen, die die mit dieser IAM-Rolle verknüpften Berechtigungen widerspiegeln, werden den Anwendungen auf der Instance über deren Instance-Metadaten zur Verfügung gestellt. Dies ist die sicherste Art, Anmeldeinformationen für einen Konsumenten zu verwalten, der auf einer EC2-Instance ausgeführt wird.

Die Beispielanwendung versucht zunächst, IAM-Anmeldeinformationen aus den Instance-Metadaten abzurufen:

```
credentialsProvider = new InstanceProfileCredentialsProvider();
```

Wenn die Beispielanwendung keine Anmeldeinformationen aus den Instance-Metadaten abrufen kann, versucht sie, Anmeldeinformationen aus einer Eigenschaftendatei abzurufen:

```
credentialsProvider = new ClasspathPropertiesFileCredentialsProvider();
```

Weitere Informationen über Instance-Metadaten finden Sie unter [Instance-Metadaten](#) im Benutzerhandbuch zu Amazon EC2 für Linux-Instances.

## Verwenden der Auftragnehmer-ID für mehrere Instances

Derselbe Initialisierungscode erstellt unter Verwendung des Namens des lokalen Computers und Anfügung eines global eindeutigen Bezeichners eine ID für den Auftragnehmer, `workerId`, wie im folgenden Codeauszug gezeigt. Dieser Ansatz unterstützt das Szenario mit mehreren Instances der Konsumentenanzwendung, die auf einem einzigen Computer ausgeführt werden.

```
String workerId = InetAddress.getLocalHost().getCanonicalHostName() + ":" +  
    UUID.randomUUID();
```



## Migration zu Version 2 der Datensatzverarbeiterschnittstelle

Wenn Sie Code migrieren möchten, der die ursprüngliche Schnittstelle verwendet, sind zusätzlich zu den zuvor beschriebenen Schritten die folgenden Schritte erforderlich:

1. Ändern der Datensatzverarbeiterklasse, um Version 2 der Datensatzverarbeiterschnittstelle zu importieren:

```
import com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor;
```

2. Ändern der Referenzen zu Eingaben, um get-Methoden für die Container-Objekte zu verwenden. In der Operation `shutdown()` ändern Sie beispielsweise „`checkpointer`“ in „`shutdownInput.getCheckpointer()`“.
3. Ändern der Datensatzverarbeiter-Factory, um Version 2 der Datensatzverarbeiter-Factory zu importieren:

```
import  
    com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessorFactory;
```

4. Ändern der Konstruktion für den Auftragnehmer, um `Worker.Builder` zu verwenden. Beispielsweise:

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

## Entwickeln eines Kinesis-Client-Library-Verbrauchers in Node.js

Sie können die Kinesis Client Library (KCL) verwenden, um Anwendungen zu erstellen, die Daten aus Ihren Kinesis-Datenströmen verarbeiten. Die Kinesis Client Library ist in mehreren Sprachen verfügbar. In diesem Thema wird Node.js behandelt.

Die KCL ist eine Java-Bibliothek. Unterstützung für andere Sprachen als Java wird über eine mehrsprachige Schnittstelle namens `bereitgestelltMultiLangDaemon`. Dieser Daemon basiert auf Java und wird im Hintergrund ausgeführt, wenn Sie eine andere KCL-Sprache als Java verwenden. Wenn Sie also die KCL für Node.js installieren und Ihre Konsumenten-App vollständig in Node.js schreiben, benötigen Sie aufgrund der trotzdem Java auf Ihrem System `MultiLangDaemon`. Darüber hinaus `MultiLangDaemon` verfügt über einige Standardeinstellungen, die Sie möglicherweise

an Ihren Anwendungsfall anpassen müssen, z. B. die AWS Region, mit der es eine Verbindung herstellt. Weitere Informationen über die MultiLangDaemon auf finden Sie auf der [KCL-MultiLangDaemon Projektseite](#).

Um die Node.js-KCL von herunterzuladen GitHub, gehen Sie zur [Kinesis Client Library \(Node.js\)](#).

Downloads von Beispiel-Code

Es gibt zwei Code-Beispiele für die KCL in Node.js:

- [basic-sample](#)

Wird in den folgenden Abschnitten verwendet, um die Grundlagen zum Erstellen einer KCL-Konsumentenanzwendung in Node.js zu zeigen.

- [click-stream-sample](#)

Etwas komplexere und verwendet ein reales Szenario, nachdem Sie sich mit dem grundlegenden Beispiel-Code vertraut gemacht haben. Dieses Beispiel wird hier nicht behandelt. Es gibt jedoch eine Readme-Datei mit weiteren Informationen.

Sie müssen die folgenden Aufgaben durchführen, wenn Sie eine KCL-Konsumentenanzwendung in Node.js implementieren:

Aufgaben

- [Implementieren des Datensatzverarbeiters](#)
- [Ändern der Konfigurationseigenschaften](#)

Implementieren des Datensatzverarbeiters

Der einfachste Konsument, der die KCL für Node.js verwenden kann, muss die Funktion `recordProcessor` implementieren. Diese enthält wiederum die Funktionen `initialize`, `processRecords`, und `shutdown`. Das Beispiel zeigt eine Implementierung, die Sie als Ausgangspunkt verwenden können (siehe `sample_kcl_app.js`).

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

`initialize`

Die KCL ruft die Funktion `initialize` auf, wenn der Datensatzverarbeiter gestartet wird. Dieser Datensatzverarbeiter verarbeitet nur die Shard-ID, die als `initializeInput.shardId` übergeben wird. In der Regel ist dies auch umgekehrt der Fall (diese Shard wird nur durch diesen Datensatzverarbeiter verarbeitet). Ihr Konsument sollte jedoch die Möglichkeit berücksichtigen, dass ein Datensatz mehr als einmal verarbeitet werden könnte. Das liegt daran, dass Kinesis Data Streams eine Semantik nach dem Grundsatz mindestens einmal hat. Das bedeutet, dass jeder Datensatz aus einer Shard mindestens einmal von einem Worker in Ihrem Konsumenten verarbeitet wird. Weitere Informationen zu Fällen, in denen eine bestimmte Shard möglicherweise durch mehr als einen Auftragnehmer verarbeitet wird, finden Sie unter [Resharding, Skalierung und Parallelverarbeitung](#).

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

### `processRecords`

Die KCL ruft diese Funktion mit einer Eingabe auf, die eine Liste von Datensätzen aus der für die Funktion `initialize` angegebenen Shard enthält. Der von Ihnen implementierte Datensatzverarbeiter verarbeitet die Daten in diesen Datensätzen entsprechend der Semantik Ihres Konsumenten. Beispielsweise kann der Auftragnehmer eine Transformation für die Daten ausführen und das Ergebnis dann in einem Amazon Simple Storage Service (Amazon S3)-Bucket speichern.

```
processRecords: function(processRecordsInput, completeCallback)
```

Zusätzlich zu den Daten enthält der Datensatz auch eine Sequenznummer und einen Partitionsschlüssel, die der Auftragnehmer beim Verarbeiten der Daten verwenden kann. Beispielsweise könnte der Auftragnehmer basierend auf dem Wert des Partitionsschlüssels den S3-Bucket wählen, in dem die Daten gespeichert werden sollen. Das `record`-Anmeldeverzeichnis stellt die folgenden Schlüssel-Wert-Paare für den Zugriff auf die Daten des Datensatzes, die Sequenznummer und den Partitionsschlüssel bereit:

```
record.data  
record.sequenceNumber  
record.partitionKey
```

Beachten Sie, dass die Daten Base64-kodiert sind.

Im einfachen Beispiel weist die Funktion `processRecords` Code auf, der zeigt, wie ein Auftragnehmer auf die Daten des Datensatzes, die Sequenznummer und den Partitionsschlüssel zugreifen kann.

Kinesis Data Streams erfordert, dass der Datensatzverarbeiter die Datensätze nachverfolgt, die bereits in einer Shard verarbeitet wurden. Die KCL übernimmt die Nachverfolgung durch ein `checkpointer`-Objekt, das als `processRecordsInput.checkpointer` übergeben wird. Der Datensatzverarbeiter ruft die Funktion `checkpointer.checkpoint` auf, um die KCL über die Fortschritte zu informieren, die er beim Verarbeiten der Datensätze in der Shard gemacht hat. Wenn der Auftragnehmer fehlschlägt, verwendet die KCL diese Informationen, wenn Sie die Verarbeitung der Shard erneut starten, damit sie den Vorgang ab dem letzten bekannten Datensatz fortsetzt.

Im Fall einer Teilungs- oder Zusammenführungsoperation beginnt die KCL erst dann mit der Verarbeitung der neuen Shards, wenn die Verarbeiter für die ursprünglichen Shards `checkpoint` aufgerufen haben, um zu signalisieren, dass die Verarbeitung der ursprünglichen Shards vollständig abgeschlossen ist.

Wenn Sie die Sequenznummer nicht an die `checkpoint`-Funktion übergeben, nimmt die KCL an, dass der Aufruf von `checkpoint` bedeutet, dass alle Datensätze bis zum letzten Datensatz, der an den Datensatzverarbeiter übergeben wurde, verarbeitet wurden. Daher sollte der Datensatzverarbeiter die Methode `checkpoint` erst aufrufen, wenn er alle Datensätze in der Liste, die ihm übergeben wurden, verarbeitet hat. Datensatzverarbeiter müssen `checkpoint` nicht bei jedem Aufruf von `processRecords` aufrufen. Ein Verarbeiter könnte beispielsweise `checkpoint` bei jedem dritten Aufruf oder beim Eintritt eines Ereignisses außerhalb Ihres Datensatzverarbeiters aufrufen, beispielsweise eines von Ihnen implementierten Verifizierung-/Validierungs-Service.

Sie können optional die exakte Sequenznummer eines Datensatzes als Parameter für `checkpoint` angeben. In diesem Fall nimmt die KCL an, dass alle Datensätze nur bis zu diesem Datensatz verarbeitet wurden.

Die einfache Beispielanwendung zeigt den einfachsten möglichen Aufruf der Funktion `checkpointer.checkpoint`. Sie können weitere Checkpoint-Logik hinzufügen, die Sie an diesem Punkt in der Funktion für Ihren Konsumenten benötigen.

## shutdown

Die KCL ruft die Funktion `shutdown` entweder auf, wenn die Verarbeitung beendet wird (`shutdownInput.reason` ist `TERMINATE`) oder wenn der Auftragnehmer nicht mehr reagiert (`shutdownInput.reason` ist `ZOMBIE`).

```
shutdown: function(shutdownInput, completeCallback)
```

Die Verarbeitung endet, wenn der Datensatzverarbeiter keine weiteren Datensätze aus der Shard erhält, entweder weil die Shard geteilt oder zusammengeführt wurde oder weil der Stream gelöscht wurde.

Die KCL übergibt auch ein `shutdownInput.checkpointer`-Objekt an `shutdown`. Wenn der Grund für das Herunterfahren `TERMINATE` ist, sollten Sie sicherstellen, dass der Datensatzverarbeiter die Verarbeitung aller Datensätze fertiggestellt hat, und dann die Funktion `checkpoint` in seiner Schnittstelle aufrufen.

## Ändern der Konfigurationseigenschaften

Das Beispiel zeigt Standardwerte für die Konfigurationseigenschaften. Sie können diese Eigenschaften mit eigenen Werten überschreiben (siehe `sample.properties` im einfachen Beispiel).

### Anwendungsname

Die KCL erfordert eine Anwendung, die unter Ihren Anwendungen sowie den Amazon-DynamoDB-Tabellen in derselben Region eindeutig ist. Sie verwendet den Wert der Anwendungsnamenkonfiguration auf folgende Arten:

- Für mit diesem Anwendungsnamen verknüpfte Auftragnehmer wird angenommen, dass sie gemeinsam im gleichen Stream arbeiten. Diese Auftragnehmer können auf mehrere Instances verteilt sein. Wenn Sie eine zusätzliche Instance desselben Anwendungscode ausführen, jedoch mit einem anderen Anwendungsnamen, behandelt die KCL die zweite Instance als eine völlig getrennte Anwendung, die ebenfalls im selben Stream arbeitet.
- Die KCL erstellt eine DynamoDB-Tabelle mit dem Namen der Anwendung und verwendet die Tabelle für die Verwaltung von Statusinformationen für die Anwendung (wie Checkpoints und Auftragnehmer-Shard-Zuweisungen). Jede Anwendung verfügt über eine eigene DynamoDB-Tabelle. Weitere Informationen finden Sie unter [Mithilfe einer Leasetabelle können Sie die von der KCL-Konsumentenanzahl verarbeiteten Shards verfolgen](#).

## Einrichten von Anmeldeinformationen

Sie müssen Ihre AWS-Anmeldeinformationen für einen der Anmeldeinformationsanbieter in der Anmeldeinformationsanbieter-Standardkette bereitstellen. Sie können die Eigenschaft `AWSCredentialsProvider` verwenden, um einen Anmeldeinformationsanbieter einzurichten. Die `sample.properties`-Datei muss Anmeldeinformationen einem der Anmeldeinformationsanbieter in der [Anmeldeinformationsanbieter-Standardkette](#) bereitstellen. Wenn Sie Ihren Konsumenten auf einer

Amazon-EC2-Instance ausführen, empfehlen wir, die Instance mit einer IAM-Rolle zu konfigurieren. AWS-Anmeldeinformationen, die die mit dieser IAM-Rolle verknüpften Berechtigungen widerspiegeln, werden den Anwendungen auf der Instance über deren Instance-Metadaten zur Verfügung gestellt. Dies ist die sicherste Art, Anmeldeinformationen für eine Konsumenten-anwendung zu verwalten, die auf einer EC2-Instance ausgeführt wird.

Im folgenden Beispiel wird eine KCL konfiguriert, um einen Kinesis-Datenstrom namens `kc1nodejssample` mittels des Datensatzverarbeiters zu verarbeiten, der in `sample_kc1_app.js` bereitgestellt wird:

```
# The Node.js executable script
executableName = node sample_kc1_app.js
# The name of an Amazon Kinesis stream to process
streamName = kc1nodejssample
# Unique KCL application name
applicationName = kc1nodejssample
# Use default AWS credentials provider chain
AWSCredentialsProvider = DefaultAWSCredentialsProviderChain
# Read from the beginning of the stream
initialPositionInStream = TRIM_HORIZON
```

## Entwickeln eines Kinesis Client Library-Verbrauchers in .NET

Sie können die Kinesis Client Library (KCL) verwenden, um Anwendungen zu erstellen, die Daten aus Ihren Kinesis-Datenströmen verarbeiten. Die Kinesis Client Library ist in mehreren Sprachen verfügbar. In diesem Thema wird .NET behandelt.

Die KCL ist eine Java-Bibliothek. Unterstützung für andere Sprachen als Java wird über eine mehrsprachige Schnittstelle namens `bereitgestelltMultiLangDaemon`. Dieser Daemon basiert auf Java und wird im Hintergrund ausgeführt, wenn Sie eine andere KCL-Sprache als Java verwenden. Wenn Sie also die KCL für .NET installieren und Ihre Konsumenten-App vollständig in .NET schreiben, benötigen Sie aufgrund der trotzdem Java auf Ihrem System `MultiLangDaemon`. Darüber hinaus `MultiLangDaemon` verfügt über einige Standardeinstellungen, die Sie möglicherweise an Ihren Anwendungsfall anpassen müssen, z. B. die AWS Region, mit der es eine Verbindung herstellt. Weitere Informationen über die `MultiLangDaemon` auf finden Sie auf der [KCL-MultiLangDaemon Projektseite](#).

Um die .NET KCL von herunterzuladen GitHub, gehen Sie zur [Kinesis Client Library \(.NET\)](#). Um Beispielcode für eine .NET-KCL-Konsumenten-anwendung herunterzuladen, gehen Sie auf die Seite [KCL für .NET-Beispielkonsumentenprojekt](#) auf GitHub.

Sie müssen die folgenden Aufgaben durchführen, wenn Sie eine KCL-Konsumentenanzwendung in .NET implementieren:

## Aufgaben

- [Implementieren der I-RecordProcessor Klassenmethoden](#)
- [Ändern der Konfigurationseigenschaften](#)

## Implementieren der I-RecordProcessor Klassenmethoden

Der Konsument muss die folgenden Methoden für `IRecordProcessor` implementieren. Der Konsument im Beispiel stellt Implementierungen bereit, die Sie als Ausgangspunkt verwenden können (siehe die `SampleRecordProcessor`-Klasse in `SampleConsumer/AmazonKinesisSampleConsumer.cs`).

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

## Initialisieren

Die KCL ruft diese Methode auf, wenn der Datensatzverarbeiter instanziiert wird, und übergibt eine spezifische Shard-ID an den `input`-Parameter (`input.ShardId`). Dieser Datensatzverarbeiter verarbeitet nur diese Shard und in der Regel ist dies auch umgekehrt der Fall (diese Shard wird nur durch diesen Datensatzverarbeiter verarbeitet). Ihr Konsument sollte jedoch die Möglichkeit berücksichtigen, dass ein Datensatz mehr als einmal verarbeitet werden könnte. Das liegt daran, dass Kinesis Data Streams eine Semantik nach dem Grundsatz mindestens einmal hat. Das bedeutet, dass jeder Datensatz aus einer Shard mindestens einmal von einem Worker in Ihrem Konsumenten verarbeitet wird. Weitere Informationen zu Fällen, in denen eine bestimmte Shard möglicherweise durch mehr als einen Auftragnehmer verarbeitet wird, finden Sie unter [Resharding, Skalierung und Parallelverarbeitung](#).

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

## ProcessRecords

Die KCL ruft diese Methode auf und übergibt eine Liste der Datensätze an den `input`-Parameter (`input.Records`) aus der Shard, die von der Methode `Initialize` angegeben wird. Der von Ihnen implementierte Datensatzverarbeiter verarbeitet die Daten in diesen Datensätzen entsprechend

der Semantik Ihres Konsumenten. Beispielsweise kann der Auftragnehmer eine Transformation für die Daten ausführen und das Ergebnis dann in einem Amazon Simple Storage Service (Amazon S3)-Bucket speichern.

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

Zusätzlich zu den Daten selbst enthält der Datensatz auch eine Sequenznummer und einen Partitionsschlüssel. Der Auftragnehmer kann diese Werte beim Verarbeiten der Daten verwenden. Beispielsweise könnte der Auftragnehmer basierend auf dem Wert des Partitionsschlüssels den S3-Bucket wählen, in dem die Daten gespeichert werden sollen. Die Klasse `Record` stellt die folgenden Methoden bereit, die Zugriff auf die Daten des Datensatzes, die Sequenznummer und den Partitionsschlüssel bieten:

```
byte[] Record.Data  
string Record.SequenceNumber  
string Record.PartitionKey
```

Im Beispiel weist die Methode `ProcessRecordsWithRetries` Code auf, der zeigt, wie ein Auftragnehmer auf die Daten des Datensatzes, die Sequenznummer und den Partitionsschlüssel zugreifen kann.

Kinesis Data Streams erfordert, dass der Datensatzverarbeiter die Datensätze nachverfolgt, die bereits in einer Shard verarbeitet wurden. Die KCL übernimmt diese Nachverfolgung für Sie, indem ein `Checkpointter`-Objekt an `ProcessRecords` (`input.Checkpointer`) übergeben wird. Der Datensatzverarbeiter ruft die Methode `Checkpointter.Checkpoint` auf, um die KCL über die Fortschritte zu informieren, die sie beim Verarbeiten der Datensätze in der Shard gemacht hat. Wenn der Auftragnehmer fehlschlägt, verwendet die KCL diese Informationen, um die Verarbeitung der Shard mit dem letzten bekannten Datensatz neu zu starten.

Im Fall einer Teilungs- oder Zusammenführungsoperation beginnt die KCL erst dann mit der Verarbeitung der neuen Shards, wenn die Verarbeiter für die ursprünglichen Shards `Checkpointter.Checkpoint` aufgerufen haben, um zu signalisieren, dass die Verarbeitung der ursprünglichen Shards vollständig abgeschlossen ist.

Wenn Sie keinen Parameter übergeben, nimmt die KCL an, dass der Aufruf von `Checkpointter.Checkpoint` bedeutet, dass alle Datensätze bis zum letzten Datensatz, der an den Datensatzverarbeiter übergeben wurde, verarbeitet wurden. Daher sollte der Datensatzverarbeiter die Methode `Checkpointter.Checkpoint` erst aufrufen, wenn er alle Datensätze in der Liste, die



ihm übergeben wurden, verarbeitet hat. Datensatzverarbeiter müssen `Checkpointter.Checkpoint` nicht bei jedem Aufruf von `ProcessRecords` aufrufen. Ein Prozessor könnte beispielsweise `Checkpointter.Checkpoint` bei jedem dritten oder vierten Aufruf aufrufen. Sie können optional die exakte Sequenznummer eines Datensatzes als Parameter für `Checkpointter.Checkpoint` angeben. In diesem Fall nimmt die KCL an, dass die Datensätze nur bis zu diesem Datensatz verarbeitet wurden.

Im Beispiel zeigt die private Methode `Checkpoint(Checkpointter checkpointter)`, wie die `Checkpointter.Checkpoint`-Methode mithilfe der entsprechenden Ausnahmebehandlung und Wiederholungslogik aufgerufen wird.

Die KCL für .NET verarbeitet Ausnahmen anders als andere KCL-Sprachbibliotheken, da sie keine Ausnahmen verarbeitet, die aus der Verarbeitung der Datensätze entstanden sind. Alle nicht abgefangenen Ausnahmen vom Benutzer-Code bringen das Programm zum Absturz.

## Herunterfahren

Die KPL ruft die Methode `Shutdown` entweder auf, wenn die Verarbeitung beendet wird (Grund für das Herunterfahren ist `TERMINATE`) oder wenn der Auftragnehmer nicht mehr reagiert (der `input.Reason`-Wert für das Herunterfahren ist `ZOMBIE`).

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

Die Verarbeitung endet, wenn der Datensatzverarbeiter keine weiteren Datensätze aus der Shard erhält, weil die Shard geteilt oder zusammengeführt wurde oder der Stream gelöscht wurde.

Die KCL übergibt auch ein `Checkpointter`-Objekt an `shutdown`. Wenn der Grund für das Herunterfahren `TERMINATE` ist, sollte der Datensatzverarbeiter alle Datensätze fertigstellen und dann die Methode `checkpoint` in seiner Schnittstelle aufrufen.

## Ändern der Konfigurationseigenschaften

Der Beispielkonsument zeigt Standardwerte für die Konfigurationseigenschaften. Sie können diese Eigenschaften mit eigenen Werten überschreiben (siehe `SampleConsumer/kcl.properties`).

## Anwendungsname

Die KCL erfordert eine Anwendung, die unter Ihren Anwendungen sowie den Amazon-DynamoDB-Tabellen in derselben Region eindeutig ist. Sie verwendet den Wert der Anwendungsnamenkonfiguration auf folgende Arten:

- Für mit diesem Anwendungsnamen verknüpfte Auftragnehmer wird angenommen, dass sie gemeinsam im gleichen Stream arbeiten. Diese Auftragnehmer können auf mehrere Instances verteilt sein. Wenn Sie eine zusätzliche Instance desselben Anwendungscode ausführen, jedoch mit einem anderen Anwendungsnamen, behandelt die KCL die zweite Instance als eine völlig getrennte Anwendung, die ebenfalls im selben Stream arbeitet.
- Die KCL erstellt eine DynamoDB-Tabelle mit dem Namen der Anwendung und verwendet die Tabelle für die Verwaltung von Statusinformationen für die Anwendung (wie Checkpoints und Auftragnehmer-Shard-Zuweisungen). Jede Anwendung verfügt über eine eigene DynamoDB-Tabelle. Weitere Informationen finden Sie unter [Mithilfe einer Leasetabelle können Sie die von der KCL-Konsumentenanzahl verarbeiteten Shards verfolgen](#).

## Einrichten von Anmeldeinformationen

Sie müssen Ihre AWS-Anmeldeinformationen für einen der Anmeldeinformationsanbieter in der Anmeldeinformationsanbieter-Standardkette bereitstellen. Sie können die Eigenschaft `AWSCredentialsProvider` verwenden, um einen Anmeldeinformationsanbieter einzurichten. Die [sample.properties](#) muss Ihre Anmeldeinformationen einem der Anmeldeinformationsanbieter in der [Anmeldeinformationsanbieter-Standardkette](#) bereitstellen. Wenn Sie Ihre Konsumentenanzahl auf einer EC2-Instance ausführen, empfehlen wir, die Instance mit einer IAM-Rolle zu konfigurieren. AWS-Anmeldeinformationen, die die mit dieser IAM-Rolle verknüpften Berechtigungen widerspiegeln, werden den Anwendungen auf der Instance über deren Instance-Metadaten zur Verfügung gestellt. Dies ist die sicherste Art, Anmeldeinformationen für einen Konsumenten zu verwalten, der auf einer EC2-Instance ausgeführt wird.

Die Eigenschaftendatei des Beispiels konfiguriert KCL, um einen Kinesis-Datenstrom namens „words“ mittels des Datensatzverarbeiters zu verarbeiten, der in `AmazonKinesisSampleConsumer.cs` bereitgestellt wird.

## Entwickeln eines Kinesis Client Library-Verbrauchers in Python

Sie können die Kinesis Client Library (KCL) verwenden, um Anwendungen zu erstellen, die Daten aus Ihren Kinesis-Datenströmen verarbeiten. Die Kinesis Client Library ist in mehreren Sprachen verfügbar. In diesem Thema wird Python behandelt.

Die KCL ist eine Java-Bibliothek. Unterstützung für andere Sprachen als Java wird über eine mehrsprachige Schnittstelle namens `bereitgestelltMultiLangDaemon`. Dieser Daemon basiert auf Java und wird im Hintergrund ausgeführt, wenn Sie eine andere KCL-Sprache als Java verwenden. Wenn Sie also die KCL für Python installieren und Ihre Konsumenten-App vollständig in Python

schreiben, benötigen Sie aufgrund der trotzdem Java auf Ihrem System MultiLangDaemon. Darüber hinaus MultiLangDaemon verfügt über einige Standardeinstellungen, die Sie möglicherweise an Ihren Anwendungsfall anpassen müssen, z. B. die AWS Region, mit der es eine Verbindung herstellt. Weitere Informationen über die MultiLangDaemon auf finden GitHubSie auf der Seite [KCL-MultiLangDaemon Projekt](#).

Um die Python-KCL von herunterzuladen GitHub, gehen Sie zur [Kinesis Client Library \(Python\)](#). Um Beispielcode für eine Python-KCL-Konsumenten-anwendung herunterzuladen, gehen Sie zur Seite [KCL für Python-Beispielprojekt](#) auf GitHub.

Sie müssen die folgenden Aufgaben durchführen, wenn Sie eine KCL-Konsumenten-anwendung in Python implementieren:

### Aufgaben

- [Implementieren der RecordProcessor Klassenmethoden](#)
- [Ändern der Konfigurationseigenschaften](#)

### Implementieren der RecordProcessor Klassenmethoden

Die RecordProcess-Klasse muss die RecordProcessorBase erweitern, um die folgenden Methoden zu implementieren. Das Beispiel stellt Implementierungen bereit, die Sie als Ausgangspunkt verwenden können (siehe `sample_kclpy_app.py`).

```
def initialize(self, shard_id)
def process_records(self, records, checkpoint)
def shutdown(self, checkpoint, reason)
```

### initialize

Die KCL ruft die Methode `initialize` auf, wenn der Datensatzverarbeiter instanziiert wird, und übergibt eine spezifische Shard-ID als Parameter. Dieser Datensatzverarbeiter verarbeitet nur diese Shard und in der Regel ist dies auch umgekehrt der Fall (diese Shard wird nur durch diesen Datensatzverarbeiter verarbeitet). Ihr Konsument sollte jedoch die Möglichkeit berücksichtigen, dass ein Datensatz mehr als einmal verarbeitet werden könnte. Das liegt daran, dass Kinesis Data Streams eine Semantik nach dem Grundsatz mindestens einmal hat. Das bedeutet, dass jeder Datensatz aus einer Shard mindestens einmal von einem Worker in Ihrem Konsumenten verarbeitet wird. Weitere Informationen zu Fällen, in denen eine bestimmte Shard möglicherweise durch mehr als einen Auftragnehmer verarbeitet wird, finden Sie unter [Resharding, Skalierung und Parallelverarbeitung](#).

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

## process\_records

Die KCL ruft diese Methode auf und übergibt eine Liste der Datensätze aus der Shard, die von der Methode `initialize` angegeben wird. Der von Ihnen implementierte Datensatzverarbeiter verarbeitet die Daten in diesen Datensätzen entsprechend der Semantik Ihres Konsumenten. Beispielsweise kann der Auftragnehmer eine Transformation für die Daten ausführen und das Ergebnis dann in einem Amazon Simple Storage Service (Amazon S3)-Bucket speichern.

```
def process_records(self, records, checkpoint)
```

Zusätzlich zu den Daten selbst enthält der Datensatz auch eine Sequenznummer und einen Partitionsschlüssel. Der Auftragnehmer kann diese Werte beim Verarbeiten der Daten verwenden. Beispielsweise könnte der Auftragnehmer basierend auf dem Wert des Partitionsschlüssels den S3-Bucket wählen, in dem die Daten gespeichert werden sollen. Das `record`-Anmeldeverzeichnis stellt die folgenden Schlüssel-Wert-Paare für den Zugriff auf die Daten des Datensatzes, die Sequenznummer und den Partitionsschlüssel bereit:

```
record.get('data')  
record.get('sequenceNumber')  
record.get('partitionKey')
```

Beachten Sie, dass die Daten Base64-kodiert sind.

Im Beispiel weist die Methode `process_records` Code auf, der zeigt, wie ein Auftragnehmer auf die Daten des Datensatzes, die Sequenznummer und den Partitionsschlüssel zugreifen kann.

Kinesis Data Streams erfordert, dass der Datensatzverarbeiter die Datensätze nachverfolgt, die bereits in einer Shard verarbeitet wurden. Die KCL übernimmt diese Nachverfolgung für Sie, indem ein `Checkpoint`-Objekt an `process_records` übergeben wird. Der Datensatzverarbeiter ruft die Methode `checkpoint` auf diesem Objekt auf, um die KCL über die Fortschritte zu informieren, die er beim Verarbeiten der Datensätze in der Shard gemacht hat. Wenn der Auftragnehmer fehlschlägt, verwendet die KCL diese Informationen, um die Verarbeitung der Shard mit dem letzten bekannten Datensatz neu zu starten.

Im Fall einer Teilungs- oder Zusammenführungsoperation beginnt die KCL erst dann mit der Verarbeitung der neuen Shards, wenn die Verarbeiter für die ursprünglichen Shards `checkpoint`

aufgerufen haben, um zu signalisieren, dass die Verarbeitung der ursprünglichen Shards vollständig abgeschlossen ist.

Wenn Sie keinen Parameter übergeben, nimmt die KCL an, dass der Aufruf von `checkpoint` bedeutet, dass alle Datensätze bis zum letzten Datensatz, der an den Datensatzverarbeiter übergeben wurde, verarbeitet wurden. Daher sollte der Datensatzverarbeiter die Methode `checkpoint` erst aufrufen, wenn er alle Datensätze in der Liste, die ihm übergeben wurden, verarbeitet hat. Datensatzverarbeiter müssen `checkpoint` nicht bei jedem Aufruf von `process_records` aufrufen. Ein Prozessor könnte beispielsweise `checkpoint` bei jedem dritten Aufruf aufrufen. Sie können optional die exakte Sequenznummer eines Datensatzes als Parameter für `checkpoint` angeben. In diesem Fall nimmt die KCL an, dass alle Datensätze nur bis zu diesem Datensatz verarbeitet wurden.

Im Beispiel zeigt die private Methode `checkpoint`, wie die `Checkpointers.checkpoint`-Methode mithilfe der entsprechenden Ausnahmebehandlung und Wiederholungslogik aufgerufen wird.

Die KCL ist bei der Behandlung von Ausnahmen, die während der Verarbeitung der Datensätze auftreten, von `process_records` abhängig. Wenn `process_records` eine Ausnahme auslöst, überspringt die KCL die Datensätze, die vor der Ausnahme an `process_records` übergeben wurden. Das heißt, diese Datensätze werden nicht erneut an den Datensatzprozessor gesendet, der die Ausnahme ausgelöst hat, oder an einen anderen Datensatzprozessor im Verbraucher.

## shutdown

Die KCL ruft die Methode `shutdown` entweder auf, wenn die Verarbeitung beendet wird (Grund für das Herunterfahren ist `TERMINATE`) oder wenn der Auftragnehmer nicht mehr reagiert (das Herunterfahren `reason` ist `ZOMBIE`).

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

Die Verarbeitung endet, wenn der Datensatzverarbeiter keine weiteren Datensätze aus der Shard erhält, entweder weil die Shard geteilt oder zusammengeführt wurde oder weil der Stream gelöscht wurde.

Die KCL übergibt auch ein `Checkpointers`-Objekt an `shutdown`. Wenn der `reason` für das Herunterfahren `TERMINATE` ist, sollte der Datensatzverarbeiter alle Datensätze fertigstellen und dann die Methode `checkpoint` in seiner Schnittstelle aufrufen.

## Ändern der Konfigurationseigenschaften

Das Beispiel zeigt Standardwerte für die Konfigurationseigenschaften. Sie können diese Eigenschaften mit eigenen Werten überschreiben (siehe `sample.properties`).

### Anwendungsname

Die KCL erfordert einen Anwendungsnamen, der unter Ihren Anwendungen sowie den Amazon-DynamoDB-Tabellen in derselben Region eindeutig ist. Sie verwendet den Wert der Anwendungsnamenkonfiguration auf folgende Arten:

- Für mit diesem Anwendungsnamen verknüpfte Auftragnehmer wird angenommen, dass sie gemeinsam im gleichen Stream arbeiten. Diese Auftragnehmer können auf mehrere Instances verteilt sein. Wenn Sie eine zusätzliche Instance desselben Anwendungscode ausführen, jedoch mit einem anderen Anwendungsnamen, behandelt die KCL die zweite Instance als eine völlig getrennte Anwendung, die ebenfalls im selben Stream arbeitet.
- Die KCL erstellt eine DynamoDB-Tabelle mit dem Namen der Anwendung und verwendet die Tabelle für die Verwaltung von Statusinformationen für die Anwendung (wie Checkpoints und Auftragnehmer-Shard-Zuweisungen). Jede Anwendung verfügt über eine eigene DynamoDB-Tabelle. Weitere Informationen finden Sie unter [Mithilfe einer Leasetabelle können Sie die von der KCL-Konsumentenanzwendung verarbeiteten Shards verfolgen](#).

### Einrichten von Anmeldeinformationen

Sie müssen Ihre AWS-Anmeldeinformationen für einen der Anmeldeinformationsanbieter in der Anmeldeinformationsanbieter-Standardkette bereitstellen. Sie können die Eigenschaft `AWSCredentialsProvider` verwenden, um einen Anmeldeinformationsanbieter einzurichten. Die [sample.properties](#) muss Ihre Anmeldeinformationen einem der Anmeldeinformationsanbieter in der [Anmeldeinformationsanbieter-Standardkette](#) bereitstellen. Wenn Sie Ihre Konsumentenanzwendung auf einer Amazon-EC2-Instance ausführen, empfehlen wir, die Instance mit einer IAM-Rolle zu konfigurieren. AWS-Anmeldeinformationen, die die mit dieser IAM-Rolle verknüpften Berechtigungen widerspiegeln, werden den Anwendungen auf der Instance über deren Instance-Metadaten zur Verfügung gestellt. Dies ist die sicherste Art, Anmeldeinformationen für eine Konsumentenanzwendung zu verwalten, die auf einer EC2-Instance ausgeführt wird.

Die Eigenschaftendatei des Beispiels konfiguriert KCL, um einen Kinesis-Datenstrom namens „words“ mittels des Datensatzverarbeiters zu verarbeiten, der in `sample_kc1py_app.py` bereitgestellt wird.

## Entwickeln eines Kinesis Client Library-Verbrauchers in Ruby

Sie können die Kinesis Client Library (KCL) verwenden, um Anwendungen zu erstellen, die Daten aus Ihren Kinesis-Datenströmen verarbeiten. Die Kinesis Client Library ist in mehreren Sprachen verfügbar. In diesem Thema wird Ruby behandelt.

Die KCL ist eine Java-Bibliothek. Unterstützung für andere Sprachen als Java wird über eine mehrsprachige Schnittstelle namens bereitgestellt `MultiLangDaemon`. Dieser Daemon basiert auf Java und wird im Hintergrund ausgeführt, wenn Sie eine andere KCL-Sprache als Java verwenden. Wenn Sie also die KCL für Ruby installieren und Ihre Konsumenten-App vollständig in Ruby schreiben, benötigen Sie aufgrund der trotzdem Java auf Ihrem System `MultiLangDaemon`. Darüber hinaus `MultiLangDaemon` verfügt über einige Standardeinstellungen, die Sie möglicherweise an Ihren Anwendungsfall anpassen müssen, z. B. die AWS Region, mit der es eine Verbindung herstellt. Weitere Informationen über die `MultiLangDaemon` auf finden GitHub Sie auf der [KCL-MultiLangDaemon Projektseite](#).

Um die Ruby-KCL von herunterzuladen GitHub, gehen Sie zur [Kinesis Client Library \(Ruby\)](#). Um Beispielcode für eine Ruby-KCL-Konsumenten-anwendung herunterzuladen, gehen Sie zur Seite für ein [Beispielprojekt von KCL für Ruby](#) unter GitHub.

Weitere Informationen über die KCL-Bibliothek für die Unterstützung von Ruby finden Sie unter [Dokumentation zu KCL Ruby Gems](#).

## Entwicklung von KCL 2.x-Verbrauchern

In diesem Thema erhalten Sie Informationen zur Nutzung der Version 2.0 der Kinesis Client Library (KCL). Weitere Informationen über die KCL finden Sie in der Übersicht in [Entwickeln von Verbrauchern mit der Kinesis Client Library 1.x](#).

### Inhalt

- [Entwickeln eines Kinesis Client Library-Verbrauchers in Java](#)
- [Entwickeln eines Kinesis Client Library-Verbrauchers in Python](#)

## Entwickeln eines Kinesis Client Library-Verbrauchers in Java

Der folgende Code zeigt eine Beispielimplementierung in Java für `ProcessorFactory` und `RecordProcessor`. Weitere Informationen zur Nutzung der Vorteile der erweiterten Rundsendefunktion finden Sie unter [Verwenden von Verbrauchern mit erweitertem Rundsenden](#).

