
Amazon Kinesis Data Streams

开发人员指南

亚马逊云科技

The Amazon logo, a curved orange arrow pointing from the letter 'a' to the letter 'z', is positioned below the Chinese text.

Amazon Kinesis Data Streams: 开发人员指南

Copyright © 2023 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon 的商标和商业外观不得用于任何非 Amazon 的商品或服务，也不得以任何可能引起客户混淆、贬低或诋毁 Amazon 的方式使用。所有非 Amazon 拥有的其它商标均为各自所有者的财产，这些所有者可能附属于 Amazon、与 Amazon 有关联或由 Amazon 赞助，也可能不是如此。

Amazon Web Services 文档中描述的 Amazon Web Services 服务或功能可能因区域而异。要查看适用于中国区域的差异，请参阅 [中国的 Amazon Web Services 服务入门 \(PDF\)](#)。

Table of Contents

什么是 Amazon Kinesis Data Streams ?	1
可以使用 Kinesis Data Streams 做什么?	1
使用 Kinesis Data Streams 的好处	1
相关服务	2
术语和概念	3
高级别架构	3
Kinesis Data Streams 术语	3
Kinesis Data Streams	3
数据记录	3
容量模式	4
保留周期	4
创建者	4
使用者	4
Amazon Kinesis Data Streams 应用程序	4
分片	4
分区键	4
序列号	5
Kinesis 客户端库	5
Application Name	5
服务器端加密	5
配额和限制	6
API 限制	6
KDS 控制层面 API 限制	7
数据层面 API 限制	8
提升配额	9
设置	10
注册 Amazon	10
下载库和工具	10
配置您的开发环境	10
入门	12
安装和配置 Amazon CLI	12
安装 Amazon CLI	12
配置 Amazon CLI	13
使用 Amazon CLI 执行基本 Kinesis 数据流操作。	13
第 1 步：创建 流	13
第 2 步：设置记录	15
第 3 步：获取记录	15
第 4 步：清除	17
示例	18
教程：使用 KPL 和 KCL 2.x 处理实时股票数据	18
先决条件	19
步骤 1：创建数据流	19
步骤 2：创建 IAM 策略和用户	20
步骤 3：下载并构建代码	23
第 4 步：实施创建器	23
第 5 步：实施使用者	26
第 6 步：(可选) 扩展使用者	29
第 7 步：收尾	30
教程：使用 KPL 和 KCL 1.x 处理实时股票数据	31
先决条件	31
步骤 1：创建数据流	32
步骤 2：创建 IAM 策略和用户	33
步骤 3：下载和构建实施代码	36
第 4 步：实施创建器	37

第 5 步：实施使用者	39
第 6 步：(可选) 扩展使用者	42
第 7 步：收尾	42
教程：使用适用于 Flink 应用程序的 Kinesis 数据分析实时股票数据	43
先决条件	44
步骤 1：设置账户	44
步骤 2：设置 Amazon CLI	46
步骤 3：创建应用程序	47
教程：使用 Amazon Lambda Kinesis Data Streams	58
Amazon 数据流式处理	58
创建和管理流	59
选择数据流容量模式	59
什么是数据流容量模式？	59
按需模式	60
预置模式	60
在容量模式之间切换	61
通过 Amazon 管理控制台创建直播	61
通过 API 创建直播	62
构建 Kinesis Data Streams 客户端	62
创建流	62
更新流	63
使用 API 更新流	64
使用 Amazon CLI 更新流	64
列出流	64
列表分片	65
ListShards API-推荐	65
DescribeStream API-已过时	67
删除流	68
对流进行重新分片	68
重新分片策略	68
拆分分片	69
合并两个分片	70
重新分片后	70
更改数据保留期	72
为您的流添加标签	72
有关标签的基本知识	73
使用标签跟踪成本	73
标签限制	73
使用 Kinesis Data Streams 控制台为流添加标签	74
使用 Amazon CLI 为流添加标签	74
使用 Kinesis Data Streams API 为流添加标签	75
写入数据流	76
使用 KPL	76
KPL 的作用	77
使用 KPL 的优势	77
何时不使用 KPL	78
安装 KPL	78
过渡到适用于 Kinesis Producer Library 的 Amazon Trust Services (ATS) 证书	78
KPL 支持的平台	78
KPL 重要概念	79
将 KPL 与生产者代码集成	80
写入您的 Kinesis 数据流	81
配置 KPL	82
使用者取消聚合	83
将 KPL 与 Kinesis Data Firehose 一起使用	85
将 KPL 与 Glue 架构注册表一起使用	85

KPL 代理配置	85
使用 API	86
向流添加数据	86
使用与数据进行交互Amazon架 Glue 注册表	90
使用代理	90
先决条件	90
下载并安装代理	91
配置并启动代理	91
代理配置设置	92
监控多个文件目录并写入多个流	94
使用代理预处理数据	94
代理 CLI 命令	97
使用其他Amazon服务	98
Amazon Amplify	98
Amazon Aurora	98
Amazon CloudFront	98
亚马逊CloudWatch日志	99
Amazon Connect	99
Amazon Database Migration Service	99
Amazon DynamoDB	99
Amazon EventBridge	99
Amazon IoT Core	99
Amazon Relational Database Service	100
Amazon Pinpoint	100
亚马逊量子账本数据库	100
使用第三方集成	100
Apache Flink	100
Fluentd	100
Debezium	101
甲骨文 GoldenGate	101
Kafka Connect	101
Adobe	101
Striim	101
故障排除	101
创建器应用程序的写入速率比预期的慢	101
未授权的 KMS 主密钥权限错误	102
面向生产者的常见问题、问题和故障排除想法	102
高级主题	103
重试和速率限制	103
使用 KPL 聚合时的注意事项	103
读取数据流	105
在 Kinesis 控制台中使用数据查看器	106
使用 Amazon Lambda	106
使用 Kinesis Data Analytics	106
使用 Kinesis Data Firehose	106
使用 Kinesis 客户端库	107
Kinesis 客户端库是什么？	107
KCL 可用的版本	108
KCL 概念	108
使用租赁表跟踪 KCL 消费者应用程序处理的碎片	109
针对 Java 消费者应用程序使用相同的 KCL 2.x 处理多个数据流	114
将 Kinesis 客户端库与AmazonGlue 架构注册	116
开发具有共享吞吐量的自定义使用者	116
使用 KCL 开发具有共享吞吐量的自定义使用者	117
使用 Amazon SDK for Java 开发具有共享吞吐量的自定义使用者	139
开发具有专用吞吐量的自定义使用者（增强扇出功能）	143
使用 KCL 2.x 开发具有增强型扇出功能的使用者	144