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 * Licensed under the Amazon Software License (the "License").
 * You may not use this file except in compliance with the License.
 * A copy of the License is located at
 *
 * http://aws.amazon.com/asl/
 *
 * or in the "license" file accompanying this file. This file is distributed
 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
 * express or implied. See the License for the specific language governing
 * permissions and limitations under the License.
 */
```

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 * Licensed under the Apache License, Version 2.0 (the "License").
 * You may not use this file except in compliance with the License.
 * A copy of the License is located at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * or in the "license" file accompanying this file. This file is distributed
 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
 * express or implied. See the License for the specific language governing
 * permissions and limitations under the License.
 */
```

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.UUID;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.ScheduledFuture;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;
```



```
import org.apache.commons.lang3.ObjectUtils;
import org.apache.commons.lang3.RandomStringUtils;
import org.apache.commons.lang3.RandomUtils;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.slf4j.MDC;

import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cloudwatch.CloudWatchAsyncClient;
import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;
import software.amazon.awssdk.services.kinesis.KinesisAsyncClient;
import software.amazon.awssdk.services.kinesis.model.PutRecordRequest;
import software.amazon.kinesis.common.ConfigsBuilder;
import software.amazon.kinesis.common.KinesisClientUtil;
import software.amazon.kinesis.coordinator.Scheduler;
import software.amazon.kinesis.exceptions.InvalidStateException;
import software.amazon.kinesis.exceptions.ShutdownException;
import software.amazon.kinesis.lifecycle.events.InitializationInput;
import software.amazon.kinesis.lifecycle.events.LeaseLostInput;
import software.amazon.kinesis.lifecycle.events.ProcessRecordsInput;
import software.amazon.kinesis.lifecycle.events.ShardEndedInput;
import software.amazon.kinesis.lifecycle.events.ShutdownRequestedInput;

import software.amazon.kinesis.processor.ShardRecordProcessor;
import software.amazon.kinesis.processor.ShardRecordProcessorFactory;
import software.amazon.kinesis.retrieval.polling.PollingConfig;

/**
 * This class will run a simple app that uses the KCL to read data and uses the AWS SDK
 * to publish data.
 * Before running this program you must first create a Kinesis stream through the AWS
 * console or AWS SDK.
 */
public class SampleSingle {

    private static final Logger log = LoggerFactory.getLogger(SampleSingle.class);

    /**
     * Invoke the main method with 2 args: the stream name and (optionally) the region.
     * Verifies valid inputs and then starts running the app.
     */
    public static void main(String... args) {
```

```
        if (args.length < 1) {
            log.error("At a minimum, the stream name is required as the first argument.
The Region may be specified as the second argument.");
            System.exit(1);
        }

        String streamName = args[0];
        String region = null;
        if (args.length > 1) {
            region = args[1];
        }

        new SampleSingle(streamName, region).run();
    }

    private final String streamName;
    private final Region region;
    private final KinesisAsyncClient kinesisClient;

    /**
     * Constructor sets streamName and region. It also creates a KinesisClient object
to send data to Kinesis.
     * This KinesisClient is used to send dummy data so that the consumer has something
to read; it is also used
     * indirectly by the KCL to handle the consumption of the data.
     */
    private SampleSingle(String streamName, String region) {
        this.streamName = streamName;
        this.region = Region.of(ObjectUtils.firstNonNull(region, "us-east-2"));
        this.kinesisClient =
KinesisClientUtil.createKinesisAsyncClient(KinesisAsyncClient.builder().region(this.region));
    }

    private void run() {

        /**
         * Sends dummy data to Kinesis. Not relevant to consuming the data with the KCL
         */
        ScheduledExecutorService producerExecutor =
Executors.newSingleThreadScheduledExecutor();
        ScheduledFuture<?> producerFuture =
producerExecutor.scheduleAtFixedRate(this::publishRecord, 10, 1, TimeUnit.SECONDS);

        /**
```

```
    * Sets up configuration for the KCL, including DynamoDB and CloudWatch
dependencies. The final argument, a
    * ShardRecordProcessorFactory, is where the logic for record processing lives,
and is located in a private
    * class below.
    */
    DynamoDbAsyncClient dynamoClient =
DynamoDbAsyncClient.builder().region(region).build();
    CloudWatchAsyncClient cloudWatchClient =
CloudWatchAsyncClient.builder().region(region).build();
    ConfigsBuilder configsBuilder = new ConfigsBuilder(streamName, streamName,
kinesisClient, dynamoClient, cloudWatchClient, UUID.randomUUID().toString(), new
SampleRecordProcessorFactory());

    /**
    * The Scheduler (also called Worker in earlier versions of the KCL) is the
entry point to the KCL. This
    * instance is configured with defaults provided by the ConfigsBuilder.
    */
    Scheduler scheduler = new Scheduler(
        configsBuilder.checkpointConfig(),
        configsBuilder.coordinatorConfig(),
        configsBuilder.leaseManagementConfig(),
        configsBuilder.lifecycleConfig(),
        configsBuilder.metricsConfig(),
        configsBuilder.processorConfig(),
        configsBuilder.retrievalConfig().retrievalSpecificConfig(new
PollingConfig(streamName, kinesisClient))
    );

    /**
    * Kickoff the Scheduler. Record processing of the stream of dummy data will
continue indefinitely
    * until an exit is triggered.
    */
    Thread schedulerThread = new Thread(scheduler);
    schedulerThread.setDaemon(true);
    schedulerThread.start();

    /**
    * Allows termination of app by pressing Enter.
    */
    System.out.println("Press enter to shutdown");
    BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
```

```
        try {
            reader.readLine();
        } catch (IOException ioex) {
            log.error("Caught exception while waiting for confirm. Shutting down.",
ioex);
        }

        /**
         * Stops sending dummy data.
         */
        log.info("Cancelling producer and shutting down executor.");
        producerFuture.cancel(true);
        producerExecutor.shutdownNow();

        /**
         * Stops consuming data. Finishes processing the current batch of data already
received from Kinesis
         * before shutting down.
         */
        Future<Boolean> gracefulShutdownFuture = scheduler.startGracefulShutdown();
        log.info("Waiting up to 20 seconds for shutdown to complete.");
        try {
            gracefulShutdownFuture.get(20, TimeUnit.SECONDS);
        } catch (InterruptedException e) {
            log.info("Interrupted while waiting for graceful shutdown. Continuing.");
        } catch (ExecutionException e) {
            log.error("Exception while executing graceful shutdown.", e);
        } catch (TimeoutException e) {
            log.error("Timeout while waiting for shutdown. Scheduler may not have
exited.");
        }
        log.info("Completed, shutting down now.");
    }

    /**
     * Sends a single record of dummy data to Kinesis.
     */
    private void publishRecord() {
        PutRecordRequest request = PutRecordRequest.builder()
            .partitionKey(RandomStringUtils.randomAlphabetic(5, 20))
            .streamName(streamName)
            .data(SdkBytes.fromByteArray(RandomUtils.nextBytes(10)))
            .build();

        try {
```

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

private static class SampleRecordProcessorFactory implements
ShardRecordProcessorFactory {
    public ShardRecordProcessor shardRecordProcessor() {
        return new SampleRecordProcessor();
    }
}

/**
 * The implementation of the ShardRecordProcessor interface is where the heart of
the record processing logic lives.
 * In this example all we do to 'process' is log info about the records.
 */
private static class SampleRecordProcessor implements ShardRecordProcessor {

    private static final String SHARD_ID_MDC_KEY = "ShardId";

    private static final Logger log =
LoggerFactory.getLogger(SampleRecordProcessor.class);

    private String shardId;

    /**
     * Invoked by the KCL before data records are delivered to the
ShardRecordProcessor instance (via
     * processRecords). In this example we do nothing except some logging.
     *
     * @param initializationInput Provides information related to initialization.
     */
    public void initialize(InitializationInput initializationInput) {
        shardId = initializationInput.shardId();
        MDC.put(SHARD_ID_MDC_KEY, shardId);
        try {
            log.info("Initializing @ Sequence: {}",
initializationInput.extendedSequenceNumber());
        } finally {
```

```
        MDC.remove(SHARD_ID_MDC_KEY);
    }
}

/**
 * Handles record processing logic. The Amazon Kinesis Client Library will
invoke this method to deliver
 * data records to the application. In this example we simply log our records.
 *
 * @param processRecordsInput Provides the records to be processed as well as
information and capabilities
 *                               related to them (e.g. checkpointing).
 */
public void processRecords(ProcessRecordsInput processRecordsInput) {
    MDC.put(SHARD_ID_MDC_KEY, shardId);
    try {
        log.info("Processing {} record(s)",
processRecordsInput.records().size());
        processRecordsInput.records().forEach(r -> log.info("Processing record
pk: {} -- Seq: {}", r.partitionKey(), r.sequenceNumber()));
    } catch (Throwable t) {
        log.error("Caught throwable while processing records. Aborting.");
        Runtime.getRuntime().halt(1);
    } finally {
        MDC.remove(SHARD_ID_MDC_KEY);
    }
}

/** Called when the lease tied to this record processor has been lost. Once the
lease has been lost,
 * the record processor can no longer checkpoint.
 *
 * @param leaseLostInput Provides access to functions and data related to the
loss of the lease.
 */
public void leaseLost(LeaseLostInput leaseLostInput) {
    MDC.put(SHARD_ID_MDC_KEY, shardId);
    try {
        log.info("Lost lease, so terminating.");
    } finally {
        MDC.remove(SHARD_ID_MDC_KEY);
    }
}
```

```
/**
 * Called when all data on this shard has been processed. Checkpointing must
 occur in the method for record
 * processing to be considered complete; an exception will be thrown otherwise.
 *
 * @param shardEndedInput Provides access to a checkpointer method for
 completing processing of the shard.
 */
public void shardEnded(ShardEndedInput shardEndedInput) {
    MDC.put(SHARD_ID_MDC_KEY, shardId);
    try {
        log.info("Reached shard end checkpointing.");
        shardEndedInput.checkpointer().checkpoint();
    } catch (ShutdownException | InvalidStateException e) {
        log.error("Exception while checkpointing at shard end. Giving up.", e);
    } finally {
        MDC.remove(SHARD_ID_MDC_KEY);
    }
}

/**
 * Invoked when Scheduler has been requested to shut down (i.e. we decide to
 stop running the app by pressing
 * Enter). Checkpoints and logs the data a final time.
 *
 * @param shutdownRequestedInput Provides access to a checkpointer, allowing a
 record processor to checkpoint
 *
 *                                     before the shutdown is completed.
 */
public void shutdownRequested(ShutdownRequestedInput shutdownRequestedInput) {
    MDC.put(SHARD_ID_MDC_KEY, shardId);
    try {
        log.info("Scheduler is shutting down, checkpointing.");
        shutdownRequestedInput.checkpointer().checkpoint();
    } catch (ShutdownException | InvalidStateException e) {
        log.error("Exception while checkpointing at requested shutdown. Giving
 up.", e);
    } finally {
        MDC.remove(SHARD_ID_MDC_KEY);
    }
}
}
```

## Entwickeln eines Kinesis Client Library-Verbrauchers in Python

Sie können die Kinesis Client Library (KCL) verwenden, um Anwendungen zu erstellen, die Daten aus Ihren Kinesis-Datenströmen verarbeiten. Die Kinesis Client Library ist in mehreren Sprachen verfügbar. In diesem Thema wird Python behandelt.

Die KCL ist eine Java-Bibliothek. Unterstützung für andere Sprachen als Java wird über eine mehrsprachige Schnittstelle bereitgestellt, die als MultiLangDaemon bezeichnet wird. Dieser Daemon basiert auf Java und wird im Hintergrund ausgeführt, wenn Sie eine andere KCL-Sprache als Java verwenden. Wenn Sie also die KCL für Python installieren und Ihre Consumer-App vollständig in Python schreiben, muss Java aufgrund der MultiLangDaemon trotzdem auf Ihrem System installiert sein. Darüber hinaus verfügt MultiLangDaemon über einige Standardeinstellungen, die Sie möglicherweise an Ihren Anwendungsfall anpassen müssen, z. B. die AWS Region, mit der eine Verbindung hergestellt wird. Weitere Informationen dazu finden Sie MultiLangDaemon auf GitHub der [MultiLangDaemon KCL-Projektseite](#).

Gehen Sie zur [Kinesis Client Library \(Python\) GitHub](#), um die Python-KCL von herunterzuladen. Um Beispielcode für eine Python-KCL-Consumer-Anwendung herunterzuladen, gehen Sie zur [KCL for Python-Beispielprojektseite](#) unter GitHub

Sie müssen die folgenden Aufgaben durchführen, wenn Sie eine KCL-Konsumenten-Anwendung in Python implementieren:

### Aufgaben

- [Implementieren Sie die Klassenmethoden RecordProcessor](#)
- [Ändern der Konfigurationseigenschaften](#)

### Implementieren Sie die Klassenmethoden RecordProcessor

Die RecordProcessor-Klasse muss die RecordProcessorBase-Klasse erweitern, um die folgenden Methoden zu implementieren:

```
initialize
process_records
shutdown_requested
```

Dieses Beispiel stellt Implementierungen bereit, die Sie als Ausgangspunkt verwenden können.



```
#!/usr/bin/env python

# Copyright 2014-2015 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#
# Licensed under the Amazon Software License (the "License").
# You may not use this file except in compliance with the License.
# A copy of the License is located at
#
# http://aws.amazon.com/asl/
#
# or in the "license" file accompanying this file. This file is distributed
# on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
# express or implied. See the License for the specific language governing
# permissions and limitations under the License.

from __future__ import print_function

import sys
import time

from amazon_kclpy import kcl
from amazon_kclpy.v3 import processor

class RecordProcessor(processor.RecordProcessorBase):
    """
    A RecordProcessor processes data from a shard in a stream. Its methods will be
    called with this pattern:

    * initialize will be called once
    * process_records will be called zero or more times
    * shutdown will be called if this MultiLangDaemon instance loses the lease to this
    shard, or the shard ends due
      a scaling change.
    """
    def __init__(self):
        self._SLEEP_SECONDS = 5
        self._CHECKPOINT_RETRIES = 5
        self._CHECKPOINT_FREQ_SECONDS = 60
        self._largest_seq = (None, None)
        self._largest_sub_seq = None
        self._last_checkpoint_time = None
```

```
def log(self, message):
    sys.stderr.write(message)

def initialize(self, initialize_input):
    """
    Called once by a KCLProcess before any calls to process_records

    :param amazon_kclpy.messages.InitializeInput initialize_input: Information
about the lease that this record
    processor has been assigned.
    """
    self._largest_seq = (None, None)
    self._last_checkpoint_time = time.time()

def checkpoint(self, checkpointer, sequence_number=None, sub_sequence_number=None):
    """
    Checkpoints with retries on retryable exceptions.

    :param amazon_kclpy.kcl.Checkpointer checkpointer: the checkpointer provided to
either process_records
    or shutdown
    :param str or None sequence_number: the sequence number to checkpoint at.
    :param int or None sub_sequence_number: the sub sequence number to checkpoint
at.
    """
    for n in range(0, self._CHECKPOINT_RETRIES):
        try:
            checkpointer.checkpoint(sequence_number, sub_sequence_number)
            return
        except kcl.CheckpointError as e:
            if 'ShutdownException' == e.value:
                #
                # A ShutdownException indicates that this record processor should
be shutdown. This is due to
                # some failover event, e.g. another MultiLangDaemon has taken the
lease for this shard.
                #
                print('Encountered shutdown exception, skipping checkpoint')
                return
            elif 'ThrottlingException' == e.value:
                #
                # A ThrottlingException indicates that one of our dependencies is
is over burdened, e.g. too many
                # dynamo writes. We will sleep temporarily to let it recover.
```

```

        #
        if self._CHECKPOINT_RETRIES - 1 == n:
            sys.stderr.write('Failed to checkpoint after {n} attempts,
giving up.\n'.format(n=n))
            return
        else:
            print('Was throttled while checkpointing, will attempt again in
{s} seconds'
                  .format(s=self._SLEEP_SECONDS))
        elif 'InvalidStateException' == e.value:
            sys.stderr.write('MultiLangDaemon reported an invalid state while
checkpointing.\n')
        else: # Some other error
            sys.stderr.write('Encountered an error while checkpointing, error
was {e}.\n'.format(e=e))
            time.sleep(self._SLEEP_SECONDS)

    def process_record(self, data, partition_key, sequence_number,
sub_sequence_number):
        """
        Called for each record that is passed to process_records.

        :param str data: The blob of data that was contained in the record.
        :param str partition_key: The key associated with this 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
        checkpointer which accepts sequence numbers
        from the records to indicate where in the stream to checkpoint.

        :param amazon_kclpy.messages.ProcessRecordsInput process_records_input: the
        records, and metadata about the
            records.
        """
        try:
            for record in process_records_input.records:
                data = record.binary_data
                seq = int(record.sequence_number)
                sub_seq = record.sub_sequence_number
                key = record.partition_key
                self.process_record(data, key, seq, sub_seq)
                if self.should_update_sequence(seq, sub_seq):
                    self._largest_seq = (seq, sub_seq)

            #
            # Checkpoints every self._CHECKPOINT_FREQ_SECONDS seconds
            #
            if time.time() - self._last_checkpoint_time >
self._CHECKPOINT_FREQ_SECONDS:
                self.checkpoint(process_records_input.checkpointer,
str(self._largest_seq[0]), self._largest_seq[1])
                self._last_checkpoint_time = time.time()

        except Exception as e:
            self.log("Encountered an exception while processing records. Exception was
{e}\n".format(e=e))

    def lease_lost(self, lease_lost_input):
        self.log("Lease has been lost")

    def shard_ended(self, shard_ended_input):
        self.log("Shard has ended checkpointing")
```

```
shard_ended_input.checkpointer.checkpoint()

def shutdown_requested(self, shutdown_requested_input):
    self.log("Shutdown has been requested, checkpointing.")
    shutdown_requested_input.checkpointer.checkpoint()

if __name__ == "__main__":
    kcl_process = kcl.KCLProcess(RecordProcessor())
    kcl_process.run()
```

## Ändern der Konfigurationseigenschaften

Das Beispiel zeigt Standardwerte für die Konfigurationseigenschaften, wie in dem folgenden Skript gezeigt. Sie können diese Eigenschaften mit eigenen Werten überschreiben.

```
# The script that abides by the multi-language protocol. This script will
# be executed by the MultiLangDaemon, which will communicate with this script
# over STDIN and STDOUT according to the multi-language protocol.
executableName = sample_kclpy_app.py

# The name of an Amazon Kinesis stream to process.
streamName = words

# Used by the KCL as the name of this application. Will be used as the name
# of an Amazon DynamoDB table which will store the lease and checkpoint
# information for workers with this application name
applicationName = PythonKCLSample

# Users can change the credentials provider the KCL will use to retrieve credentials.
# The DefaultAWSCredentialsProviderChain checks several other providers, which is
# described here:
# http://docs.aws.amazon.com/AWSJavaSDK/latest/javadoc/com/amazonaws/auth/
DefaultAWSCredentialsProviderChain.html
AWSCredentialsProvider = DefaultAWSCredentialsProviderChain

# Appended to the user agent of the KCL. Does not impact the functionality of the
# KCL in any other way.
processingLanguage = python/2.7

# Valid options at TRIM_HORIZON or LATEST.
# See http://docs.aws.amazon.com/kinesis/latest/APIReference/
API_GetShardIterator.html#API_GetShardIterator_RequestSyntax
```

```
initialPositionInStream = TRIM_HORIZON

# The following properties are also available for configuring the KCL Worker that is
  created
# by the MultiLangDaemon.

# The KCL defaults to us-east-1
#regionName = us-east-1

# Fail over time in milliseconds. A worker which does not renew it's lease within this
  time interval
# will be regarded as having problems and it's shards will be assigned to other
  workers.
# For applications that have a large number of shards, this msy be set to a higher
  number to reduce
# the number of DynamoDB IOPS required for tracking leases
#failoverTimeMillis = 10000

# A worker id that uniquely identifies this worker among all workers using the same
  applicationName
# If this isn't provided a MultiLangDaemon instance will assign a unique workerId to
  itself.
#workerId =

# Shard sync interval in milliseconds - e.g. wait for this long between shard sync
  tasks.
#shardSyncIntervalMillis = 60000

# Max records to fetch from Kinesis in a single GetRecords call.
#maxRecords = 10000

# Idle time between record reads in milliseconds.
#idleTimeBetweenReadsInMillis = 1000

# Enables applications flush/checkpoint (if they have some data "in progress", but
  don't get new data for while)
#callProcessRecordsEvenForEmptyRecordList = false

# Interval in milliseconds between polling to check for parent shard completion.
# Polling frequently will take up more DynamoDB IOPS (when there are leases for shards
  waiting on
# completion of parent shards).
#parentShardPollIntervalMillis = 10000
```

```
# Cleanup leases upon shards completion (don't wait until they expire in Kinesis).
# Keeping leases takes some tracking/resources (e.g. they need to be renewed,
  assigned), so by default we try
# to delete the ones we don't need any longer.
#cleanupLeasesUponShardCompletion = true

# Backoff time in milliseconds for Amazon Kinesis Client Library tasks (in the event of
  failures).
#taskBackoffTimeMillis = 500

# Buffer metrics for at most this long before publishing to CloudWatch.
#metricsBufferTimeMillis = 10000

# Buffer at most this many metrics before publishing to CloudWatch.
#metricsMaxQueueSize = 10000

# KCL will validate client provided sequence numbers with a call to Amazon Kinesis
  before checkpointing for calls
# to 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
```

## Anwendungsname

Die KCL erfordert einen Anwendungsnamen, der unter Ihren Anwendungen sowie den Amazon-DynamoDB-Tabellen in derselben Region eindeutig ist. Sie verwendet den Wert der Anwendungsnamenkonfiguration auf folgende Arten:

- Für mit diesem Anwendungsnamen verknüpfte Auftragnehmer wird angenommen, dass sie gemeinsam im gleichen Stream arbeiten. Diese Worker können auf mehrere Instances verteilt sein. Wenn Sie eine zusätzliche Instance desselben Anwendungscode ausführen, jedoch mit einem anderen Anwendungsnamen, behandelt die KCL die zweite Instance als eine völlig getrennte Anwendung, die ebenfalls im selben Stream arbeitet.
- Die KCL erstellt eine DynamoDB-Tabelle mit dem Namen der Anwendung und verwendet die Tabelle für die Verwaltung von Statusinformationen für die Anwendung (wie Checkpoints und Auftragnehmer-Shard-Zuweisungen). Jede Anwendung verfügt über eine eigene DynamoDB-

Tabelle. Weitere Informationen finden Sie unter [Mithilfe einer Leasetabelle können Sie die von der KCL-Konsumentenanzwendung verarbeiteten Shards verfolgen](#).

## Anmeldeinformationen

Sie müssen Ihre AWS-Anmeldeinformationen für einen der Anmeldeinformationsanbieter in der [Anmeldeinformationsanbieter-Standardkette](#) bereitstellen. Sie können die Eigenschaft `AWSCredentialsProvider` verwenden, um einen Anmeldeinformationsanbieter einzurichten. Wenn Sie Ihre Konsumentenanzwendung auf einer Amazon-EC2-Instance ausführen, empfehlen wir, die Instance mit einer IAM-Rolle zu konfigurieren. AWS-Anmeldeinformationen, die die mit dieser IAM-Rolle verknüpften Berechtigungen widerspiegeln, werden den Anwendungen auf der Instance über deren Instance-Metadaten zur Verfügung gestellt. Dies ist die sicherste Art, Anmeldeinformationen für eine Konsumentenanzwendung zu verwalten, die auf einer EC2-Instance ausgeführt wird.

## Entwickeln benutzerdefinierter Verbraucher mit gemeinsam genutztem Durchsatz unter Verwendung von AWS SDK for Java

Eine der Methoden für die Entwicklung benutzerdefinierter Verbraucher von Kinesis Data Streams mit gemeinsamem Zugriff ist die Verwendung der APIs von Amazon Kinesis Data Streams. In diesem Abschnitt wird die Verwendung der APIs von Kinesis Data Streams mit dem AWS-SDK für Java beschrieben. Anhand des Java-Beispiel-Codes in diesem Abschnitt, der nach Operationstyp unterteilt ist, wird gezeigt, wie Sie grundlegende KDS-API-Operationen durchführen.

Diese Beispiele stellen keinen produktionsbereiten Code dar. Sie überprüfen nicht alle möglichen Ausnahmen und es werden nicht alle möglichen Sicherheits- oder Leistungsüberlegungen berücksichtigt.

Sie können die APIs für Kinesis Data Streams mit anderen Programmiersprachen aufrufen. Weitere Informationen über verfügbare AWS-SDKs finden Sie unter [Entwickeln Sie Ihre Apps mit Amazon Web Services](#).

### Important

Die empfohlene Methode zur Entwicklung benutzerdefinierter Verbraucher von Kinesis Data Streams mit durchgängiger gemeinsamer Nutzung ist die Verwendung der Kinesis Client Library (KCL). KCL unterstützt Sie bei der Nutzung und Verarbeitung von Daten aus einem Kinesis-Datenstrom, indem sie sich um viele der komplexen Aufgaben kümmert, die



mit verteilter Datenverarbeitung verbunden sind. Weitere Informationen finden Sie unter [Entwickeln benutzerdefinierter Verbraucher mit gemeinsamem Durchsatz mithilfe von KCL](#).

## Themen

- [Abrufen von Daten aus einem Stream](#)
- [Verwenden von Shard-Iteratoren](#)
- [Verwenden von GetRecords](#)
- [Anpassung an einen Reshard](#)
- [Interaktion mit Daten mithilfe des AWS Glue Schema-Registry](#)

## Abrufen von Daten aus einem Stream

Die APIs von Kinesis Data Streams enthalten die Methoden `getShardIterator` und `getRecords`, die Sie aufrufen können, um Datensätze aus einem Datenstrom abzurufen. Dies ist das PULL-Modell, bei dem der Code Datensätze direkt aus den Shards des Datenstroms abrufen.

### Important

Wir empfehlen Ihnen, die von KCL bereitgestellte Unterstützung für Datensatzprozessoren zu verwenden, um Datensätze aus Ihren Datenströmen abzurufen. Dies ist das PUSH-Modell, bei dem Sie den Code implementieren, der die Daten verarbeitet. Die KCL ruft Datensätze aus dem Datenstrom ab und übermittelt diese an Ihren Anwendungscode. Darüber hinaus bietet die KCL Funktionen für einen Failover, eine Wiederherstellung und eine Lastverteilung. Weitere Informationen finden Sie unter [Entwickeln benutzerdefinierter Verbraucher mit gemeinsamem Durchsatz mithilfe von KCL](#).

In einigen Fällen können Sie jedoch die APIs von Kinesis Data Streams verwenden. Dies ist beispielsweise der Fall, wenn Sie benutzerdefinierte Tools für das Überwachen und Debuggen Ihrer Datenströme implementieren.

**⚠ Important**

Kinesis Data Streams unterstützt Änderungen des Zeitraums der Datensatzaufbewahrung für einen Datenstrom. Weitere Informationen finden Sie unter [Ändern des Zeitraums der Datenaufbewahrung](#).

## Verwenden von Shard-Iteratoren

Sie rufen Datensätze aus dem Stream pro Shard ab. Für jeden Shard und jeden Datensatzstapel, den Sie aus dem Shard abrufen, benötigen Sie einen Shard-Iterator. Der Shard-Iterator wird im `getRecordsRequest`-Objekt verwendet, um den Shard anzugeben, aus dem die Datensätze abgerufen werden. Der Typ, der dem Shard-Iterator zugeordnet ist, gibt die Stelle im Shard an, von der die Datensätze abgerufen werden sollen (weitere Informationen dazu finden Sie später in diesem Abschnitt). Bevor Sie mit dem Shard-Iterator arbeiten können, müssen Sie den Shard abrufen, wie in [DescribeStream API — Veraltet](#) beschrieben.

Rufen Sie diesen ersten Shard-Iterator mit der `getShardIterator`-Methode ab. Rufen Sie Shard-Iteratoren für weitere Datensatzstapel mit der `getNextShardIterator`-Methode des `getRecordsResult`-Objekts ab, das von der `getRecords`-Methode zurückgegeben wird. Ein Shard-Iterator verliert seine Gültigkeit nach 5 Minuten. Wenn Sie einen Shard-Iterator verwenden, solange er gültig ist, erhalten Sie einen neuen. Jeder Shard-Iterator ist 5 Minuten lang gültig, selbst wenn er bereits verwendet wurde.

Instanzieren Sie `GetShardIteratorRequest`, um den ersten Shard-Iterator abzurufen. Übergeben Sie ihn an die `getShardIterator`-Methode. Geben Sie zum Konfigurieren der Anforderung den Stream und die Shard-ID an. Weitere Informationen zum Abrufen des Streams in Ihrem AWS-Konto finden Sie unter [Auflisten von Streams](#). Informationen zum Abrufen der Shards in einem Stream finden Sie unter [DescribeStream API — Veraltet](#).

```
String shardIterator;
GetShardIteratorRequest getShardIteratorRequest = new GetShardIteratorRequest();
getShardIteratorRequest.setStreamName(myStreamName);
getShardIteratorRequest.setShardId(shard.getShardId());
getShardIteratorRequest.setShardIteratorType("TRIM_HORIZON");

GetShardIteratorResult getShardIteratorResult =
    client.getShardIterator(getShardIteratorRequest);
shardIterator = getShardIteratorResult.getShardIterator();
```

Der Beispiel-Code gibt beim Abrufen des ersten Shard-Iterators `TRIM_HORIZON` als Iterator-Typ an. Dieser Iterator-Typ bedeutet, dass Datensätze zurückgegeben werden sollen. Und zwar beginnend mit dem ersten zum Shard hinzugefügten Datensatz und nicht mit dem letzten hinzugefügten Datensatz, auch als Spitze bezeichnet. Folgende Iterator-Typen werden unterstützt:

- `AT_SEQUENCE_NUMBER`
- `AFTER_SEQUENCE_NUMBER`
- `AT_TIMESTAMP`
- `TRIM_HORIZON`
- `LATEST`

Weitere Informationen finden Sie unter [ShardIteratorType](#).

Bei einigen Iterator-Typen müssen Sie zusätzlich eine Sequenznummer angeben, z. B.:

```
getShardIteratorRequest.setShardIteratorType("AT_SEQUENCE_NUMBER");
getShardIteratorRequest.setStartingSequenceNumber(specialSequenceNumber);
```

Nachdem Sie einen Datensatz mit `getRecords` abgerufen haben, erhalten Sie die Sequenznummer für den Datensatz, indem Sie die `getSequenceNumber`-Methode des Datensatzes aufrufen.

```
record.getSequenceNumber()
```

Darüber hinaus erhält der Code, durch den Datensätze zum Stream hinzugefügt werden, die Sequenznummer für einen hinzugefügten Datensatz, indem `getSequenceNumber` auf dem Ergebnis von `putRecord` aufgerufen wird.

```
lastSequenceNumber = putRecordResult.getSequenceNumber();
```

Sie können mit diesen Sequenznummern eine strenge aufsteigende Anordnung der Datensätze gewährleisten. Weitere Informationen finden Sie im Code-Beispiel unter [PutRecord – Beispiel](#).

## Verwenden von GetRecords

Instanzieren Sie nach Abruf des Shard-Iterators ein `GetRecordsRequest`-Objekt. Geben Sie den Iterator für die Anforderung mit der `setShardIterator`-Methode an.

Optional können Sie auch die Anzahl der abzurufenden Datensätze mithilfe der `setLimit`-Methode angeben. Die Anzahl der Datensätze, die `getRecords` zurückgibt, ist stets gleich oder kleiner als dieses Limit. Wenn Sie kein Limit angeben, gibt `getRecords` 10 MB abgerufener Datensätze zurück. Beim unten stehenden Beispiel-Code wird das Limit auf 25 Datensätze festgelegt.

Wenn keine Datensätze zurückgegeben werden, bedeutet dies, dass derzeit keine Datensätze von diesem Shard für die vom Shard-Iterator angegebene Sequenznummer verfügbar sind. Wenn dies der Fall ist, sollte Ihre Anwendung so lange warten, wie dies für die Datenquellen des Streams angemessen ist. Versuchen Sie dann erneut mit dem Shard-Iterator, der vom vorherigen Aufruf von `getRecords` zurückgegeben wurde, Daten aus dem Shard abzurufen.

Übergeben Sie `getRecordsRequest` an die `getRecords`-Methode und erfassen Sie die zurückgegebenen Werte als `getRecordsResult`-Objekt. Rufen Sie die `getRecords`-Methode auf dem `getRecordsResult`-Objekt auf, um die Datensätze abzurufen.

```
GetRecordsRequest getRecordsRequest = new GetRecordsRequest();
getRecordsRequest.setShardIterator(shardIterator);
getRecordsRequest.setLimit(25);

GetRecordsResult getRecordsResult = client.getRecords(getRecordsRequest);
List<Record> records = getRecordsResult.getRecords();
```

Zur Vorbereitung auf einen weiteren Aufruf von `getRecords` rufen Sie den nächsten Shard-Iterator von `getRecordsResult` ab.

```
shardIterator = getRecordsResult.getNextShardIterator();
```

Die besten Ergebnisse erzielen Sie, wenn Sie mindestens 1 Sekunde (1.000 Millisekunden) zwischen den Aufrufen von `getRecords` warten, um ein Überschreiten des Limits für die Aufrufe von `getRecords` zu vermeiden.

```
try {
    Thread.sleep(1000);
}
catch (InterruptedException e) {}
```

In der Regel sollten Sie `getRecords` in einer Schleife aufrufen, auch wenn Sie einen einzelnen Datensatz in einem Testszenario abrufen. Ein einzelner Aufruf von `getRecords` gibt möglicherweise eine leere Datensatzliste zurück, auch wenn der Shard mehrere Datensätze mit höheren Sequenznummern enthält. In diesem Fall verweist der zurückgegebene `NextShardIterator` zusammen mit der leeren Datensatzliste auf eine höhere Sequenznummer im Shard. Nachfolgende Aufrufe von `getRecords` führen dann zu einem erfolgreichen Abruf. Das folgende Beispiel zeigt die Verwendung einer Schleife.

#### Beispiel: `getRecords`

Das folgende Codebeispiel spiegelt die `getRecords`-Tipps in diesem Abschnitt wieder, einschließlich der Aufrufe in Schleifen.

```
// Continuously read data records from a shard
List<Record> records;

while (true) {

    // Create a new getRecordsRequest with an existing shardIterator
    // Set the maximum records to return to 25

    GetRecordsRequest getRecordsRequest = new GetRecordsRequest();
    getRecordsRequest.setShardIterator(shardIterator);
    getRecordsRequest.setLimit(25);

    GetRecordsResult result = client.getRecords(getRecordsRequest);

    // Put the result into record list. The result can be empty.
    records = result.getRecords();

    try {
        Thread.sleep(1000);
    }
    catch (InterruptedException exception) {
        throw new RuntimeException(exception);
    }

    shardIterator = result.getNextShardIterator();
}
```

Bei der Nutzung der Kinesis Client Library müssen möglicherweise mehrere Aufrufe durchgeführt werden, ehe Daten zurückgegeben werden. Dieses Verhalten ist Design-bedingt und kein Fehler der KCL oder Ihrer Daten.

## Anpassung an einen Reshard

Wenn `getRecordsResult.getNextShardIterator` null zurückgibt, bedeutet dies, dass eine Aufteilung oder Zusammenführung des Shards stattgefunden hat, die diesen Shard betrifft. Dieser Shard befindet sich jetzt in einem CLOSED-Status und Sie haben alle verfügbaren Datensätze von diesem Shard gelesen.

In diesem Szenario können Sie `getRecordsResult.childShards` verwenden, um etwas über die neuen untergeordneten Shards des zu verarbeitenden Shards zu erfahren, die durch die Aufteilung oder Zusammenführung entstanden sind. Weitere Informationen finden Sie unter [ChildShard](#).

Bei einer Teilung ist die `parentShardId` der beiden neuen Shards gleich der Shard-ID des zuvor verarbeiteten Shards. Der Wert von `adjacentParentShardId` für beide Shards ist null.

Bei einer Zusammenführung ist bei dem entstandenen einzelnen Datensatz die `parentShardId` identisch mit der Shard-ID eines übergeordneten Shards und die `adjacentParentShardId` ist gleich der Shard-ID des anderen übergeordneten Shards. Ihre Anwendung hat bereits alle Daten aus einem dieser Shards ausgelesen. Dies ist der Shard, für den `getRecordsResult.getNextShardIterator` null zurückgegeben hat. Wenn die Reihenfolge der Daten für Ihre Anwendung von Bedeutung ist, stellen Sie sicher, dass die Anwendung auch alle Daten des anderen übergeordneten Shards ausliest, ehe neue Daten aus dem durch die Zusammenführung entstandenen untergeordneten Shards ausgelesen werden.

Wenn Sie mehrere Prozessoren zum Abrufen von Daten aus dem Stream verwenden (beispielsweise einen Prozessor pro Shard), und es kommt zu einer Teilung oder Zusammenführung von Shards, sollten Sie die Anzahl der Prozessoren entsprechend anpassen.

Weitere Informationen zum Resharding, einschließlich einer Diskussion über Shard-Status, beispielsweise CLOSED finden Sie unter [Resharding eines Streams](#).

## Interaktion mit Daten mithilfe des AWS Glue Schema-Registry

Sie können Ihre Kinesis-Datenströme in das AWS Glue Schema Registry integrieren. Mit dem AWS Glue Schema Registry können Sie Schemata zentral erkennen, steuern und weiterentwickeln

und gleichzeitig sicherstellen, dass die erstellten Daten kontinuierlich durch ein registriertes Schema validiert werden. Ein Schema definiert die Struktur und das Format eines Datensatzes. Ein Schema ist eine versionierte Spezifikation für zuverlässige Datenveröffentlichung, -nutzung oder -speicherung. Mit dem AWS Glue Schema Registry können Sie die durchgängige Datenqualität und Datenverwaltung in Ihren Streaming-Anwendungen verbessern. Weitere Informationen finden Sie unter [AWS Glue Schema Registry](#). Eine der Möglichkeiten, diese Integration einzurichten, ist die `GetRecords`-API von Kinesis Data Streams, die im AWS-Java SDK verfügbar ist.

Detaillierte Anweisungen zur Einrichtung der Integration von Kinesis Data Streams mit Schema Registry mithilfe der `GetRecords`-APIs Kinesis Data Streams finden Sie im Abschnitt „Interaktion mit Daten mithilfe der APIs für Kinesis Data Streams“ unter [Anwendungsfall: Integration von Amazon Kinesis Data Streams mit der AWS Glue Schema Registry](#).

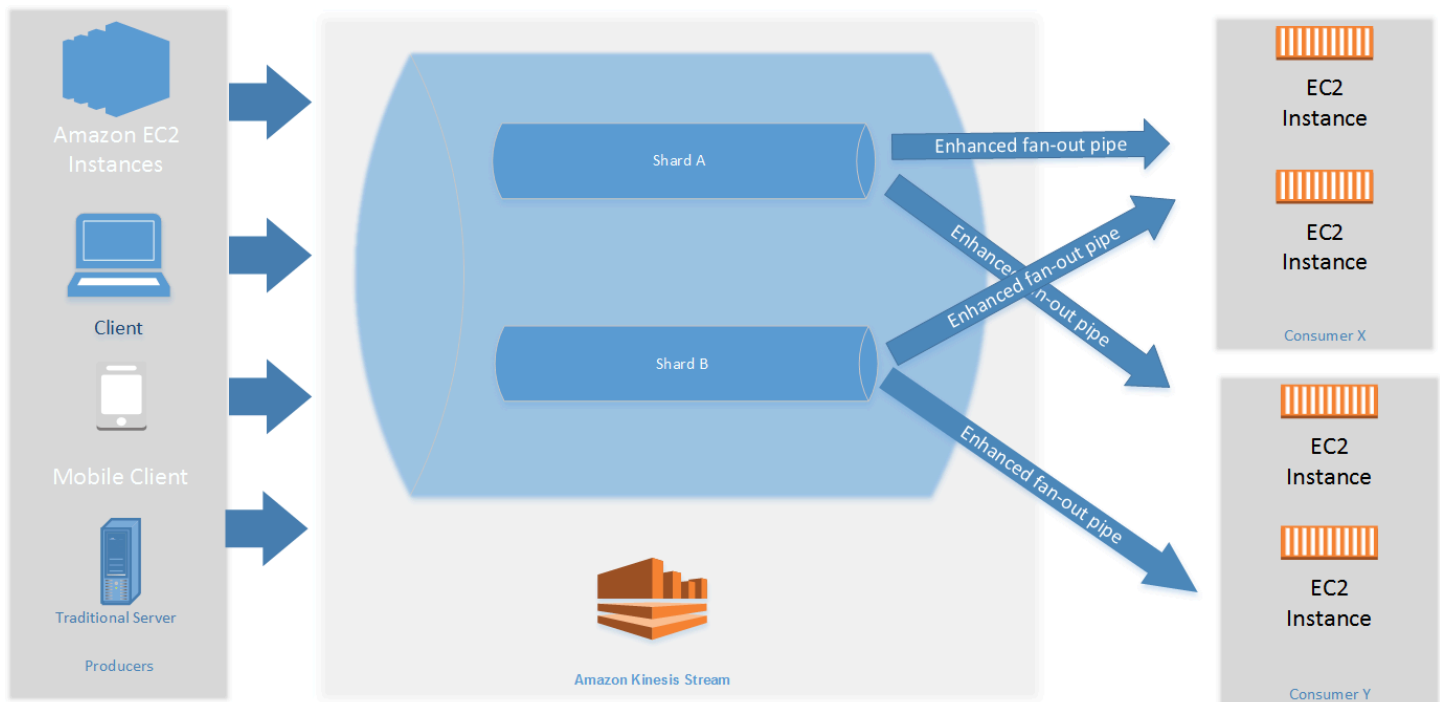
## Entwickeln benutzerdefinierter Verbraucher mit dediziertem Durchsatz (Erweitertes Rundsenden)

In Amazon Kinesis Data Streams können Sie Verbraucher erstellen, die ein Feature namens erweiterte Rundsendungen verwenden. Mit dieser Funktion können Verbraucher Datensätze aus einem Stream empfangen, mit einem Durchsatz von bis zu 2 MB Daten pro Sekunde pro Shard. Dieser Durchsatz ist dediziert, d. h. dass Verbraucher, die ein erweitertes Rundsenden verwenden, nicht mit anderen Verbrauchern konkurrieren müssen, die Daten aus dem Stream empfangen. Kinesis Data Streams überträgt Datensätze aus dem Stream zu den Verbrauchern die ein erweitertes Rundsenden verwenden. Aus diesem Grund müssen diese Verbraucher keine Daten abfragen.

### Important

Sie können bis zu zwanzig Nutzer pro Stream registrieren, um das erweiterte Rundsenden zu verwenden.

Das folgende Diagramm zeigt die Architektur für das erweiterte Rundsenden. Wenn Sie die Version 2.0 oder höher der Amazon Kinesis Client Library (KCL) verwenden, um einen Verbraucher zu erstellen, richtet die KCL den Verbraucher so ein, dass er erweitertes Rundsenden verwendet, um Daten von allen Shards des Streams zu empfangen. Wenn Sie die API verwenden, um einen Verbraucher zu erstellen, der ein erweitertes Rundsenden verwendet, können Sie einzelne Shards abonnieren.



Das Diagramm zeigt Folgendes:

- Einen Stream mit zwei Shards.
- Zwei Verbraucher, die erweitertes Rundsenden verwenden, um Daten vom Stream zu empfangen: Consumer X und Consumer Y. Beide Verbraucher haben alle Shards und alle Datensätze im Stream abonniert. Wenn Sie die Version 2.0 oder höher der KCL verwenden, um einen Verbraucher zu erstellen, abonniert die KCL automatisch alle Shards des Streams. Wenn Sie dagegen die API verwenden, um einen Verbraucher zu erstellen, können Sie einzelne Shards abonnieren.
- Pfeile stellen die Pipes für das erweiterte Rundsenden dar, die die Verbraucher verwenden, um Daten aus dem Stream zu erhalten. Eine Pipe für erweitertes Rundsenden liefert bis zu 2 MB/s Daten pro Shard, unabhängig von anderen Pipes oder der Gesamtzahl der Verbraucher.

## Themen

- [Entwicklung von Verbrauchern für erweitertes Rundsenden mit KCL 2.x](#)
- [Entwicklung Verbrauchern für erweitertes Rundsenden mit der API für Kinesis Data Streams](#)
- [Verwalten von Verbrauchern für erweitertes Rundsenden mit dem AWS Management Console](#)



## Entwicklung von Verbrauchern für erweitertes Rundsenden mit KCL 2.x

Verbraucher, die ein erweitertes Rundsenden in Amazon Kinesis Data Streams verwenden, können Datensätze aus einem Datenstrom mit einem dedizierten Durchsatz von bis zu 2 MB Daten pro Sekunde pro Shard empfangen. Diese Art Verbraucher muss nicht mit anderen Verbrauchern konkurrieren, die Daten aus dem Stream empfangen. Weitere Informationen finden Sie unter [Entwickeln benutzerdefinierter Verbraucher mit dediziertem Durchsatz \(Erweitertes Rundsenden\)](#).

Sie können die Version 2.0 oder höher der Kinesis Client Library (KCL) verwenden, um Anwendungen zu entwickeln, die erweitertes Rundsenden verwenden, um Daten aus Streams zu empfangen. Die KCL abonniert Ihre Anwendung automatisch auf alle Shards eines Streams und stellt sicher, dass Ihre Verbraucheranwendung mit einem Durchsatzwert von 2 MB/s pro Shard lesen kann. Weitere Informationen zur Verwendung der KCL ohne Aktivierung von erweitertem Rundsenden finden Sie unter [Entwickeln von Verbrauchern mit der Kinesis Client Library 2.0](#).

### Themen

- [Entwickeln von Verbrauchern für erweitertes Rundsenden mit KCL 2.x in Java](#)

## Entwickeln von Verbrauchern für erweitertes Rundsenden mit KCL 2.x in Java

Sie können die Version 2.0 oder höher der Kinesis Client Library (KCL) verwenden, um Anwendungen in Amazon Kinesis Data Streams zu entwickeln, die Daten aus Streams mit erweitertem Rundsenden empfangen. Der folgende Code zeigt eine Beispielimplementierung in Java für `ProcessorFactory` und `RecordProcessor`.

Es wird empfohlen, dass Sie `KinesisClientUtil` zum Erstellen von `KinesisAsyncClient` und zum Konfigurieren von `maxConcurrency` in `KinesisAsyncClient` verwenden.

### Important

Für den Amazon Kinesis Client kann sich die Latenz möglicherweise signifikant erhöhen, sofern Sie `KinesisAsyncClient` nicht für einen `maxConcurrency`-Wert konfigurieren, der hoch genug ist, um alle Leases plus zusätzliche Verwendungen von `KinesisAsyncClient` zu ermöglichen.

```
/*  
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
```

```
*
* Licensed under the Amazon Software License (the "License").
* You may not use this file except in compliance with the License.
* A copy of the License is located at
*
* http://aws.amazon.com/asl/
*
* or in the "license" file accompanying this file. This file is distributed
* on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
* express or implied. See the License for the specific language governing
* permissions and limitations under the License.
*/

/*
* Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
*
* Licensed under the Apache License, Version 2.0 (the "License").
* You may not use this file except in compliance with the License.
* A copy of the License is located at
*
* http://www.apache.org/licenses/LICENSE-2.0
*
* or in the "license" file accompanying this file. This file is distributed
* on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
* express or implied. See the License for the specific language governing
* permissions and limitations under the License.
*/

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.UUID;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.ScheduledFuture;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

import org.apache.commons.lang3.ObjectUtils;
import org.apache.commons.lang3.RandomStringUtils;
import org.apache.commons.lang3.RandomUtils;
import org.slf4j.Logger;
```

```
import org.slf4j.LoggerFactory;
import org.slf4j.MDC;

import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cloudwatch.CloudWatchAsyncClient;
import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;
import software.amazon.awssdk.services.kinesis.KinesisAsyncClient;
import software.amazon.awssdk.services.kinesis.model.PutRecordRequest;
import software.amazon.kinesis.common.ConfigsBuilder;
import software.amazon.kinesis.common.KinesisClientUtil;
import software.amazon.kinesis.coordinator.Scheduler;
import software.amazon.kinesis.exceptions.InvalidStateException;
import software.amazon.kinesis.exceptions.ShutdownException;
import software.amazon.kinesis.lifecycle.events.InitializationInput;
import software.amazon.kinesis.lifecycle.events.LeaseLostInput;
import software.amazon.kinesis.lifecycle.events.ProcessRecordsInput;
import software.amazon.kinesis.lifecycle.events.ShardEndedInput;
import software.amazon.kinesis.lifecycle.events.ShutdownRequestedInput;
import software.amazon.kinesis.processor.ShardRecordProcessor;
import software.amazon.kinesis.processor.ShardRecordProcessorFactory;

public class SampleSingle {

    private static final Logger log = LoggerFactory.getLogger(SampleSingle.class);

    public static void main(String... args) {
        if (args.length < 1) {
            log.error("At a minimum, the stream name is required as the first argument.
The Region may be specified as the second argument.");
            System.exit(1);
        }

        String streamName = args[0];
        String region = null;
        if (args.length > 1) {
            region = args[1];
        }

        new SampleSingle(streamName, region).run();
    }

    private final String streamName;
    private final Region region;
```

```
private final KinesisAsyncClient kinesisClient;

private SampleSingle(String streamName, String region) {
    this.streamName = streamName;
    this.region = Region.of(ObjectUtils.firstNonNull(region, "us-east-2"));
    this.kinesisClient =
KinesisClientUtil.createKinesisAsyncClient(KinesisAsyncClient.builder().region(this.region));
}

private void run() {
    ScheduledExecutorService producerExecutor =
Executors.newSingleThreadScheduledExecutor();
    ScheduledFuture<?> producerFuture =
producerExecutor.scheduleAtFixedRate(this::publishRecord, 10, 1, TimeUnit.SECONDS);

    DynamoDbAsyncClient dynamoClient =
DynamoDbAsyncClient.builder().region(region).build();
    CloudWatchAsyncClient cloudWatchClient =
CloudWatchAsyncClient.builder().region(region).build();
    ConfigsBuilder configsBuilder = new ConfigsBuilder(streamName, streamName,
kinesisClient, dynamoClient, cloudWatchClient, UUID.randomUUID().toString(), new
SampleRecordProcessorFactory());

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

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

    System.out.println("Press enter to shutdown");
    BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
    try {
        reader.readLine();
    } catch (IOException ioex) {
        log.error("Caught exception while waiting for confirm. Shutting down.",
ioex);
    }
}
```

```
    }

    log.info("Cancelling producer, and shutting down executor.");
    producerFuture.cancel(true);
    producerExecutor.shutdownNow();

    Future<Boolean> gracefulShutdownFuture = scheduler.startGracefulShutdown();
    log.info("Waiting up to 20 seconds for shutdown to complete.");
    try {
        gracefulShutdownFuture.get(20, TimeUnit.SECONDS);
    } catch (InterruptedException e) {
        log.info("Interrupted while waiting for graceful shutdown. Continuing.");
    } catch (ExecutionException e) {
        log.error("Exception while executing graceful shutdown.", e);
    } catch (TimeoutException e) {
        log.error("Timeout while waiting for shutdown. Scheduler may not have
exited.");
    }
    log.info("Completed, shutting down now.");
}

private void publishRecord() {
    PutRecordRequest request = PutRecordRequest.builder()
        .partitionKey(RandomStringUtils.randomAlphabetic(5, 20))
        .streamName(streamName)
        .data(SdkBytes.fromByteArray(RandomUtils.nextBytes(10)))
        .build();

    try {
        kinesisClient.putRecord(request).get();
    } catch (InterruptedException e) {
        log.info("Interrupted, assuming shutdown.");
    } catch (ExecutionException e) {
        log.error("Exception while sending data to Kinesis. Will try again next
cycle.", e);
    }
}

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

```
private static class SampleRecordProcessor implements ShardRecordProcessor {

    private static final String SHARD_ID_MDC_KEY = "ShardId";

    private static final Logger log =
LoggerFactory.getLogger(SampleRecordProcessor.class);

    private String shardId;

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

    public void processRecords(ProcessRecordsInput processRecordsInput) {
        MDC.put(SHARD_ID_MDC_KEY, shardId);
        try {
            log.info("Processing {} record(s)",
processRecordsInput.records().size());
            processRecordsInput.records().forEach(r -> log.info("Processing record
pk: {} -- Seq: {}", r.partitionKey(), r.sequenceNumber()));
        } catch (Throwable t) {
            log.error("Caught throwable while processing records. Aborting.");
            Runtime.getRuntime().halt(1);
        } finally {
            MDC.remove(SHARD_ID_MDC_KEY);
        }
    }

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

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

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

## Entwicklung Verbrauchern für erweitertes Rundsenden mit der API für Kinesis Data Streams

Erweitertes Rundsenden ist ein Feature in Amazon Kinesis Data Streams, die es Verbrauchern ermöglicht, Datensätze aus einem Datenstrom mit einem dedizierten Durchsatz von bis zu 2 MB Daten pro Sekunde pro Shard zu empfangen. Ein Verbraucher, der ein erweitertes Rundsenden verwendet, muss nicht mit anderen Verbrauchern konkurrieren, die Daten aus dem Stream empfangen. Weitere Informationen finden Sie unter [Entwickeln benutzerdefinierter Verbraucher mit dediziertem Durchsatz \(Erweitertes Rundsenden\)](#).

Sie können API-Operationen zum Erstellen eines Verbrauchers in Kinesis Data Streams verwenden der erweitertes Rundsenden verwendet.

## Einen Verbraucher mit erweitertem Rundsenden unter Verwendung der API für Kinesis Data Streams registrieren

1. Rufen Sie [RegisterStreamConsumer](#) auf, um Ihre Anwendung als Verbraucher zu registrieren, der ein erweitertes Rundsenden verwendet. Kinesis Data Streams generiert einen Amazon-Ressourcennamen (ARN) für den Verbraucher und gibt ihn in der Antwort zurück.
2. Um einen bestimmten Shard zu überwachen, übergeben Sie den Verbraucher-ARN in einem Aufruf an [SubscribeToShard](#). Kinesis Data Streams beginnt dann damit, Ihnen die Datensätze aus diesem Shard zu übertragen, nämlich in Form von Ereignissen des Typs [SubscribeToShardEvent](#) über eine HTTP/2-Verbindung. Die Verbindung bleibt für bis zu 5 Minuten offen. Rufen Sie [SubscribeToShard](#) erneut auf, wenn Sie weiter Datensätze von dem Shard empfangen wollen, nachdem die `future`, die vom Aufruf von [SubscribeToShard](#) zurückgegeben wurde, normal oder außergewöhnlich beendet wurde.

### Note

Die `SubscribeToShard`-API gibt auch die Liste der untergeordneten Shards des aktuellen Shards zurück, wenn das Ende des aktuellen Shards erreicht ist.

3. Um die Registrierung eines Verbrauchers aufzuheben, der ein erweitertes Rundsenden verwendet, rufen Sie [DeregisterStreamConsumer](#) auf.

Der folgende Code ist ein Beispiel dafür, wie Sie für Ihren Verbraucher ein Abonnement für einen Shard einrichten, das Abonnement regelmäßig erneuern und die Ereignisse verarbeiten können.

```
import software.amazon.awssdk.services.kinesis.KinesisAsyncClient;
import software.amazon.awssdk.services.kinesis.model.ShardIteratorType;
import software.amazon.awssdk.services.kinesis.model.SubscribeToShardEvent;
import software.amazon.awssdk.services.kinesis.model.SubscribeToShardRequest;
import
software.amazon.awssdk.services.kinesis.model.SubscribeToShardResponseHandler;

import java.util.concurrent.CompletableFuture;

/**
 * See https://github.com/awsdocs/aws-doc-sdk-examples/blob/master/javav2/
example_code/kinesis/src/main/java/com/example/kinesis/KinesisStreamEx.java
 * for complete code and more examples.
 */
```



```
public class SubscribeToShardSimpleImpl {

    private static final String CONSUMER_ARN = "arn:aws:kinesis:us-
east-1:123456789123:stream/foobar/consumer/test-consumer:1525898737";
    private static final String SHARD_ID = "shardId-000000000000";

    public static void main(String[] args) {

        KinesisAsyncClient client = KinesisAsyncClient.create();

        SubscribeToShardRequest request = SubscribeToShardRequest.builder()
            .consumerARN(CONSUMER_ARN)
            .shardId(SHARD_ID)
            .startingPosition(s -> s.type(ShardIteratorType.LATEST)).build();

        // Call SubscribeToShard iteratively to renew the subscription
periodically.
        while(true) {
            // Wait for the CompletableFuture to complete normally or
exceptionally.
            callSubscribeToShardWithVisitor(client, request).join();
        }

        // Close the connection before exiting.
        // client.close();
    }

    /**
     * Subscribes to the stream of events by implementing the
SubscribeToShardResponseHandler.Visitor interface.
     */
    private static CompletableFuture<Void>
callSubscribeToShardWithVisitor(KinesisAsyncClient client, SubscribeToShardRequest
request) {
        SubscribeToShardResponseHandler.Visitor visitor = new
SubscribeToShardResponseHandler.Visitor() {
            @Override
            public void visit(SubscribeToShardEvent event) {
                System.out.println("Received subscribe to shard event " + event);
            }
        };
        SubscribeToShardResponseHandler responseHandler =
SubscribeToShardResponseHandler
```

```
        .builder()
        .onError(t -> System.err.println("Error during stream - " +
t.getMessage()))
        .subscriber(visitor)
        .build();
    return client.subscribeToShard(request, responseHandler);
}
}
```

Wenn `event.ContinuationSequenceNumber` null zurückgibt, bedeutet dies, dass eine Aufteilung oder Zusammenführung des Shards stattgefunden hat, die diesen Shard betrifft. Dieser Shard befindet sich jetzt in einem CLOSED-Status und Sie haben alle verfügbaren Datensätze von diesem Shard gelesen. In diesem Szenario können Sie, wie im obigen Beispiel, `event.childShards` verwenden, um etwas über die neuen untergeordneten Shards des zu verarbeitenden Shards zu erfahren, die durch die Aufteilung oder Zusammenführung entstanden sind. Weitere Informationen finden Sie unter [ChildShard](#).

## Interaktion mit Daten mithilfe des AWS Glue Schema-Registry

Sie können Ihre Kinesis-Datenströme in das AWS Glue Schema Registry integrieren. Mit dem AWS Glue Schema Registry können Sie Schemata zentral erkennen, steuern und weiterentwickeln und gleichzeitig sicherstellen, dass die erstellten Daten kontinuierlich durch ein registriertes Schema validiert werden. Ein Schema definiert die Struktur und das Format eines Datensatzes. Ein Schema ist eine versionierte Spezifikation für zuverlässige Datenveröffentlichung, -nutzung oder -speicherung. Mit dem AWS Glue Schema Registry können Sie die durchgängige Datenqualität und Datenverwaltung in Ihren Streaming-Anwendungen verbessern. Weitere Informationen finden Sie unter [AWS Glue Schema Registry](#). Eine der Möglichkeiten, diese Integration einzurichten, ist die `GetRecords`-API von Kinesis Data Streams, die im AWS-Java SDK verfügbar ist.

Detaillierte Anweisungen zur Einrichtung der Integration von Kinesis Data Streams mit Schema Registry mithilfe der `GetRecords`-APIs Kinesis Data Streams finden Sie im Abschnitt „Interaktion mit Daten mithilfe der APIs für Kinesis Data Streams“ unter [Anwendungsfall: Integration von Amazon Kinesis Data Streams mit der AWS Glue Schema Registry](#).

## Verwalten von Verbrauchern für erweitertes Rundsenden mit dem AWS Management Console

Verbraucher, die ein erweitertes Rundsenden in Amazon Kinesis Data Streams verwenden, können Datensätze aus einem Datenstrom mit einem dedizierten Durchsatz von bis zu 2 MB

Daten pro Sekunde pro Shard empfangen. Weitere Informationen finden Sie unter [Entwickeln benutzerdefinierter Verbraucher mit dediziertem Durchsatz \(Erweitertes Rundsenden\)](#).

Sie können die AWS Management Console verwenden, um eine Liste aller Verbraucher zu verwenden, die für ein erweitertes Rundsenden bei einem bestimmten Stream registriert sind. Für jeden dieser Verbraucher können Sie Details wie ARN, Status, Erstellungsdatum und Überwachungsmetriken sehen.

Anzeige der Verbraucher, die registriert sind, um das erweiterte Rundsenden zu verwenden, ihres Status, ihres Erstellungsdatums ihrer Metriken in der Konsole

1. Melden Sie sich bei AWS Management Console an und öffnen Sie die Kinesis-Konsole unter <https://console.aws.amazon.com/kinesis>.
2. Klicken Sie im Navigationsbereich auf Data Streams (Daten-Streams).
3. Wählen Sie einen Kinesis-Datenstrom aus, um dessen Details anzuzeigen.
4. Wählen Sie auf der Detailseite für den Stream die Registerkarte Enhanced fan-out (Erweitertes Rundsenden).
5. Wählen Sie einen Verbraucher, um seinen Namen, den Status und das Datum der Registrierung anzuzeigen.

Einen Verbraucher abmelden

1. Öffnen Sie die Kinesis-Konsole unter <https://console.aws.amazon.com/kinesis>.
2. Klicken Sie im Navigationsbereich auf Data Streams (Daten-Streams).
3. Wählen Sie einen Kinesis-Datenstrom aus, um dessen Details anzuzeigen.
4. Wählen Sie auf der Detailseite für den Stream die Registerkarte Enhanced fan-out (Erweitertes Rundsenden).
5. Wählen Sie das Kontrollkästchen links neben dem Namen jedes Verbrauchers, den Sie abmelden möchten.
6. Wählen Sie Deregister consumer (Verbraucher abmelden).

## Migrieren von Verbrauchern von KCL 1.x zu KCL 2.x

In diesem Thema werden die Unterschiede zwischen den Versionen 1.x und 2.x der Kinesis Client Library (KCL) erläutert. Außerdem erfahren Sie, wie Sie Ihren Konsumenten von Version 1.x nach

Version 2.x der KCL migrieren. Nach der Migration des Clients werden Datensätze vom letzten Checkpoint-Speicherort verarbeitet.

Version 2.0 der KCL enthält die folgenden Schnittstellenänderungen:

### KCL-Schnittstellenänderungen

KCL 1.x-Schnittstelle	KCL 2.0-Schnittstelle
<code>com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor</code>	<code>software.amazon.kinesis.processor.ShardRecordProcessor</code>
<code>com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessorFactory</code>	<code>software.amazon.kinesis.processor.ShardRecordProcessorFactory</code>
<code>com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IShutdownNotificationAware</code>	Umgewandelt in <code>software.amazon.kinesis.processor.ShardRecordProcessor</code>

### Themen

- [Migrieren des Datensatzprozessors](#)
- [Migrieren der Datensatzprozessor-Factory](#)
- [Migrieren der Auftragnehmer](#)
- [Konfiguration des Amazon-Kinesis-Clients](#)
- [Entfernen der Leerlaufzeit](#)
- [Entfernen von Client-Konfigurationen](#)

## Migrieren des Datensatzprozessors

Das folgende Beispiel zeigt einen Datensatzprozessor, der für KCL 1.x implementiert wurde:

```
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();
    }
}

```

So migrieren Sie die Datensatzprozessorklasse

1. Ändern Sie die Schnittstellen von

`com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor`  
und

`com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IShutdownNotificationAware`  
folgendermaßen nach `software.amazon.kinesis.processor.ShardRecordProcessor`:

```

// import
com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor;
// import
com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IShutdownNotificationAware;
import software.amazon.kinesis.processor.ShardRecordProcessor;

// public class TestRecordProcessor implements IRecordProcessor,
// IShutdownNotificationAware {
public class TestRecordProcessor implements ShardRecordProcessor {

```

2. Aktualisieren Sie die `import`-Anweisungen für die Methoden `initialize` und `processRecords`.

```

// import com.amazonaws.services.kinesis.clientlibrary.types.InitializationInput;
import software.amazon.kinesis.lifecycle.events.InitializationInput;

//import com.amazonaws.services.kinesis.clientlibrary.types.ProcessRecordsInput;
import software.amazon.kinesis.lifecycle.events.ProcessRecordsInput;

```

3. Ersetzen Sie die Methode `shutdown` durch die folgenden neuen Methoden: `leaseLost`, `shardEnded` und `shutdownRequested`.

```

// @Override
// public void shutdownRequested(IRecordProcessorCheckpointter checkpointter) {
//     //
//     // This is moved to shardEnded(...)
//     //

```

```
//      try {
//          checkpointer.checkpoint();
//      } catch (ShutdownException | InvalidStateException e) {
//          //
//          // Swallow exception
//          //
//          e.printStackTrace();
//      }
//  }

@Override
public void leaseLost(LeaseLostInput leaseLostInput) {

}

@Override
public void shardEnded(ShardEndedInput shardEndedInput) {
    try {
        shardEndedInput.checkpointer().checkpoint();
    } catch (ShutdownException | InvalidStateException e) {
        //
        // Swallow the exception
        //
        e.printStackTrace();
    }
}

// @Override
// public void shutdownRequested(IRecordProcessorCheckpointer checkpointer) {
//     //
//     // This is moved to shutdownRequested(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();
        }
    }
}
```

Nachstehend finden Sie die aktualisierte Version der Datensatzprozessorklasse.

```
package com.amazonaws.kcl;

import software.amazon.kinesis.exceptions.InvalidStateException;
import software.amazon.kinesis.exceptions.ShutdownException;
import software.amazon.kinesis.lifecycle.events.InitializationInput;
import software.amazon.kinesis.lifecycle.events.LeaseLostInput;
import software.amazon.kinesis.lifecycle.events.ProcessRecordsInput;
import software.amazon.kinesis.lifecycle.events.ShardEndedInput;
import software.amazon.kinesis.lifecycle.events.ShutdownRequestedInput;
import software.amazon.kinesis.processor.ShardRecordProcessor;

public class TestRecordProcessor implements ShardRecordProcessor {
    @Override
    public void initialize(InitializationInput initializationInput) {

    }

    @Override
    public void processRecords(ProcessRecordsInput processRecordsInput) {

    }

    @Override
    public void leaseLost(LeaseLostInput leaseLostInput) {

    }

    @Override
    public void shardEnded(ShardEndedInput shardEndedInput) {
        try {
```



```
        shardEndedInput.checkpointer().checkpoint();
    } catch (ShutdownException | InvalidStateException e) {
        //
        // Swallow the exception
        //
        e.printStackTrace();
    }
}

@Override
public void shutdownRequested(ShutdownRequestedInput shutdownRequestedInput) {
    try {
        shutdownRequestedInput.checkpointer().checkpoint();
    } catch (ShutdownException | InvalidStateException e) {
        //
        // Swallow the exception
        //
        e.printStackTrace();
    }
}
}
```

## Migrieren der Datensatzprozessor-Factory

Die Datensatzprozessor-Factory ist für das Erstellen von Prozessoren verantwortlich, wenn eine Lease erworben wird. Nachfolgend sehen Sie ein Beispiel für eine KCL-1.x-Factory.

```
package com.amazonaws.kcl;

import com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor;
import
    com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessorFactory;

public class TestRecordProcessorFactory implements IRecordProcessorFactory {
    @Override
    public IRecordProcessor createProcessor() {
        return new TestRecordProcessor();
    }
}
```

## So migrieren Sie die Datensatzprozessor-Factory

1. Ändern Sie die implementierte Schnittstelle von `com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessorFactory` folgendermaßen nach `software.amazon.kinesis.processor.ShardRecordProcessorFactory`:

```
// import
com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor;
import software.amazon.kinesis.processor.ShardRecordProcessor;

// import
com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessorFactory;
import software.amazon.kinesis.processor.ShardRecordProcessorFactory;

// public class TestRecordProcessorFactory implements IRecordProcessorFactory {
public class TestRecordProcessorFactory implements ShardRecordProcessorFactory {
```

2. Ändern Sie die Rückgabesignatur für `createProcessor`.

```
// public IRecordProcessor createProcessor() {
public ShardRecordProcessor shardRecordProcessor() {
```

Es folgt ein Beispiel für die Verwendung der Datensatzprozessor-Factory in 2.0:

```
package com.amazonaws.kcl;

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

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

## Migrieren der Auftragnehmer

In Version 2.0 des KCL, einer neuen Klasse mit der Bezeichnung `Scheduler`, wird die `Worker`-Klasse ersetzt. Nachfolgend sehen Sie ein Beispiel für einen KCL-1.x-Auftragnehmer.

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

So migrieren Sie den Auftragnehmer

1. Ändern Sie die `import`-Anweisung für die `Worker`-Klasse, um Anweisungen für die Klassen `Scheduler` und `ConfigsBuilder` zu importieren.

```
// import com.amazonaws.services.kinesis.clientlibrary.lib.worker.Worker;
import software.amazon.kinesis.coordinator.Scheduler;
import software.amazon.kinesis.common.ConfigsBuilder;
```

2. Erstellen Sie `ConfigsBuilder` und `Scheduler` wie im folgenden Beispiel gezeigt.

Es wird empfohlen, dass Sie `KinesisClientUtil` zum Erstellen von `KinesisAsyncClient` und zum Konfigurieren von `maxConcurrency` in `KinesisAsyncClient` verwenden.

### Important

Für den Amazon Kinesis Client kann sich die Latenz möglicherweise signifikant erhöhen, sofern Sie `KinesisAsyncClient` nicht für einen `maxConcurrency`-Wert konfigurieren, der hoch genug ist, um alle Leases plus zusätzliche Verwendungen von `KinesisAsyncClient` zu ermöglichen.

```
import java.util.UUID;

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;
import software.amazon.awssdk.services.cloudwatch.CloudWatchAsyncClient;
```

```
import software.amazon.awssdk.services.kinesis.KinesisAsyncClient;
import software.amazon.kinesis.common.ConfigsBuilder;
import software.amazon.kinesis.common.KinesisClientUtil;
import software.amazon.kinesis.coordinator.Scheduler;

...

Region region = Region.AP_NORTHEAST_2;
KinesisAsyncClient kinesisClient =
    KinesisClientUtil.createKinesisAsyncClient(KinesisAsyncClient.builder().region(region));
DynamoDbAsyncClient dynamoClient =
    DynamoDbAsyncClient.builder().region(region).build();
CloudWatchAsyncClient cloudWatchClient =
    CloudWatchAsyncClient.builder().region(region).build();

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

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

## Konfiguration des Amazon-Kinesis-Clients

Mit Version 2.0 der Kinesis Client Library wurde die Konfiguration des Clients aus einer einzelnen Konfigurationsklasse (`KinesisClientLibConfiguration`) in sechs Konfigurationsklassen verlagert. Die folgende Tabelle beschreibt die Migration.

## Konfigurationsfelder und ihre neue Klassen

Originalfeld	Neue Konfigurationsklasse	Beschreibung
<code>applicationName</code>	<code>ConfigsBuilder</code>	Der Name für die KCL-Anwendung. Wird als Standard für <code>tableName</code> und <code>consumerName</code> verwendet.
<code>tableName</code>	<code>ConfigsBuilder</code>	Ermöglicht das Überschreiben des für die Lease-Tabelle von Amazon DynamoDB verwendeten Tabellennamens.
<code>streamName</code>	<code>ConfigsBuilder</code>	Der Name des Streams, dessen Datensätze diese Anwendung verarbeitet.
<code>kinesisEndpoint</code>	<code>ConfigsBuilder</code>	Diese Option wurde entfernt. Siehe "Entfernen von Client-Konfigurationen".
<code>dynamoDBEndpoint</code>	<code>ConfigsBuilder</code>	Diese Option wurde entfernt. Siehe "Entfernen von Client-Konfigurationen".
<code>initialPositionInStreamExtended</code>	<code>RetrievalConfig</code>	Der Speicherort im Shard, an dem die KCL mit der Ausführung der Anwendung durch Abrufen von Datensätzen beginnt.
<code>kinesisCredentialsProvider</code>	<code>ConfigsBuilder</code>	Diese Option wurde entfernt. Siehe "Entfernen von Client-Konfigurationen".
<code>dynamoDBCredentialsProvider</code>	<code>ConfigsBuilder</code>	Diese Option wurde entfernt. Siehe "Entfernen von Client-Konfigurationen".
<code>cloudWatchCredentialsProvider</code>	<code>ConfigsBuilder</code>	Diese Option wurde entfernt. Siehe "Entfernen von Client-Konfigurationen".

Originalfeld	Neue Konfigurationsklasse	Beschreibung
<code>failoverTimeMillis</code>	<code>LeaseManagementConfig</code>	Anzahl Millisekunden, nach deren Ablauf unterstellt werden kann, dass ein Lease-Eigentümer fehlgeschlagen ist.
<code>workerIdentifier</code>	<code>ConfigsBuilder</code>	Eine eindeutige Kennung, die die Instanziierung des Anwendungsprozessors repräsentiert. Dieser Wert muss eindeutig sein.
<code>shardSyncIntervalMillis</code>	<code>LeaseManagementConfig</code>	Die Zeit zwischen Shard-Synchronisierungsaufrufen.
<code>maxRecords</code>	<code>PollingConfig</code>	Ermöglicht das Einstellen der maximalen Anzahl an Datensätzen, die Kinesis zurückgibt.
<code>idleTimeBetweenReadsInMillis</code>	<code>CoordinatorConfig</code>	Diese Option wurde entfernt. Siehe "Entfernen der Leerlaufzeit".
<code>callProcessorRecordsEvenForEmptyRecordList</code>	<code>ProcessorConfig</code>	Wenn diese Option aktiviert ist, wird der Datensatzprozessor aufgerufen, auch wenn Kinesis keine Datensätze bereitgestellt hat.
<code>parentShardPollIntervalMillis</code>	<code>CoordinatorConfig</code>	Gibt an, wie oft ein Datensatzprozessor abfragen soll, ob der übergeordnete Shard abgeschlossen wurde.
<code>cleanupLeasesUponShardCompletion</code>	<code>LeaseManagementConfig</code>	Wenn diese Option aktiviert ist, werden Leases entfernt, sobald die untergeordneten Leases die Verarbeitung gestartet haben.

Originalfeld	Neue Konfigurationsklasse	Beschreibung
<code>ignoreUnexpectedChildShards</code>	<code>LeaseManagementConfig</code>	Wenn diese Option aktiviert ist, werden untergeordnete Shards, die einen offenen Shard aufweisen, ignoriert. Dies gilt hauptsächlich für DynamoDB Streams.
<code>kinesisClientConfig</code>	<code>ConfigsBuilder</code>	Diese Option wurde entfernt. Siehe "Entfernen von Client-Konfigurationen".
<code>dynamoDBClientConfig</code>	<code>ConfigsBuilder</code>	Diese Option wurde entfernt. Siehe "Entfernen von Client-Konfigurationen".
<code>cloudWatchClientConfig</code>	<code>ConfigsBuilder</code>	Diese Option wurde entfernt. Siehe "Entfernen von Client-Konfigurationen".
<code>taskBackoffTimeMillis</code>	<code>LifecycleConfig</code>	Die Wartezeit beim erneuten Versuchen von fehlgeschlagenen Aufgaben.
<code>metricsBufferTimeMillis</code>	<code>MetricsConfig</code>	Steuert die Veröffentlichung der CloudWatch-Metrik.
<code>metricsMaxQueueSize</code>	<code>MetricsConfig</code>	Steuert die Veröffentlichung der CloudWatch-Metrik.
<code>metricsLevel</code>	<code>MetricsConfig</code>	Steuert die Veröffentlichung der CloudWatch-Metrik.
<code>metricsEnabledDimensions</code>	<code>MetricsConfig</code>	Steuert die Veröffentlichung der CloudWatch-Metrik.
<code>validateSequenceNumberBeforeCheckpointing</code>	<code>CheckpointConfig</code>	Diese Option wurde entfernt. Siehe "Checkpoint Sequence Number Validation".

Originalfeld	Neue Konfigurationsklasse	Beschreibung
regionName	ConfigsBuilder	Diese Option wurde entfernt. Siehe "Entfernen von Client-Konfigurationen".
maxLeasesForWorker	LeaseManagementConfig	Die maximale Anzahl der Leases, die eine einzelne Instance der Anwendung akzeptieren sollte.
maxLeasesToStealAtOneTime	LeaseManagementConfig	Die maximale Anzahl der Leases, die eine Anwendung zu einem gegebenen Zeitpunkt zu stehlen versuchen sollte.
initialLeaseTableReadCapacity	LeaseManagementConfig	Die DynamoDB-Lese-IOPs, die verwendet werden, wenn die Kinesis Client Library eine neue DynamoDB-Lease-Tabelle erstellen muss.
initialLeaseTableWriteCapacity	LeaseManagementConfig	Die DynamoDB-Lese-IOPs, die verwendet werden, wenn die Kinesis Client Library eine neue DynamoDB-Lease-Tabelle erstellen muss.
initialPositionInStreamExtended	LeaseManagementConfig	Die ursprüngliche Position im Stream, an der die Anwendung starten soll. Dieser Wert wird nur im Rahmen der Lease-Erstellung verwendet.
skipShardSyncAtWorkerInitializationIfLeasesExist	CoordinatorConfig	Synchronisieren der Shard-Daten deaktivieren, wenn die Lease-Tabelle Leases enthält. TODO: KinesisEco-438
shardPrioritization	CoordinatorConfig	Welche Shard-Priorisierung verwendet werden soll.
shutdownGraceMillis	–	Diese Option wurde entfernt. Siehe "MultiLang Removals".