使用 Kinesis Data Streams API 开发具有增强型扇出功能的使用者	148
使用 Amazon Web Services Management Console 管理具有增强扇出功能的使用者	150
将使用者从 KCL 1.x 迁移到 KCL 2.x	150
迁移记录处理器	151
迁移记录处理器工厂	154
迁移工作程序	155
配置 Amazon Kinesis 客户端	156
闲置时间删除	158
客户端配置删除	158
使用他Amazon服务	158
EMR	159
使用亚马逊 EventBridge 管道	159
使用 Amazon Glue	159
Amazon Redshift	159
使用第三方集成	159
Apache Flink	160
Adobe 体验平台	160
Apache Duid	160
Apache Spark	160
Databricks	160
Kafka	160
Kinesumer	160
塔伦德	160
排查 Kinesis Data Streams 使用者的问题	160
使用 Kinesis 客户端库时会跳过某些 Kinesis 数据流记录	161
属于同一分片的记录通过不同的记录处理器同时处理	161
使用者应用程序的读取速率比预期的慢	161
GetRecords即使流中有数据，也会返回空记录数组	162
分片迭代器意外过期	162
使用者记录处理滞后	162
未授权的 KMS 主密钥权限错误	163
消费者常见问题、疑问和疑难解答建议	163
高级主题	163
低延迟处理	163
使用Amazon Lambda与 Kinesis 创建者库一起使用	164
重新分片、扩展和并行处理	164
处理重复记录	165
处理启动、关闭和限制	166
监控数据流	168
使用 CloudWatch 监控服务	168
Amazon Kinesis Data Streams 维度和指标	168
访问 Amazon CloudWatch Kinesis Data Streams 的指标	177
使用 CloudWatch 监控代理	178
使用 CloudWatch 进行监控	178
记录 Amazon Kinesis Data Streams API 调用Amazon CloudTrail	178
CloudTrail 中的 Kinesis Data Streams	179
例如：Kinesis Data Streams 日志文件条目	180
使用 CloudWatch 监控 KCL	182
指标和命名空间	182
指标级别和维度	182
指标配置	183
指标的列表	183
使用 CloudWatch 监控 KPL	191
指标、维度和命名空间	191
指标级别和粒度	191
本地访问和亚马逊 CloudWatch 上传	192
指标的列表	192

安全性	195
数据保护	195
什么是 Kinesis 数据流的服务器端加密？	196
费用、区域和性能注意事项	196
如何开始使用服务器端加密？	197
创建和使用用户生成的 KMS 主密钥	197
使用用户生成的 KMS 主密钥的权限	197
验证 KMS 密钥权限和排查问题	199
使用 接口 VPC 终端节点	199
控制访问	201
策略语法	202
针对 Kinesis 数据流的操作	202
Kinesis 数据流的亚马逊资源名称 (ARN)	203
Kinesis 数据流的策略示例	203
合规性验证	205
故障恢复能力	205
灾难恢复	205
基础设施安全性	206
安全最佳实践	206
实施最低权限访问	206
使用 IAM 角色	206
实施从属资源中的服务器端加密	207
使用 CloudTrail 监控 API 调用	207
文档历史记录	208
Amazon 词汇表	210
.....	ccxi

什么是 Amazon Kinesis Data Streams？

您可以使用 Amazon Kinesis Data Streams 收集和处理大数据流。[流](#)实时的数据记录。可以创建称为“数据处理应用程序”（称 Kinesis Data Streams 应用程序。典型的 Kinesis Data Streams 应用程序从数据流作为数据记录。这些应用程序可使用 Kinesis 客户端库，并且可在 Amazon EC2 实例上运行。您可以将处理后的记录发送到控制面板，使用这些记录生成警报、动态更改定价和广告战略或将数据发送给多种其他控制面板。Amazon 服务。有关 Kinesis Data Streams 功能和定价的信息，请参阅[Amazon Kinesis Data Streams](#)。

与 Kinesis Data Streams 和 Kinesis Data Streams 都是 Kinesis 流数据平台的一部分。[Kinesis Data Firehose](#)、[Kinesis Video Streams](#)，和[Kinesis Data Analytics](#)。

有关 的更多信息 Amazon 大数据解决方案，请参阅[开启大数据 Amazon](#)。有关 的更多信息 Amazon 流数据解决方案，请参阅[什么是流数据？](#)。

主题

- [可以使用 Kinesis Data Streams 做什么？ \(p. 1\)](#)
- [使用 Kinesis Data Streams 的好处 \(p. 1\)](#)
- [相关服务 \(p. 2\)](#)

可以使用 Kinesis Data Streams 做什么？

您可以使用 Kinesis Data Streams 进行快速而持续的数据引入和聚合。使用的数据类型可以包括 IT 基础设施日志数据、应用程序日志、社交媒体、市场数据源和 Web 点击流数据。由于数据引入和处理的响应时间是实时的，因此处理通常是轻量级的。

以下是使用 Kinesis Data Streams 的典型场景：

加速的日志和数据源引入和处理

您可以让创建者直接将数据推入流。例如，推送系统和应用程序日志，它们可在几秒内就绪，以用于处理。这可以防止因前端或应用程序服务器失败而造成日志数据丢失。因为您在提交数据以备引入之前，未在服务器上批处理数据，因此 Kinesis Data Streams 提供加速的数据源引入。

实时指标和报告

您可以将数据收集到 Kinesis Data Streams 中的数据用于实时的简单数据分析和报告。例如，您的数据处理应用程序可以处理系统和应用程序日志的指标和报告，因为数据将流入而不是等待接收批量数据。

实时数据分析

这可将并行处理的强大功能与实时数据的价值相结合。例如，使用多个并行运行的不同的 Kinesis Data Streams 应用程序，实时处理网站点击流，然后分析站点可用性参与度。

复杂流处理

可以创建 Kinesis Data Streams 应用程序和数据流的有向无环图 (DAG)。这通常会涉及将数据从多个 Kinesis Data Streams 应用程序放入其他流，以供其他 Kinesis Data Streams 应用程序进行下游处理。

使用 Kinesis Data Streams 的好处

虽然可使用 Kinesis Data Streams 解决各种流式处理数据问题，但其常见用途是实时聚合数据，然后将聚合数据加载到数据仓库或 map-reduce 群集。

将数据放入 Kinesis Data Streams，以确保持久性和弹性。将记录放入流的时间与可检索记录的时间之间的延迟（put-to-get延迟）通常少于 1 秒。换言之，在添加数据之后，Kinesis Data Streams 应用程序几乎立即可以开始使用流中的数据。Kinesis Data Streams 的托管服务方面可减轻您创建和运行数据引入管道的操作负担。可以创建流式处理 map-reduce 类型应用程序。利用 Kinesis Data Streams 的弹性，可以扩大或缩小流，以便绝不会在数据记录过期前丢失它们。

多个 Kinesis Data Streams 应用程序可以使用流中的数据，以便多个操作（如存档和处理）可以并发且独立进行。例如，两个应用程序可读取同一流中的数据。第一个应用程序计算正在运行的聚合并更新 Amazon DynamoDB 表，第二个应用程序压缩数据并将数据存档至像 Amazon Simple Storage Service (Amazon S3) 这样的数据存储。然后，控制面板将读取带正在运行的聚合的 DynamoDB 表以下内容：up-to-the分钟的报告。

Kinesis 客户端库支持容错使用流中的数据，并提供针对 Kinesis Data Streams 应用程序的扩展支持。

相关服务

有关使用 Amazon EMR 群集直接读取和处理 Kinesis 数据流的信息，请参阅[Kinesis 连接器](#)。

Amazon Kinesis Data Streams 术语和概念

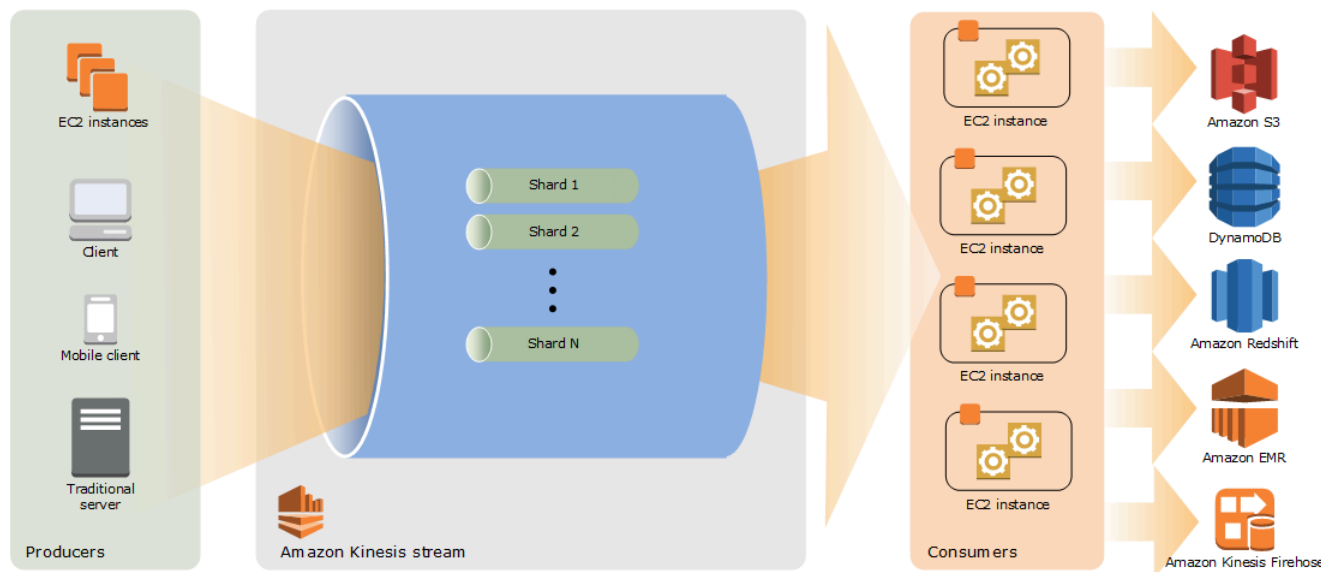
当您开始使用 Amazon Kinesis Data Streams 时，了解其架构和术语将大有裨益。

主题

- [Kinesis Data Streams 高级架构 \(p. 3\)](#)
- [Kinesis Data Streams 术语 \(p. 3\)](#)

Kinesis Data Streams 高级架构

下图演示 Kinesis Data Streams 的高级别架构。这些区域有：生产商们不断将数据推送到 Kinesis Data Streams，消费者们实时处理数据。使用者（如在 Amazon EC2 上运行的自定义应用程序或 Amazon Kinesis Data Firehose 传输流）可以使用 Amazon 服务，例如 Amazon DynamoDB、Amazon Redshift 或 Amazon S3。



Kinesis Data Streams 术语

Kinesis Data Streams

一个 Kinesis Data Streams 是一套 [鞘翅 \(p. 4\)](#)。每个分片都有一系列数据记录。每个数据记录都有一个 [序列号 \(p. 5\)](#) 这是由 Kinesis Data Streams 分配的。

数据记录

一个数据记录是存储在中的数据单位 [Kinesis Data Streams \(p. 3\)](#)。数据记录由 [序列号 \(p. 5\)](#)，[分区键 \(p. 4\)](#) 和数据 Blob，这是一个不可变的字节序列。Kinesis Data Streams 不会以任何形式检查、解释或更改 Blob 中的数据。数据 Blob 最大可达 1 MB。

容量模式

数据流容量模式确定如何管理容量以及如何为数据流的使用量收费。目前，在 Kinesis Data Streams 中，您可以在按需模式和供应数据流的模式。有关更多信息，请参阅 [选择数据流容量模式 \(p. 59\)](#)。

使用按需模式下，Kinesis Data Streams 自动管理分片以提供必要的吞吐量。您只需为使用的实际吞吐量付费，而 Kinesis Data Streams 在工作负载上升或下降时自动适应它们的吞吐量需求。有关更多信息，请参阅 [按需模式 \(p. 60\)](#)。

使用供应模式，必须指定数据流的分片数。数据流的总容量是其分片容量的总和。您可以根据需要增加或减少数据流中的分片数，并按小时费率支付分片数。有关更多信息，请参阅 [预置模式 \(p. 60\)](#)。

保留周期

保留期是数据记录在添加到流中后可供访问的时间长度。在创建之后，流的保留期设置为默认值 24 小时。您可以将保留期增加至 8760 小时 (365 天) 使用 [IncreaseStreamRetentionPeriod](#) 操作，并使用将保留期缩短至 24 小时 (最小值) [DecreaseStreamRetentionPeriod](#) operation。对于保留期设置为 24 小时以上的流，将收取额外费用。有关更多信息，请参阅 [Amazon Kinesis 数据流定价](#)。

创建者

创建者将记录放入 Amazon Kinesis Data Streams 中。例如，发送日志数据到流的 Web 服务器是创建器。

使用者

使用者从 Amazon Kinesis Data Streams 获取记录并进行处理。这些使用者称为 [Amazon Kinesis Data Streams 应用程序 \(p. 4\)](#)。

Amazon Kinesis Data Streams 应用程序

网络 ACL 和安全组都允许 (因此可到达您的实例) 的发起 ping 的 Amazon Kinesis Data Streams 应用程序是通常在 EC2 实例队列上运行的流的使用者。

可以开发的使用者有两种：共享扇出功能使用者和增强型扇出功能使用者。要了解它们之间的区别，以及了解如何创建每种使用者，请参阅 [从 Amazon Kinesis Data Streams \(p. 105\)](#)。

Kinesis Data Streams 应用程序的输出可能是另一个流的输入，这使您能够创建实时处理数据的复杂拓扑。应用程序也可以将数据发送到各种其他应用程序。Amazon 服务。一个流可以有多个应用程序，每个应用程序可同时单独使用流中的数据。

分片

分片是流中数据记录的唯一标识序列。一个流由一个或多个分片组成，每个分片提供一个固定的容量单位。每个分片最多可以支持每秒 5 个事务处理用于读取，最高总数据读取速率为每秒 2 MB，写入最多每秒 1000 条记录，最高总数据写入速率为每秒 1 MB (包括分区键)。流的数据容量是您为流指定的分片数的函数。流的总容量是其分片容量的总和。

如果数据速率增加，您可以增加或减少分配给流的分片数量。有关更多信息，请参阅 [对流进行重新分片 \(p. 68\)](#)。

分区键

一个分区键用于按分片对流中的数据进行分组。Kinesis Data Streams 将属于一个流的数据记录隔离到多个分片中。它使用与每个数据记录关联的分区键确定指定数据记录属于哪个分片。分区键是 Unicode 字符串，

每个键的最大长度限制为 256 个字符。MD5 哈希函数用于将分区键映射到 128 位整数值，并使用分片的哈希键范围将关联的数据记录映射到分片。当应用程序将数据放入流中时，它必须指定一个分区键。

序列号

每个数据记录都有一个序列号在分片中，每个分区键都是唯一的。在您使用写入流之后，Kinesis Data Streams 将分配序列号 `client.putRecords` 要么 `client.putRecord`。同一分区键的序列号通常会随时间推移增加。写入请求之间的时间段越长，序列号越大。