Originalfeld	Neue Konfigurationsklasse	Beschreibung
<code>timeoutInSeconds</code>	–	Diese Option wurde entfernt. Siehe "MultiLang Removals".
<code>retryGetRecordsInSeconds</code>	<code>PollingConfig</code>	Konfiguriert die Verzögerung zwischen <code>GetRecords</code> -Versuchen bei Fehlschlägen.
<code>maxGetRecordsThreadPool</code>	<code>PollingConfig</code>	Die Thread-Pool-Größe für <code>GetRecords</code> .
<code>maxLeaseRenewalThreads</code>	<code>LeaseManagementConfig</code>	Steuert die Größe des Lease-Renewer-Thread-Pools. Dieser Pool muss größer sein, wenn die Anwendung mehr Leases annehmen kann.
<code>recordsFetcherFactory</code>	<code>PollingConfig</code>	Ermöglicht es, die Voreinstellung zu ersetzen, die zum Erstellen von Fetchers verwendet wird, die von Streams abrufen.
<code>logWarningForTaskAfterMillis</code>	<code>LifecycleConfig</code>	Wartezeit, bevor eine Warnung protokolliert wird, wenn eine Aufgabe nicht abgeschlossen wurde.
<code>listShardsBackoffTimeInMillis</code>	<code>RetrievalConfig</code>	Die Anzahl der zwischen Aufrufen von <code>ListShards</code> abzuwartenden Millisekunden, wenn es zu Fehlern kommt.
<code>maxListShardsRetryAttempts</code>	<code>RetrievalConfig</code>	Die maximale Anzahl Wiederholungsversuche durch <code>ListShards</code> , bevor abgebrochen wird.

## Entfernen der Leerlaufzeit

In Version 1.x von KCL wurde `idleTimeBetweenReadsInMillis` für zwei Werte verwendet:

- Zeitraum zwischen Task-Versandprüfungen. Sie können diesen Zeitraum zwischen Tasks jetzt mit `CoordinatorConfig#shardConsumerDispatchPollIntervalMillis` konfigurieren.
- Zeit im Ruhemodus, wenn keine Datensätze von Kinesis Data Streams zurückgegeben wurden. In Version 2.0 werden Datensätze im Rahmen der verbesserten Rundsendung von den jeweiligen Abrufern im Push-Verfahren übermittelt. Aktivitäten auf dem Shard-Konsumenten treten nur auf, wenn eine Anforderung per Push ankommt.

## Entfernen von Client-Konfigurationen

In Version 2.0 erstellt KCL keine Clients mehr. Der Benutzer muss einen gültigen Client bereitstellen. Mit dieser Änderung wurden alle Konfigurationsparameter für die Client-Erstellung entfernt. Wenn Sie diese Parameter benötigen, können Sie sie in den Clients einstellen, bevor die Clients für `ConfigsBuilder` bereitgestellt werden.

Entferntes Field	Äquivalente Konfiguration
<code>kinesisEndpoint</code>	Konfigurieren Sie das SDK <code>KinesisAsyncClient</code> mit dem bevorzugten Endpunkt: <code>KinesisAsyncClient.builder().endpointOverride(URI.create("https://&lt;kinesis endpoint&gt;")).build()</code> .
<code>dynamoDBEndpoint</code>	Konfigurieren Sie das SDK <code>DynamoDbAsyncClient</code> mit dem bevorzugten Endpunkt: <code>DynamoDbAsyncClient.builder().endpointOverride(URI.create("https://&lt;dynamodb endpoint&gt;")).build()</code> .
<code>kinesisClientConfig</code>	Konfigurieren Sie das SDK <code>KinesisAsyncClient</code> mit der benötigten Konfiguration: <code>KinesisAsyncClient.builder().overrideConfiguration(&lt;your configuration&gt;).build()</code> .
<code>dynamoDBClientConfig</code>	Konfigurieren Sie das SDK <code>DynamoDbAsyncClient</code> mit der benötigten Konfiguration: <code>DynamoDbAsyncClient.builder().overrideConfiguration(&lt;your configuration&gt;).build()</code> .
<code>cloudWatchClientConfig</code>	Konfigurieren Sie das SDK <code>CloudWatchAsyncClient</code> mit der benötigten Konfiguration: <code>CloudWatchAsyncClient.builder().overrideConfiguration(&lt;your configuration&gt;).build()</code> .

Entferntes Field	Äquivalente Konfiguration
<code>regionName</code>	Konfigurieren Sie das SDK mit der bevorzugten Region. Dies ist für alle SDK-Clients gleich. Zum Beispiel <code>KinesisAsyncClient.builder().region(Region.US_WEST_2).build()</code> .

## Nutzung anderer AWS-Services zum Lesen von Daten aus Kinesis Data Streams

Im Folgenden finden Sie eine Liste anderer AWS-Services, die direkt in Kinesis Data Streams integriert werden können, um Kinesis Data Streams zu lesen:

Themen

- [Nutzung von Amazon EMR](#)
- [Amazon EventBridge Pipes verwenden](#)
- [Verwenden von AWS Glue](#)
- [Verwenden von Amazon Redshift](#)

### Nutzung von Amazon EMR

Amazon EMR-Cluster können Amazon Kinesis Kinesis-Streams direkt lesen und verarbeiten, indem sie vertraute Tools aus dem Hadoop-Ökosystem wie Hive, Pig MapReduce, die Hadoop Streaming API und Cascading verwenden. Außerdem können Sie Echtzeitdaten aus Amazon Kinesis mit vorhandenen Daten auf Amazon S3, Amazon DynamoDB und HDFS in einem aktiven Cluster zusammenführen. Zur Nachbearbeitung können Sie die Daten aus Amazon EMR direkt in Amazon S3 oder DynamoDB laden.

Weitere Informationen finden Sie unter [Amazon Kinesis](#) im Handbuch zu Amazon-EMR-Releases.

### Amazon EventBridge Pipes verwenden

Amazon EventBridge Pipes unterstützt Amazon Kinesis Data Streams als Quelle. Amazon EventBridge Pipes unterstützt Sie bei der Erstellung von point-to-point Integrationen zwischen Veranstaltern und Verbrauchern mit optionalen Transformations-, Filter- und Anreicherungsschritten.

Sie können EventBridge Pipes verwenden, um Datensätze in einem Kinesis Data Stream zu empfangen und diese Datensätze optional zu filtern oder zu verbessern, bevor Sie sie zur Verarbeitung an eines der verfügbaren Ziele senden, einschließlich Kinesis Data Streams.

Weitere Informationen finden Sie unter [Amazon Kinesis Stream as a source](#) im Amazon EventBridge Release Guide.

## Verwenden von AWS Glue

Mit AWS Glue-Streaming-ETL können Sie Streaming-Aufträge für Extract, Transform, Load (ETL) erstellen, die kontinuierlich ausgeführt werden und Daten aus Amazon Kinesis Data Streams konsumieren. Die Aufträge bereinigen und transformieren die Daten und laden die Ergebnisse dann in Amazon-S3-Data-Lakes oder JDBC-Datenspeicher.

Weitere Informationen finden Sie unter [Streaming-ETL-Aufträge in AWS Glue](#) im AWS Glue-Release-Handbuch.

## Verwenden von Amazon Redshift

Amazon Redshift unterstützt die Streaming-Erfassung von Amazon Kinesis Data Streams. Das Streaming-Erfassungs-Feature Amazon Redshift ermöglicht das Erfassen von Streaming-Daten mit geringer Latenz und hoher Geschwindigkeit aus Amazon Kinesis Data Streams in einer materialisierten Ansicht von Amazon Redshift. Dank der Streaming-Erfassung von Amazon Redshift ist es nicht mehr notwendig, Daten in Amazon S3 bereitzustellen, bevor sie in Amazon Redshift erfasst werden.

Weitere Informationen finden Sie unter [Streaming-Erfassung](#) im Handbuch zum Release von Amazon Redshift.

## Integrationen von Drittanbietern verwenden

Sie können Daten aus Datenströmen von Amazon Kinesis Data Streams mit einer der folgenden Drittanbieteroptionen lesen, die in Kinesis Data Streams integriert sind:

### Themen

- [Apache Flink](#)
- [Adobe Experience Platform](#)
- [Apache Druid](#)
- [Apache Spark](#)

- [Databricks](#)
- [Kafka Confluent Plattform](#)
- [Kinesumer](#)
- [Talend](#)

## Apache Flink

Apache Flink ist ein Framework und eine verteilte Verarbeitungs-Engine für statusbehaftete Berechnungen über unbegrenzte und begrenzte Datenströme. Weitere Informationen zur Nutzung von Kinesis Data Streams mit Apache Flink finden Sie unter [Amazon Kinesis Data Streams Connector](#).

## Adobe Experience Platform

Die Adobe Experience Platform ermöglicht es Unternehmen, Kundendaten aus jedem System zu zentralisieren und zu standardisieren. Anschließend werden Datenwissenschaft und Machine Learning angewendet, um das Design und die Bereitstellung umfassender, personalisierter Erlebnisse erheblich zu verbessern. Weitere Informationen zur Nutzung von Kinesis-Datenströmen mit der Adobe Experience Platform finden Sie unter [Amazon Kinesis Connector](#).

## Apache Druid

Druid ist eine hochleistungsfähige Echtzeit-Analysedatenbank, die Abfragen auf Streaming- und Batch-Daten in Sekundenschnelle und bei hoher Auslastung ermöglicht. Weitere Informationen zur Aufnahme von Kinesis-Datenströmen mit Apache Druid finden Sie unter [Amazon Kinesis-Erfassung](#).

## Apache Spark

Apache Spark ist eine einheitliche Analytics-Engine für die großflächige Datenverarbeitung. Sie bietet High-Level-APIs in Java, Scala, Python und R sowie eine optimierte Engine, die allgemeine Ausführungsdiagramme unterstützt. Sie können Apache Spark verwenden, um Stream-Verarbeitungsanwendungen zu erstellen, die die Daten in Ihren Kinesis-Datenströmen verbrauchen.

Um Kinesis-Datenströme mit strukturiertem Streaming von Apache Spark zu verwenden, verwenden Sie den Amazon Kinesis-Data-Streams-[Konnektor](#). Dieser Konnektor unterstützt den Verbrauch mit Enhanced Fan-Out, der Ihrer Anwendung einen dedizierten Lesedurchsatz von bis zu 2 MB Daten pro Sekunde pro Shard bietet. Weitere Informationen finden Sie unter [Entwickeln benutzerdefinierter Verbraucher mit dediziertem Durchsatz \(Erweitertes Rundsenden\)](#).

Informationen zur Nutzung von Kinesis-Datenströmen mit Spark Streaming finden Sie unter [Spark Streaming + Kinesis Integration](#).

## Databricks

Databricks ist eine Cloud-basierte Plattform, die eine kollaborative Umgebung für Datentechnik, Datenwissenschaft und Machine Learning bietet. Weitere Informationen zur Nutzung von Kinesis-Datenströmen mit Databricks finden Sie unter [Verbindung zu Amazon Kinesis herstellen](#).

## Kafka Confluent Plattform

Confluent Plattform basiert auf Kafka und bietet zusätzliche Features und Funktionen, mit denen Unternehmen Daten-Pipelines und Streaming-Anwendungen in Echtzeit aufbauen und verwalten können. Weitere Informationen zur Nutzung von Kinesis-Datenströmen mit der Confluent-Plattform finden Sie unter [Amazon Kinesis Source Connector für Confluent Plattform](#).

## Kinesumer

Kinesumer ist ein Go-Client, der einen clientseitigen Client für verteilte Verbrauchergruppen für Kinesis-Datenströme implementiert. Weitere Informationen finden Sie im [Kinesumer-GitHub-Repository](#).

## Talend

Talend ist eine Datenintegrations- und Verwaltungssoftware, die es Benutzern ermöglicht, Daten aus verschiedenen Quellen auf skalierbare und effiziente Weise zu sammeln, zu transformieren und miteinander zu verbinden. Weitere Informationen zur Nutzung von Kinesis-Datenströmen mit Talend finden Sie unter [Verbinden von Talend mit einem Amazon Kinesis-Stream](#).

## Problembehandlung bei Verbrauchern von Kinesis Data Streams

In den folgenden Abschnitten finden Sie Lösungen für einige Probleme, die möglicherweise bei der Arbeit mit Konsumenten von Amazon Kinesis Data Streams auftreten.

- [Einige Datensätze in Kinesis Data Streams werden bei der Nutzung der Kinesis Client Library übersprungen](#)
- [Datensätze, die zum selben Shard gehören, werden gleichzeitig von verschiedenen Datensatzprozessoren verarbeitet](#)
- [Die Konsumentenanzahl liest Daten langsamer aus als erwartet](#)

- [GetRecords Gibt leeres Datensatz-Array zurück, auch wenn sich Daten im Stream befinden](#)
- [Der Shard-Iterator verliert unerwartet seine Gültigkeit](#)
- [Die Verarbeitung der Konsumentendatensätze hängt hinterher](#)
- [Fehler aufgrund fehlender KMS-Masterschlüsselberechtigung](#)
- [Häufig gestellte Probleme, Fragen und Ideen zur Problemlösung für Konsumenten](#)

## Einige Datensätze in Kinesis Data Streams werden bei der Nutzung der Kinesis Client Library übersprungen

Der häufigste Grund für übersprungene Datensätzen ist ein Ausnahmefehler, der von `processRecords` ausgelöst wird. Die Kinesis Client Library (KCL) verlässt sich bei der Behandlung von Ausnahmen, die während der Verarbeitung von Datensätzen auftreten, auf Ihrem `processRecords`-Code. Alle von `processRecords` ausgelösten Ausnahmen werden von der KCL aufgenommen. Um unbegrenzte Wiederholungen eines Fehlers zu vermeiden, sendet die KCL den Datensatzstapel, der während der Ausnahme verarbeitet wurde, nicht erneut. Die KCL ruft stattdessen `processRecords` für den nächsten Datensatzstapel auf, ohne den Datensatzprozessor neu zu starten. Dies führt dazu, dass Konsumenten Anwendungen übersprungene Datensätze erkennen. Zur Vermeidung übersprungener Datensätze müssen Sie alle Ausnahmen innerhalb von `processRecords` entsprechend behandeln.

## Datensätze, die zum selben Shard gehören, werden gleichzeitig von verschiedenen Datensatzprozessoren verarbeitet

Ein Shard hat für jede ausgeführte Anwendung der Kinesis Client Library (KCL) nur einen Besitzer. Allerdings kann es vorkommen, dass mehrere Datensatzprozessoren vorübergehend denselben Shard verarbeiten. Bei einer Worker-Instance mit unterbrochener Netzwerkverbindung geht die KCL davon aus, dass der nicht erreichbare Worker die Datensätze nicht mehr verarbeitet, nachdem die Failover-Zeit abgelaufen ist und veranlasst andere Worker-Instances zur Übernahme der Verarbeitung. Für eine kurze Zeit werden dann möglicherweise Daten aus demselben Shard von neuen Datensatzprozessoren und Datensatzprozessoren des nicht erreichbaren Workers verarbeitet.

Legen Sie deshalb eine für Ihre Anwendung angemessene Failover-Zeit fest. Für Anwendungen mit geringer Latenz entspricht der Standardwert von 10 Sekunden möglicherweise dem Zeitraum, den Sie maximal bereit sind zu warten. In den Fällen jedoch, in denen Sie Verbindungsprobleme erwarten, beispielsweise bei Aufrufen über verschiedene geografische Bereiche hinweg, in denen häufiger mit Verbindungsproblemen zu rechnen ist, ist die Zeitspanne möglicherweise zu gering.

Ihre Anwendung sollte diesem Szenario abhelfen können, insbesondere, weil die Netzwerkanbindung in der Regel für den zuvor nicht erreichbaren Worker wiederhergestellt wird. Wenn die Shards eines Datensatzprozessors von einem anderen Datensatzprozessor übernommen werden, müssen folgende Fälle für ein ordnungsgemäßes Herunterfahren korrekt behandelt werden:

1. Nach Abschluss des aktuellen Aufrufs von `processRecords` ruft die KCL die Methode für das Herunterfahren auf dem Datensatzprozessor mit Grund „ZOMBIE“ auf. Von Ihren Datensatzprozessoren wird erwartet, dass alle Ressourcen entsprechend bereinigt werden und die Datensatzprozessoren anschließend beendet werden.
2. Wenn Sie versuchen, ein Checkpointing mit einem „Zombie“-Worker durchzuführen, löst die KCL `ShutdownException` aus. Nach Empfang der Ausnahme ist die aktuelle Methode durch Ihren Code sauber zu beenden.

Weitere Informationen finden Sie unter [Umgang mit doppelten Datensätzen](#).

## Die Konsumentenanzahl liest Daten langsamer aus als erwartet

Die häufigsten Gründe für einen langsameren Lesedurchsatz sind:

1. Bei mehreren Konsumentenanzahlungen übersteigt die Anzahl der Lesevorgänge die pro Shard festgelegten Limits. Weitere Informationen finden Sie unter [Kontingente und Einschränkungen](#). Erhöhen Sie in diesem Fall die Anzahl der Shards im Kinesis-Datenstrom.
2. Das [Limit](#) für die maximale Anzahl von `GetRecords` pro Aufruf ist möglicherweise zu niedrig festgelegt. Wenn Sie die KCL verwenden, haben Sie möglicherweise einen niedrigen Wert bei der `maxRecords`-Eigenschaft des Workers angegeben. Grundsätzlich empfehlen wir die Verwendung der Systemvoreinstellungen für diese Eigenschaft.
3. Die Logik in Ihrem `processRecords`-Aufruf kann aus verschiedenen Gründen länger brauchen als erwartet. Möglicherweise erfordert Sie eine intensive Beteiligung der CPU, blockiert Ein- und Ausgaben oder es kommt zu Engpässen (Bottlenecks) bei der Synchronisierung. Führen Sie leere Datensatzprozessoren aus und vergleichen Sie den Lesedurchsatz, um festzustellen, ob dies der Fall ist. Weitere Informationen dazu, wie Sie eingehende Daten zeitgerecht verarbeiten können, finden Sie unter [Resharding, Skalierung und Parallelverarbeitung](#).

Wenn Sie nur eine Konsumentenanzahl haben, ist es immer möglich, mindestens zwei Mal schneller als die PUT-Rate zu lesen. Das liegt daran, dass Sie bis zu 1 000 Datensätze pro Sekunde für Schreibvorgänge bis zu einem maximalen Datenschreibvolumen von 1 MB pro



Sekunde (einschließlich Partitionsschlüsseln) schreiben können. Jeder offene Shard kann bis zu 5 Lesetransaktionen pro Sekunde unterstützen, bis zu einer maximalen Gesamtdatenleserate von 2 MB pro Sekunde. Beachten Sie, dass jeder Lesevorgang (GetRecords-Aufruf) einen Stapel an Datensätzen erhält. Die Menge der Daten, die von GetRecords zurückgegeben werden, hängt von der Shard-Auslastung ab. Die maximale Größe der Daten, die GetRecords zurückgeben kann, ist 10 MB. Wenn ein Aufruf das Limit zurückgibt, lösen nachfolgende Aufrufe innerhalb der nächsten 5 Sekunden `ProvisionedThroughputExceededException` aus.

## GetRecords Gibt leeres Datensatz-Array zurück, auch wenn sich Daten im Stream befinden

Beim Verbrauchen oder Empfangen von Datensätzen kommt ein PULL-Modell zum Einsatz. Es wird erwartet, dass Entwickler [GetRecords](#) in einer kontinuierlichen Schleife ohne Back-offs aufrufen. Jeder Aufruf von GetRecords gibt auch einen `ShardIterator`-Wert zurück, der in der nächsten Iteration der Schleife verwendet wird.

Die GetRecords-Operation blockiert nicht. Stattdessen erfolgt sofort eine Rückgabe, entweder mit relevanten Datensätzen oder mit einem leeren `Records-Element`. Ein leeres `Records-Element` wird unter zwei Bedingungen zurückgegeben:

1. Im Shard sind derzeit keine weiteren Daten vorhanden.
2. Es gibt keine Daten in der Nähe des Shard-Inhalts, auf den `ShardIterator` zeigt.

Die letzte Bedingung ist subtil, aber eine notwendige Design-Komponente, durch die unbegrenzte Suchzeiten (Latenzen) beim Abrufen von Datensätzen vermieden werden. Daher sollte die Anwendung, die Daten aus dem Stream verbraucht, eine Schleife durchlaufen, GetRecords aufrufen und leere Datensätze verarbeiten.

In einer Produktionsumgebung sollte die Endlosschleife nur verlassen werden, wenn der Wert von `NextShardIterator` NULL ist. Wenn `NextShardIterator` NULL ist, bedeutet dies, dass der aktuelle Shard geschlossen wurde und der Wert von `ShardIterator` ansonsten auf eine Position hinter dem letzten Datensatz verweisen würde. Wenn die datenverbrauchende Anwendung `SplitShard` oder `MergeShards` nicht aufruft, bleibt der Shard offen und die Aufrufe von GetRecords geben niemals einen `NextShardIterator`-Wert von NULL zurück.

Wenn Sie die Kinesis Client Library (KCL) nutzen, wird das obige Verbrauchsmuster für Sie abstrahiert. Dies umfasst die automatische Handhabung einer Gruppe von Shards, die sich

dynamisch ändert. Mit der KCL übermittelt der Entwickler nur die Logik für die Verarbeitung eingehender Datensätze. Dies ist möglich, da die Bibliothek `GetRecords` kontinuierlich für Sie aufruft.

## Der Shard-Iterator verliert unerwartet seine Gültigkeit

Bei jeder `GetRecords`-Anforderung (wie `NextShardIterator`) wird ein neuer Shard-Iterator zurückgegeben, den Sie bei der nächsten `GetRecords`-Anforderung (wie `ShardIterator`) verwenden. In der Regel läuft dieser Shard-Iterator nicht vor der Verwendung ab. Es kann jedoch sein, dass Shard-Iteratoren ihre Gültigkeit verlieren, weil Sie `GetRecords` länger als 5 Minuten nicht aufgerufen oder die Konsumentenanzahl neu gestartet haben.

Wenn der Shard-Iterator abläuft, ehe Sie ihn verwenden können, ist dies möglicherweise ein Hinweis darauf, dass die von Kinesis genutzte DynamoDB-Tabelle nicht genügend Kapazität hat, um die Lease-Daten zu speichern. Diese Situation tritt häufiger auf, wenn Sie viele Shards haben. Beheben Sie das Problem, indem Sie die Schreibkapazität der Shard-Tabelle erhöhen. Weitere Informationen finden Sie unter [Mithilfe einer Leasetabelle können Sie die von der KCL-Konsumentenanzahl verarbeiteten Shards verfolgen](#).

## Die Verarbeitung der Konsumentendatensätze hängt hinterher

Bei den meisten Anwendungsfällen lesen die Konsumentenanzahl die neuesten Daten aus dem Stream. Unter bestimmten Umständen hängt das Lesen des Konsumenten hinterher. Dieses Verhalten möglicherweise nicht erwünscht. Sehen Sie sich die häufigsten Gründe für ein zu langsames Lesen des Datenkonsumenten an, wenn Sie ermittelt haben, wie weit die Lesevorgänge des Konsumenten hinterherhängen.

Beginnen Sie mit der `GetRecords.IteratorAgeMilliseconds`-Metrik. Diese verfolgt die Lesezeitpunkt aller Shards und Konsumenten im Stream. Beachten Sie: Wenn das Alter eines Iterators 50 % des Aufbewahrungszeitraums (standardmäßig 24 Stunden, anpassbar auf bis zu 365 Tage) überschreitet, besteht die Gefahr des Datenverlusts durch Datensatzablauf. Als schnelle Übergangslösung eignet sich eine Verlängerung des Aufbewahrungszeitraums. Auf diese Weise wird der Verlust von wichtigen Daten gestoppt, während Sie sich der weiteren Fehlerbehebung widmen. Weitere Informationen finden Sie unter [Überwachung des Services von Amazon Kinesis Data Streams für Amazon CloudWatch](#). Bestimmen Sie als Nächstes anhand einer benutzerdefinierten CloudWatch Metrik, die von der Kinesis Client Library (KCL) ausgegeben wird, wie weit Ihre Verbraucheranzahl hinter jedem Shard zurückliegt `MillisBehindLatest`. Weitere Informationen finden Sie unter [Überwachen der Kinesis Client Library mit Amazon CloudWatch](#).

Im Folgenden sind die häufigsten Gründe für ein Hinterherhängen von Konsumenten aufgeführt:

- Plötzliche Anstiege auf `GetRecords.IteratorAgeMilliseconds` oder `MillisBehindLatest` weisen in der Regel auf ein vorübergehendes Problem hin, beispielsweise Fehler bei einer API-Operation in einer nachgelagerten Anwendung. Prüfen Sie bei solch plötzlichen Anstiegen, ob eine der Metriken durchgängig dieses Verhalten aufweist.
- Ein allmählicher Anstieg dieser Metriken deutet darauf hin, dass ein Konsument Datensätze nicht schnell genug verarbeitet. Die häufigsten Ursachen für ein solches Verhalten sind unzureichende physische Ressourcen oder eine Datensatzverarbeitungslogik, die nicht für einen Anstieg des Stream-Durchsatzes skaliert wurde. Sie können dieses Verhalten überprüfen, indem Sie sich die anderen CloudWatch benutzerdefinierten Metriken ansehen, die die KCL im Zusammenhang mit der `processTask` Operation ausgibt, einschließlich `RecordProcessor.processRecords.TimeSuccess`, und `RecordsProcessed`.
- Wenn Sie eine Erhöhung in der `processRecords.Time`-Metrik feststellen, die mit einem Anstieg des Durchsatzes korreliert, sollten Sie Ihre Datensatzverarbeitungslogik analysieren, um herauszufinden, warum sich diese nicht an den erhöhten Durchsatz anpasst.
- Wenn Sie eine Erhöhung der `processRecords.Time`-Werte feststellen, die nicht in Zusammenhang mit einem erhöhten Durchsatz steht, prüfen Sie, ob im kritischen Pfad blockierende Aufrufe durchgeführt werden. Diese sind häufig der Grund für eine verlangsamte Datensatzverarbeitung. Alternativ können Sie auch die Parallelität durch eine Erhöhung der Anzahl der Shards erhöhen. Bestätigen Sie schließlich, dass Sie während der Zeiten mit höchster Auslastung über eine entsprechende Menge an physischen Ressourcen (Arbeitsspeicher, CPU-Nutzung usw.) auf den zugrunde liegenden Verarbeitungsknoten verfügen.

## Fehler aufgrund fehlender KMS-Masterschlüsselberechtigung

Dieser Fehler tritt auf, wenn eine Konsumentenapplication ohne Berechtigungen für den KMS-Masterschlüssel Daten aus einem verschlüsselten Stream ausliest. Informationen zum Zuweisen von Berechtigungen zu einer Anwendung für den Zugriff auf einen KMS-Schlüssel finden Sie unter [Verwenden von Schlüsselrichtlinien in AWS KMS](#) und [Verwenden von IAM-Richtlinien mit AWS KMS](#).

## Häufig gestellte Probleme, Fragen und Ideen zur Problemlösung für Konsumenten

- [Warum kann der Kinesis-Data-Streams-Trigger meine Lambda-Funktion nicht aufrufen?](#)
- [Wie erkenne und behebe `ReadProvisionedThroughputExceeded` ich Ausnahmen in Kinesis Data Streams?](#)

- [Warum habe ich Probleme mit hoher Latenz bei Kinesis Data Streams?](#)
- [Warum gibt mein Kinesis-Datenstrom einen 500 Internal Server Error zurück?](#)
- [Wie behebe ich Fehler bei einer blockierten oder hängengebliebenen KCL-Anwendung für Kinesis Data Streams?](#)
- [Kann ich verschiedene Anwendungen der Amazon Kinesis Client Library mit der gleichen Amazon-DynamoDB-Tabelle verwenden?](#)

## Fortgeschrittene Themen für Amazon Kinesis Data Streams-Verbraucher

Erfahren Sie, wie Sie Ihren Verbraucher von Amazon Kinesis Data Streams optimieren können.

### Inhalt

- [Verarbeitung mit geringer Latenz](#)
- [Verwenden von AWS Lambda mit der Kinesis-Producer-Bibliothek](#)
- [Resharding, Skalierung und Parallelverarbeitung](#)
- [Umgang mit doppelten Datensätzen](#)
- [Umgang mit Startup, Herunterfahren und Drosselung](#)

## Verarbeitung mit geringer Latenz

Die Verbreitungsverzögerung ist definiert als eine Ende-zu-Ende-Latenz und umfasst den Zeitraum zwischen dem Schreiben eines Datensatzes in den Stream und dem Auslesen des Datensatzes von einer Konsumenten-anwendung aus dem Stream. Diese Verzögerung variiert und hängt von vielen Faktoren ab, im Wesentlichen aber vom Abrufintervall der Konsumenten-anwendungen.

Für die meisten Anwendungen empfehlen wir, jeden Shard pro Anwendung einmal pro Sekunde abzurufen. So können mehrere Konsumenten-anwendungen einen Stream gleichzeitig unabhängig voneinander verarbeiten, ohne an die Limits von 5 `GetRecords`-Aufrufen pro Sekunde von Amazon Kinesis Data Streams zu stoßen. Darüber hinaus ist die Verarbeitung größerer Datenstapel in Bezug auf Netzwerk- und andere nachgelagerte Latenzzeiten effizienter.

Die Standardeinstellungen der KCL entsprechend der bewährten Einstellung von einem Abruf pro Sekunde. Dieser Standard führt zu einer durchschnittlichen Verbreitungsverzögerung von weniger als 1 Sekunde.

Die Datensätze von Kinesis Data Streams können sofort nach dem Schreiben gelesen werden. In Fällen, in denen Daten sofort nach Bereitstellung verarbeitet werden müssen, ist dies ein wichtiger Aspekt. Sie können die Verbreitungsverzögerung erheblich verringern, indem Sie die Standardeinstellungen der KCL überschreiben, um häufiger Datensätze abzurufen. Sehen Sie sich dazu die folgenden Beispiele an.

Java KCL-Konfigurationscode:

```
kinesisClientLibConfiguration = new
    KinesisClientLibConfiguration(applicationName,
        streamName,
        credentialsProvider,

workerId).withInitialPositionInStream(initialPositionInStream).withIdleTimeBetweenReadsInMilli
```

Einstellung der Eigenschaftendatei für Python- und Ruby-KCL:

```
idleTimeBetweenReadsInMillis = 250
```

#### Note

Da für Kinesis Data Streams ein Limit von 5 `GetRecords`-Aufrufen pro Sekunde pro Shard gilt, kann das Festlegen der `idleTimeBetweenReadsInMillis`-Eigenschaft auf einen Wert von unter 200 ms dazu führen, dass Ihre Anwendung die `ProvisionedThroughputExceededException`-Ausnahme erkennt. Zu viele dieser Ausnahmen können zu exponentiellen Backoffs führen und bedeutsame unerwartete Latenzzeiten bei der Verarbeitung nach sich ziehen. Wenn Sie diese Eigenschaft auf 200 ms oder etwas mehr setzen und mehr als eine Verarbeitungsanwendung haben, kommt es zu einer ähnlichen Drosselung.

## Verwenden von AWS Lambda mit der Kinesis-Producer-Bibliothek

Die [Kinesis Producer Library](#) (KPL) aggregiert kleine vom Benutzer formatierte Datensätze in größere Datensätze von bis zu 1 MB, um den Durchsatz von Amazon Kinesis Data Streams zu optimieren. Die KCL für Java unterstützt eine Disaggregation dieser Datensätze, Sie müssen aber ein spezielles Modul für die Disaggregation nutzen, wenn Sie die AWS Lambda als Konsument für Ihre Streams nutzen. Sie erhalten den notwendigen Projektcode sowie erforderliche Anleitungen von GitHub

unter [Kinesis Producer Library Deaggregation Modules for AWS Lambda](#). Mit den Komponenten dieses Projekts können Sie serialisierte Daten der KPL in AWS Lambda, Java, Node.js und Python nutzen. Diese Komponenten können auch als Teil einer [mehrsprachigen KCL-Anwendung](#) verwendet werden.

## Resharding, Skalierung und Parallelverarbeitung

Das Resharding ermöglicht Ihnen das Erhöhen oder Verringern der Anzahl der Shards in einem Stream zur Anpassung an die Datenrate. Das Resharding wird üblicherweise von einer administrativen Anwendung durchgeführt, die Datenverwaltungsmetriken zu Shards überwacht. Obwohl die KCL selbst keine Resharding-Vorgänge auslöst, kann sie sich an die geänderte Anzahl von Shards anpassen, die sich aus dem Resharding ergibt.

Wie unter [Mithilfe einer Leasetabelle können Sie die von der KCL-Konsumentenanzahl verfolgte Shards verfolgen](#) erwähnt, verfolgt die KCL die Shards im Stream mit einer Amazon-DynamoDB-Tabelle. Wenn aufgrund einer Resharding-Operation neue Shards erstellt werden, werden diese von der KCL erkannt, die dann neue Zeilen in die Tabelle einfügt. Die Auftragnehmer entdecken die neuen Shards automatisch und erstellen Prozessoren, um die Daten dieser neuen Shards zu verwalten. Die KCL verteilt die Shards zudem auf alle verfügbaren Auftragnehmer und Datensatzprozessoren im Stream.

Die KCL stellt sicher, dass alle Daten, die vor dem Resharding bereits in einem Shard vorhanden waren, als erstes verarbeitet werden. Nachdem diese Daten verarbeitet wurden, werden die Daten der neuen Shards an die Datensatzprozessoren gesendet. So behält die KCL die Reihenfolge bei, in der die Datensätze für einen bestimmten Partitionsschlüssel zum Stream hinzugefügt wurden.

### Beispiel: Resharding, Skalierung und Parallelverarbeitung

Das folgende Beispiel veranschaulicht, wie Sie die KCL bei der Skalierung und beim Resharding unterstützt:

- Angenommen, Ihre Anwendung wird auf einer EC2 Instance ausgeführt und verarbeitet einen Kinesis-Datenstrom mit vier Shards. Diese eine Instance verfügt über einen KCL-Auftragnehmer und vier Datensatzprozessoren (ein Datensatzprozessor für jeden Shard). Diese vier Datensatzprozessoren werden parallel innerhalb desselben Prozesses ausgeführt.
- Wenn Sie dann die Anwendung für die Verwendung einer weiteren Instance skalieren, verarbeiten zwei Instances einen Stream mit vier Shards. Wenn der KCL-Auftragnehmer auf der zweiten Instance gestartet wird, führt er einen Lastausgleich mit der ersten Instance durch, sodass nun jede Instance zwei Shards verarbeitet.

- Dann beschließen Sie, Ihre vier Shards auf fünf Shards aufzuteilen. Die KCL koordiniert erneut die Verarbeitung auf den Instances: Eine Instance verarbeitet drei Shards und die andere zwei. Eine ähnliche Koordinierung erfolgt, wenn Sie Shards zusammenführen.

In der Regel sollten Sie bei Verwendung der KCL sicherstellen, dass die Anzahl der Instances nicht die Anzahl der Shards übersteigt (außer für Standby-Zwecke im Falle eines Ausfalls). Jeder Shard wird exakt von einem KCL-Auftragnehmer verarbeitet und verfügt über genau einen entsprechenden Datensatzprozessor, deshalb benötigen Sie für die Verarbeitung eines Shards niemals mehrere Instances. Ein Auftragnehmer kann allerdings eine beliebige Anzahl von Shards verarbeiten, deshalb ist es kein Problem, wenn Sie mehr Shards als Instances haben.

Testen Sie eine Kombination der folgenden Ansätze, wenn Sie die Verarbeitung in Ihrer Anwendung optimieren möchten:

- Erhöhung der Instance-Größe (da alle Datensatzprozessoren innerhalb eines Prozesses parallel ausgeführt werden)
- Erhöhung der Anzahl der Instances bis zur maximalen Anzahl offener Shards (da Shards unabhängig verarbeitet werden können)
- Erhöhung der Anzahl der Shards (fördert die Parallelität)

Beachten Sie, dass Sie Auto Scaling für die automatische Skalierung Ihrer Instances basierend auf entsprechenden Metriken nutzen können. Weitere Informationen hierzu finden Sie unter [Amazon EC2 Auto Scaling-Benutzerhandbuch](#).

Wenn durch das Resharding die Anzahl der Shards im Stream erhöht wird, führt eine entsprechende Erhöhung der Datensatzprozessoren zu einer höheren Last der EC2 Instances, auf denen sie gehostet werden. Wenn die Instances Teil einer Auto Scaling-Gruppe sind und die Last entsprechend steigt, fügt die Auto Scaling-Gruppe weitere Instances hinzu, damit die höhere Last ausgeglichen wird. Konfigurieren Sie Ihre Instances so, dass Ihre Anwendung Amazon Kinesis Data Streams beim Startup gestartet wird. So werden zusätzliche Auftragnehmer und Datensatzprozessoren in der neuen Instance sofort aktiviert.

Weitere Informationen zum Resharding finden Sie unter [Resharding eines Streams](#).

## Umgang mit doppelten Datensätzen

Es gibt zwei primäre Gründe, warum Datensätze mehr als einmal an Ihre Anwendung von Amazon Kinesis Data Streams übermittelt werden: Wiederholungsversuche des Produzenten und

Wiederholungsversuche des Konsumenten. Ihre Anwendung muss in der Lage sein, einzelne Datensätze mehrere Male angemessen zu verarbeiten.

## Wiederholungsversuche des Produzenten

Stellen Sie sich einen Produzenten vor, der nach dem Aufruf von `PutRecord` aber noch vor dem Erhalt einer Bestätigung von Amazon Kinesis Data Streams einen netzwerkbedingten Timeout erfährt. In diesem Fall ist es fraglich, ob der Datensatz auch an Kinesis Data Streams übermittelt wurde. Davon ausgehend, dass jeder Datensatz für die Anwendung wichtig ist, ist der Produzent so aufgebaut, dass er den Aufruf mit denselben Daten wiederholt. Wenn beide `PutRecord`-Aufrufe mit denselben Daten erfolgreich an Kinesis Data Streams übergeben werden, sind anschließend zwei Datensätze von Kinesis Data Streams vorhanden. Obwohl die beiden Datensätze identische Daten enthalten, haben Sie trotzdem eindeutige Sequenznummern. Anwendungen, die Garantien benötigen, sollten einen Primärschlüssel in den Datensatz einbetten, sodass doppelte Datensätze später bei der Verarbeitung entfernt werden. Beachten Sie, dass die Anzahl der Duplikate aufgrund von Wiederholungen durch den Produzenten verglichen mit der Anzahl der Duplikate aufgrund von Wiederholungen durch den Konsumenten in der Regel nicht sehr hoch ist.