Note

序列号不能用作相同流中的数据集的索引。为了在逻辑上分隔数据集，请使用分区键或者为每个数据集创建单独的流。

Kinesis 客户端库

Kinesis 客户端库已编译成应用程序，从而支持以容错方式使用流中的数据。Kinesis 客户端库确保每个分片都有一个运行和处理该分片的记录处理器。库还可以简化流中的数据读取。Kinesis 客户端库使用 Amazon DynamoDB 表存储控制数据。它会为每个正在处理数据的应用程序创建一个表。

Kinesis 客户端库有两个主要版本。使用哪个版本取决于要创建的使用者的类型。有关更多信息，请参阅 [从 Amazon Kinesis Data Streams \(p. 105\)](#)。

Application Name

Amazon Kinesis Data Streams 应用程序的名称标识应用程序。每个应用程序必须具有一个唯一名称，此名称的范围限定于 Amazon 应用程序使用的账户和地区。此名称用作 Amazon DynamoDB 中的控制表名称和 Amazon 的命名空间。CloudWatch 指标。

服务器端加密

当创建者将敏感数据输入流时，Amazon Kinesis Data Streams 可以自动加密这些数据。使用 Kinesis Data Streams [Amazon KMS](#) 主密钥进行加密。有关更多信息，请参阅 [亚马逊 Kinesis 数据流中的数据保护 \(p. 195\)](#)。

Note

要读取或写入加密的流、创建者和使用者应用程序必须有权访问主密钥。有关向创建者应用程序和使用者应用程序授予权限的信息，请参阅 [the section called “使用用户生成的 KMS 主密钥的权限” \(p. 197\)](#)。

Note

使用服务器端加密将产生 Amazon Key Management Service (Amazon KMS) 费用。有关更多信息，请参阅 [Amazon Key Management Service](#)。

配额和限制

Amazon Kinesis Data Streams 具有以下数据流和分片配额和限制。

配额	按需模式	预调配模式
数据负载大小	您的Amazon账户内的直播数量没有上限。原定设置情况下，您可以为按需容量模式结合使用。如果您需要增加此配额，请提出 支持请求 。	账户内采用预配模式的直播数量没有配额上限。
分片数量	没有上限。分片数量取决于摄取的数据量和所需的吞吐量水平。Kinesis Data Streams 会根据数据量和流量的变化自动调整分片数量。	没有上限。以下Amazon区域的默认分区配额为每个Amazon账户 500 个分片：美国东部（弗吉尼亚北部）、美国西部（俄勒冈）和欧洲（爱尔兰）。对于所有其他区域，默认分片配额为每个Amazon账户 200 个分片。要申请增加 shards-per-data 直播配额，请参阅 申请增加配额 。
数据流吞吐量	默认情况下，使用按需容量模式创建的新数据流具有 4 MB/s 的写入吞吐量和 8 MB/s 的读取吞吐量。随着流量的增加，采用按需容量模式的数据流可扩展至 200 MB/s 的写入吞吐量和 400 MB/s 的读取吞吐量。如果您需要将写入容量增加到 1 Gb/s 和 2Gb/s 读取容量，请提交 支持请求	没有上限。最大吞吐量取决于为流配置的分片数量。每个分片可以支持高达 1 MB/秒或 1,000 条记录/秒的写入吞吐量或高达 2 MB/秒或 2,000 条记录/秒的读取吞吐量。如果您需要更多的采集容量，您可以使用Amazon Web Services Management Console或 UpdateShardCount API 轻松增加流中的分区数量。
数据负载大小	之前记录的数据负载的最大大小base64-encoding为 1 MB。	
GetRecords事务大小	GetRecords 每次调用最多可以从单个分区检索 10 MB 的数据，每次调用最多可检索 10,000 条记录。对 GetRecords 的每次调用都算作一个读取事务。每个分片每秒可支持多达 5 个读取事务。每个读取事务可提供多达 10000 个记录，每个事务的配额上限为 10 MB。	
每个分片的数据读取率	每个分片最多可通过以下方式支持每秒 2 MB 的最大总数据读取速率 GetRecords 。如果对 GetRecords 的调用返回 10 MB，则在接下来的 5 秒内发出的后续调用将会引发异常。	
每个数据流的注册消费者数量	您最多可以为每个数据流创建 20 个注册用户（增强型扇出限制）。	
在预配模式和按需模式之间切换	对于Amazon账户中的每个数据流，您可以在 24 小时内两次在按需和预配置容量模式之间切换。	

API 限制

与大多数Amazon API 一样，Kinesis Data Streams API 操作也有速率限制。以下限制适用于每个地区的每个Amazon账户。有关 Kinesis Data Streams API 的更多信息，请参阅 [Amazon Kinesis API 参考](#)。

KDS 控制层面 API 限制

下一节介绍 KDS 控制层面 API 限制。利用 KDS 控制平面 API，您可以创建和管理数据流。这些限制适用于每个地区的每个 Amazon 账户。

控制层面 API 限制

API	API 调用限制	每个账户/直播	描述
AddTagsToStream	每秒 5 个事务 (TPS)	每场直播	每个数据流 50 个标签
CreateStream	5 TPS	每个账户	在账户中可以拥有的流数量没有配额上限。LimitExceededException 当您尝试执行以下任一操作时，您会在 CreateStream 发出请求时收到： - 在任何时间点，CREATING 状态中有五个以上的流。 - 创建的分片数量超过了为您的账户授权的数量。
DecreaseStreamRetentionPeriod	5 TPS	每场直播	数据流的保留期的最小值为 24 小时。
DeleteStream	5 TPS	每个账户	
DeregisterStreamConsumer	5 TPS	每场直播	
DescribeLimits	1 TPS	每个账户	
DescribeStream	10 TPS	每场直播	
DescribeStreamConsumer	20 TPS	每场直播	
DescribeStreamSummary	20 TPS	每场直播	
DisableEnhancedMonitoring	5 TPS	每场直播	
EnableEnhancedMonitoring	5 TPS	每场直播	
IncreaseStreamRetentionPeriod	5 TPS	每场直播	直播保留期的最大值为 8760 小时 (365 天)。
ListShards	1000 TPS	每场直播	
ListStreamConsumers	5 TPS	每场直播	
ListStreams	5 TPS	每个账户	
ListTagsForStream	5 TPS	每场直播	
MergeShards	5 TPS	每场直播	仅适用于预配置。
RegisterStreamConsumer	5 TPS	每场直播	您最多可以为每个数据流注册 20 个使用者。每次只能在一个数据流

API	API 调用限制	每个账户/直播	描述
			中注册给定的使用者。只能同时创建 5 个使用者。换句话说，您不能同时建立多路复用有效CREATING负载大小 5。在 a 中有 5 个消费者时注册第 6 个消费者CREATING
RemoveTagsFromStream	5 TPS	每场直播	
SplitShard	5 TPS	每场直播	仅适用于已配置
StartStreamEncryption		每场直播	在连续的 24 小时期间内，您可以为服务器端加密成功应用 25 次新的 KMS 密Amazon钥。
StopStreamEncryption		每场直播	在连续 24 小时期间，您可以成功禁用服务器端加密 25 次。
UpdateShardCount		每场直播	仅适用于预配置。分区数为 10000。此 API 还有其他限制。有关更多信息，请参阅 UpdateShardCount 。
UpdateStreamMode		每条直播	对于Amazon账户中的每个数据流，您可以在 24 小时内两次在按需和预配置容量模式之间切换。

数据层面 API 限制

下一部分介绍 KDS 数据平面 API 的限制。利用 KDS 数据平面 API，您可以使用数据流实时收集和处理数据记录。下面是您的数据流中的每分片限制。

数据层面 API 限制

API	API 调用限制	负载限制	其他详细信息
GetRecords	5 TPS	每次调用可返回的最大记录数为 10000 个。GetRecords可以返回的数据的最大大小为 10 MB。	如果调用返回此数量的数据，则在接下来的 5 秒内进行的后续调用将抛出ProvisionedThroughputExceed 如果流上的预置吞吐量不足，则在接下来的 1 秒钟内进行的后续调用会抛出ProvisionedThroughputExceed
GetShardIterator	5 TPS		分片迭代器在其返回给请求者的 5 分

API	API 调用限制	负载限制	其他详细信息
			钟后过期。如果 GetShardIterator 请求过于频繁，您会收到 ProvisionedThroughputExceededException。
PutRecord	1000 TPS	每个分片最多可以支持每秒写入 1000 条记录，最大数据写入总数为每秒 1 MB。	
PutRecords		每个 PutRecords 请求最多可支持 500 条记录。请求中的每一个记录最大可以为 1 MB，整个请求的上限为 5 MB，包括分区键。每个分片最多可以支持每秒写入 1000 条记录，最大数据写入总数为每秒 1 MB。	
SubscribeToShard	对于每个分片，每个注册使用者 SubscribeToShard 每秒可以调用一次。		如果您使用相同的 CoConsumerARN SubscribeToShard 再次拨打电话，并且 ShardId 在通话成功后的 5 秒钟内，您将获得 ResourceInUseException。

提升配额

如果配额是可调整的，则可以使用服务配额来请求增加配额。有些请求会自动解决，而另一些请求则提交给 SuAmazon pport 部门。您可以跟踪提交给 Support 人员的配额增加请求的 Amazon 状态。提高服务配额的请求没有得到优先支持。如果您有紧急请求，请联系 SuAmazon pport 部门。有关更多信息，请参阅[什么是服务配额？](#)。

要请求增加服务配额，请按照[请求增加配额](#)中概述的步骤操作。

Amazon Kinesis Data Streams

首次使用 Amazon Kinesis Data Streams abase Database Database Database

任务

- [注册 Amazon \(p. 10\)](#)
- [下载库和工具 \(p. 10\)](#)
- [配置您的开发环境 \(p. 10\)](#)

注册 Amazon

当您注册 Amazon Web Services (Amazon) 时，您的 Amazon 账户会自动注册中的 Amazon 所有服务，包括 Kinesis Data Streams。您只需为使用的服务付费。

如果您已有 Amazon 账户，请跳到下一个任务。如果您还没有 Amazon 账户，请使用以下步骤创建。

如何注册 Amazon 账户

1. 打开 <https://portal.aws.amazon.com/billing/signup>。
2. 按照屏幕上的说明进行操作。

在注册时，您将接到一通电话，要求您使用电话键盘输入一个验证码。

当您注册 Amazon Web Services 账户时，系统将会创建一个 Amazon Web Services 账户根用户。根用户有权访问该账户中的所有 Amazon Web Services 和资源。作为安全最佳实践，请 [为管理用户分配管理访问权限](#)，并且只使用根用户执行 [需要根用户访问权限的任务](#)。

下载库和工具

以下库和工具将帮助您使用 Kinesis Data Streams：

- [Amazon Kinesis API 参考](#)是 Kinesis Data Streams 支持的一组基本操作。有关使用 Java 代码执行基本操作的更多信息，请参阅以下主题：
 - [将 Amazon Kinesis Data Streams API 与结合使用来开发创建者 Amazon SDK for Java \(p. 86\)](#)
 - [使用 Amazon SDK for Java 开发具有共享吞吐量的自定义使用者 \(p. 139\)](#)
 - [创建和管理流 \(p. 59\)](#)
- [Go](#)、[Java](#)、[.NET](#)、[Node.js](#)、[JavaScriptPHP](#)、[Python](#) 和 [Ruby](#) 的软件开发工具包包括 Kinesis Amazon Data Streams 支持和示例。如果您的版本 Amazon SDK for Java 不包含 Kinesis Data Streams 的示例，也可以从中下载 [GitHub](#)。
- Kinesis CL 客户端库 (KCL) 提供用于处理数据的 easy-to-use 编程模型。KCL 可以帮助您快速开始使用 Java、Node.js、.NET、Python 和 Ruby 中的 Kinesis Data Streams。有关更多信息，请参阅 [读取流中的数据 \(p. 105\)](#)。
- [Amazon Command Line Interface](#)支持 Kinesis Data Streams。Amazon CLI 使您能够从命令行控制多个 Amazon 服务并通过脚本自动执行这些服务。

配置您的开发环境

要使用 KCL，请确保您的 Java 开发环境满足以下要求：

- Java 1.7 (Java SE 7 JDK) 或更高版本。您可以从 Oracle 网站上的 [Java SE 下载](#) 页面下载最新的 Java 软件。
- Apache Commons 程序包 (代码、HTTP 客户端和日志记录)
- Jackson JSON 处理器

请注意，[Amazon SDK for Java](#) 将 Apache Commons 和 Jackson 包含在第三方文件夹中。但是，SDK for Java Java 1.6，而 Kinesis 客户端库需要 Java 1.7。

Amazon Kinesis Data Streams 入门

本节中的信息可帮助您开始使用 Amazon Kinesis Data Streams。如果您不熟悉 Kinesis Data Streams，请先熟悉中介绍的概念和术语[Amazon Kinesis Data Streams 术语和概念 \(p. 3\)](#)。

本节向您展示如何使用执行基本的 Amazon Kinesis Data Streams 操作 Amazon Command Line Interface。您将学习 Kinesis Data Streams 数据流的基本原理以及从 Kinesis 数据流中输入和获取数据所需的步骤。

主题

- [安装和配置 Amazon CLI \(p. 12\)](#)
- [使用 Amazon CLI 执行基本 Kinesis 数据流操作。\(p. 13\)](#)

要进行 CLI 访问，您需要访问密钥 ID 和秘密访问密钥。如果可能，请使用临时凭证代替长期访问密钥。临时凭证包括访问密钥 ID、秘密访问密钥，以及一个指示凭证何时到期的安全令牌。有关详细信息，请参阅《Amazon Web Services 一般参考》中的[管理 Amazon 访问密钥的最佳实践](#)。

您可以在[创建 step-by-step IAM 用户中找到详细的 IAM](#) 和安全密钥设置说明。

在本节中，讨论到的特定命令将一字不差地提供，但其特定值在每次运行时必然会不同。此外，示例使用的是美国西部（俄勒冈）区域，但本节中的步骤适用于[支持 Kinesis Data Streams 的任何区域](#)。

安装和配置 Amazon CLI

安装 Amazon CLI

有关如何安装详细步骤 Amazon CLI 有关 Windows 以及 Linux、OS X 和 Unix 操作系统，请参阅[安装 Amazon CLI](#)。

使用以下命令列出可用的选项和服务：

```
aws help
```

您将要使用 Kinesis Data Streams 服务，因此您可以使用 Kinesis 数据流服务审查 Amazon CLI 使用以下命令使用与 Kinesis Data Streams 相关的子命令：

```
aws kinesis help
```