### Note

Wenn Sie das AWS-SDK `PutRecord` verwenden, wird in der standardmäßigen [Konfiguration ein fehlgeschlagener `PutRecord`-Aufruf bis zu drei Mal wiederholt.](#)

## Wiederholungsversuche des Konsumenten

Wiederholungen durch den Konsumenten (Datenverarbeitungsanwendungen) kommen vor, wenn Datenprozessoren neu gestartet werden. Datensatzprozessoren für denselben Shard werden in den folgenden Fällen neu gestartet:

1. Ein Auftragnehmer wird unerwartet beendet
2. Es werden Auftragnehmer-Instances hinzugefügt oder entfernt
3. Es werden Shards zusammengeführt oder getrennt
4. Die Anwendung wird bereitgestellt

In all diesen Fällen wird die Zuordnung zwischen Shards, Auftragnehmer und Datensatzprozessor für eine Lastverteilung kontinuierlich aktualisiert. Shard-Prozessoren, die in andere Instances migriert



wurden, starten das Verarbeiten von Datensätzen am letzten Prüfpunkt neu. Dies führt zu einer doppelten Datensatzverarbeitung, wie in dem folgenden Beispiel dargestellt. Weitere Informationen zur Lastverteilung finden Sie unter [Resharding, Skalierung und Parallelverarbeitung](#).

Beispiel: Wiederholungsversuche des Konsumenten führen zu erneut zugestellten Datensätzen

In diesem Beispiel haben Sie eine Anwendung, die kontinuierlich Datensätze aus einem Stream liest, Datensätze in eine lokale Datei aggregiert und die Datei in Amazon S3 hochlädt. Der Einfachheit halber gehen wir davon aus, dass nur 1 Shard und 1 Auftragnehmer den Shard verarbeiten. Beachten Sie die folgende Beispielabfolge von Ereignissen und gehen Sie davon aus, dass der letzte Prüfpunkt bei Datensatznummer 10.000 gesetzt wurde.

1. Ein Auftragnehmer liest den nächsten Datensatzstapel aus dem Shard (Datensätze 10.001 bis 20.000).
2. Anschließend übermittelt der Auftragnehmer diesen an den zugehörigen Datensatzprozessor.
3. Der Datensatzprozessor aggregiert die Daten, erstellt eine Amazon-S3-Datei und lädt die Datei erfolgreich in Amazon S3 hoch.
4. Der Auftragnehmer wird unerwartet beendet, noch ehe er einen neuen Prüfpunkt setzen kann.
5. Anwendung, Auftragnehmer und Datensatzprozessor starten neu.
6. Der Auftragnehmer beginnt beim letzten erfolgreich gesetzten Prüfpunkt mit dem Lesen, hier Datensatznummer 10.001.

Somit werden die Datensätze 10.001 bis 20.000 mehr als einmal verwendet.

Resistenz gegen Wiederholungsversuche durch den Konsumenten

Auch wenn Datensätze mehrmals verarbeitet werden, kann die Anwendung vorgeben, dass Datensätze nur einmal verarbeitet wurden (idempotente Verarbeitung). Die Lösungen für dieses Problem variieren in Komplexität und Genauigkeit. Wenn das Ziel der resultierenden Daten doppelte Datensätze ordnungsgemäß handhaben kann, sollten Sie sich auf diese Anwendung verlassen, um eine idempotente Verarbeitung zu erreichen. Mit [Opensearch](#) können Sie zur Vermeidung einer doppelten Verarbeitung beispielsweise eine Kombination aus Versioning und eindeutigen IDs verwenden.

Die Beispielanwendung aus dem vorherigen Abschnitt liest kontinuierlich Datensätze aus dem Stream, aggregiert diese in eine lokale Datei und lädt diese Datei in Amazon S3 hoch. Wie gezeigt, werden die Datensätze 10 001 bis 20 000 mehr als einmal verarbeitet, sodass es mehrere Amazon-

S3-Dateien mit identischen Daten gibt. Eine Möglichkeit, Duplikate in diesem Beispiel zu vermeiden, besteht darin, sicherzustellen, dass bei Schritt 3 das folgende Schema verwendet wird:

1. Der Datensatzprozessor verwendet eine feste Anzahl von Datensätzen pro Amazon-S3-Datei, beispielsweise 5 000.
2. Der Dateiname nutzt folgendes Schema: Amazon-S3-Präfix, Shard-ID und `First-Sequence-Num`. In diesem Fall könnte dies `sample-shard000001-10001` sein.
3. Nach dem Hochladen der Amazon-S3-Datei setzen Sie durch Angabe von `Last-Sequence-Num` einen Prüfpunkt. In diesem Fall würden Sie bei Datensatznummer 15.000 einen Prüfpunkt setzen.

Bei diesem Schema hat die resultierende Amazon-S3-Datei auch bei einer mehrfachen Verarbeitung der Datensätze denselben Namen und Dateninhalt. Die Wiederholungen führen nur dazu, dass dieselben Daten mehrmals in dieselbe Datei geschrieben werden.

Bei einer Reshard-Operation ist die Anzahl der im Shard verbliebenen Datensätze möglicherweise kleiner als benötigte Anzahl Ihrer gewünschten, festen Anzahl. In diesem Fall muss die `shutdown()`-Methode die Datei an Amazon S3 übertragen und bei der letzten Sequenznummer einen Prüfpunkt setzen. Die obige Schema ist auch mit Reshard-Operationen kompatibel.

## Umgang mit Startup, Herunterfahren und Drosselung

Nachfolgend einige Überlegungen, die Sie in das Design Ihrer Amazon Kinesis Data Streams einfließen lassen sollten.

### Inhalt

- [Starten von Datenproduzenten und Datenkonsumenten](#)
- [Herunterfahren einer Amazon-Kinesis-Daten-Streams-Anwendung](#)
- [Drosselung bei Lesevorgängen](#)

## Starten von Datenproduzenten und Datenkonsumenten

Standardmäßig beginnt die KCL am vorderen Ende des Streams mit dem Lesen von Datensätzen, also mit dem zuletzt hinzugefügten Datensatz. Wenn bei dieser Konfiguration eine datenproduzierende Anwendung Datensätze zum Stream hinzufügt, ehe empfangsbereite Datensatzprozessoren ausgeführt werden, werden diese Datensätze nach dem Start der Datensatzprozessoren nicht gelesen.

Damit Datenprozessoren immer am Anfang des Streams mit dem Lesen beginnen, legen Sie den folgenden Wert in der Eigenschaftendatei Ihrer Amazon Kinesis Data Streams fest:

```
initialPositionInStream = TRIM_HORIZON
```

Standardmäßig speichert Amazon Kinesis Data Streams alle Daten für 24 Stunden. Er unterstützt auch eine erweiterte Aufbewahrung von bis zu 7 Tagen und die langfristige Aufbewahrung von bis zu 365 Tagen. Dieser Zeitrahmen wird als Aufbewahrungszeitraum bezeichnet. Wenn Sie die Startposition auf TRIM\_HORIZON festsetzen, beginnt der Datensatzprozessor entsprechend des definierten Aufbewahrungszeitraums mit den ältesten Daten. Wenn ein Datensatzprozessor nach Ablauf des Aufbewahrungszeitraums gestartet wird, sind trotz TRIM\_HORIZON-Einstellung einige der Datensätze aus dem Stream nicht mehr verfügbar. Aus diesem Grund sollten Sie Konsumenten Anwendungen immer Daten aus dem Stream lesen lassen und mit der CloudWatch-Metrik `GetRecords.IteratorAgeMilliseconds` überwachen, ob die Anwendungen dem eingehenden Datenvolumen gerecht werden.

In einigen Fällen mag es unerheblich sein, wenn Datensatzprozessoren die ersten Datensätze im Stream auslassen. Sie können beispielsweise einige erste Datensätze durch den Stream schicken, um zu testen, ob die Ende-zu-Ende-Verarbeitung wie erwartet funktioniert. Nach dieser anfänglichen Verifizierung starten Sie dann die Auftragnehmer und beginnen mit der Einleitung von Produktionsdaten in den Stream.

Weitere Informationen zur TRIM\_HORIZON-Einstellung finden Sie unter [Verwenden von Shard-Iteratoren](#).

## Herunterfahren einer Amazon-Kinesis-Daten-Streams-Anwendung

Wenn Ihre Anwendung für Amazon Kinesis Data Streams die beabsichtigten Aufgaben abgeschlossen hat, sollten Sie sie herunterfahren, indem Sie die EC2-Instances beenden, auf denen sie ausgeführt wird. Sie können die Instances über die [AWS Management Console](#) oder die [AWS CLI](#) beenden.

Nach dem Herunterfahren der Anwendung für Amazon Kinesis Data Streams sollten Sie die Amazon DynamoDB-Tabelle löschen, mit der die KCL den Anwendungsstatus verfolgt hat.

## Drosselung bei Lesevorgängen

Der Durchsatz eines Streams wird auf Shard-Ebene bereitgestellt. Jeder Shard hat einen Lesedurchsatz von bis zu 5 Transaktionen pro Sekunde für Lesevorgänge, bis zu einer maximalen

Gesamtdatenleaserate von 2 MB pro Sekunde. Wenn eine Anwendung (oder eine Gruppe von Anwendungen im selben Stream) versucht, Daten schneller aus einem Shard abzurufen, drosselt Kinesis Data Streams die entsprechenden GET-Operationen.

Wenn in einer Amazon Kinesis Data Streams ein Datensatzprozessor Daten schneller als das Limit verarbeitet, beispielsweise bei einem Ausfall, kommt es zu einem Failover. Da die KCL die Interaktionen zwischen der Anwendung und Kinesis Data Streams verwaltet, treten Ablehnungsausnahmen im Code der KCL und nicht im Anwendungscode auf. Da die KCL diese Ausnahmen jedoch protokolliert, sind sie in den Protokollen zu sehen.

Wenn Sie feststellen, dass Ihre Anwendung permanent gedrosselt wird, sollten Sie eine Erhöhung der Shards für diesen Stream in Betracht ziehen.

# Überwachung von Amazon Kinesis Data Streams

Sie können Datenströme in Amazon Kinesis Data Streams mit den folgenden Features überwachen:

- [CloudWatch-Metriken](#) – Kinesis Data Streams sendet Amazon-CloudWatch-Metriken mit detaillierter Überwachung für jeden Stream.
- [Kinesis Agent](#) – Der Kinesis Agent veröffentlicht benutzerdefinierte CloudWatch-Metriken, um festzustellen, ob der Agent wie erwartet funktioniert.
- [API-Protokollierung](#) – Kinesis Data Streams verwendet AWS CloudTrail zur Protokollierung von API-Aufrufen und speichert die Daten in einem Amazon-S3-Bucket.
- [Kinesis Client Library](#) – Die Kinesis Client Library (KCL) bietet Metriken pro Shard, Arbeiter und KCL-Anwendung.
- [Kinesis Producer Library](#) – Die Kinesis Producer Library (KPL) bietet Metriken pro Shard, Arbeiter und KPL-Anwendung.

Weitere Informationen zu allgemeinen Überwachungsproblemen, Fragen und zur Fehlerbehebung finden Sie im Folgenden:

- [Welche Metriken sollte ich verwenden, um Probleme mit Kinesis Data Streams zu überwachen und zu beheben?](#)
- [Warum steigt der Wert für `IteratorAgeMilliseconds` in Kinesis Data Streams ständig an?](#)

## Überwachung des Services von Amazon Kinesis Data Streams für Amazon CloudWatch

Amazon Kinesis Data Streams und Amazon CloudWatch sind integriert, sodass Sie CloudWatch-Metriken für Ihre Kinesis-Datenströme sammeln, anzeigen und analysieren können. Um beispielsweise die Shard-Nutzung zu verfolgen, können Sie die Kennzahlen `IncomingBytes` und `OutgoingBytes` überwachen und mit der Anzahl der Shards im Stream vergleichen.

Die Metriken, die Sie für Ihre Streams konfigurieren, werden automatisch gesammelt und alle fünf Minuten an CloudWatch übergeben. Die Metriken werden für zwei Wochen archiviert. Nach Ablauf dieses Zeitraums werden die Daten verworfen.

Die folgende Tabelle beschreibt die Überwachung von Kinesis-Datenströmen auf grundlegender Stream- und erweiterter Shard-Ebene.

Typ	Beschreibung
Grundlegend (Stream-Ebene)	Daten auf Stream-Ebene werden automatisch und kostenlos jede Minute gesendet.
Erweitert (Shard-Ebene)	Daten auf Shard-Ebene werden für zusätzliche Kosten jede Minute gesendet. Für diese Datenebene müssen Sie eine spezielle Aktivierung für den Stream mit der Operation <a href="#">EnableEnhancedMonitoring</a> vornehmen.  Weitere Informationen zu Preisen finden Sie auf der <a href="#">Produktseite Amazon CloudWatch</a> .

## Amazon Kinesis Data Streams – Metriken und Dimensionen

Kinesis Data Streams sendet auf zwei Ebenen Metriken an CloudWatch: auf Stream-Ebene und optional auf Shard-Ebene. Metriken auf Stream-Ebene eignen sich unter normalen Bedingungen für die häufigsten Überwachungsanwendungsfälle. Metriken auf Shard-Ebene sind für bestimmte Überwachungsaufgaben gedacht, gewöhnlich im Zusammenhang mit der Fehlerbehebung; sie werden normalerweise mit der Operation [EnableEnhancedMonitoring](#) aktiviert.

Eine Erläuterung der anhand von CloudWatch-Metriken gesammelten Statistiken finden Sie unter [CloudWatch-Statistiken](#) im Amazon-CloudWatch-Benutzerhandbuch.

### Themen

- [Grundlegende Stream-Metriken](#)
- [Erweiterte Metriken auf Shard-Ebene:](#)
- [Dimensionen für Metriken für Amazon Kinesis Data Streams](#)
- [Empfohlene Metriken für Amazon Kinesis Data Streams](#)

### Grundlegende Stream-Metriken

Der AWS/Kinesis-Namespace enthält die folgenden Metriken auf Stream-Ebene.

Kinesis-Datenströme sendet diese Metriken auf Streams-Ebene minütlich an CloudWatch. Diese Metriken sind jederzeit verfügbar:

Kennzahl	Beschreibung
<code>GetRecords.Bytes</code>	<p>Anzahl der Byte, die im angegebenen Zeitraum vom Kinesis-Stream abgerufen wurden. Die Statistiken Minimum, Maximum und Durchschnitt geben die Byte in einem einzelnen <code>GetRecords</code> -Vorgang für den Stream im angegebenen Zeitraum an.</p> <p>Metrikname auf Shard-Ebene: <code>OutgoingBytes</code></p> <p>Dimensionen: <code>StreamName</code></p> <p>Gültige Statistiken: Minimum, Maximum, Durchschnitt, Summe, Stichproben</p> <p>Einheiten: Byte</p>
<code>GetRecords.IteratorAge</code>	<p>Diese Metrik ist veraltet. Verwenden Sie <code>GetRecords.IteratorAgeMilliseconds</code>.</p>
<code>GetRecords.IteratorAgeMilliseconds</code>	<p>Alter des letzten Datensatzes in allen <code>GetRecords</code> -Aufrufen an einen Kinesis-Stream, gemessen im angegebenen Zeitraum. Das Alter ist die Differenz zwischen der aktuellen Zeit und der Zeit, zu der der letzte Datensatz des <code>GetRecords</code> -Aufrufs in den Stream geschrieben wurde. Die Statistiken Minimum und Maximum können zum Nachverfolgen des Fortschritts von Kinesis-Verbraucheranwendungen verwendet werden. Wenn der Wert null beträgt, sind die gelesenen Datensätze vollständig mit dem Stream aktualisiert.</p> <p>Metrikname auf Shard-Ebene: <code>IteratorAgeMilliseconds</code></p> <p>Dimensionen: <code>StreamName</code></p>

Kennzahl	Beschreibung
	<p>Gültige Statistiken: Minimum, Maximum, Durchschnitt, Stichproben</p> <p>Einheiten: Millisekunden</p>
GetRecords.Latency	<p>Die Dauer pro GetRecords -Vorgang, gemessen im angegebenen Zeitraum.</p> <p>Dimensionen: StreamName</p> <p>Statistiken: Minimum, Maximum, Durchschnitt</p> <p>Einheiten: Millisekunden</p>
GetRecords.Records	<p>Anzahl der Datensätze, die im angegebenen Zeitraum vom Shard abgerufen wurden. Die Statistiken Minimum, Maximum und Durchschnitt geben die Datensätze in einem einzelnen GetRecords -Vorgang für den Stream im angegebenen Zeitraum an.</p> <p>Metrikname auf Shard-Ebene: OutgoingRecords</p> <p>Dimensionen: StreamName</p> <p>Gültige Statistiken: Minimum, Maximum, Durchschnitt, Summe, Stichproben</p> <p>Einheiten: Anzahl</p>
GetRecords.Success	<p>Anzahl der erfolgreichen GetRecords -Vorgänge pro Stream im angegebenen Zeitraum.</p> <p>Dimensionen: StreamName</p> <p>Gültige Statistiken: Durchschnitt, Summe, Stichproben</p> <p>Einheiten: Anzahl</p>



Kennzahl	Beschreibung
IncomingBytes	<p>Anzahl der Byte, die im angegebenen Zeitraum erfolgreich an den Kinesis-Stream übertragen wurden. Diese Metrik enthält die Byte von PutRecord - und PutRecords -Vorgängen. Die Statistiken Minimum, Maximum und Durchschnitt geben die Byte in einem einzelnen PUT-Vorgang für den Stream im angegebenen Zeitraum an.</p> <p>Metrikname auf Shard-Ebene: IncomingBytes</p> <p>Dimensionen: StreamName</p> <p>Gültige Statistiken: Minimum, Maximum, Durchschnitt, Summe, Stichproben</p> <p>Einheiten: Byte</p>
IncomingRecords	<p>Anzahl der Datensätze, die im angegebenen Zeitraum erfolgreich an den Kinesis-Stream übertragen wurden. Diese Metrik enthält die Anzahlen der Datensätze von PutRecord - und PutRecords -Vorgängen. Die Statistiken Minimum, Maximum und Durchschnitt geben die Datensätze in einem einzelnen PUT-Vorgang für den Stream im angegebenen Zeitraum an.</p> <p>Metrikname auf Shard-Ebene: IncomingRecords</p> <p>Dimensionen: StreamName</p> <p>Gültige Statistiken: Minimum, Maximum, Durchschnitt, Summe, Stichproben</p> <p>Einheiten: Anzahl</p>

Kennzahl	Beschreibung
PutRecord.Bytes	<p>Anzahl der Bytes, die im angegebenen Zeitraum mithilfe des Vorgangs PutRecord an den Kinesis-Stream übertragen wurden.</p> <p>Dimensionen: StreamName</p> <p>Gültige Statistiken: Minimum, Maximum, Durchschnitt, Summe, Stichproben</p> <p>Einheiten: Byte</p>
PutRecord.Latency	<p>Die Dauer pro PutRecord -Vorgang, gemessen im angegebenen Zeitraum.</p> <p>Dimensionen: StreamName</p> <p>Statistiken: Minimum, Maximum, Durchschnitt</p> <p>Einheiten: Millisekunden</p>
PutRecord.Success	<p>Anzahl der erfolgreichen PutRecord -Vorgänge pro Kinesis-Stream im angegebenen Zeitraum. Der Durchschnitt gibt den Prozentsatz der erfolgreichen Schreibvorgänge in einem Stream an.</p> <p>Dimensionen: StreamName</p> <p>Gültige Statistiken: Durchschnitt, Summe, Stichproben</p> <p>Einheiten: Anzahl</p>

Kennzahl	Beschreibung
<code>PutRecords.Bytes</code>	<p>Anzahl der Bytes, die im angegebenen Zeitraum mithilfe des Vorgangs <code>PutRecords</code> an den Kinesis-Stream übertragen wurden.</p> <p>Dimensionen: <code>StreamName</code></p> <p>Gültige Statistiken: Minimum, Maximum, Durchschnitt, Summe, Stichproben</p> <p>Einheiten: Byte</p>
<code>PutRecords.Latency</code>	<p>Die Dauer pro <code>PutRecords</code> -Vorgang, gemessen im angegebenen Zeitraum.</p> <p>Dimensionen: <code>StreamName</code></p> <p>Statistiken: Minimum, Maximum, Durchschnitt</p> <p>Einheiten: Millisekunden</p>
<code>PutRecords.Records</code>	<p>Diese Metrik ist veraltet. Verwenden Sie <code>PutRecords.SuccessfulRecords</code>.</p> <p>Dimensionen: <code>StreamName</code></p> <p>Gültige Statistiken: Minimum, Maximum, Durchschnitt, Summe, Stichproben</p> <p>Einheiten: Anzahl</p>
<code>PutRecords.Success</code>	<p>Anzahl der <code>PutRecords</code> -Vorgänge pro Kinesis-Stream mit mindestens einem erfolgreichen Datensatz, gemessenen im angegebenen Zeitraum.</p> <p>Dimensionen: <code>StreamName</code></p> <p>Gültige Statistiken: Durchschnitt, Summe, Stichproben</p> <p>Einheiten: Anzahl</p>

Kennzahl	Beschreibung
<code>PutRecords.TotalRecords</code>	<p>Die gesamte Anzahl der Datensätze in einem <code>PutRecords</code> -Vorgang pro Kinesis-Datenstrom, gemessen im angegebenen Zeitraum.</p> <p>Dimensionen: <code>StreamName</code></p> <p>Gültige Statistiken: Minimum, Maximum, Durchschnitt, Summe, Stichproben</p> <p>Einheiten: Anzahl</p>
<code>PutRecords.SuccessfulRecords</code>	<p>Anzahl der erfolgreichen Datensätze in einem <code>PutRecords</code> -Vorgang pro Kinesis-Datenstrom, gemessen im angegebenen Zeitraum.</p> <p>Dimensionen: <code>StreamName</code></p> <p>Gültige Statistiken: Minimum, Maximum, Durchschnitt, Summe, Stichproben</p> <p>Einheiten: Anzahl</p>
<code>PutRecords.FailedRecords</code>	<p>Anzahl der Datensätze, die aufgrund von internen Fehlern in einem <code>PutRecords</code> -Vorgang pro Kinesis-Datenstrom zurückgewiesen wurden, gemessen über den angegebenen Zeitraum. Gelegentliche interne Fehler sind zu erwarten und sollten erneut versucht werden.</p> <p>Dimensionen: <code>StreamName</code></p> <p>Gültige Statistiken: Minimum, Maximum, Durchschnitt, Summe, Stichproben</p> <p>Einheiten: Anzahl</p>

Kennzahl	Beschreibung
<code>PutRecords.ThrottledRecords</code>	<p>Anzahl der Datensätze, die aufgrund von Drosselung in einem <code>PutRecords</code> -Vorgang pro Kinesis-Datenstrom zurückgewiesen wurden, gemessen über den angegebenen Zeitraum.</p> <p>Dimensionen: <code>StreamName</code></p> <p>Gültige Statistiken: Minimum, Maximum, Durchschnitt, Summe, Stichproben</p> <p>Einheiten: Anzahl</p>
<code>ReadProvisionedThroughputExceeded</code>	<p>Anzahl von <code>GetRecords</code> -Aufrufen, die im angegebenen Zeitraum für den Stream gedrosselt wurden. Die am häufigsten verwendete Statistik für diese Metrik ist Durchschnitt.</p> <p>Wenn die Statistik Minimum den Wert 1 hat, wurden im angegebenen Zeitraum sämtliche Datensätze für den Stream gedrosselt.</p> <p>Wenn die Statistik Maximum den Wert 0 (null) hat, wurden im angegebenen Zeitraum keine Datensätze für den Stream gedrosselt.</p> <p>Metrikname auf Shard-Ebene: <code>ReadProvisionedThroughputExceeded</code></p> <p>Dimensionen: <code>StreamName</code></p> <p>Gültige Statistiken: Minimum, Maximum, Durchschnitt, Summe, Stichproben</p> <p>Einheiten: Anzahl</p>

Kennzahl	Beschreibung
<code>SubscribeToShard.RateExceeded</code>	<p>Diese Metrik wird ausgelöst, wenn ein neuer Abonnementsversuch fehlschlägt, da bereits ein aktives Abonnement von demselben Verbraucher existiert, oder wenn Sie die Anzahl der Aufrufe pro Sekunde überschreiten, die für diese Operation zulässig ist.</p> <p>Dimensionen: StreamName, ConsumerName</p>
<code>SubscribeToShard.Success</code>	<p>Diese Metrik zeichnet auf, ob das Abonnement <code>SubscribeToShard</code> erfolgreich erstellt wurde. Das Abonnement besteht nur für maximal 5 Minuten. Aus diesem Grund wird diese Metrik mindestens einmal alle 5 Minuten ausgegeben.</p> <p>Dimensionen: StreamName, ConsumerName</p>
<code>SubscribeToShardEvent.Bytes</code>	<p>Anzahl der Bytes, die im angegebenen Zeitraum vom Shard empfangen wurden. Die Statistiken Minimum, Maximum und Durchschnitt geben die Bytes an, die in einem einzelnen Ereignis für den angegebenen Zeitraum veröffentlicht wurden.</p> <p>Metrikname auf Shard-Ebene: <code>OutgoingBytes</code></p> <p>Dimensionen: StreamName, ConsumerName</p> <p>Gültige Statistiken: Minimum, Maximum, Durchschnitt, Summe, Stichproben</p> <p>Einheiten: Byte</p>

Kennzahl	Beschreibung
<code>SubscribeToShardEvent.MillisBehindLatest</code>	<p>Die Differenz zwischen der aktuellen Zeit und der Zeit, zu der der letzte Datensatz des Ereignisses <code>SubscribeToShard</code> in den Stream geschrieben wurde.</p> <p>Dimensionen: <code>StreamName</code>, <code>ConsumerName</code></p> <p>Gültige Statistiken: Minimum, Maximum, Durchschnitt, Stichproben</p> <p>Einheiten: Millisekunden</p>
<code>SubscribeToShardEvent.Records</code>	<p>Anzahl der Datensätze, die im angegebenen Zeitraum vom Shard empfangen wurden. Die Statistiken Minimum, Maximum und Durchschnitt geben die Datensätze in einem einzelnen Ereignis für den angegebenen Zeitraum an.</p> <p>Metrikname auf Shard-Ebene: <code>OutgoingRecords</code></p> <p>Dimensionen: <code>StreamName</code>, <code>ConsumerName</code></p> <p>Gültige Statistiken: Minimum, Maximum, Durchschnitt, Summe, Stichproben</p> <p>Einheiten: Anzahl</p>
<code>SubscribeToShardEvent.Success</code>	<p>Diese Metrik wird jedes Mal ausgegeben, wenn ein Ereignis erfolgreich veröffentlicht wird. Es wird nur ausgegeben, wenn ein aktives Abonnement besteht.</p> <p>Dimensionen: <code>StreamName</code>, <code>ConsumerName</code></p> <p>Gültige Statistiken: Minimum, Maximum, Durchschnitt, Summe, Stichproben</p> <p>Einheiten: Anzahl</p>

Kennzahl	Beschreibung
<p><code>WriteProvisionedThroughputExceeded</code></p>	<p>Anzahl der Datensätze, die im angegebenen Zeitraum infolge der Drosselung für den Stream abgelehnt wurden. Diese Metrik enthält die Drosselung infolge von <code>PutRecord</code> - und <code>PutRecords</code> -Vorgängen. Die am häufigsten verwendete Statistik für diese Metrik ist Durchschnitt.</p> <p>Wenn die Statistik Minimum einen Wert ungleich null hat, wurden im angegebenen Zeitraum Datensätze für den Stream gedrosselt.</p> <p>Wenn die Statistik Maximum den Wert 0 (null) hat, wurden im angegebenen Zeitraum keine Datensätze für den Stream gedrosselt.</p> <p>Metrikname auf Shard-Ebene: <code>WriteProvisionedThroughputExceeded</code></p> <p>Dimensionen: <code>StreamName</code></p> <p>Gültige Statistiken: Minimum, Maximum, Durchschnitt, Summe, Stichproben</p> <p>Einheiten: Anzahl</p>

## Erweiterte Metriken auf Shard-Ebene:

Der AWS/Kinesis-Namespaces enthält die folgenden Metriken auf Shard-Ebene.

Kinesis sendet jede Minute die folgenden Metriken auf Shard-Ebene an CloudWatch. Jede Metrikdimension erstellt eine CloudWatch-Metrik und führt ungefähr 43 200 `PutMetricData`-API-Aufrufe pro Monat durch. Diese Metriken sind nicht standardmäßig aktiviert. Von Kinesis übermittelte erweiterte Metriken sind kostenpflichtig. Weitere Informationen finden Sie unter [Amazon CloudWatch – Preise](#) unter der Überschrift Benutzerdefinierte Amazon CloudWatch-Metriken. Die Kosten fallen pro Shard pro Metrik pro Monat an.



Kennzahl	Beschreibung
IncomingBytes	<p>Anzahl der Byte, die im angegebenen Zeitraum erfolgreich an den Shard übertragen wurden. Diese Metrik enthält die Byte von PutRecord - und PutRecords -Vorgängen. Die Statistiken Minimum, Maximum und Durchschnitt geben die Byte in einem einzelnen PUT-Vorgang für den Shard im angegebenen Zeitraum an.</p> <p>Metrikname auf Stream-Ebene: IncomingBytes</p> <p>Dimensionen: StreamName, ShardId</p> <p>Gültige Statistiken: Minimum, Maximum, Durchschnitt, Summe, Stichproben</p> <p>Einheiten: Byte</p>
IncomingRecords	<p>Anzahl der Datensätze, die im angegebenen Zeitraum erfolgreich an den Shard übertragen wurden. Diese Metrik enthält die Anzahlen der Datensätze von PutRecord - und PutRecords -Vorgängen. Die Statistiken Minimum, Maximum und Durchschnitt geben die Datensätze in einem einzelnen PUT-Vorgang für den Shard im angegebenen Zeitraum an.</p> <p>Metrikname auf Stream-Ebene: IncomingRecords</p> <p>Dimensionen: StreamName, ShardId</p> <p>Gültige Statistiken: Minimum, Maximum, Durchschnitt, Summe, Stichproben</p> <p>Einheiten: Anzahl</p>
IteratorAgeMilliseconds	<p>Alter des letzten Datensatzes in allen GetRecords - Aufrufen an einen Shard, gemessen im angegebenen Zeitraum. Das Alter ist die Differenz zwischen der aktuellen Zeit und der Zeit, zu der der letzte Datensatz</p>

Kennzahl	Beschreibung
	<p>des <code>GetRecords</code> -Aufrufs in den Stream geschrieben wurde. Die Statistiken Minimum und Maximum können zum Nachverfolgen des Fortschritts von Kinesis-Verbraucheranwendungen verwendet werden. Wenn der Wert 0 (null) beträgt, sind die gelesenen Datensätze vollständig mit dem Stream aktualisiert.</p> <p>Metrikname auf Stream-Ebene: <code>GetRecords.IteratorAgeMilliseconds</code></p> <p>Dimensionen: <code>StreamName</code>, <code>ShardId</code></p> <p>Gültige Statistiken: Minimum, Maximum, Durchschnitt, Stichproben</p> <p>Einheiten: Millisekunden</p>
OutgoingBytes	<p>Anzahl der Bytes, die im angegebenen Zeitraum vom Shard abgerufen wurden. Die Statistiken Minimum, Maximum und Durchschnitt geben die Bytes an, die für den Shard im angegebenen Zeitraum in einem einzelnen <code>GetRecords</code> -Vorgang zurückgegeben oder bei einem einzelnen <code>SubscribeToShard</code> -Ereignis veröffentlicht wurden.</p> <p>Metrikname auf Stream-Ebene: <code>GetRecords.Bytes</code></p> <p>Dimensionen: <code>StreamName</code>, <code>ShardId</code></p> <p>Gültige Statistiken: Minimum, Maximum, Durchschnitt, Summe, Stichproben</p> <p>Einheiten: Byte</p>

Kennzahl	Beschreibung
OutgoingRecords	<p>Anzahl der Datensätze, die im angegebenen Zeitraum vom Shard abgerufen wurden. Die Statistiken Minimum, Maximum und Durchschnitt geben die Datensätze an, die für den Shard im angegebenen Zeitraum in einem einzelnen <code>GetRecords</code> -Vorgang zurückgegeben oder bei einem einzelnen <code>SubscribeToShard</code> -Ereignis veröffentlicht wurden.</p> <p>Metrikname auf Stream-Ebene: <code>GetRecords.Records</code></p> <p>Dimensionen: <code>StreamName</code>, <code>ShardId</code></p> <p>Gültige Statistiken: Minimum, Maximum, Durchschnitt, Summe, Stichproben</p> <p>Einheiten: Anzahl</p>

Kennzahl	Beschreibung
ReadProvisionedThroughputExceeded	<p>Anzahl von <code>GetRecords</code> -Aufrufen, die im angegebenen Zeitraum für den Shard gedrosselt wurden. Diese Anzahl von Ausnahmen deckt alle Dimensionen mit folgenden Einschränkungen ab: 5 Lesevorgänge pro Shard pro Sekunde oder 2 MB pro Sekunde pro Shard. Die am häufigsten verwendete Statistik für diese Metrik ist Durchschnitt.</p> <p>Wenn die Statistik Minimum den Wert 1 hat, wurden im angegebenen Zeitraum sämtliche Datensätze für den Shard gedrosselt.</p> <p>Wenn die Statistik Maximum den Wert 0 (null) hat, wurden im angegebenen Zeitraum keine Datensätze für den Shard gedrosselt.</p> <p>Metrikname auf Stream-Ebene: <code>ReadProvisionedThroughputExceeded</code></p> <p>Dimensionen: <code>StreamName</code>, <code>ShardId</code></p> <p>Gültige Statistiken: Minimum, Maximum, Durchschnitt, Summe, Stichproben</p> <p>Einheiten: Anzahl</p>

Kennzahl	Beschreibung
WriteProvisionedThroughputExceeded	<p>Anzahl der Datensätze, die im angegebenen Zeitraum infolge der Drosselung für den Shard abgelehnt wurden. Diese Metrik enthält die Drosselung infolge von PutRecord - und PutRecords -Vorgängen und deckt alle Dimensionen mit folgenden Einschränkungen ab: 1 000 Lesevorgänge pro Shard pro Sekunde oder 1 MB pro Sekunde pro Shard. Die am häufigsten verwendete Statistik für diese Metrik ist Durchschnitt.</p> <p>Wenn die Statistik Minimum einen Wert ungleich null hat, wurden im angegebenen Zeitraum Datensätze für den Shard gedrosselt.</p> <p>Wenn die Statistik Maximum den Wert 0 (null) hat, wurden im angegebenen Zeitraum keine Datensätze für den Shard gedrosselt.</p> <p>Metrikname auf Stream-Ebene: WriteProvisionedThroughputExceeded</p> <p>Dimensionen: StreamName, ShardId</p> <p>Gültige Statistiken: Minimum, Maximum, Durchschnitt, Summe, Stichproben</p> <p>Einheiten: Anzahl</p>

## Dimensionen für Metriken für Amazon Kinesis Data Streams

Dimension	Beschreibung
StreamName	Der Name des Kinesis-Streams. Alle verfügbaren Statistiken werden von StreamName gefiltert.

## Empfohlene Metriken für Amazon Kinesis Data Streams

Verschiedene Metriken für Amazon Kinesis Data Streams könnten für Kunden von Kinesis Data Streams von besonderem Interesse sein. Die folgende Liste enthält empfohlene Metriken und ihre Verwendung.

Kennzahl	Nutzungshinweise
<code>GetRecords.IteratorAgeMilliseconds</code>	Verfolgt die Leseposition über alle Shards und Kunden in dem Stream. Wenn das Alter eines Iterators 50 % des Aufbewahrungszeitraums (standardmäßig 24 Stunden, anpassbar auf bis zu 7 Tage) überschreitet, besteht die Gefahr des Datenverlusts durch Ablauf des Datensatzes. Wir empfehlen die Verwendung von CloudWatch-Alarmen für den statistischen Maximum-Wert, damit Sie gewarnt werden, bevor dieses Verlustrisiko real wird. Ein Beispielszenario mit dieser Metrik finden Sie unter <a href="#">Die Verarbeitung der Konsumentendatensätze hängt hinterher</a> .
<code>ReadProvisionedThroughputExceeded</code>	Wenn Ihre konsumentenseitige Datensatzverarbeitung zurückfällt, ist es oft schwierig zu erkennen, wo der Engpass ist. Verwenden Sie diese Metrik, um zu bestimmen, ob Ihre Lesevorgänge aufgrund einer Überschreitung Ihrer Durchsatzgrenzwerte gedrosselt werden. Die am häufigsten verwendete Statistik für diese Metrik ist Durchschnitt.
<code>WriteProvisionedThroughputExceeded</code>	Diese dient dem gleichen Zweck wie die Metrik <code>ReadProvisionedThroughputExceeded</code> , jedoch für die Producer- (Put) Seite des Streams. Die am häufigsten verwendete Statistik für diese Metrik ist Durchschnitt.
<code>PutRecords.Success</code> , <code>PutRecords.Success</code>	Wir empfehlen die Verwendung von CloudWatch-Alarmen für den statistischen "Average"-Wert, um zu melden, wenn Datensätze für den Stream fehlschlagen. Wählen Sie einen oder beide Put-Typen, je nachdem, was Ihr Producer verwendet. Wenn Sie die Kinesis Producer Library (KPL) verwenden, nehmen Sie <code>PutRecords.Success</code> .
<code>GetRecords.Success</code>	Wir empfehlen die Verwendung von CloudWatch-Alarmen für den statistischen Average-Wert, um zu melden, wenn Datensätze aus dem Stream fehlschlagen.

## Zugreifen auf Amazon-CloudWatch-Metriken für Kinesis Data Streams

Sie können Metriken für Kinesis Data Streams mithilfe der CloudWatch-Konsole, der Befehlszeile oder der CloudWatch-API überwachen. Die folgenden Verfahren zeigen, wie Sie mithilfe dieser verschiedenen Verfahren auf die Metriken zugreifen können.

So greifen Sie auf Metriken mit der CloudWatch-Konsole zu:

1. Öffnen Sie die CloudWatch-Konsole unter <https://console.aws.amazon.com/cloudwatch/>.
2. Wählen Sie in der Navigationsleiste eine Region aus.
3. Wählen Sie im Navigationsbereich Metriken aus.
4. Wählen Sie im Fenster CloudWatch Metrics by Category (CloudWatch-Metriken nach Kategorie) die Option Kinesis Metrics (Kinesis-Metriken).
5. Klicken Sie auf die entsprechende Zeile, um die Statistiken für den angegebenen MetricName und den StreamName anzuzeigen.