此命令将生成包含可用 Kinesis Data Streams 命令的输出：

```
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
```

此命令列表对应于中记录的 Kinesis Data Streams API [Amazon Kinesis 服务 API 参考](#)。例如，create-stream 命令与 CreateStream API 操作对应。

Amazon CLI 现已成功安装，但未配置。这将在下一节展示。

配置 Amazon CLI

对于一般用途，aws configure 命令是设置 Amazon CLI 安装的最快方法。有关更多信息，请参阅 [配置 Amazon CLI](#)。

使用 Amazon CLI 执行基本 Kinesis 数据流操作。

本节介绍如何通过从命令行对执行 Kinesis 本操作。Amazon CLI。确保您熟悉 [Amazon Kinesis Data Streams 术语和概念 \(p. 3\)](#) 中讨论的概念。

Note

在创建流后，将象征性地向您的账户收取 Kinesis Data Streams 使用费，因为 Kinesis Data Streams 没有获得 Amazon 免费套餐。当您完成本教程时，请删除 Amazon 用于停止产生费用的资源。有关更多信息，请参阅 [第 4 步：清除 \(p. 17\)](#)。

主题

- [第 1 步：创建流 \(p. 13\)](#)
- [第 2 步：设置记录 \(p. 15\)](#)
- [第 3 步：获取记录 \(p. 15\)](#)
- [第 4 步：清除 \(p. 17\)](#)

第 1 步：创建流

您的第一步是创建一个流并验证它是否已创建成功。使用以下命令创建一个名为“Foo”的流：

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

接下来，发出以下命令以检查流的创建进度：

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

您应获得类似于以下示例的输出：

```
{
  "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
  }
}
```

在此示例中，流的状态为 CREATING，这表示它还未完全做好使用准备。在几分钟后再次检查，您应看到类似于以下示例的输出：

```
{
  "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
  }
}
```

此输出包含您在本教程中无需关注的信息。目前，您需要重点关注的是 "StreamStatus": "ACTIVE"（告知您流已做好使用准备）和有关您请求的单个分片的信息。您还可以通过使用 `list-streams` 命令验证您的新流是否存在，如下所示：

```
aws kinesis list-streams
```

输出：

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

```
}
```

第 2 步：设置记录

既然您已经拥有活动的流，您便已做好放置一些数据的准备。在本教程中，您将使用最简单的命令 `put-record`，该命令会将一个包含文本“testdata”的数据记录放入流中：

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

如果成功，此命令将生成类似于以下示例的输出：

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

恭喜，您刚刚已将数据添加到流！接下来您将了解如何从流中获取数据。

第 3 步：获取记录

GetShard迭代器

您需要先为您感兴趣的分片获取分片迭代器，然后才能从流中获取数据。分片迭代器表示使用者（在本例中为 `get-record` 命令）要从中读取数据的流和分片的位置。您将使用 `get-shard-iterator` 命令，如下所示：

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

请记住，`aws kinesis`命令后面有一个 Kinesis Data Streams API，因此如果您对显示的任何参数感兴趣，都可以在[GetShardIterator](#) API 参考主题。执行成功将产生与以下示例类似的输出（水平滚动可查看完整输出）：

```
{
  "ShardIterator": "AAAAAAAAAAHSywljv0zEgPX4NyKdZ5wryMzP9yALs8NeKbUjp1IxtZs1Sp
+KEd9I6AJ9ZG4lNR1EMi+9Md/nHvtLyxpfhEzYvkTZ4D9DQVz/mBYWR060TZRNW9gd+efGN2aHFdkH1rJl4BL9Wyrk
+ghYG22D2T1Da2EyNSH1+LabK33gQweTJADBdyMwlo5r6PqcP2dzhg="
}
```

看起来像随机字符的长字符串就是分片迭代器（您的字符串将与此不同）。您需要将分片迭代器复制/粘贴到接下来显示的 `get` 命令中。分片迭代器的有效生命周期为 300 秒，应该足以让您将分片迭代器复制/粘贴到下一个命令中。请注意，在将分片迭代器粘贴到写一个命令之前，您需要从中删除所有换行符。如果您收到分片迭代器不再有效的错误消息，只需再次执行 `get-shard-iterator` 命令。

GetRecords

这些区域有：`get-records`命令从流中获取数据，然后它解析为调用[GetRecords](#)在 Kinesis Data Streams API 中。分片迭代器指定了分片中的一个位置，您希望从该位置开始按顺序读取数据记录。如果迭代器指向的分片中的部分没有可用的记录，`GetRecords` 将返回空白列表。请注意，可能需要进行多次调用才能到达分片中包含记录的部分。

在以下 `get-records` 命令示例中（水平滚动可查看完整命令）：

```
aws kinesis get-records --shard-iterator
  AAAAAAAAAAAHSywljv0zEgPX4NyKdZ5wryMzP9yALs8NeKbUjp1IxtZs1Sp+KEd9I6AJ9ZG4lNR1EMi
```

```
+9Md/nHvtLyxpFhEzYvkTZ4D9DQVz/mBYWR060TZRNW9gd+efGN2aHFdkH1rJl4BL9Wyrk  
+ghYG22D2T1Da2EyNSH1+LAbK33gQweTJADbDyMwlo5r6PqcP2dzhg=
```

如果您是从 Unix 类型的命令处理器（如 bash）运行本教程，则可以使用嵌套命令自动执行分片迭代器的获取，如下所示（水平滚动可查看完整命令）：

```
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
```

如果你是从支持的系统上运行本教程 PowerShell，您可以使用如下所示的命令自动执行分片迭代器的获取（水平滚动可查看完整命令）：

```
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])
```

get-records 命令的成功结果将从您在获取分片迭代器时指定的分片的流中请求记录，如下示例所示（水平滚动可查看完整输出）：

```
{  
  "Records": [ {  
    "Data": "dGVzdGRhdGE=",  
    "PartitionKey": "123",  
    "ApproximateArrivalTimestamp": 1.441215410867E9,  
    "SequenceNumber": "49544985256907370027570885864065577703022652638596431874"  
  } ],  
  "MillisBehindLatest": 24000,  
  
  "NextShardIterator": "AAAAAAAAAED0W3ugseWPE4503kqN1yN1UaodY8unE0sYs1MUmc6lX9hlig5+t4RtZM0/  
tALfiI4QGjunVgJvQsjxjh2aLyxaAaPr  
+LaoENQ7eVs4EdYXgKyThTZGPcca2fVXYJWL3yafv9dsDwsYVedI66dbMZFC8rPMwc797zxQkv4pSKvP0ZvrUIudb8UkH3VMzx58Is=  
}"
```

请注意，get-records 在上面被描述为请求，这意味着即使您的流中有记录，您可能也会收到零个或零个以上的记录，并且任何返回的记录都无法表示当前您的流中的所有记录。这是完全正常的，并且生产代码只会以适当的时间间隔轮询流中的记录（此轮询速度因您的特定应用程序设计要求而异）。

在本教程的这一部分你可能会注意到你的记录的第一件事是数据似乎是垃圾——而不是明文 testdata 我们发送的。这归因于 put-record 使用 Base64 编码支持您发送二进制数据的方式。但是，Kinesis Data Streams 支持 Amazon CLI 不提供 Base64 解码中因为 Base64 解码到打印到 stdout 的原始二进制内容可能会导致某些平台和终端上的不良行为和潜在的安全问题。如果您使用 Base64 解码程序（例如，<https://www.base64decode.org/>）对 dGVzdGRhdGE= 进行手动解码，您将看到它实际上是 testdata。这对本教程来说已足够，在实践中，Amazon CLI 很少用于使用数据，更多时候是用于监控流的状态和获取信息，如前面所示（describe-stream 和 list-streams）。将来的教程将向您展示如何使用 Kinesis 客户端库（KCL）构建生产质量的使用者应用程序，KCL 将会为您处理 Base64。有关 KCL 的更多信息，请参阅[使用 KCL 开发具有共享吞吐量的自定义使用者](#)。

get-records 并非总是会返回在流/分片中指定的所有记录。当出现这种情况时，请使用最后一个结果中的 NextShardIterator 获取下一组记录。因此，如果更多数据正在被放入流中（正常情况下在生产应用程序中），您每次都可以使用 get-records 持续轮询数据。但是，如果您在 300 秒的分片迭代器生命周期内未使用下一个分片迭代器调用 get-records，则会收到一条错误消息，并且需要使用 get-shard-iterator 命令来获取新的分片迭代器。

此输出中还提供了 MillisBehindLatest，它是从流的末端响应 GetRecords 操作的毫秒数，指示使用者落后当前时间多远。零值指示正在进行记录处理，此时没有新的记录要处理。在本教程中，如果您一边阅读教程一边操作，则可能会看到这个数值非常大。默认情况下，数据记录在流中保留 24 小时以等待您检索它们。此时间范围称为保留期，可以配置为最多 365 天。

请注意，一个成功的 `get-records` 结果总是有一个 `NextShardIterator`，即使目前流中没有更多记录。这是一个假定创建器在任何给定时间内正在将更多记录放入流中的轮询模型。虽然您可编写自己的轮询例程，但如果您使用之前提到的 KCL 开发使用者应用程序，则系统将会为您执行此轮询。

如果您调用 `get-records`，直到您正在提取的流和分片中沒有更多记录，您将看到带有空白记录的输出，类似于以下示例（水平滚动可查看完整输出）：

```
{
  "Records": [],
  "NextShardIterator": "AAAAAAAAAGCJ5jzQNjmdh06B/YDIDE56jmZmrmMA/r1WjoHXC/
kPJXc1rckt3TFL55dENfe5meNgdkyCRpUPGzJpMgYHaJ53C3nCAjQ6s7ZupjXeJGoUFs5oCuFwhP+Wul/
EhyNeSs5DYXLSSC5XCcapMCAYGFjYER69QsdQjxMmBPE/hiybFDi5qtkT6/PsZNz6kFoqtDk="
}
```

第 4 步：清除

最后，如前所述，您希望删除您的流以释放资源和避免您的账户产生意外费用。在实践中，每当您创建了不会使用的流时，请执行此操作，因为费用是按流量计算的，无论您是否在使用流放入和获取数据。清除命令很简单：

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

成功之后不会生成输出，因此您可能希望使用 `describe-stream` 来检查删除进度：

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

如果您在执行删除命令后立即执行此命令，您可能会看到其中的输出部分类似于以下示例：

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

在流完全删除后，`describe-stream` 将生成“未找到”错误：

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

Amazon Kinesis Data Streams 示例教程

本节中的示例教程旨在进一步帮助您了解 Amazon Kinesis Data Streams 的概念和功能。

主题

- [教程：使用 KPL 和 KCL 2.x 处理实时股票数据 \(p. 18\)](#)
- [教程：使用 KPL 和 KCL 1.x 处理实时股票数据 \(p. 31\)](#)
- [教程：使用适用于 Flink 应用程序的 Kinesis 数据分析实时股票数据 \(p. 43\)](#)
- [教程：使用 Amazon Lambda Kinesis Data Streams \(p. 58\)](#)
- [Amazon Kinesis 的流数据解决方案 \(p. 58\)](#)

教程：使用 KPL 和 KCL 2.x 处理实时股票数据

本教程的场景包括将股票交易提取到数据流中，然后编写一个简单的 Amazon Kinesis Data Streams 应用程序来对数据流进行计算。您将学习如何向 Kinesis Data Streams 发送记录流，以及如何实现近乎实时地使用和处理记录的应用程序。

Important

创建直播后，由于 Kinesis Data Streams 不符合 Amazon 免费套餐的资格，您的账户将收取 Kinesis Data Streams 的使用费。消费者应用程序启动后，使用 Amazon DynamoDB 也会产生象征性费用。消费者应用程序使用 DynamoDB 来跟踪处理状态。完成此应用程序后，删除您的 Amazon 资源以停止产生费用。有关更多信息，请参阅[第 7 步：收尾 \(p. 30\)](#)：

代码不访问实际股票市场数据，而是模拟股票交易流。它通过使用随机股票交易生成器（将截至 2015 年 2 月市值排名前 25 位的股票的实际市场数据作为起始点）来执行此操作。如果您有权访问实时的股票交易流，则可能有兴趣从该流派生有用且及时的统计数据。例如，您可能希望执行滑动窗口分析，从而确定前 5 分钟内购买的最热门股票。或者，您可能希望在销售订单过大（即具有过多股份）时收到通知。可以扩展此系列代码以提供此类功能。

您可以在台式计算机或笔记本电脑上演练本教程中的步骤，然后在同一台计算机或支持已定义要求的任何平台上同时运行创建器和使用代码。

显示的示例使用美国西部（俄勒冈）区域，但它们适用于支持 Kinesis Data Streams 的任何[Amazon 区域](#)。

任务

- [先决条件 \(p. 19\)](#)
- [步骤 1：创建数据流 \(p. 19\)](#)
- [步骤 2：创建 IAM 策略和用户 \(p. 20\)](#)
- [步骤 3：下载并构建代码 \(p. 23\)](#)
- [第 4 步：实施创建器 \(p. 23\)](#)
- [第 5 步：实施使用者 \(p. 26\)](#)
- [第 6 步：\(可选\) 扩展使用者 \(p. 29\)](#)
- [第 7 步：收尾 \(p. 30\)](#)

先决条件

您必须满足以下要求才能完成本教程：

Amazon Web Services 账户

在开始之前，请确保熟悉[Amazon Kinesis Data Streams 术语和概念 \(p. 3\)](#)中讨论的概念，特别是流、分片、创建器和使用。完成以下指南中的步骤也很有帮助：[安装和配置 Amazon CLI \(p. 12\)](#)。

您必须具有 Amazon 账户和 Web 浏览器才能访问 Amazon Web Services Management Console。

要访问控制台，请使用您的 IAM 用户名和密码[Amazon Web Services Management Console](#)从 [IAM 登录页面登录](#)。IAM 让您可以安全地控制对您的 Amazon Web Services 账户中 Amazon Web Services 和资源的访问。有关控制台和编程证书的详细信息，请参阅中的[了解和获取您的安全证书Amazon Web Services 一般参考](#)。

有关 IAM 和安全密钥设置说明的更多信息，请参阅[创建 IAM 用户](#)。

系统软件要求

用于运行应用程序的系统必须已安装 Java 7 或更高版本。要下载和安装最新 Java 开发工具包 (JDK)，请转到 [Oracle 的 Java SE 安装站点](#)。