Hinweis: Die meisten Konsolen-Statistiknamen entsprechen den CloudWatch-Metrikenamen (s. o.), ausgenommen sind Read Throughput (Lesedurchsatz) und Write Throughput (Schreibdurchsatz). Diese Statistiken werden über 5-Minuten-Intervalle berechnet: Write Throughput (Schreibdurchsatz) überwacht die IncomingBytes-CloudWatch-Metrik und Read Throughput (Lesedurchsatz) überwacht GetRecords.Bytes.

6. (Optional) Wählen Sie im Diagrammbereich einen statistischen Wert und einen Zeitraum und erstellen Sie dann mit diesen Einstellungen einen CloudWatch-Alarm.

Zugriff auf Metriken über die AWS CLI

Verwenden Sie die Befehle [list-metrics](#) und [get-metric-statistics](#).

So greifen Sie mit der CloudWatch-CLI auf Metriken zu

Verwenden Sie die Befehle [mon-list-metrics](#) und [mon-get-stats](#).

So greifen Sie mit der CloudWatch-API auf Metriken zu

Verwenden Sie die Operationen [ListMetrics](#) und [GetMetricStatistics](#).

# Überwachung von Kinesis Data Streams Agent Health für Amazon CloudWatch

Der Agent veröffentlicht benutzerdefinierte CloudWatch-Metriken mit dem `AWSKinesisAgent`. Diese Metriken ermöglichen die Einschätzung, ob der Agent Daten wie angegeben an Kinesis Data Streams sendet, ob er in gutem Zustand ist und angemessene CPU- und Arbeitsspeicherressourcen auf dem Datenproduzenten konsumiert. Metriken wie die Anzahl der gesendeten Datensätze und Bytes sind nützlich, um zu verstehen, wie der Agent Daten an den Stream; übergibt. Wenn diese Metriken um einige Prozent unter die erwarteten Schwellenwerte oder auf Null sinken, kann dies auf Probleme mit der Konfiguration, Netzwerkfehler oder Probleme mit dem Zustand des Agenten hinweisen. Metriken wie On-Host CPU- und Speicherbelegung sowie die Anzahl der Agentenfehler zeigen die Nutzung der Ressourcen des Datenproduzenten an und informieren über potenzielle Konfigurations- oder Hostfehler. Schließlich protokolliert der Agent auf Serviceausnahmen, um die Untersuchung von Agentenproblemen zu unterstützen. Diese Kennzahlen werden in der Region gemeldet, die in der Agentenkonfigurationseinstellung `cloudwatch.endpoint` angegeben ist. Cloudwatch-Metriken, die von mehreren Kinesis-Agenten veröffentlicht wurden, werden aggregiert oder kombiniert. Weitere Informationen zur Agentenkonfiguration finden Sie unter [Konfigurationseinstellungen für den Agenten](#)

## Überwachung mit CloudWatch

Der Kinesis-Data-Streams-Agent sendet die folgenden Metriken an CloudWatch.

Kennzahl	Beschreibung
<code>BytesSent</code>	Die Anzahl von Byte, die über den angegebenen Zeitraum an Kinesis Data Streams gesendet wurden.  Einheiten: Byte
<code>RecordSendAttempts</code>	Die Anzahl der versuchten Datensätze (zum ersten Mal oder wiederholt) in einem Aufruf an <code>PutRecords</code> in dem angegebenen Zeitraum.  Einheiten: Anzahl
<code>RecordSendErrors</code>	Die Anzahl der Datensätze mit Fehlerstatus in einem Aufruf an <code>PutRecords</code> , einschließlich wiederholter Versuche, in dem angegebenen Zeitraum.



Kennzahl	Beschreibung
	Einheiten: Anzahl
<code>ServiceErrors</code>	Die Anzahl der Aufrufe an <code>PutRecords</code> , die zu einem Servicefehler führten (außer Ablehnungsfehlern) in dem angegebenen Zeitraum.  Einheiten: Anzahl

## Protokollieren von Amazon-Kinesis-Data-Streams-API-Aufrufen mithilfe von AWS CloudTrail

Amazon Kinesis Data Streams ist in AWS CloudTrail integriert, einem Service, der eine Aufzeichnung der von einem Benutzer, einer Rolle oder einem AWS-Service durchgeführten Aktionen in Kinesis Data Streams bereitstellt. CloudTrail erfasst alle API-Aufrufe für Kinesis Data Streams als Ereignisse. Zu den erfassten Aufrufen gehören Aufrufe von der Kinesis-Data-Streams-Konsole und Code-Aufrufe der API-Operationen von Kinesis Data Streams. Wenn Sie einen Trail erstellen, können Sie die kontinuierliche Bereitstellung von CloudTrail-Ereignissen an einen Amazon S3-Bucket, einschließlich Ereignisse für Kinesis Data Streams aktivieren. Wenn Sie keinen Trail konfigurieren, können Sie die neuesten Ereignisse in der CloudTrail-Konsole trotzdem in Ereignisverlauf anzeigen. Mit den von CloudTrail erfassten Informationen können Sie die an Kinesis Data Streams gestellte Anfrage, die IP-Adresse, von der die Anfrage gestellt wurde, den Initiator der Anfrage, den Zeitpunkt der Anfrage und zusätzliche Details bestimmen.

Weitere Informationen über CloudTrail, einschließlich Konfiguration und Aktivierung, finden Sie im [AWS CloudTrail-Benutzerhandbuch](#).

### Informationen zu Kinesis Data Streams in CloudTrail

CloudTrail wird beim Erstellen Ihres AWS-Kontos für Sie aktiviert. Wenn die unterstützte Ereignisaktivität in Kinesis Data Streams eintritt, wird diese Aktivität in einem CloudTrail-Ereignis zusammen mit anderen AWS-Serviceereignissen im Ereignisverlauf aufgezeichnet. Sie können die neusten Ereignisse in Ihr AWS-Konto herunterladen und dort suchen und anzeigen. Weitere Informationen finden Sie unter [Anzeigen von Ereignissen mit dem CloudTrail-Ereignisverlauf](#).

Erstellen Sie für eine fortlaufende Aufzeichnung der Ereignisse in Ihrem AWS-Konto, einschließlich Ereignissen für Kinesis Data Streams, einen Trail. Ein Trail ermöglicht es CloudTrail, Protokolldateien in einem Amazon-S3-Bucket bereitzustellen. Wenn Sie einen Trail in der Konsole anlegen, gilt

dieser für alle AWS-Regionen. Der Trail protokolliert Ereignisse aus allen Regionen in der AWS-Partition und stellt die Protokolldateien in dem von Ihnen angegebenen Amazon-S3-Bucket bereit. Darüber hinaus können Sie andere AWS-Services konfigurieren, um die in den CloudTrail-Protokollen erfassten Ereignisdaten weiter zu analysieren und entsprechend zu agieren. Weitere Informationen finden Sie unter:

- [Übersicht zum Erstellen eines Trails](#)
- [Von CloudTrail unterstützte Dienste und Integrationen](#)
- [Konfigurieren von Amazon-SNS-Benachrichtigungen für CloudTrail](#)
- [Empfangen von CloudTrail-Protokolldateien aus mehreren Regionen](#) und [Empfangen von CloudTrail-Protokolldateien aus mehreren Konten](#)

Kinesis Data Streams unterstützt die Protokollierung der folgenden Aktionen als Ereignisse in CloudTrail-Protokolldateien:

- [AddTagsToStream](#)
- [CreateStream](#)
- [DecreaseStreamRetentionPeriod](#)
- [DeleteStream](#)
- [DeregisterStreamConsumer](#)
- [DescribeStream](#)
- [DescribeStreamConsumer](#)
- [DisableEnhancedMonitoring](#)
- [EnableEnhancedMonitoring](#)
- [IncreaseStreamRetentionPeriod](#)
- [ListStreamConsumers](#)
- [ListStreams](#)
- [ListTagsForStream](#)
- [MergeShards](#)
- [RegisterStreamConsumer](#)
- [RemoveTagsFromStream](#)
- [SplitShard](#)
- [StartStreamEncryption](#)

- [StopStreamEncryption](#)
- [UpdateShardCount](#)
- [UpdateStreamMode](#)

Jeder Ereignis- oder Protokolleintrag enthält Informationen zu dem Benutzer, der die Anforderung generiert hat. Anhand der Identitätsinformationen zur Benutzeridentität können Sie Folgendes bestimmen:

- Ob die Anfrage mit Stammbenutzer- oder AWS Identity and Access Management (IAM)-Benutzeranmeldeinformationen ausgeführt wurde.
- Ob die Anforderung mit temporären Sicherheitsanmeldeinformationen für eine Rolle oder einen Verbundbenutzer ausgeführt wurde.
- Ob die Anforderung aus einem anderen AWS-Service gesendet wurde

Weitere Informationen finden Sie unter dem [CloudTrail userIdentity-Element](#).

## Beispiel: Einträge in der Protokolldatei für Kinesis Data Streams

Ein Trail ist eine Konfiguration, durch die Ereignisse als Protokolldateien an den von Ihnen angegebenen Amazon-S3-Bucket übermittelt werden. CloudTrail-Protokolldateien können einen oder mehrere Einträge enthalten. Ein Ereignis stellt eine einzelne Anforderung aus einer beliebigen Quelle dar und enthält unter anderem Informationen über die angeforderte Aktion, das Datum und die Uhrzeit der Aktion sowie über die Anforderungsparameter. CloudTrail-Protokolleinträge sind kein geordnetes Stack-Trace der öffentlichen API-Aufrufe und erscheinen daher in keiner bestimmten Reihenfolge.

Das folgende Beispiel zeigt einen CloudTrail-Protokolleintrag, der die Aktionen CreateStream, DescribeStream, ListStreams, DeleteStream, SplitShard und MergeShards demonstriert.

```
{
  "Records": [
    {
      "eventVersion": "1.01",
      "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::012345678910:user/Alice",
        "accountId": "012345678910",
```

```

        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice"
    },
    "eventTime": "2014-04-19T00:16:31Z",
    "eventSource": "kinesis.amazonaws.com",
    "eventName": "CreateStream",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "127.0.0.1",
    "userAgent": "aws-sdk-java/unknown-version Linux/x.xx",
    "requestParameters": {
        "shardCount": 1,
        "streamName": "GoodStream"
    },
    "responseElements": null,
    "requestID": "db6c59f8-c757-11e3-bc3b-57923b443c1c",
    "eventID": "b7acfd0-6ca9-4ee1-a3d7-c4e8d420d99b"
},
{
    "eventVersion": "1.01",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::012345678910:user/Alice",
        "accountId": "012345678910",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice"
    },
    "eventTime": "2014-04-19T00:17:06Z",
    "eventSource": "kinesis.amazonaws.com",
    "eventName": "DescribeStream",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "127.0.0.1",
    "userAgent": "aws-sdk-java/unknown-version Linux/x.xx",
    "requestParameters": {
        "streamName": "GoodStream"
    },
    "responseElements": null,
    "requestID": "f0944d86-c757-11e3-b4ae-25654b1d3136",
    "eventID": "0b2f1396-88af-4561-b16f-398f8eaea596"
},
{
    "eventVersion": "1.01",
    "userIdentity": {
        "type": "IAMUser",

```

```

        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::012345678910:user/Alice",
        "accountId": "012345678910",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice"
    },
    "eventTime": "2014-04-19T00:15:02Z",
    "eventSource": "kinesis.amazonaws.com",
    "eventName": "ListStreams",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "127.0.0.1",
    "userAgent": "aws-sdk-java/unknown-version Linux/x.xx",
    "requestParameters": {
        "limit": 10
    },
    "responseElements": null,
    "requestID": "a68541ca-c757-11e3-901b-cbcfe5b3677a",
    "eventID": "22a5fb8f-4e61-4bee-a8ad-3b72046b4c4d"
},
{
    "eventVersion": "1.01",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::012345678910:user/Alice",
        "accountId": "012345678910",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice"
    },
    "eventTime": "2014-04-19T00:17:07Z",
    "eventSource": "kinesis.amazonaws.com",
    "eventName": "DeleteStream",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "127.0.0.1",
    "userAgent": "aws-sdk-java/unknown-version Linux/x.xx",
    "requestParameters": {
        "streamName": "GoodStream"
    },
    "responseElements": null,
    "requestID": "f10cd97c-c757-11e3-901b-cbcfe5b3677a",
    "eventID": "607e7217-311a-4a08-a904-ec02944596dd"
},
{
    "eventVersion": "1.01",

```

```

    "userIdentity": {
      "type": "IAMUser",
      "principalId": "EX_PRINCIPAL_ID",
      "arn": "arn:aws:iam::012345678910:user/Alice",
      "accountId": "012345678910",
      "accessKeyId": "EXAMPLE_KEY_ID",
      "userName": "Alice"
    },
    "eventTime": "2014-04-19T00:15:03Z",
    "eventSource": "kinesis.amazonaws.com",
    "eventName": "SplitShard",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "127.0.0.1",
    "userAgent": "aws-sdk-java/unknown-version Linux/x.xx",
    "requestParameters": {
      "shardToSplit": "shardId-000000000000",
      "streamName": "GoodStream",
      "newStartingHashKey": "11111111"
    },
    "responseElements": null,
    "requestID": "a6e6e9cd-c757-11e3-901b-cbcfe5b3677a",
    "eventID": "dcd2126f-c8d2-4186-b32a-192dd48d7e33"
  },
  {
    "eventVersion": "1.01",
    "userIdentity": {
      "type": "IAMUser",
      "principalId": "EX_PRINCIPAL_ID",
      "arn": "arn:aws:iam::012345678910:user/Alice",
      "accountId": "012345678910",
      "accessKeyId": "EXAMPLE_KEY_ID",
      "userName": "Alice"
    },
    "eventTime": "2014-04-19T00:16:56Z",
    "eventSource": "kinesis.amazonaws.com",
    "eventName": "MergeShards",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "127.0.0.1",
    "userAgent": "aws-sdk-java/unknown-version Linux/x.xx",
    "requestParameters": {
      "streamName": "GoodStream",
      "adjacentShardToMerge": "shardId-000000000002",
      "shardToMerge": "shardId-000000000001"
    },
  },

```

```
        "responseElements": null,  
        "requestID": "e9f9c8eb-c757-11e3-bf1d-6948db3cd570",  
        "eventID": "77cf0d06-ce90-42da-9576-71986fec411f"  
    }  
]  
}
```

## Überwachen der Kinesis Client Library mit Amazon CloudWatch

Die [Kinesis Client Library](#) (KCL) für Amazon Kinesis Data Streams veröffentlicht in Ihrem Namen benutzerdefinierte Amazon-CloudWatch-Metriken, wobei der Name Ihrer KCL-Anwendung als Namespace verwendet wird. Sie können diese Metriken anzeigen, indem Sie zur [CloudWatch-Konsole](#) navigieren und Benutzerdefinierte Metriken auswählen. Weitere Informationen zu benutzerdefinierten Metriken finden Sie unter [Veröffentlichen von benutzerdefinierten Metriken](#) im Benutzerhandbuch für Amazon CloudWatch.

Für die vom KCL zu CloudWatch hochgeladenen Metriken wird eine Nominalgebühr erhoben, insbesondere gelten die Gebühren für benutzerdefinierte Amazon CloudWatch-Metriken und Amazon-CloudWatch-API-Anfragen. Weitere Informationen hierzu finden Sie unter [Amazon CloudWatch – Preise](#).

### Themen

- [Metriken und Namespaces](#)
- [Metrikstufen und Dimensionen](#)
- [Metrik-Konfiguration](#)
- [Liste der Metriken](#)

## Metriken und Namespaces

Der zum Hochladen von Kennzahlen verwendete Namespace ist der Anwendungsname, den Sie beim Starten von KCL angeben.

## Metrikstufen und Dimensionen

Es gibt zwei Möglichkeiten, um zu steuern, welche Metriken zu CloudWatch hochgeladen werden:

## Metrikstufen

Jeder Metrik ist eine individuelle Stufe zugewiesen. Wenn Sie die Berichtsstufe einer Metrik einrichten, werden Metriken, deren Stufe unter der Berichtsstufe liegt, nicht an CloudWatch gesendet. Die Stufen sind: NONE, SUMMARY und DETAILED. Die Standardeinstellung ist DETAILED, d. h., alle Metriken werden an CloudWatch gesendet. Die Berichtsstufe NONE bedeutet, dass keine Kennzahlen gesendet werden. Informationen dazu, welchen Metriken welche Stufen zugewiesen werden, finden Sie unter [Liste der Metriken](#).

## aktivierte Dimensionen

Jeder KCL-Metrik sind Dimensionen zugeordnet, die ebenfalls an CloudWatch gesendet werden. Wenn KCL in KCL 2.x für die Verarbeitung eines einzelnen Datenstroms konfiguriert ist, sind alle Metrikdimensionen (`Operation`, `ShardId` und `WorkerIdentifier`) standardmäßig aktiviert. Außerdem kann in KCL 2.x die Dimension `Operation` nicht deaktiviert werden, wenn KCL für die Verarbeitung eines einzelnen Datenstroms konfiguriert ist. Wenn KCL in KCL 2.x für die Verarbeitung mehrerer Datenströme konfiguriert ist, sind alle Metrikdimensionen (`Operation`, `ShardId`, `StreamId` und `WorkerIdentifier`) standardmäßig aktiviert. Wenn KCL für die Verarbeitung mehrerer Datenströme konfiguriert ist, können die Dimensionen `Operation` und `StreamId` in KCL 2.x nicht deaktiviert werden. Die Dimension `StreamId` ist nur für die Metriken pro Shard verfügbar.

In KCL 1.x sind nur die Dimensionen `Operation` und `ShardId` standardmäßig aktiviert, die Dimension `WorkerIdentifier` ist deaktiviert. In KCL 1.x kann die `Operation`-Dimension nicht deaktiviert werden.

Weitere Informationen zu den metrischen Dimensionen von CloudWatch finden im Abschnitt [Dimensionen](#) des Themas Amazon-CloudWatch-Konzepte im Amazon-CloudWatch-Benutzerhandbuch.

Wenn die Dimension `WorkerIdentifier` aktiviert ist, gilt: Wenn bei jedem Neustart eines bestimmten KCL-Auftragnehmers ein anderer Wert für die Eigenschaft Auftragnehmer-ID verwendet wird, werden neue Metriksätze mit neuen `WorkerIdentifier`-Dimensionswerten an CloudWatch gesendet. Wenn der `WorkerIdentifier`-Dimensionswert für verschiedene KCL-Worker-Neustarts identisch sein muss, müssen Sie explizit denselben Worker-ID-Wert bei der Initialisierung für jeden Worker angeben. Beachten Sie, dass der Worker-ID-Wert für jeden aktiven KCL-Worker unter allen KCL-Workern eindeutig sein muss.



## Metrik-Konfiguration

Metrikstufen und aktivierte Dimensionen können mit der `KinesisClientLibConfiguration`-Instance konfiguriert werden, die beim Start der KCL-Anwendung an den Worker übergeben wird. Im `MultiLangDaemon`-Fall können die Eigenschaften `metricsLevel` und `metricsEnabledDimensions` in der Eigenschaftendatei angegeben werden, die zum Start der KCL-Anwendung `MultiLangDaemon` verwendet wird.

Metrikstufen kann einer von drei Werten zugewiesen werden: `NONE`, `SUMMARY` oder `DETAILED`. Aktivierte Dimensionenwerte müssen durch Kommata getrennte Zeichenfolgen mit der Liste der Dimensionen sein, die für die CloudWatch-Metriken zulässig sind. Die von der KCL-Anwendung verwendeten Dimensionen sind `Operation`, `ShardId` und `WorkerIdentifier`.

## Liste der Metriken

Die folgenden Tabellen listen die KCL-Metriken, gruppiert nach Umfang und Operation, auf.

Themen

- [Metriken nach KCL-Anwendung](#)
- [Metriken pro Worker](#)
- [Metriken pro Shard](#)

## Metriken nach KCL-Anwendung

Diese Metriken werden über alle KCL-Auftragnehmer im Bereich der Anwendung, gemäß der Definition durch den Amazon-CloudWatch-Namespace, aggregiert.

Themen

- [InitializeTask](#)
- [ShutdownTask](#)
- [ShardSyncTask](#)
- [BlockOnParentTask](#)
- [PeriodicShardSyncManager](#)
- [MultistreamTracker](#)

## InitializeTask

Die Operation `InitializeTask` ist verantwortlich für die Initialisierung des Datensatzprozessors für die KCL-Anwendung. Zur Logik dieser Operation gehört der Abruf eines Shard-Iterators von Kinesis Data Streams und die Initialisierung des Datensatzprozessors.

Kennzahl	Beschreibung
<code>KinesisDataFetcher.getIterator.Success</code>	<p>Anzahl der erfolgreichen <code>GetShardIterator</code> -Operationen pro KCL-Anwendung.</p> <p>Metrikstufe: Detailed</p> <p>Einheiten: Anzahl</p>
<code>KinesisDataFetcher.getIterator.Time</code>	<p>Zeitbedarf für die <code>GetShardIterator</code> -Operation für die jeweilige KCL-Anwendung.</p> <p>Metrikstufe: Detailed</p> <p>Einheiten: Millisekunden</p>
<code>RecordProcessor.initialize.Time</code>	<p>Zeitbedarf für die Initialisierungsmethode des Datensatzprozessors.</p> <p>Metrikstufe: Summary</p> <p>Einheiten: Millisekunden</p>
Herzlichen Glückwunsch	<p>Anzahl der erfolgreichen Initialisierungen des Datensatzprozessors.</p> <p>Metrikstufe: Summary</p> <p>Einheiten: Anzahl</p>
Zeit	<p>Zeitbedarf des KCL-Workers für die Initialisierung des Datensatzprozessors.</p> <p>Metrikstufe: Summary</p> <p>Einheiten: Millisekunden</p>

## ShutdownTask

Die - ShutdownTask Operation initiiert die Abschaltsequenz für die Shard-Verarbeitung. Dies kann auftreten, da ein Shard geteilt oder zusammengeführt ist, oder wenn das Shard-Lease vom Worker verloren ging. In beiden Fällen wird die Datensatzprozessorfunktion shutdown() aufgerufen. Neue Shards werden auch erkannt, wenn ein Shard geteilt oder zusammengeführt wurde, was zur Erstellung von einem oder zwei neuen Shards führt.

Kennzahl	Beschreibung
CreateLease.Success	<p>Häufigkeit, mit der neue untergeordnete Shards nach dem Schließen des übergeordneten Shards erfolgreich der DynamoDB-Tabelle der KCL-Anwendung hinzugefügt wurden.</p> <p>Metrikstufe: Detailed</p> <p>Einheiten: Anzahl</p>
CreateLease.Time	<p>Zeitbedarf für das Hinzufügen von Informationen zu neuen untergeordneten Shards in der DynamoDB-Tabelle der KCL-Anwendung.</p> <p>Metrikstufe: Detailed</p> <p>Einheiten: Millisekunden</p>
UpdateLease.Success	<p>Anzahl der erfolgreichen endgültigen Checkpoints während des Schließens des Datensatzprozessors.</p> <p>Metrikstufe: Detailed</p> <p>Einheiten: Anzahl</p>
UpdateLease.Time	<p>Zeitbedarf für die Checkpoint-Operation während des Schließens des Datensatzprozessors.</p> <p>Metrikstufe: Detailed</p> <p>Einheiten: Millisekunden</p>
RecordProcessor.shutdown.Time	<p>Zeitbedarf für die Schließungsmethode des Datensatzprozessors.</p>

Kennzahl	Beschreibung
	Metrikstufe: Summary  Einheiten: Millisekunden
Herzlichen Glückwunsch	Anzahl der erfolgreichen Schließungsvorgänge.  Metrikstufe: Summary  Einheiten: Anzahl
Zeit	Zeitbedarf des KCL-Workers für den Schließungsvorgang.  Metrikstufe: Summary  Einheiten: Millisekunden

## ShardSyncTask

Die Operation `ShardSyncTask` erkennt Änderungen an Shard-Informationen für den Kinesis-Datenstrom, damit die KCL-Anwendung neue Shards verarbeiten kann.

Kennzahl	Beschreibung
<code>CreateLease.Success</code>	Anzahl der erfolgreichen Versuche zum Hinzufügen neuer Shard-Informationen in der DynamoDB-Tabelle der KCL-Anwendung.  Metrikstufe: Detailed  Einheiten: Anzahl
<code>CreateLease.Time</code>	Zeitbedarf für das Hinzufügen von Informationen zu neuen Shards in der DynamoDB-Tabelle der KCL-Anwendung.  Metrikstufe: Detailed  Einheiten: Millisekunden
Herzlichen Glückwunsch	Anzahl der erfolgreichen Shard Synchronisierungsoperationen.

Kennzahl	Beschreibung
	Metrikstufe: Summary Einheiten: Anzahl
Zeit	Zeitbedarf für die Shard-Synchronisierungsoperation. Metrikstufe: Summary Einheiten: Millisekunden

### BlockOnParentTask

Wenn der Shard geteilt oder mit anderen Shards zusammengeführt wird, werden neue untergeordnete Shards erstellt. Die Operation `BlockOnParentTask` stellt sicher, dass die Datensatzverarbeitung für die neuen Shards erst beginnt, wenn die übergeordneten Shards von dem KCL vollständig verarbeitet wurden.

Kennzahl	Beschreibung
Herzlichen Glückwunsch	Anzahl der erfolgreichen Prüfungen für den Abschluss der übergeordneten Shards. Metrikstufe: Summary Einheiten: Anzahl
Zeit	Zeitbedarf für den Abschluss der übergeordneten Shards. Metrikstufe: Summary Einheit: Millisekunden

### PeriodicShardSyncManager

`PeriodicShardSyncManager` hat die Aufgabe, die von der KCL-Konsumentenanzwendung verarbeiteten Datenströme zu prüfen, Datenströme mit Teil-Leases zu identifizieren und sie zur Synchronisierung weiterzuleiten.

Die folgenden Metriken sind verfügbar, wenn KCL für die Verarbeitung eines einzelnen Datenstroms konfiguriert ist (dann ist der Wert von NumStreamsToSync und NumStreamsWithPartialLeases auf 1 gesetzt) und wenn KCL für die Verarbeitung mehrerer Datenströme konfiguriert ist.

Kennzahl	Beschreibung
NumStreamsToSync	<p>Die Anzahl der Datenströme (pro AWS-Konto), die von der Verbrauchieranwendung verarbeitet werden und Teil-Leases enthalten und zur Synchronisation übergeben werden müssen.</p> <p>Metrikstufe: Summary</p> <p>Einheiten: Anzahl</p>
NumStreamsWithPartialLeases	<p>Die Anzahl der Datenströme (pro AWS-Konto), die die Verbrauchieranwendung verarbeitet und die Teil-Leases enthalten.</p> <p>Metrikstufe: Summary</p> <p>Einheiten: Anzahl</p>
Herzlichen Glückwunsch	<p>Die Anzahl der Male, in denen <code>PeriodicShardSyncManager</code> erfolgreich Teil-Leases in den Datenströmen identifizieren konnte, die die Verbrauchieranwendung verarbeitet.</p> <p>Metrikstufe: Summary</p> <p>Einheiten: Anzahl</p>
Zeit	<p>Die Zeit (in Millisekunden), die <code>PeriodicShardSyncManager</code> benötigt wird, um die Datenströme zu untersuchen, die von der Verbrauchieranwendung verarbeitet werden, um festzustellen, welche Datenströme eine Shard-Synchronisierung erfordern.</p> <p>Metrikstufe: Summary</p> <p>Einheiten: Millisekunden</p>

## MultistreamTracker

Die `MultistreamTracker`-Schnittstelle ermöglicht es Ihnen, KCL-Konsumenten Anwendungen zu erstellen, die mehrere Datenströme gleichzeitig verarbeiten können.

Kennzahl	Beschreibung
<code>DeletedStreams.Count</code>	Die Anzahl der in diesem Zeitraum gelöschten Datenströme.  Metrikstufe: Summary  Einheiten: Anzahl
<code>ActiveStreams.Count</code>	Die Anzahl der aktiven Datenströme, die verarbeitet werden.  Metrikstufe: Summary  Einheiten: Anzahl
<code>StreamsPendingDeletion.Count</code>	Die Anzahl der Datenströme, die aufgrund von <code>FormerStreamsLeaseDeletionStrategy</code> zur Löschung anstehen.  Metrikstufe: Summary  Einheiten: Anzahl

## Metriken pro Worker

Diese Metriken werden über alle Datensatzprozessoren, die Daten von einem Kinesis-Datenstrom verbrauchen, etwa einer Amazon-EC2-Instance, aggregiert.

### Themen

- [RenewAllLeases](#)
- [TakeLeases](#)

### RenewAllLeases

Die Operation `RenewAllLeases` erneuert periodisch alle Shard-Leases, die zu einer bestimmten Worker-Instance gehören.

Kennzahl	Beschreibung
RenewLease.Success	Anzahl der erfolgreichen Lease-Erneuerungen durch den Worker.  Metrikstufe: Detailed  Einheiten: Anzahl
RenewLease.Time	Zeitbedarf für die Operation der Lease-Erneuerung.  Metrikstufe: Detailed  Einheiten: Millisekunden
CurrentLeases	Anzahl der Shard-Leases, die einem Worker nach der Erneuerung aller Leases gehören.  Metrikstufe: Summary  Einheiten: Anzahl
LostLeases	Anzahl der Shard-Leases, die nach einem Versuch zur Erneuerung aller Leases eines Workers verloren gingen.  Metrikstufe: Summary  Einheiten: Anzahl
Herzlichen Glückwunsch	Häufigkeit erfolgreicher Lease-Erneuerungsoperationen für den Worker.  Metrikstufe: Summary  Einheiten: Anzahl
Zeit	Zeitbedarf für die Erneuerung aller Leases für den Worker.  Metrikstufe: Summary  Einheiten: Millisekunden



## TakeLeases

Die Operation TakeLeases sorgt für den Ausgleich aller Datensatzverarbeitungsvorgänge zwischen allen KCL-Workern. Wenn der aktuelle KCL-Worker weniger Shard-Leases als erforderlich hat, werden Shard-Leases von einem anderen, überlasteten, Worker genommen.

Kennzahl	Beschreibung
ListLeases.Success	Häufigkeit, mit der alle Shard-Leases erfolgreich aus der DynamoDB-Tabelle der KCL-Anwendung abgerufen wurden.  Metrikstufe: Detailed  Einheiten: Anzahl
ListLeases.Time	Zeitbedarf für das Abrufen aller Shard-Leases aus der DynamoDB-Tabelle der KCL-Anwendung.  Metrikstufe: Detailed  Einheiten: Millisekunden
TakeLease.Success	Häufigkeit der erfolgreichen Übernahme von Shard-Leases von anderen KCL-Workern.  Metrikstufe: Detailed  Einheiten: Anzahl
TakeLease.Time	Zeitbedarf für die Aktualisierung der Lease-Tabelle mit Leases des Workers.  Metrikstufe: Detailed  Einheiten: Millisekunden
NumWorkers	Gesamtzahl der Worker, wie von einem spezifischen Worker definiert.  Metrikstufe: Summary  Einheiten: Anzahl

Kennzahl	Beschreibung
NeededLeases	<p>Anzahl der Shard-Leases, die der aktuelle Worker für eine ausgeglichene Shard-Verarbeitungslast benötigt.</p> <p>Metrikstufe: Detailed</p> <p>Einheiten: Anzahl</p>
LeasesToTake	<p>Anzahl der Leases, die der Worker zu übernehmen versuchen wird.</p> <p>Metrikstufe: Detailed</p> <p>Einheiten: Anzahl</p>
TakenLeases	<p>Anzahl der von dem Worker erfolgreich übernommenen Leases.</p> <p>Metrikstufe: Summary</p> <p>Einheiten: Anzahl</p>
TotalLeases	<p>Gesamtzahl der Shards, die die KCL-Anwendung verarbeitet.</p> <p>Metrikstufe: Detailed</p> <p>Einheiten: Anzahl</p>
ExpiredLeases	<p>Gesamtzahl der Shards, die nicht von einem Worker verarbeitet werden, wie von einem spezifischen Worker identifiziert.</p> <p>Metrikstufe: Summary</p> <p>Einheiten: Anzahl</p>
Herzlichen Glückwunsch	<p>Häufigkeit des erfolgreichen Abschlusses der TakeLeases -Operation.</p> <p>Metrikstufe: Summary</p> <p>Einheiten: Anzahl</p>

Kennzahl	Beschreibung
Zeit	Zeitbedarf für die TakeLeases -Operation für einen Worker.  Metrikstufe: Summary  Einheiten: Millisekunden

## Metriken pro Shard

Diese Metriken werden über einen einzigen Datensatzprozessor aggregiert.

### ProcessTask

Die ProcessTask-Operation ruft [GetRecords](#) mit der aktuellen Iterator-Position auf, um Datensätze aus dem Stream abzurufen, und ruft die Datensatzprozessorfunktion processRecords auf.

Kennzahl	Beschreibung
KinesisDataFetcher .getRecords.Success	Anzahl der erfolgreichen GetRecords -Operationen pro Kinesis-Datenstrom-Shard.  Metrikstufe: Detailed  Einheiten: Anzahl
KinesisDataFetcher .getRecords.Time	Zeitbedarf für die GetRecords -Operation für den Kinesis-Datenstrom-Shard.  Metrikstufe: Detailed  Einheiten: Millisekunden
UpdateLease.Success	Anzahl der erfolgreichen Checkpoints durch den Datensatzprozessor für einen bestimmten Shard.  Metrikstufe: Detailed  Einheiten: Anzahl
UpdateLease.Time	Zeitbedarf für jede Checkpoint-Operation eines bestimmten Shards.

Kennzahl	Beschreibung
	<p>Metrikstufe: Detailed</p> <p>Einheiten: Millisekunden</p>
DataBytesProcessed	<p>Gesamtgröße der verarbeiteten Datensätze in Byte bei jedem <code>ProcessTask</code> -Aufruf.</p> <p>Metrikstufe: Summary</p> <p>Einheiten: Byte</p>
RecordsProcessed	<p>Anzahl der bei jedem <code>ProcessTask</code> -Aufruf verarbeiteten Datensätze.</p> <p>Metrikstufe: Summary</p> <p>Einheiten: Anzahl</p>
ExpiredIterator	<p>Anzahl der erhaltenen <code>ExpiredIteratorException</code>-Vorgänge beim Aufruf von <code>GetRecords</code> .</p> <p>Metrikstufe: Summary</p> <p>Einheiten: Anzahl</p>
MillisBehindLatest	<p>Zeitraum, um den der aktuelle Iterator seit dem letzten Datensatz (Tip) in dem Shard verspätet ist. Dieser Wert ist kleiner oder gleich der Zeitdifferenz zwischen dem letzten Datensatz in einer Antwort und dem aktuellen Zeitpunkt. Dies ist eine genauere Angabe dafür, wie weit ein Shard vom Tip entfernt ist, als durch den Vergleich der Zeitstempel im letzten Antwortdatensatz möglich ist. Dieser Wert gilt für den letzten Datensatzstapel und ist kein Durchschnittswert aller Zeitstempel in den Datensätzen.</p> <p>Metrikstufe: Summary</p> <p>Einheiten: Millisekunden</p>

Kennzahl	Beschreibung
RecordProcessor.processRecords.Time	<p>Zeitbedarf für die processRecords -Methode des Datensatzprozessors.</p> <p>Metrikstufe: Summary</p> <p>Einheiten: Millisekunden</p>
Herzlichen Glückwunsch	<p>Anzahl der erfolgreichen Verarbeitungsaufgabenoperationen.</p> <p>Metrikstufe: Summary</p> <p>Einheiten: Anzahl</p>
Zeit	<p>Zeitbedarf für die Verarbeitungsaufgabenoperation.</p> <p>Metrikstufe: Summary</p> <p>Einheiten: Millisekunden</p>

## Überwachen der Kinesis Producer Library mit Amazon CloudWatch

Die [Kinesis Producer Library](#) (KPL) für Amazon Kinesis Data Streams veröffentlicht in Ihrem Namen benutzerdefinierte Amazon-CloudWatch-Metriken. Sie können diese Metriken anzeigen, indem Sie zur [CloudWatch-Konsole](#) navigieren und Benutzerdefinierte Metriken auswählen. Weitere Informationen zu benutzerdefinierten Metriken finden Sie unter [Veröffentlichen von benutzerdefinierten Metriken](#) im Benutzerhandbuch für Amazon CloudWatch.

Für die vom KPL zu CloudWatch hochgeladenen Metriken wird eine Nominalgebühr erhoben, insbesondere gelten die Gebühren für benutzerdefinierte Amazon CloudWatch-Metriken und Amazon-CloudWatch-API-Anfragen. Weitere Informationen hierzu finden Sie unter [Amazon CloudWatch – Preise](#). Die Erfassung lokaler Metriken ist nicht mit CloudWatch-Gebühren verbunden.

### Themen

- [Metriken, Dimensionen und Namespaces](#)
- [Metrikstufe und Granularität](#)
- [Lokaler Zugriff und Amazon-CloudWatch-Upload](#)

- [Liste der Metriken](#)

## Metriken, Dimensionen und Namespaces

Sie können einen Anwendungsnamen festlegen, wenn Sie das KPL starten, das dann beim Hochladen von Metriken als Teil des Namespaces verwendet wird. Dies ist optional. Das KPL bietet einen Standardwert, wenn kein Anwendungsname eingerichtet wurde.

Sie können das KPL auch so konfigurieren, dass beliebige zusätzliche Dimensionen zu den Metriken hinzugefügt werden. Dies ist nützlich, wenn Sie in Ihren CloudWatch-Metriken detailliertere Daten wünschen. Sie können beispielsweise den Host-Namen als Dimension hinzufügen, damit Sie ungleichmäßige Lastverteilungen in der Flotte erkennen. Alle KPL-Konfigurationseinstellungen sind unveränderlich, diese zusätzlichen Dimensionen können nach der Initialisierung der KPL-Instance also nicht mehr geändert werden.

## Metrikstufe und Granularität

Es gibt zwei Möglichkeiten, um zu steuern, wie viele Metriken zu CloudWatch hochgeladen werden:

### Metrikstufe

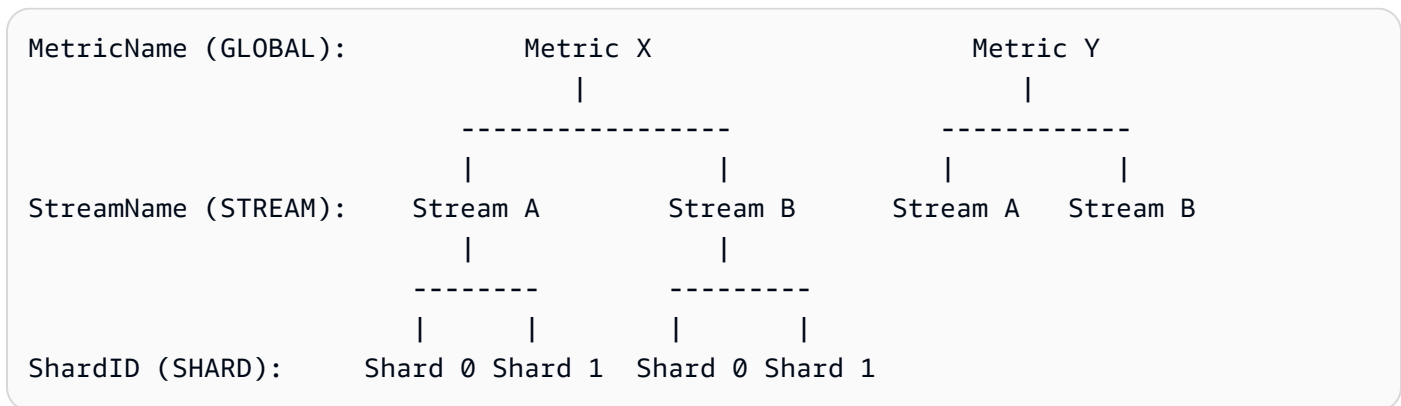
Dies ist ein grobes Maß für die Wichtigkeit der Metrik. Jede Metrik wird eine Stufe zugewiesen. Wenn Sie eine Stufe einstellen, werden Metriken mit darunter liegenden Stufen nicht an CloudWatch gesendet. Die Stufen sind NONE, SUMMARY und DETAILED. Die Standardeinstellung ist DETAILED; das heißt, alle Metriken. NONE bedeutet keine Metriken, dieser Stufe werden daher keine Metriken zugewiesen.

### Granularität

Dies steuert, ob deine Metrik mit weiteren Granularitätsstufen ausgegeben werden soll. Die Stufen sind GLOBAL, STREAM und SHARD. Die Standardeinstellung ist SHARD mit den detailliertesten Metriken.

Bei Auswahl von SHARD werden Metriken mit dem Stream-Namen und der Shard-ID als Dimensionen ausgegeben. Dazu wird die gleiche Metrik auch nur mit der Stream-Name-Dimension und ohne die Stream-Name-Dimension ausgegeben. Dies bedeutet, dass für eine bestimmte Metrik zwei Streams mit jeweils zwei Shards sieben CloudWatch-Metriken produzieren: jeweils eine für jeden Shard, eine für jeden Stream und eine insgesamt; alle beschreiben die gleichen statistischen Werte, jedoch mit unterschiedlicher Granularität. Eine Illustration finden Sie im folgenden Diagramm.

Die unterschiedlichen Granularitätsebenen bilden eine Hierarchie, und alle Metriken in dem System bilden einen Baum, dessen Wurzel die Metrikenamen bilden:



Nicht alle Metriken sind auf Shard-Ebene verfügbar; einige sind ihrer Natur nach global oder auf die Stream-Ebene beschränkt. Sie werden nicht auf Shard-Ebene produziert, auch wenn Sie Kennzahlen auf Shard-Ebene aktiviert haben (Metric Y im Diagramm oben).

Wenn Sie eine zusätzliche Dimension angeben, müssen Sie Werte für `tuple:<DimensionName, DimensionValue, Granularity>` angeben. Die Granularität legt fest, wo die benutzerdefinierte Dimension in die Hierarchie eingefügt wird: GLOBAL bedeutet, dass die zusätzliche Dimension hinter dem Kennzahlnamen eingefügt wird, STREAM bedeutet, dass sie hinter dem Stream-Namen und SHARD, dass sie hinter der Shard-ID eingefügt wird. Wenn mehrere Dimensionen pro Granularitätsstufe angegeben sind, werden sie in der angegebenen Reihenfolge eingefügt.

## Lokaler Zugriff und Amazon-CloudWatch-Upload

Metriken für die aktuelle KPL-Instance sind lokal in Echtzeit verfügbar; Sie können dazu jederzeit das KPL abfragen. Das KPL berechnet lokal Summe, Durchschnitt, Minimum, Maximum und Anzahl jeder Metrik, wie in CloudWatch.

Sie erhalten statistische Werte, die seit dem Start des Programms bis zum aktuellen Zeitpunkt kumuliert wurden, oder Sie können ein laufendes Zeitfenster über die letzten N Sekunden verwenden, wobei N eine ganze Zahl zwischen 1 und 60 ist.

Alle Metriken sind für den Upload zu CloudWatch verfügbar. Dies ist besonders nützlich für die Aggregation von Daten über mehrere Hosts sowie Überwachungs- und Alarmvorgänge. Diese Funktionalität ist nicht lokal verfügbar.

Wie zuvor beschrieben, können Sie wählen, welche Metriken mit der Metrik-Ebene und Granularität hochgeladen werden sollen. Metriken, die nicht hochgeladen wurde, sind lokal verfügbar.

Das Hochladen einzelner Datenpunkte ist nicht sinnvoll, da dies zu Millionen von Uploads pro Sekunde führen könnte, wenn der Datenverkehr hoch ist. Aus diesem Grund aggregiert das KPL Metriken lokal in 1-Minuten-Buckets und lädt einmal pro Minute pro aktivierter Metrik ein Statistikobjekt zu CloudWatch.

## Liste der Metriken

Kennzahl	Beschreibung
UserRecordsReceived	<p>Anzahl der logischen Benutzerdatensätze, die vom KPL-Core für Put-Operationen erhalten wurden. Auf Shard-Ebene nicht verfügbar.</p> <p>Metrikstufe: Detailed</p> <p>Einheit: Anzahl</p>
UserRecordsPending	<p>Regelmäßige Stichprobe, wie viele Benutzer Datensätze derzeit ausstehen. Ein Datensatz ist ausstehend, wenn er entweder derzeit gepuffert ist und darauf wartet, gesendet zu werden, oder wenn er direkt zu dem Backend-Service gesendet wird. Auf Shard-Ebene nicht verfügbar.</p> <p>Das KPL bietet eine dedizierte Methode zum Abrufen dieser Metrik auf globaler Ebene, damit Kunden ihre Put-Rate verwalten können.</p> <p>Metrikstufe: Detailed</p> <p>Einheit: Anzahl</p>
UserRecordsPut	<p>Anzahl der logischen Benutzerdatensätze mit erfolgreichen Put-Operationen.</p> <p>Das KPL zählt für diese Metrik fehlgeschlagene Datensätze nicht. Dadurch kann der Durchschnittswert die Erfolgsrate, die Anzahl die Gesamtzahl der Versuche und die Differenz zwischen beiden die Anzahl der Fehlschläge angeben.</p>



Kennzahl	Beschreibung
	<p>Metrikstufe: Summary</p> <p>Einheit: Anzahl</p>
UserRecordsDataPut	<p>Bytes in den logischen Benutzer-Datensätzen mit erfolgreichen Put-Operationen.</p> <p>Metrikstufe: Detailed</p> <p>Einheit: Byte</p>
KinesisRecordsPut	<p>Anzahl der Datensätze der Kinesis Data Streams mit erfolgreichen Put-Operationen (jeder Datensatz der Kinesis Data Streams kann mehrere Benutzerdatensätze enthalten).</p> <p>Das KPL gibt eine Null für fehlgeschlagene Datensätze aus. Dadurch kann der Durchschnittswert die Erfolgsrate, die Anzahl die Gesamtzahl der Versuche und die Differenz zwischen beiden die Anzahl der Fehlschläge angeben.</p> <p>Metrikstufe: Summary</p> <p>Einheit: Anzahl</p>
KinesisRecordsDataPut	<p>Bytes in den Datensätzen der Kinesis Data Streams.</p> <p>Metrikstufe: Detailed</p> <p>Einheit: Byte</p>

Kennzahl	Beschreibung
ErrorsByCode	<p>Anzahl der einzelnen Arten von Fehlercodes. Dies führt eine weitere Dimension von <code>ErrorCode</code> zusätzlich zu den normalen Dimensionen ein, wie z. B. <code>StreamName</code> und <code>ShardId</code>. Nicht jeder Fehler kann zu einem Shard verfolgt werden. Die Fehler, die nicht verfolgt werden können, werden nur auf Stream- oder globaler Ebene ausgegeben. Diese Metrik erfasst Informationen zu Dingen wie Drosselung, Änderungen der Shard-Zuordnung, internen Fehlern, nicht verfügbaren Services, Zeitüberschreitungen u. dgl.</p> <p>API-Fehler der Kinesis Data Streams werden einmal pro Datensatz der Kinesis Data Streams gezählt. Mehrere Benutzerdatensätze innerhalb eines Datensatzes von Kinesis Data Streams erzeugen keine Mehrfachzählungen.</p> <p>Metrikstufe: Summary</p> <p>Einheit: Anzahl</p>
AllErrors	<p>Dies wird von den gleichen Fehlern ausgelöst wie Fehler nach Code, jedoch ohne Unterscheidung zwischen den Typen. Dies ist nützlich zur allgemeinen Überwachung der Fehlerrate, ohne dass eine manuelle Summe der Anzahlen der verschiedenen Arten von Fehlern erforderlich ist.</p> <p>Metrikstufe: Summary</p> <p>Einheit: Anzahl</p>

Kennzahl	Beschreibung
<code>RetriesPerRecord</code>	<p>Anzahl der pro Benutzerdatensatz durchgeführten erneuten Versuche. Für Datensätze, die bei einem Versuch erfolgreich waren, wird Null ausgegeben.</p> <p>Die Daten werden in dem Moment ausgegeben, in dem ein Benutzerdatensatz abgeschlossen wird (wenn er entweder erfolgreich war, oder wenn kein Wiederholungsversuch mehr möglich ist). Wenn die Aufbewahrungsdauer eines Datensatz einen hohen Wert hat, kann diese Metrik stark verzögert werden.</p> <p>Metrikstufe: Detailed</p> <p>Einheit: Anzahl</p>
<code>BufferingTime</code>	<p>Der Zeitraum zwischen dem Eingang eines Benutzerdatensatzes beim KPL und seinem Abgang zum Backend. Diese Informationen werden an den Benutzer pro Datensatz zurückgegeben, sind jedoch auch als aggregierte Statistik verfügbar.</p> <p>Metrikstufe: Summary</p> <p>Einheit: Millisekunden</p>
<code>Request Time</code>	<p>Die Zeit zum Ausführen von <code>PutRecordsRequests</code>.</p> <p>Metrikstufe: Detailed</p> <p>Einheit: Millisekunden</p>
<code>User Records per Kinesis Record</code>	<p>Die Anzahl der logischen Benutzerdatensätze, die in einem einzigen Datensatz der Kinesis Data Streams zusammengefasst sind.</p> <p>Metrikstufe: Detailed</p> <p>Einheit: Anzahl</p>

Kennzahl	Beschreibung
Amazon Kinesis Records per PutRecord sRequest	<p>Die Anzahl der Datensätze von Kinesis Data Streams, die zu einem einzigen PutRecordsRequest aggregiert wurden. Auf Shard-Ebene nicht verfügbar.</p> <p>Metrikstufe: Detailed</p> <p>Einheit: Anzahl</p>
User Records per PutRecord sRequest	<p>Die Gesamtanzahl der Benutzerdatensätze in einer PutRecord sRequest . Diese Zahl entspricht ungefähr dem Produkt der beiden vorherigen Metriken. Auf Shard-Ebene nicht verfügbar.</p> <p>Metrikstufe: Detailed</p> <p>Einheit: Anzahl</p>

# Sicherheit in Amazon Kinesis Data Streams

Cloud-Sicherheit bei AWS hat höchste Priorität. Als - AWS Kunde profitieren Sie von einer Rechenzentrums- und Netzwerkarchitektur, die entwickelt wurde, um die Anforderungen der sicherheitssensibelsten Organisationen zu erfüllen.

Sicherheit ist eine geteilte Verantwortung zwischen AWS und Ihnen. Das [Modell der geteilten Verantwortung](#) beschreibt dies als Sicherheit der Cloud und Sicherheit in der Cloud:

- Sicherheit der Cloud – AWS ist für den Schutz der Infrastruktur verantwortlich, die AWS Services in der AWS Cloud ausführt. stellt Ihnen AWS außerdem Services bereit, die Sie sicher nutzen können. Die Wirksamkeit unserer Sicherheitsfunktionen wird regelmäßig von externen Prüfern im Rahmen des [AWS -Compliance-Programms getestet und überprüft](#). Weitere Informationen zu den Compliance-Programmen für Kinesis Data Streams finden Sie unter [Durch das Compliance-Programm abgedeckte AWS -Services](#).
- Sicherheit in der Cloud – Ihre Verantwortung wird durch den - AWS Service bestimmt, den Sie verwenden. In Ihre Verantwortung fallen außerdem weitere Faktoren, wie z. B. die Vertraulichkeit der Daten, die Anforderungen Ihrer Organisation sowie geltende Gesetze und Vorschriften.

Diese Dokumentation hilft Ihnen zu verstehen, wie Sie das Modell der geteilten Verantwortung bei der Verwendung von Kinesis Data Streams einsetzen können. Die folgenden Themen veranschaulichen, wie Sie Kinesis Data Streams zur Erfüllung Ihrer Sicherheits- und Compliance-Ziele konfigurieren können. Sie erfahren auch, wie Sie andere - AWS Services verwenden, die Ihnen bei der Überwachung und Sicherung Ihrer Ressourcen von Kinesis Data Streams helfen können.

## Themen

- [Datenschutz in Amazon Kinesis Data Streams](#)
- [Steuern des Zugriffs auf Ressourcen von Amazon Kinesis Data Streams mithilfe von IAM](#)
- [Konformitätsprüfung für Amazon Kinesis Data Streams](#)
- [Resilienz in Amazon Kinesis Data Streams](#)
- [Infrastruktursicherheit in Kinesis Data Streams](#)
- [Bewährte Methoden für die Sicherheit für Kinesis Data Streams](#)

# Datenschutz in Amazon Kinesis Data Streams

Die serverseitige Verschlüsselung mit AWS Key Management Service (AWS KMS)-Schlüsseln erleichtert es Ihnen, strenge Datenverwaltungsanforderungen zu erfüllen, indem Sie Ihre Daten im Ruhezustand in Amazon Kinesis Data Streams verschlüsseln.

## Note

Wenn Sie für den Zugriff auf AWS über eine Befehlszeilenschnittstelle oder eine API FIPS-140-2-validierte kryptografische Module benötigen, verwenden Sie einen FIPS-Endpunkt. Weitere Informationen über verfügbare FIPS-Endpunkte finden Sie unter [Federal Information Processing Standard \(FIPS\) 140-2](#).

## Themen

- [Was bedeutet eine serverseitige Verschlüsselung in Kinesis Data Streams?](#)
- [Kosten, Regionen und Leistungsanforderungen](#)
- [Wie beginne ich mit der serverseitigen Verschlüsselung?](#)
- [Erstellen und Verwenden benutzergenerierter KMS-Masterschlüssel](#)
- [Berechtigungen für die Nutzung benutzergenerierter KMS-Masterschlüssel](#)
- [Überprüfung und Fehlerbehebung bei KMS-Schlüssel Berechtigungen](#)
- [Verwenden von Amazon Kinesis Data Streams für Schnittstellen-VPC-Endpunkten](#)

## Was bedeutet eine serverseitige Verschlüsselung in Kinesis Data Streams?

Die serverseitige Verschlüsselung ist eine Funktion in Amazon Kinesis Data Streams, die Daten automatisch verschlüsselt, bevor sie sich im Ruhezustand befinden, indem sie einen von Ihnen angegebenen AWS KMS -Kundenmasterschlüssel (CMK) verwenden. Die Daten werden verschlüsselt, bevor sie in die Speicherschicht des Kinesis-Streams geschrieben werden. Nach Abruf aus dem Speicher werden Sie entschlüsselt. Dadurch sind Ihre ruhenden Daten innerhalb des Services Kinesis Data Streams verschlüsselt. Sie erfüllen so die strengen regulatorischen Anforderungen und erhöhen die Sicherheit Ihrer Daten.

Durch die serverseitige Verschlüsselung müssen Ihre Kinesis-Stream-Produzenten und -Verbraucher keine Masterschlüssel oder kryptographische Operationen verwalten. Ihre Daten werden automatisch

verschlüsselt, wenn sie in den Service Kinesis Data Streams ein- und ausgehen, sodass Ihre Daten im Ruhezustand verschlüsselt werden. AWS KMS stellt alle Masterschlüssel bereit, die von der serverseitigen Verschlüsselungsfunktion verwendet werden. AWS KMS erleichtert die Verwendung eines CMK für Kinesis, der von verwaltet wird AWS, eines AWS KMS benutzerdefinierten CMK oder eines Masterschlüssels, der in den AWS KMS Service importiert wird.

 Note

Bei der serverseitigen Verschlüsselung werden eingehende Daten nur dann verschlüsselt, wenn die Funktion aktiviert ist. Bereits in einem nicht verschlüsselten Stream vorhandene Daten werden nach Aktivierung der serverseitigen Verschlüsselung nicht verschlüsselt.

Wenn Sie Ihre Datenströme verschlüsseln und den Zugriff für andere Prinzipal(e) freigeben, müssen Sie sowohl in der Schlüsselrichtlinie für den AWS KMS Schlüssel als auch in den IAM-Richtlinien im externen Konto die Berechtigung erteilen. Weitere Informationen finden Sie unter [Benutzern in anderen Konten die Verwendung eines KMS-Schlüssels erlauben](#).

Wenn Sie die serverseitige Verschlüsselung für einen Datenstrom mit AWS verwaltetem KMS-Schlüssel aktiviert haben und den Zugriff über eine Ressourcenrichtlinie freigeben möchten, müssen Sie zur Verwendung von vom Kunden verwalteten Schlüsseln (CMK) wechseln, wie im Folgenden gezeigt:

## Edit encryption for test\_encryption

### Encryption [Info](#)

**Enable server-side encryption**

Kinesis Data Stream uses AWS Key Management Service (KMS) to encrypt your data. You can choose the AWS managed customer master key (CMK) to encrypt your data or specify a customer-managed CMK.

**Use AWS managed CMK**

The AWS managed CMK (aws/kinesis) in your account is created, managed, and used on your behalf by Kinesis Data Streams.

**Use customer-managed CMK**

Customer-managed CMKs in your AWS account are created, owned, and managed by you.

Customer-managed CMK in KMS

Choose customer-managed CMK ▼



Create key ↗

Cancel

Save changes

Darüber hinaus müssen Sie Ihren freigebenden Prinzipal-Entitäten Zugriff auf Ihren CMK gewähren, indem Sie die kontoübergreifenden KMS-Freigabe-Funktionen nutzen. Stellen Sie sicher, dass Sie auch die IAM-Richtlinien für die freigebenden Prinzipal-Entitäten ändern. Weitere Informationen finden Sie unter [Benutzern in anderen Konten die Verwendung eines KMS-Schlüssels erlauben](#).

## Kosten, Regionen und Leistungsanforderungen

Wenn Sie die serverseitige Verschlüsselung anwenden, unterliegen Sie den AWS KMS API-Nutzungs- und Schlüsselkosten. Im Gegensatz zu benutzerdefinierten KMS-Masterschlüsseln wird der benutzerdefinierte (Default) aws/kinesis-Masterschlüssel (Customer Master Key, CMK) kostenlos zur Verfügung gestellt. Sie müssen jedoch weiterhin die Kosten für die API-Nutzung tragen, die Amazon Kinesis Data Streams für Sie verursacht.

Kosten für die API-Nutzung fallen für alle CMKs an, einschließlich benutzerdefinierten. Kinesis Data Streams ruft AWS KMS ungefähr alle fünf Minuten ab, wenn der Datenschlüssel gewechselt wird. In einem Monat mit 30 Tagen sollten die Gesamtkosten für AWS KMS API-Aufrufe, die von einem Kinesis-Stream initiiert werden, unter ein paar Dollar liegen. Diese Kosten skalieren mit der Anzahl der Benutzeranmeldeinformationen, die Sie auf Ihren Datenproduzenten und -konsumenten verwenden, da jede Benutzeranmeldeinformation einen eindeutigen API-Aufruf von erfordert AWS KMS. Wenn Sie eine IAM-Rolle für die Authentifizierung verwenden, führt jeder



Rollenübernahmeaufruf zu eindeutigen Benutzeranmeldeinformationen. Um KMS-Kosten zu sparen, können Sie Benutzeranmeldeinformationen zwischenspeichern, die vom Rollenübernahmeaufruf zurückgegeben werden.

Im Folgenden werden die Kosten pro Ressource aufgeführt:

## Schlüssel

- Der CMK für Kinesis, der von verwaltet wird AWS (Alias = `aws/kinesis`), ist kostenlos.
- Von Benutzern generierte KMS-Schlüssel unterliegen den KMS-Schlüssel-Kosten. Weitere Informationen finden Sie unter [AWS Key Management Service – Preise](#).

Kosten für die API-Nutzung fallen für alle CMKs an, einschließlich benutzerdefinierten. Kinesis Data Streams ruft KMS ungefähr alle fünf Minuten ab, wenn der Datenschlüssel gewechselt wird. In einem Monat mit 30 Tagen sollten die Gesamtkosten für KMS-API-Aufrufe, die von einem Kinesis-Datenstrom initiiert werden, nur wenige Dollar betragen. Bitte beachten Sie, dass diese Kosten mit der Anzahl der Benutzeranmeldeinformationen skalieren, die Sie auf Ihren Datenproduzenten und -konsumenten verwenden, da jede Benutzeranmeldeinformation einen eindeutigen API-Aufruf an AWS KMS erfordert. Wenn Sie die IAM-Rolle für die Authentifizierung verwenden, führt jede `assume-role-call` zu eindeutigen Benutzeranmeldeinformationen, und Sie möchten möglicherweise Benutzeranmeldeinformationen zwischenspeichern, die vom zurückgegeben werden, um KMS-Kosten `assume-role-call` zu sparen.

## KMS-API-Nutzung

Für jeden verschlüsselten Stream ruft der Kinesis-Dienst den AWS KMS -Service etwa 12 Mal alle 5 Minuten auf, wenn aus TIP gelesen und ein einziger IAM-Konto/Benutzerzugriffsschlüssel für Leser und Schreiber verwendet wird. Das Nichtlesen von TIP kann zu höheren AWS KMS Serviceaufrufen führen. API-Anfragen zur Generierung neuer Datenverschlüsselungsschlüssel unterliegen AWS KMS Nutzungskosten. Weitere Informationen finden Sie unter [Preise für AWS Key Management Service – Verwendung](#).

## Verfügbarkeit der serverseitigen Verschlüsselung nach Region

Derzeit ist die serverseitige Verschlüsselung von Kinesis-Streams in allen Regionen verfügbar, die für Kinesis Data Streams unterstützt werden, einschließlich AWS GovCloud (USA-West) und den Regionen China. Weitere Informationen zu den unterstützten Regionen für Kinesis Data Streams finden Sie unter <https://docs.aws.amazon.com/general/latest/gr/ak.html>.

## Performanceaspekte

Aufgrund des Serviceaufwands bei der Verschlüsselung erhöht die serverseitige Verschlüsselung die typische Latenzzeit von `PutRecord`, `PutRecords` und `GetRecords` um 100µs.

## Wie beginne ich mit der serverseitigen Verschlüsselung?

Der einfachste Weg, um mit der serverseitigen Verschlüsselung zu beginnen, ist die Verwendung der AWS Management Console und des Amazon Kinesis-KMS-Serviceschlüssels, `aws/kinesis`.

Im Folgenden wird die Aktivierung der serverseitigen Verschlüsselung für einen Kinesis-Stream beschrieben.

So aktivieren Sie die serverseitige Verschlüsselung für einen Kinesis-Stream

1. Melden Sie sich bei der an AWS Management Console und öffnen Sie die [Amazon Kinesis Data Streams-Konsole](#).
2. Wählen oder erstellen Sie einen Kinesis-Stream in der AWS Management Console.
3. Wählen Sie die Registerkarte Details aus.
4. Wählen Sie unter Server-side encryption (Serverseitige Verschlüsselung) die Option edit (Bearbeiten) aus.
5. Wenn Sie einen benutzergenerierten KMS-Masterschlüssel verwenden möchten, stellen Sie sicher, dass der (standardmäßige) `aws/kinesis-KMS-Masterschlüssel` ausgewählt ist. Dies ist die KMS-Masterschlüssel, der vom Kinesis-Service generiert wurde. Wählen Sie Enabled (Aktiviert) und anschließend Save (Speichern) aus.

### Note

Der Standard-Hauptschlüssel des Kinesis-Service ist kostenlos. Die API-Aufrufe von Kinesis an den AWS KMS Service unterliegen jedoch den KMS-Nutzungskosten.

6. Der Stream ist vorübergehend im Zustand ausstehend. Sobald sich der Stream in einem aktiven Zustand befindet und die Verschlüsselung aktiviert ist, werden alle in den Stream eingeleiteten Daten mit dem von Ihnen ausgewählten KMS-Masterschlüssel verschlüsselt.
7. Um die serverseitige Verschlüsselung zu deaktivieren, wählen Sie unter Serverseitige Verschlüsselung in der die Option Deaktiviert AWS Management Console und dann Speichern aus.

## Erstellen und Verwenden benutzergenerierter KMS-Masterschlüssel

In diesem Abschnitt wird beschrieben, wie Sie anstelle des von Amazon Kinesis verwalteten Masterschlüssels Ihre eigenen KMS-Masterschlüssel erstellen und verwenden.

### Erstellen benutzergenerierter KMS-Masterschlüssel

Informationen zum Erstellen eigener Masterschlüssel finden Sie unter [Erstellen von Schlüsseln](#) im AWS Key Management Service -Entwicklerhandbuch. Nachdem Sie Schlüssel für Ihr Konto erstellt haben, gibt der Service Kinesis Data Streams diese Schlüssel über die Liste KMS-Masterschlüsse zurück.

### Verwenden benutzergenerierter KMS-Masterschlüssel

Nachdem die richtigen Berechtigungen auf Ihre Verbraucher, Produzenten und Administratoren angewendet wurden, können Sie benutzerdefinierte KMS-Masterschlüssel in Ihrem eigenen AWS Konto oder einem anderen AWS Konto verwenden. Alle KMS-Masterschlüssel in Ihrem Konto werden in der Liste für KMS-Masterschlüssel AWS Management Console angezeigt.

Für die Nutzung benutzerdefinierter KMS-Schlüssel eines anderen Kontos benötigen Sie entsprechende Berechtigungen. Darüber hinaus müssen Sie den ARN des KMS-Masterschlüssels in der AWS Management Console im ARN-Eingabefeld eingeben.

## Berechtigungen für die Nutzung benutzergenerierter KMS-Masterschlüssel

Bevor Sie die serverseitige Verschlüsselung mit einem benutzergenerierten KMS-Masterschlüssel verwenden können, müssen Sie AWS KMS Schlüsselrichtlinien konfigurieren, um die Verschlüsselung von Streams und die Verschlüsselung und Entschlüsselung von Stream-Datensätzen zu ermöglichen. Beispiele und weitere Informationen zu AWS KMS Berechtigungen finden Sie unter [AWS KMS-API-Berechtigungen: Referenz zu Aktionen und Ressourcen](#).

#### Note

Für die Nutzung des Standard-Serviceschlüssels für die Verschlüsselung sind keine benutzerdefinierten IAM-Berechtigungen erforderlich.

Ehe Sie benutzergenerierte KMS-Masterschlüssel verwenden, müssen Sie sicherstellen, dass Ihre Kinesis-Stream-Produzenten und -Konsumenten (IAM-Prinzipale) Benutzer im Sinne der Richtlinie für KMS-Masterschlüssel sind. Andernfalls schlägt das Schreiben in und das Lesen aus dem

Stream fehl. Dies kann zu einem Datenverlust, einer verspäteten Verarbeitung oder abgestürzten Anwendungen führen. Sie können Berechtigungen für KMS-Schlüssel mit IAM-Richtlinien verwalten. Weitere Informationen finden Sie unter [Verwenden von IAM-Richtlinien mit AWS KMS](#).

## Berechtigungen für Produzenten – Beispiel

Die Kinesis-Stream-Produzenten benötigen die Berechtigung `kms:GenerateDataKey`.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kms:GenerateDataKey"
      ],
      "Resource": "arn:aws:kms:us-west-2:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab"
    },
    {
      "Effect": "Allow",
      "Action": [
        "kinesis:PutRecord",
        "kinesis:PutRecords"
      ],
      "Resource": "arn:aws:kinesis:*:123456789012:MyStream"
    }
  ]
}
```

## Berechtigungen für Konsumenten – Beispiel

Ihre Kinesis-Stream-Konsumenten benötigen die Berechtigung `kms:Decrypt`.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kms:Decrypt"
      ],
    },
  ]
}
```

```
    "Resource": "arn:aws:kms:us-  
west-2:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab"  
  },  
  {  
    "Effect": "Allow",  
    "Action": [  
      "kinesis:GetRecords",  
      "kinesis:DescribeStream"  
    ],  
    "Resource": "arn:aws:kinesis:*:123456789012:MyStream"  
  }  
]
```

Amazon Managed Service für Apache Flink und AWS Lambda verwenden Rollen, um Kinesis-Streams zu nutzen. Stellen Sie sicher, dass Sie die `kms:Decrypt`-Berechtigung für die Rollen erteilen, die diese Konsumenten innehaben.

## Stream-Administrator-Berechtigungen

Kinesis-Stream-Administratoren benötigen eine Autorisierung für den Aufruf von `kms:List*` und `kms:DescribeKey*`.

## Überprüfung und Fehlerbehebung bei KMS-Schlüssel Berechtigungen

Nachdem Sie die Verschlüsselung für einen Kinesis-Stream aktiviert haben, empfehlen wir IhnenputRecord, den Erfolg Ihrer putRecords-, - und -getRecordsAufrufe mit den folgenden Amazon- CloudWatch Metriken zu überwachen:

- PutRecord.Success
- PutRecords.Success
- GetRecords.Success

Weitere Informationen finden Sie unter [Überwachung von Amazon Kinesis Data Streams](#).

## Verwenden von Amazon Kinesis Data Streams für Schnittstellen-VPC-Endpunkten

Sie können einen Schnittstellen-VPC-Endpunkt verwenden, um zu verhindern, dass Datenverkehr zwischen Ihrer Amazon VPC und Kinesis Data Streams das Amazon-Netzwerk verlässt.

Schnittstellen-VPC-Endpunkte benötigen kein Internet-Gateway, kein NAT-Gerät, keine VPN-Verbindung und keine - AWS Direct Connect Verbindung. Schnittstellen-VPC-Endpunkte werden von AWS unterstützt PrivateLink, einer - AWS Technologie, die private Kommunikation zwischen - AWS Services über eine Elastic-Network-Schnittstelle mit privaten IPs in Ihrer Amazon VPC ermöglicht. Weitere Informationen finden Sie unter [Amazon Virtual Private Cloud](#) und [Schnittstellen-VPC-Endpunkte \(AWS PrivateLink\)](#).

## Themen

- [Verwenden von Schnittstellen-VPC-Endpunkten für Kinesis Data Streams](#)
- [Steuern des Zugriffs auf VPC-Endpunkte für Kinesis Data Streams](#)
- [Verfügbarkeit von VPC-Endpunktrichtlinien für Kinesis Data Streams](#)

## Verwenden von Schnittstellen-VPC-Endpunkten für Kinesis Data Streams

Für den Einstieg müssen Sie die Einstellungen für Ihre Streams, Produzenten oder Verbraucher nicht ändern. Erstellen Sie einfach einen Schnittstellen-VPC-Endpunkt, um Ihren Datenverkehr für Kinesis Data Streams von und zu Ihren Amazon-VPC-Ressourcen über den Schnittstellen-VPC-Endpunkt fließen zu lassen. Weitere Informationen finden Sie unter [Erstellen eines Schnittstellenendpunkts](#).

Die Kinesis Producer Library (KPL) und die Kinesis Consumer Library (KCL) rufen AWS Services wie Amazon CloudWatch und Amazon DynamoDB entweder über öffentliche Endpunkte oder über VPC-Endpunkte mit privater Schnittstelle auf, je nachdem, was gerade verwendet wird. Beispiel: Wenn Ihre KCL-Anwendung in einer VPC mit aktivierten DynamoDB-Schnittstellen-VPC-Endpunkten ausgeführt wird, fließen Aufrufe zwischen DynamoDB und Ihrer KCL-Anwendung über den Schnittstellen-VPC-Endpunkt.

## Steuern des Zugriffs auf VPC-Endpunkte für Kinesis Data Streams

VPC-Endpunktrichtlinien ermöglichen Ihnen, den Zugriff zu steuern, indem Sie entweder eine Richtlinie an einen VPC-Endpunkt anfügen oder zusätzliche Felder in einer Richtlinie verwenden, die einem IAM-Benutzer, einer IAM-Gruppe oder einer IAM-Rolle zugeordnet sind, um den Zugriff darauf zu beschränken, dass er nur über den angegebenen VPC-Endpunkt erfolgt. Diese Richtlinien können verwendet werden, um den Zugriff auf bestimmte Streams auf einen bestimmten VPC-Endpunkt zu beschränken, wenn sie in Verbindung mit den IAM-Richtlinien verwendet werden, um nur den Zugriff auf Kinesis-Datenstromaktionen über den angegebenen VPC-Endpunkt zu gewähren.

Es folgen Beispiele für Endpunktrichtlinien für den Zugriff auf Kinesis-Daten-Streams.

- Beispiel für eine VPC-Richtlinie: schreibgeschützter Zugriff – diese Beispielrichtlinie kann einem VPC-Endpoint zugeordnet werden. Weitere Informationen finden Sie unter [Steuern des Zugriffs auf Amazon VPC-Ressourcen](#). Sie beschränkt Aktionen auf das Auflisten und Beschreiben eines Kinesis-Daten-Streams über den VPC-Endpoint, dem er zugeordnet ist.

```
{
  "Statement": [
    {
      "Sid": "ReadOnly",
      "Principal": "*",
      "Action": [
        "kinesis:List*",
        "kinesis:Describe*"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

- Beispiel für eine VPC-Richtlinie: Beschränken des Zugriffs auf einen bestimmten Kinesis-Daten-Stream – diese Beispielrichtlinie kann einem VPC-Endpoint zugeordnet werden. Sie schränkt den Zugriff auf einen bestimmten Daten-Stream über den VPC-Endpoint ein, dem er zugeordnet ist.

```
{
  "Statement": [
    {
      "Sid": "AccessToSpecificDataStream",
      "Principal": "*",
      "Action": "kinesis:*",
      "Effect": "Allow",
      "Resource": "arn:aws:kinesis:us-east-1:123456789012:stream/MyStream"
    }
  ]
}
```

- Beispiel für eine IAM-Richtlinie: Beschränken des Zugriffs auf einen bestimmten Stream von einem bestimmten VPC-Endpoint aus – diese Beispielrichtlinie kann einem IAM-Benutzer, einer IAM-

Rolle oder einer IAM-Gruppe zugeordnet werden. Sie beschränkt den Zugriff auf einen bestimmten Kinesis-Daten-Stream auf einen bestimmten VPC-Endpunkt.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AccessFromSpecificEndpoint",
      "Action": "kinesis:*",
      "Effect": "Deny",
      "Resource": "arn:aws:kinesis:us-east-1:123456789012:stream/MyStream",
      "Condition": { "StringNotEquals" : { "aws:sourceVpce": "vpce-11aa22bb" } }
    }
  ]
}
```

## Verfügbarkeit von VPC-Endpunktrichtlinien für Kinesis Data Streams

VPC-Endpunkte der Schnittstelle Kinesis Data Streams mit Richtlinien werden in den folgenden Regionen unterstützt:

- Europa (Paris)
- Europa (Irland)
- USA Ost (Nord-Virginia)
- Europa (Stockholm)
- USA Ost (Ohio)
- Europa (Frankfurt)
- Südamerika (São Paulo)
- Europa (London)
- Asien-Pazifik (Tokio)
- USA West (Nordkalifornien)
- Asien-Pazifik (Singapur)
- Asien-Pazifik (Sydney)
- China (Peking)



- China (Ningxia)
- Asia Pacific (Hong Kong)
- Naher Osten (Bahrain)
- Naher Osten (VAE)
- Europa (Milan)
- Afrika (Kapstadt)
- Asien-Pazifik (Mumbai)
- Asien-Pazifik (Seoul)
- Kanada (Zentral)
- USA West (Oregon) außer usw2-az4
- AWS GovCloud (USA-Ost)
- AWS GovCloud (USA-West)
- Asien-Pazifik (Osaka)
- Europa (Zürich)
- Asien-Pazifik (Hyderabad)

## Steuern des Zugriffs auf Ressourcen von Amazon Kinesis Data Streams mithilfe von IAM

AWS Identity and Access Management (IAM) haben Sie folgende Möglichkeiten:

- Erstellen von Benutzern und Gruppen unter Ihrem AWS Konto
- Weisen Sie jedem Benutzer in Ihrem AWS Konto eindeutige Sicherheitsanmeldeinformationen zu
- Steuern der Berechtigungen der einzelnen Benutzer zum Ausführen von Aufgaben mithilfe von - AWS Ressourcen
- Benutzern in einem anderen AWS Konto erlauben, Ihre AWS Ressourcen gemeinsam zu nutzen
- Erstellen Sie Rollen für Ihr AWS Konto und definieren Sie die Benutzer oder Services, die sie übernehmen können
- Verwenden vorhandener Identitäten für Ihr Unternehmen, um Berechtigungen zum Ausführen von Aufgaben mithilfe von - AWS Ressourcen zu erteilen

Wenn Sie IAM zusammen mit Kinesis Data Streams verwenden, können Sie steuern, ob Benutzer im Unternehmen Aufgaben mit bestimmten API-Aktionen von Kinesis Data Streams ausführen und spezifische AWS -Ressourcen verwenden können.

Wenn Sie eine Anwendung mit der Kinesis Client Library (KCL) entwickeln, muss Ihre Richtlinie Berechtigungen für Amazon DynamoDB und Amazon enthalten CloudWatch. Die KCL verwendet DynamoDB, um Statusinformationen für die Anwendung CloudWatch zu verfolgen und in Ihrem Namen KCL-Metriken an zu CloudWatch senden. Weitere Informationen zur KCL finden Sie unter [Entwicklung von KCL 1.x-Verbrauchern](#).

Weitere Informationen zu IAM finden Sie unter:

- [AWS Identity and Access Management \(IAM\)](#)
- [Erste Schritte](#)
- [IAM Benutzerhandbuch](#)

Weitere Informationen zu IAM und Amazon DynamoDB finden Sie unter [Verwendung von IAM zur Steuerung des Zugriffs auf Amazon-DynamoDB-Ressourcen](#) im Amazon DynamoDB-Entwicklerhandbuch.

Weitere Informationen zu IAM und Amazon CloudWatch finden Sie unter [Steuern des Benutzerzugriffs auf Ihr - AWS Konto](#) im Amazon- CloudWatch Benutzerhandbuch.

Inhalt

- [Richtliniensyntax](#)
- [Aktionen für Kinesis Data Streams](#)
- [Amazon-Ressourcennamen \(ARNs\) für Kinesis Data Streams](#)
- [Beispielrichtlinien für Kinesis Data Streams](#)
- [Freigabe Ihres Datenstroms für ein anderes Konto](#)
- [Konfigurieren einer - AWS Lambda Funktion zum Lesen aus Kinesis Data Streams in einem anderen Konto](#)
- [Freigabe des Zugriffs mithilfe von ressourcenbasierten Richtlinien](#)

## Richtliniensyntax

Eine IAM-Richtlinie ist ein JSON-Dokument, das eine oder mehrere Anweisungen enthält. Jede Anweisung ist folgendermaßen strukturiert:

```
{
  "Statement": [{
    "Effect": "effect",
    "Action": "action",
    "Resource": "arn",
    "Condition": {
      "condition": {
        "key": "value"
      }
    }
  }
]
```

Eine Anweisung kann aus verschiedenen Elementen bestehen:

- **Effect:** Der effect-Wert kann Allow oder Deny lauten. IAM-Benutzer verfügen standardmäßig nicht über die Berechtigung zur Verwendung von Ressourcen und API-Aktionen. Daher werden alle Anfragen abgelehnt. Dieser Standardwert kann durch eine explizite Zugriffserlaubnis überschrieben werden. Eine explizite Zugriffsverweigerung überschreibt jedwede Zugriffserlaubnis.
- **Action:** Mit action wird die API-Aktion spezifiziert, für die Sie Berechtigungen erteilen oder verweigern.
- **Resource:** Die von einer Aktion betroffene Ressource. Um eine Ressource in der Anweisung anzugeben, benötigen Sie deren Amazon-Ressourcennamen (ARN).
- **Condition:** Bedingungen sind optional. Mit ihrer Hilfe können Sie bestimmen, wann Ihre Richtlinie wirksam wird.

Beim Erstellen und Verwalten von IAM-Richtlinien sollten Sie den [IAM-Richtliniengenerator](#) und den [IAM-Richtliniensimulator](#) verwenden.

## Aktionen für Kinesis Data Streams

In einer IAM-Richtlinienanweisung können Sie jede API-Aktion von jedem Service, der IAM unterstützt, angeben. Bei Kinesis Data Streams setzen Sie folgendes Präfix vor den Namen der

API-Aktion: `kinesis:.` Beispiel: `kinesis:CreateStream`, `kinesis:ListStreams` und `kinesis:DescribeStreamSummary`.

Um mehrere Aktionen in einer einzigen Anweisung anzugeben, trennen Sie sie wie folgt durch Kommata:

```
"Action": ["kinesis:action1", "kinesis:action2"]
```

Sie können auch mehrere Aktionen mittels Platzhaltern angeben. Beispielsweise können Sie alle Aktionen festlegen, deren Name mit dem Wort "Get" beginnt:

```
"Action": "kinesis:Get*"
```

Um alle Operationen von Kinesis Data Streams anzugeben, verwenden Sie den Platzhalter \* folgendermaßen:

```
"Action": "kinesis:*"
```

Die vollständige Liste der Kinesis-Data-Streams-API-Aktionen finden Sie unter [Amazon-Kinesis-API-Referenz](#).

## Amazon-Ressourcennamen (ARNs) für Kinesis Data Streams

Jede IAM-Richtlinienanweisung gilt für die Ressourcen, die Sie mithilfe ihrer ARNs angegeben haben.

Verwenden Sie das folgende ARN-Ressourcenformat für Kinesis-Datenströme:

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

Beispielsweise:

```
"Resource": arn:aws:kinesis:*:111122223333:stream/my-stream
```

## Beispielrichtlinien für Kinesis Data Streams

Die folgenden Beispielrichtlinien zeigen, wie Sie den Benutzerzugriff auf Kinesis-Datenströme steuern könnten.

## Example 1: Allow users to get data from a stream

### Example

Diese Richtlinie erlaubt einem Benutzer oder einer Gruppe, die Operationen `DescribeStreamSummary`, `GetShardIterator` und `GetRecords` auf dem angegebenen Stream und `ListStreams` auf einem beliebigen Stream auszuführen. Diese Richtlinie könnte auf Benutzer angewendet werden, die Daten aus einem spezifischen Stream abrufen können sollten.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kinesis:Get*",
        "kinesis:DescribeStreamSummary"
      ],
      "Resource": [
        "arn:aws:kinesis:us-east-1:111122223333:stream/stream1"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "kinesis:ListStreams"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

## Example 2: Allow users to add data to any stream in the account

### Example

Diese Richtlinie erlaubt einem Benutzer oder einer Gruppe, die Operation `PutRecord` mit einem beliebigen Stream des Kontos zu verwenden. Diese Richtlinie könnte auf Benutzer angewendet werden, die Daten zu allen Streams in einem Konto hinzufügen können sollten.

```
{
```

```
"Version": "2012-10-17",
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "kinesis:PutRecord"
    ],
    "Resource": [
      "arn:aws:kinesis:us-east-1:111122223333:stream/*"
    ]
  }
]
```

### Example 3: Allow any Kinesis Data Streams action on a specific stream

#### Example

Diese Richtlinie erlaubt einem Benutzer oder einer Gruppe, eine beliebige Operation von Kinesis Data Streams auf dem angegebenen Stream zu verwenden. Diese Richtlinie könnte auf Benutzer angewendet werden, die administrative Kontrolle über einen bestimmten Stream haben sollten.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "kinesis:*",
      "Resource": [
        "arn:aws:kinesis:us-east-1:111122223333:stream/stream1"
      ]
    }
  ]
}
```

### Example 4: Allow any Kinesis Data Streams action on any stream

#### Example

Diese Richtlinie erlaubt einem Benutzer oder einer Gruppe, eine beliebige Operation von Kinesis Data Streams auf einem beliebigen Stream in einem Konto zu verwenden. Da diese Richtlinie vollen Zugriff auf alle Ihre Streams gewährt, sollten Sie sie auf Administratoren beschränken.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "kinesis:*",
      "Resource": [
        "arn:aws:kinesis:*:111122223333:stream/*"
      ]
    }
  ]
}
```

## Freigabe Ihres Datenstroms für ein anderes Konto

Fügen Sie eine [ressourcenbasierte Richtlinie](#) an Ihren Datenstrom an, um einem anderen Konto, einem IAM-Benutzer oder einer IAM-Rolle Zugriff zu gewähren. Ressourcenbasierte Richtlinien sind JSON-Richtliniendokumente, die Sie an eine Ressource, beispielsweise einen Datenstrom, anfügen. Diese Richtlinien erteilen dem [angegebenen Prinzipal](#) die Berechtigung zum Ausführen bestimmter Aktionen für diese Ressource und definieren, unter welchen Bedingungen diese gilt. Eine Richtlinie kann mehrere Anweisungen enthalten. Sie müssen in einer ressourcenbasierten Richtlinie einen Prinzipal angeben. Prinzipale können Konten, Benutzer, Rollen, Verbundbenutzer oder - AWS Services umfassen. Sie können Richtlinien in der Konsole von Kinesis Data Streams, API oder SDK konfigurieren.

Beachten Sie, dass die Freigabe des Zugriffs für registrierte Verbraucher, wie beispielsweise [Enhanced Fan-Out](#), eine Richtlinie sowohl für den Datenstrom-ARN als auch für den Verbraucher-ARN erfordert.

## Aktivierung des kontoübergreifenden Zugriffs

Um kontoübergreifenden Zugriff zu ermöglichen, können Sie ein gesamtes Konto oder IAM-Entitäten in einem anderen Konto als Prinzipal in einer ressourcenbasierten Richtlinie angeben. Durch das Hinzufügen eines kontoübergreifenden Auftraggebers zu einer ressourcenbasierten Richtlinie ist nur die halbe Vertrauensbeziehung eingerichtet. Wenn sich der Prinzipal und die Ressource in separaten AWS Konten befinden, müssen Sie auch eine identitätsbasierte Richtlinie verwenden, um dem Prinzipal Zugriff auf die Ressource zu gewähren. Wenn jedoch eine ressourcenbasierte Richtlinie Zugriff auf einen Prinzipal in demselben Konto gewährt, ist keine zusätzliche identitätsbasierte Richtlinie erforderlich.

Weitere Informationen zur Verwendung ressourcenbasierter Richtlinien für den kontoübergreifenden Zugriff finden Sie unter [Kontoübergreifender Ressourcenzugriff in IAM](#).

Datenstrom-Administratoren können AWS Identity and Access Management Richtlinien verwenden, um festzulegen, wer Zugriff auf was hat. Das heißt, welcher Prinzipal kann Aktionen für welche Ressourcen und unter welchen Bedingungen ausführen. Das Element `Action` einer JSON-Richtlinie beschreibt die Aktionen, mit denen Sie den Zugriff in einer Richtlinie zulassen oder verweigern können. Richtlinienaktionen haben in der Regel denselben Namen wie die zugehörige AWS API-Operation.

Aktionen von Kinesis Data Streams, die freigegeben werden können:

Aktion	Zugriffsebene
<a href="#">DescribeStreamConsumer</a>	Konsument
<a href="#">DescribeStreamSummary</a>	Datenstrom
<a href="#">GetRecords</a>	Datenstrom
<a href="#">GetShardIterator</a>	Datenstrom
<a href="#">ListShards</a>	Datenstrom
<a href="#">PutRecord</a>	Datenstrom
<a href="#">PutRecords</a>	Datenstrom
<a href="#">SubscribeToShard</a>	Konsument

Im Folgenden finden Sie Beispiele für die Verwendung einer ressourcenbasierten Richtlinie, um kontoübergreifenden Zugriff auf Ihren Datenstrom oder registrierten Verbraucher zu gewähren.

Um eine kontoübergreifende Aktion durchzuführen, müssen Sie den Stream-ARN für den Zugriff auf den Datenstrom und den Verbraucher-ARN für den Zugriff registrierter Verbraucher angeben.



## Beispielhafte ressourcenbasierte Richtlinien für Kinesis Data Streams

Die Freigabe eines registrierten Verbrauchers erfordert aufgrund der erforderlichen Maßnahmen sowohl eine Datenstromrichtlinie als auch eine Verbraucherrichtlinie.

### Note

Nachfolgend finden Sie Beispiele für gültige Werte für `Principal`:

- `{"AWS": "123456789012"}`
- IAM-Benutzer – `{"AWS": "arn:aws:iam::123456789012:user/user-name"}`
- IAM-Rolle – `{"AWS": ["arn:aws:iam::123456789012:role/role-name"]}`
- Mehrere Prinzipale (kann eine Kombination aus Konto, Benutzer, Rolle sein) – `{"AWS": ["123456789012", "123456789013", "arn:aws:iam::123456789012:user/user-name"]}`

### Example 1: Write access to the data stream

#### Example

```
{
  "Version": "2012-10-17",
  "Id": "__default_write_policy_ID",
  "Statement": [
    {
      "Sid": "writestatement",
      "Effect": "Allow",
      "Principal": {
        "AWS": "Account12345"
      },
      "Action": [
        "kinesis:DescribeStreamSummary",
        "kinesis:ListShards",
        "kinesis:PutRecord",
        "kinesis:PutRecords"
      ],
      "Resource": "arn:aws:kinesis:us-east-2:123456789012:stream/
datastreamABC"
    }
  ]
}
```

```
}

```

## Example 2: Read access to the data stream

### Example

```
{
  "Version": "2012-10-17",
  "Id": "__default_sharedthroughput_read_policy_ID",
  "Statement": [
    {
      "Sid": "sharedthroughputreadstatement",
      "Effect": "Allow",
      "Principal": {
        "AWS": "Account12345"
      },
      "Action": [
        "kinesis:DescribeStreamSummary",
        "kinesis:ListShards",
        "kinesis:GetRecords",
        "kinesis:GetShardIterator"
      ],
      "Resource": "arn:aws:kinesis:us-east-2:123456789012:stream/
datastreamABC"
    }
  ]
}
```

## Example 3: Share enhanced fan-out read access to a registered consumer

### Example

#### Erklärung zur Datenstromrichtlinie:

```
{
  "Version": "2012-10-17",
  "Id": "__default_sharedthroughput_read_policy_ID",
  "Statement": [
    {
      "Sid": "consumerreadstatement",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::Account12345:role/role-name"
      },
    },
  ],
}
```

```

        "Action": [
            "kinesis:DescribeStreamSummary",
            "kinesis:ListShards"
        ],
        "Resource": "arn:aws:kinesis:us-east-2:123456789012:stream/
datastreamABC"
    }
]
}

```

### Erklärung zur Verbraucherrichtlinie:

```

{
  "Version": "2012-10-17",
  "Id": "__default_efo_read_policy_ID",
  "Statement": [
    {
      "Sid": "eforeadstatement",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::Account12345:role/role-name"
      },
      "Action": [
        "kinesis:DescribeStreamConsumer",
        "kinesis:SubscribeToShard"
      ],
      "Resource": "arn:aws:kinesis:us-east-2:123456789012:stream/
datastreamABC/consumer/consumerDEF:1674696300"
    }
  ]
}

```

Platzhalter (\*) werden für Aktionen oder Prinzipal-Felder nicht unterstützt, um das Prinzip der geringsten Berechtigung aufrecht zu erhalten.

## Programmgesteuertes Verwalten der Richtlinie für Ihren Datenstrom

Außerhalb der verfügt AWS Management Console Kinesis Data Streams über drei APIS für die Verwaltung Ihrer Datenstromrichtlinie:

- [PutResourcePolicy](#)

- [GetResourcePolicy](#)
- [DeleteResourcePolicy](#)

Verwenden Sie `PutResourcePolicy`, um eine Richtlinie für einen Datenstrom oder Verbraucher anzufügen oder zu überschreiben. Verwenden Sie `GetResourcePolicy`, um eine Richtlinie für den angegebenen Datenstrom oder Verbraucher zu überprüfen und anzuzeigen. Verwenden Sie `DeleteResourcePolicy`, um eine Richtlinie für den angegebenen Datenstrom oder Verbraucher zu löschen.

## Richtlinienbeschränkungen

Für die Ressourcenrichtlinien von Kinesis Data Streams gelten folgende Einschränkungen:

- Um das Prinzip der geringsten Berechtigung aufrecht zu erhalten, werden Platzhalter (\*) für Aktionen oder Prinzipal-Abschnitte nicht unterstützt.
- AWS Serviceprinzipale werden für Prinzipale nicht unterstützt, um potenzielle [verwirrte Stellvertreter](#) zu verhindern.
- Verbund-Prinzipale werden nicht unterstützt.
- Kanonische Benutzer-IDs werden nicht unterstützt.
- Die Größe der Richtlinie darf 20 KB nicht überschreiten.

## Freigabe des Zugriffs auf verschlüsselte Daten

Wenn Sie die serverseitige Verschlüsselung für einen Datenstrom mit AWS verwaltetem KMS-Schlüssel aktiviert haben und den Zugriff über eine Ressourcenrichtlinie freigeben möchten, müssen Sie zur Verwendung von vom Kunden verwalteten Schlüsseln (CMK) wechseln. Weitere Informationen finden Sie unter [Was bedeutet eine serverseitige Verschlüsselung in Kinesis Data Streams?](#). Darüber hinaus müssen Sie Ihren freigebenden Prinzipal-Entitäten Zugriff auf Ihren CMK gewähren, indem Sie die kontoübergreifenden KMS-Freigabe-Funktionen nutzen. Stellen Sie sicher, dass Sie auch die IAM-Richtlinien für die freigebenden Prinzipal-Entitäten ändern. Weitere Informationen finden Sie unter [Benutzern in anderen Konten die Verwendung eines KMS-Schlüssels erlauben](#).

## Konfigurieren einer - AWS Lambda Funktion zum Lesen aus Kinesis Data Streams in einem anderen Konto

Ein Beispiel dafür, wie Sie eine Lambda-Funktion zum Lesen von Kinesis Data Streams in einem anderen Konto konfigurieren können, finden Sie unter [Freigeben des Zugriffs mit kontoübergreifenden AWS Lambda Funktionen](#).

### Freigabe des Zugriffs mithilfe von ressourcenbasierten Richtlinien

#### Note

Das Aktualisieren einer vorhandenen ressourcenbasierten Richtlinie bedeutet, dass die bestehende Richtlinie ersetzt wird. Stellen Sie daher sicher, dass Sie in Ihrer neuen Richtlinie alle erforderlichen Informationen angeben.

### Freigeben des Zugriffs mit kontoübergreifenden AWS Lambda Funktionen

#### Lambda-Operator

1. Gehen Sie zur [IAM-Konsole](#), um eine IAM-Rolle zu erstellen, die als [Lambda-Ausführungsrolle](#) für Ihre AWS Lambda Funktion verwendet wird. Fügen Sie die verwaltete IAM-Richtlinie hinzu `AWSLambdaKinesisExecutionRole`, die über die erforderlichen Kinesis Data Streams- und Lambda-Aufrufberechtigungen verfügt. Diese Richtlinie gewährt auch Zugriff auf alle potenziellen Ressourcen von Kinesis Data Streams, auf die Sie möglicherweise Zugriff haben.
2. Erstellen Sie in der [AWS Lambda Konsole](#) eine - AWS Lambda Funktion, [um Datensätze in einem Kinesis-Data-Streams-Datenstrom zu verarbeiten](#), und wählen Sie während der Einrichtung für die Ausführungsrolle die Rolle aus, die Sie im vorherigen Schritt erstellt haben.
3. Stellen Sie dem Ressourcenbesitzer in Kinesis Data Streams die Ausführungsrolle zur Konfiguration der Ressourcenrichtlinie bereit.
4. Schließen Sie die Einrichtung der Lambda-Funktion ab.

#### Besitzer der Ressource in Kinesis Data Streams

1. Rufen Sie die kontoübergreifende Lambda-Ausführungsrolle ab, welche die Lambda-Funktion aufruft.

2. Wählen Sie in der Konsole von Amazon Kinesis Data Streams den Datenstrom aus. Wählen Sie die Registerkarte Datenstrom-Freigabe und anschließend die Schaltfläche Freigaberichtlinie erstellen, um den visuellen Richtlinieneditor zu starten. Um einen registrierten Verbraucher innerhalb eines Datenstroms freizugeben, wählen Sie den Verbraucher aus und wählen Sie dann Freigaberichtlinie erstellen aus. Sie können die JSON-Richtlinie auch direkt schreiben.
3. Geben Sie die kontoübergreifende Lambda-Ausführungsrolle als Prinzipal und die genauen Aktionen von Kinesis Data Streams an, auf die Sie Zugriff gewähren. Stellen Sie sicher, dass Sie die Aktion `kinesis:DescribeStream` einbeziehen. Weitere Informationen über beispielhafte Ressourcenrichtlinien für Kinesis Data Streams finden Sie unter [Beispielhafte ressourcenbasierte Richtlinien für Kinesis Data Streams](#).
4. Wählen Sie Richtlinie erstellen oder verwenden Sie die [PutResourcePolicy](#), um die Richtlinie an Ihre Ressource anzuhängen.

## Freigabe des Zugriffs mit kontoübergreifenden KCL-Verbrauchern

- Wenn Sie KCL 1.x verwenden, stellen Sie sicher, dass Sie KCL 1.15.0 oder höher verwenden.
- Wenn Sie KCL 2.x verwenden, stellen Sie sicher, dass Sie KCL 2.5.3 oder höher verwenden.

### KCL-Operator

1. Stellen Sie dem Ressourcenbesitzer Ihren IAM-Benutzer oder Ihre IAM-Rolle zur Ausführung der KCL-Anwendung zur Verfügung.
2. Fragen Sie den Ressourcenbesitzer nach dem Datenstrom oder dem Verbraucher-ARN.
3. Stellen Sie sicher, dass Sie den bereitgestellten Stream-ARN als Teil Ihrer KCL-Konfiguration angeben.
  - Für KCL 1.x: Verwenden Sie den [KinesisClientLibConfiguration](#) Konstruktor und geben Sie den Stream-ARN an.
  - Für KCL 2.x: Sie können nur den Stream-ARN oder [StreamTracker](#) die Kinesis Client Library angeben [ConfigsBuilder](#). StreamTrackerGeben Sie für den Stream-ARN und die Erstellungs-Epoche aus der DynamoDB-Lease-Tabelle an, die von der Bibliothek generiert wird. Wenn Sie aus einem freigegebenen registrierten Verbraucher wie Enhanced Fan-Out lesen möchten, verwenden Sie StreamTracker und geben Sie auch den Verbraucher-ARN an.

## Besitzer der Ressource in Kinesis Data Streams

1. Rufen Sie den kontoübergreifenden IAM-Benutzer oder die IAM-Rolle ab, welcher die KCL-Anwendung ausführt.
2. Wählen Sie in der Konsole von Amazon Kinesis Data Streams den Datenstrom aus. Wählen Sie die Registerkarte Datenstrom-Freigabe und anschließend die Schaltfläche Freigaberichtlinie erstellen, um den visuellen Richtlinieneditor zu starten. Um einen registrierten Verbraucher innerhalb eines Datenstroms freizugeben, wählen Sie den Verbraucher aus und wählen Sie dann Freigaberichtlinie erstellen aus. Sie können die JSON-Richtlinie auch direkt schreiben.
3. Geben Sie den IAM-Benutzer oder die IAM-Rolle der kontoübergreifenden KCL-Anwendung als Prinzipal und die genauen Aktionen in Kinesis Data Streams an, auf die Sie Zugriff gewähren. Weitere Informationen über beispielhafte Ressourcenrichtlinien für Kinesis Data Streams finden Sie unter [Beispielhafte ressourcenbasierte Richtlinien für Kinesis Data Streams](#).
4. Wählen Sie Richtlinie erstellen oder verwenden Sie die [PutResourcePolicy](#), um die Richtlinie an Ihre Ressource anzuhängen.

## Freigabe des Zugriffs auf verschlüsselte Daten

Wenn Sie die serverseitige Verschlüsselung für einen Datenstrom mit AWS verwaltetem KMS-Schlüssel aktiviert haben und den Zugriff über eine Ressourcenrichtlinie freigeben möchten, müssen Sie zur Verwendung von vom Kunden verwalteten Schlüsseln (CMK) wechseln. Weitere Informationen finden Sie unter [Was bedeutet eine serverseitige Verschlüsselung in Kinesis Data Streams?](#). Darüber hinaus müssen Sie Ihren freigebenden Prinzipal-Entitäten Zugriff auf Ihren CMK gewähren, indem Sie die kontoübergreifenden KMS-Freigabe-Funktionen nutzen. Stellen Sie sicher, dass Sie auch die IAM-Richtlinien für die freigebenden Prinzipal-Entitäten ändern. Weitere Informationen finden Sie unter [Benutzern in anderen Konten die Verwendung eines KMS-Schlüssels erlauben](#).

## Konformitätsprüfung für Amazon Kinesis Data Streams

Externe Prüfer bewerten im Rahmen verschiedener AWS -Compliance-Programme die Sicherheit und Compliance von Amazon Kinesis Data Streams. Hierzu zählen unter anderem SOC, PCI, FedRAMP und HIPAA.

Eine Liste der - AWS Services, die in den Geltungsbereich bestimmter Compliance-Programme fallen, finden Sie unter [AWS -Services im Geltungsbereich nach Compliance-Programm](#). Allgemeine Informationen finden Sie unter [AWS -Compliance-Programme](#).

Sie können Auditberichte von Drittanbietern mit heruntergeladenen AWS Artifact. Weitere Informationen finden Sie unter [Berichte in AWS Artifact herunterladen](#).

Ihre Compliance-Verantwortung bei Verwendung von Kinesis Data Streams hängt von der Vertraulichkeit der Daten, den Compliance-Zielen des Unternehmens und den geltenden Gesetzen und Vorschriften ab. Wenn Ihre Nutzung von Kinesis Data Streams der Einhaltung von Standards wie HIPAA, PCI oder FedRAMP unterliegt, AWS stellt Ressourcen zur Unterstützung bereit:

- [Schnellstartanleitungen für Sicherheit und Compliance](#) – In diesen Bereitstellungsleitfäden werden Überlegungen zur Architektur erörtert und Schritte zur Bereitstellung von sicherheits- und Compliance-orientierten Basisumgebungen in beschrieben AWS.
- [Whitepaper zur Erstellung einer Architektur mit HIPAA-konformer Sicherheit und Compliance](#) – In diesem Whitepaper wird beschrieben, wie Unternehmen mithilfe AWS von HIPAA-konforme Anwendungen erstellen können.
- [AWS Compliance-Ressourcen](#) – Diese Sammlung von Arbeitsmappen und Leitfäden könnte für Ihre Branche und Ihren Standort relevant sein
- [AWS Config](#) – Dieser AWS Service bewertet, wie gut Ihre Ressourcenkonfigurationen den internen Praktiken, Branchenrichtlinien und Vorschriften entsprechen.
- [AWS Security Hub](#) – Dieser AWS Service bietet einen umfassenden Überblick über Ihren Sicherheitsstatus in AWS , mit dem Sie Ihre Compliance mit den Sicherheitsstandards und bewährten Methoden der Branche überprüfen können.

## Resilienz in Amazon Kinesis Data Streams

Die AWS globale -Infrastruktur ist um - AWS Regionen und Availability Zones herum aufgebaut. - AWS Regionen bieten mehrere physisch getrennte und isolierte Availability Zones, die mit einem Netzwerk mit niedriger Latenz, hohem Durchsatz und hoher Redundanz verbunden sind. Mithilfe von Availability Zones können Sie Anwendungen und Datenbanken erstellen und ausführen, die automatisch Failover zwischen Availability Zones ausführen, ohne dass es zu Unterbrechungen kommt. Availability Zones sind besser hoch verfügbar, fehlertoleranter und skalierbarer als herkömmliche Infrastrukturen mit einem oder mehreren Rechenzentren.

Weitere Informationen zu AWS Regionen und Availability Zones finden Sie unter [AWS Globale Infrastruktur](#).

Neben der AWS globalen -Infrastruktur stellt Kinesis Data Streams verschiedene Funktionen bereit, um Ihren Anforderungen an Ausfallsicherheit und Datensicherung gerecht zu werden.



## Notfallwiederherstellung in Amazon Kinesis Data Streams

Wenn Sie Daten aus einem Stream mit einer Anwendung von Amazon Kinesis Data Streams verarbeiten, sind Ausfälle auf folgenden Ebenen möglich:

- Ausfall eines Datensatzprozessors
- Ausfall eines Auftragnehmers oder einer Instance der Anwendung, die den Auftragnehmer instanziiert hat
- Ausfall einer EC2 Instance, die eine oder mehrere Instances der Anwendung hostet

### Ausfall des Datensatzprozessors

Der Worker ruft Datensatzprozessormethoden mithilfe von Java-[ExecutorService](#)-Aufgaben auf. Schlägt die Ausführung einer Aufgabe fehl, behält der Auftragnehmer die Kontrolle über den Shard, den der Datensatzprozessor verarbeitet hat. Der Auftragnehmer startet eine neue Datensatzprozessoraufgabe für die Verarbeitung des Shards. Weitere Informationen finden Sie unter [Drosselung bei Lesevorgängen](#).

### Ausfall von Auftragnehmer oder Anwendung

Wenn ein Worker oder eine Instance der Anwendung Amazon Kinesis Data Streams ausfällt, sollten Sie die Situation erkennen können und angemessen reagieren. Löst beispielsweise die `Worker.run`-Methode eine Ausnahme aus, müssen Sie diese abfangen und verwalten.

Fällt die Anwendung selbst aus, sollten Sie dies erkennen, damit Sie einen Neustart durchführen können. Wenn die Anwendung gestartet wird, instanziiert sie einen neuen Auftragnehmer, der wiederum neue Datensatzprozessoren instanziiert, denen automatisch Shards zur Verarbeitung zugewiesen werden. Dies können dieselben Shards sein, die der Datensatzprozessor vor dem Ausfall verarbeitet hat, oder neue Shards.

Falls der Auftragnehmer oder die Anwendung ausfällt, der Ausfall nicht bemerkt wird und noch weitere Instances der Anwendung auf anderen EC2-Instances ausgeführt werden, wird der Ausfall von den Auftragnehmern dieser weiteren Instances verwaltet. Sie erstellen weitere Datensatzprozessoren für die Verarbeitung der Shards des ausgefallenen Auftragnehmers. Die Verarbeitungslast für diese EC2 Instances erhöht sich entsprechend.

Bei dem hier beschriebenen Szenario wird davon ausgegangen, dass die hostende EC2-Instance trotz des Ausfalls des Auftragnehmers oder der Anwendung weiterhin ausgeführt wird und deshalb nicht von einer Auto-Scaling-Gruppe neu gestartet wird.

## Amazon-EC2-Instance-Fehler

Wir empfehlen die Ausführung der EC2 Instances Ihrer Anwendung in einer Auto-Scaling-Gruppe. Wenn dann eine der EC2 Instances ausfällt, startet die Auto-Scaling-Gruppe automatisch eine neue Instance. Sie sollten die Instances so konfigurieren, dass Ihre Anwendung Amazon Kinesis Data Streams beim Start gestartet wird.

## Infrastruktursicherheit in Kinesis Data Streams

Als verwalteter Service ist Amazon Kinesis Data Streams durch die AWS globalen Verfahren zur Gewährleistung der Netzwerksicherheit von geschützt, die im Whitepaper [Amazon Web Services: Übersicht über die Sicherheitsprozesse](#) beschrieben sind.

Sie verwenden durch AWS veröffentlichte API-Aufrufe, um über das Netzwerk auf Kinesis Data Streams zuzugreifen. Clients müssen Transport Layer Security (TLS) 1.2 oder höher unterstützen. Clients müssen außerdem Verschlüsselungssammlungen mit PFS (Perfect Forward Secrecy) wie DHE (Ephemeral Diffie-Hellman) oder ECDHE (Elliptic Curve Ephemeral Diffie-Hellman) unterstützen. Die meisten modernen Systemen wie Java 7 und höher unterstützen diese Modi.

Außerdem müssen Anforderungen mit einer Zugriffsschlüssel-ID und einem geheimen Zugriffsschlüssel signiert sein, der einem IAM-Prinzipal zugeordnet ist. Alternativ können Sie mit [AWS Security Token Service](#) (AWS STS) temporäre Sicherheitsanmeldeinformationen erstellen, um die Anforderungen zu signieren.

## Bewährte Methoden für die Sicherheit für Kinesis Data Streams

Amazon Kinesis Data Streams enthält eine Reihe von Sicherheitsfunktionen, die Sie bei der Entwicklung und Implementierung Ihrer eigenen Sicherheitsrichtlinien berücksichtigen sollten. Die folgenden bewährten Methoden sind allgemeine Richtlinien und keine vollständige Sicherheitslösung. Da diese bewährten Methoden für Ihre Umgebung möglicherweise nicht angemessen oder ausreichend sind, sollten Sie sie als hilfreiche Überlegungen und nicht als bindend ansehen.

## Implementieren des Zugriffs mit geringsten Berechtigungen

Beim Erteilen von Berechtigungen entscheiden Sie, wer welche Berechtigungen für welche Ressourcen von Kinesis Data Streams erhält. Sie aktivieren die spezifischen Aktionen, die daraufhin für die betreffenden Ressourcen erlaubt sein sollen. Aus diesem Grund sollten Sie nur Berechtigungen gewähren, die zum Ausführen einer Aufgabe erforderlich sind. Die Implementierung

der geringstmöglichen Zugriffsrechte ist eine grundlegende Voraussetzung zum Reduzieren des Sicherheitsrisikos und der Auswirkungen, die aufgrund von Fehlern oder böswilligen Absichten entstehen könnten.

## Verwenden von IAM-Rollen

Produzenten- und Client-Anwendungen müssen über gültige Anmeldeinformationen für den Zugriff auf Kinesis-Datenströme verfügen. Sie sollten AWS Anmeldeinformationen nicht direkt in einer Clientanwendung oder in einem Amazon S3-Bucket speichern. Dabei handelt es sich um langfristige Anmeldeinformationen, die nicht automatisch rotiert werden und bedeutende geschäftliche Auswirkungen haben könnten, wenn sie kompromittiert werden.

Stattdessen sollten Sie eine IAM-Rolle zum Verwalten von temporären Anmeldeinformationen für Ihre Produzenten- und Client-Anwendungen für den Zugriff auf Kinesis-Datenströme verwenden. Wenn Sie eine Rolle verwenden, müssen Sie keine langfristigen Anmeldeinformationen (z. B. Benutzername und Passwort oder Zugriffsschlüssel) für den Zugriff auf andere Ressourcen verwenden.

Weitere Informationen finden Sie unter folgenden Themen im IAM-Benutzerhandbuch:

- [IAM-Rollen](#)
- [Gängige Szenarien für Rollen: Benutzer, Anwendungen und Services](#)

## Implementieren einer serverseitigen Verschlüsselung in abhängigen Ressourcen

Daten im Ruhezustand und Daten während der Übertragung können in Kinesis Data Streams verschlüsselt werden. Weitere Informationen finden Sie unter [Datenschutz in Amazon Kinesis Data Streams](#).

## Verwenden von CloudTrail zur Überwachung von API-Aufrufen

Kinesis Data Streams ist in integriert, einem Service AWS CloudTrail, der die Aktionen eines Benutzers, einer Rolle oder eines - AWS Services in Kinesis Data Streams aufzeichnet.

Anhand der von CloudTrailgesammelten Informationen können Sie die an Kinesis Data Streams gestellte Anfrage, die IP-Adresse, von der die Anfrage gestellt wurde, den Initiator der Anfrage, den Zeitpunkt der Anfrage und zusätzliche Details bestimmen.

Weitere Informationen finden Sie unter [the section called “Protokollieren von Amazon-Kinesis-Data-Streams-API-Aufrufen mithilfe von AWS CloudTrail”](#).

# Dokumentverlauf

Die folgende Tabelle enthält wichtige Änderungen an der Dokumentation zu Amazon Kinesis Data Streams.

Änderung	Beschreibung	Änderungsdatum
Unterstützung für die gemeinsame Nutzung von Datenströmen zwischen Konten hinzugefügt.	<a href="#">Freigabe Ihres Datenstroms für ein anderes Konto</a> hinzugefügt.	22. November 2023
Unterstützung für die Kapazitätsmodi „On-Demand“ und „Bereitgestellter Datenstrom“ wurde hinzugefügt.	<a href="#">Auswahl des Datenstrom-Kapazitätsmodus</a> hinzugefügt.	29. November 2021
Neue Inhalte für die serverseitige Verschlüsselung.	<a href="#">Datenschutz in Amazon Kinesis Data Streams</a> hinzugefügt.	7. Juli 2017
Neue Inhalte für verbesserte CloudWatch Metriken.	Aktualisiert <a href="#">Überwachung von Amazon Kinesis Data Streams</a> .	19. April 2016
Neue Inhalte zur Erweiterung von Kinesis-Agenten.	Aktualisiert <a href="#">Schreiben in Amazon Kinesis Data Streams mit Kinesis Agent</a> .	11. April 2016
Neue Inhalte für die Nutzung von Kinesis-Agenten.	<a href="#">Schreiben in Amazon Kinesis Data Streams mit Kinesis Agent</a> hinzugefügt.	2. Oktober 2015

Änderung	Beschreibung	Änderungsdatum
Aktualisierter KPL-Inhalt für Version 0.10.0.	<a href="#">Entwickeln von Produzenten mit der Amazon Kinesis Producer Library</a> hinzugefügt.	15. Juli 2015
Aktualisierter Inhalt zum Thema KCL-Metriken für konfigurierbare Metriken.	<a href="#">Überwachen der Kinesis Client Library mit Amazon CloudWatch</a> hinzugefügt.	9. Juli 2015
Inhalte neu organisiert.	Wichtige Neuorganisation von Inhalten für einen bessere Strukturansicht und eine logischere Gruppierung.	01. Juli 2015
Neues Thema für das KPL-Entwicklerhandbuch.	<a href="#">Entwickeln von Produzenten mit der Amazon Kinesis Producer Library</a> hinzugefügt.	02. Juni 2015
Neues Thema zu KCL-Metriken.	<a href="#">Überwachen der Kinesis Client Library mit Amazon CloudWatch</a> hinzugefügt.	19. Mai 2015
KCL-Support in .NET	<a href="#">Entwickeln eines Kinesis Client Library-Verbrauchers in .NET</a> hinzugefügt.	1. Mai 2015
KCL-Support in Node.js	<a href="#">Entwickeln eines Kinesis-Client-Library-Verbrauchers in Node.js</a> hinzugefügt.	26. März 2015
KCL-Support in Ruby	Links zur KCL Ruby-Bibliothek hinzugefügt.	12. Januar 2015
Neue API PutRecords	Informationen über die neue PutRecords API wurden hinzugefügt <a href="#">the section called "Hinzufügen mehrerer Datensätze mit PutRecords"</a> .	15. Dezember 2014
Support für Markierungen	<a href="#">Tagging von Streams in Amazon Kinesis Data Streams</a> hinzugefügt.	11. September 2014

Änderung	Beschreibung	Änderungsdatum
Neue CloudWatch Metrik	<code>GetRecords.IteratorAgeMilliseconds</code> - Metrik zu <a href="#">Amazon Kinesis Data Streams – Metriken und Dimensionen</a> hinzugefügt.	3. September 2014
Neues Kapitel zur Überwachung	<a href="#">Überwachung von Amazon Kinesis Data Streams</a> und <a href="#">Überwachung des Services von Amazon Kinesis Data Streams für Amazon CloudWatch</a> hinzugefügt.	30. Juli 2014
Standard-Shard-Limit	<a href="#">Kontingente und Einschränkungen</a> aktualisiert: Das Standard-Shard-Limit wurde von 5 auf 10 erhöht.	25. Februar 2014
Standard-Shard-Limit	<a href="#">Kontingente und Einschränkungen</a> aktualisiert: Das Standard-Shard-Limit wurde von 2 auf 5 erhöht.	28. Januar 2014
Aktualisierungen der API-Version	Aktualisierungen von Version 2013-12-02 der API für Kinesis Data Streams.	12. Dezember 2013
Erstversion	Erstveröffentlichung des Entwicklerhandbuchs für Amazon Kinesis.	14. November 2013

# AWS-Glossar

Die neueste AWS-Terminologie finden Sie im [AWS-Glossar](#) in der AWS-Glossar-Referenz.



Die vorliegende Übersetzung wurde maschinell erstellt. Im Falle eines Konflikts oder eines Widerspruchs zwischen dieser übersetzten Fassung und der englischen Fassung (einschließlich infolge von Verzögerungen bei der Übersetzung) ist die englische Fassung maßgeblich.