如果您具有 Java IDE (如 [Eclipse](#))，则可使用它来打开、编辑、构建并运行源代码。

您需要最新的 [Amazon SDK for Java](#) 版本。如果你使用 Eclipse 作为 IDE，你可以改为[安装 EclipseAmazon 工具包](#)。

消费者应用程序需要 Kinesis 客户端库 (KCL) 版本 2.2.9 或更高版本，您可以[通过 GitHub https://github.com/aws-labs/amazon-kinesis-client/tree/master](#) 获取。

后续步骤

[步骤 1：创建数据流 \(p. 19\)](#)

步骤 1：创建数据流

首先，您必须创建将在本教程的后续步骤中使用的数据流。

创建流

1. 登录到 Amazon Web Services Management Console，然后通过以下网址打开 Kinesis 控制台：<https://console.aws.amazon.com/kinesisvideo/home>。
2. 在导航窗格中，选择 Data Streams (数据流)。
3. 在导航栏中，展开区域选择器并选择一个区域。
4. 选择 Create Kinesis stream (创建 Kinesis 流)。
5. 输入数据流的名称 (例如，**StockTradeStream**)。
6. 在分片数量中输入 **1**，但保留 Estimate the number of shards you'll need (估计您需要的分片数量) 为折叠状态。
7. 选择 Create Kinesis stream (创建 Kinesis 流)。

在 Kinesis 直播列表页面上，您的直播状态显示为 CREATING 正在创建直播的状态。当流可以使用时，状态会更改为 ACTIVE。

如果您选择流的名称，在显示的页面中，Details (详细信息) 选项卡会显示数据流的配置摘要。Monitoring (监控) 部分显示流的监控信息。

后续步骤

[步骤 2：创建 IAM 策略和用户 \(p. 20\)](#)

步骤 2：创建 IAM 策略和用户

Amazon 规定使用细粒度权限来控制对不同资源的访问权限的安全最佳实践。Amazon Identity and Access Management (IAM) 可让用户和权限 Amazon。IAM 策略明确列出了允许的操作以及这些操作适用于的资源。

以下是 Kinesis Data Streams 制作者和使用者通常需要的最低权限。

创建者

操作	资源	目的
DescribeStream, DescribeStreamSummary, DescribeStreamConsumer	Kinesis 数据流	在尝试读取记录前，使用者会检查数据流是否存在，数据流中。
SubscribeToShard, RegisterStreamConsumer	Kinesis 数据流	订阅使用者并将其注册到分片。
PutRecord, PutRecords	Kinesis 数据流	将记录写入 Kinesis Data Streams。

使用者

操作	资源	目的
DescribeStream	Kinesis 数据流	在尝试读取记录前，使用者会检查数据流是否存在，数据流中。
GetRecords, GetShardIterator	Kinesis 数据流	从分片读取记录。
CreateTable, DescribeTable, GetItem, PutItem, Scan, UpdateItem	Amazon DynamoDB 表	如果使用者是使用 Kinesis 客户端库 (KCL) (版本 1.x 或 2.x) 跟踪应用程序的处理状态。
DeleteItem	Amazon DynamoDB 表	适用于使用者在 Kinesis Data Streams 分区上执行拆分/合并。
PutMetricData	亚马逊 CloudWatch 日志	KCL 还会将指标上传到 CloudWatch，这对于监控应用程序。

在本教程中，您将创建一个授予上述所有权限的单个 IAM 策略。在生产中，您可能需要创建两个策略，一个针对创建器，另一个针对使用者。

创建 IAM policy

1. 找到您在上述步骤中创建的新数据流的 Amazon 资源名称 (ARN)。您可以在“详细信息”选项卡的顶部找到以 Stream ARN 形式列出的此 ARN。ARN 格式如下所示：

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

区域

Amazon 区域代码；例如 us-west-2。有关更多信息，请参阅 [区域和可用区域概念](#)。

账户

Amazon 账户 ID，如 [账户设置](#) 中所示。

name

您在上述步骤中创建的数据流的名称，即 StockTradeStream。

2. 确定使用者要使用的 DynamoDB 表的 ARN（并由第一个使用者实例创建）。它必须采用以下格式：

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

区域和账户 ID 与您在教程中使用的数据流 ARN 中的值相同，但名称是使用者应用程序创建和使用的 DynamoDB 表的名称。KCL 使用应用程序名称作为表名称。在此步骤中，使用 StockTradesProcessor DynamoDB 表名称，因为这是本教程后续步骤中使用的应用程序名称。

3. 在 IAM 控制台的策略 (<https://console.aws.amazon.com/iam/home#policies>) 中，选择创建策略。如果这是您第一次使用 IAM 策略，请选择“开始，创建策略”。
4. 在 Policy Generator 旁，选择 Select。
5. 选择 Amazon Kinesis 作为 Amazon 服务。
6. 选择 DescribeStream、GetShardIterator、GetRecords、PutRecord 和 PutRecords 作为允许的操作。
7. 输入您在本教程中使用的数据流的 ARN。
8. 对以下各项使用 Add Statement (添加语句)：

Amazon 服务	操作	ARN
Amazon DynamoDB	CreateTable, DeleteItem, DescribeTable, GetItem, PutItem, Scan, UpdateItem	您在本过程的步骤 2 中创建的 DynamoDB 表的 ARN。
亚马逊 CloudWatch	PutMetricData	*

在不需要指定 ARN 时使用的星号 (*)。在这种情况下，这是因为没有可调用 PutMetricData 操作 CloudWatch 的特定资源。

9. 选择 Next Step。
10. 将 Policy Name (策略名称) 更改为 StockTradeStreamPolicy，审阅代码，然后选择 Create Policy (创建策略)。

生成的策略文档应该如下所示：

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

```

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

创建 IAM 用户

1. 通过以下网址打开 IAM 控制台：<https://console.aws.amazon.com/iam/>。
2. 在 Users (用户) 页面上，选择 Add user (添加用户)。
3. 对于 User name，键入 StockTradeStreamUser。
4. 对于 Access type (访问类型)，选择 Programmatic access (编程访问)，然后选择 Next: Permissions (下一步: 权限)。
5. 选择 Attach existing policies directly (直接附上现有策略)。
6. 按名称搜索您在上述过程中创建的策略 (StockTradeStreamPolicy)。选中策略名称左侧的框，然后选择 Next: Review (下一步: 审核)。
7. 查看详细信息和摘要，然后选择 Create user (创建用户)。
8. 复制 Access key ID (访问密钥 ID)，并将其私下保存。在 Secret access key (私有访问密钥) 下面选择 Show (显示)，然后也将该密钥私下保存。
9. 将访问密钥和私有密钥粘贴到一个只有您可以访问的位于安全位置的本地文件中。对于此应用程序，请创建名为 ~/.aws/credentials (具有严格权限) 的文件。该文件应采用以下格式：

```
[default]
```

```
aws_access_key_id=access key  
aws_secret_access_key=secret access key
```

将 IAM 策略附加到用户

1. 在 IAM 控制台中，打开 [策略](#) 并选择策略操作。
2. 选择 StockTradeStreamPolicy 和 Attach (附加)。
3. 选择 StockTradeStreamUser 和 Attach Policy (附加策略)。

后续步骤

[步骤 3：下载并构建代码 \(p. 23\)](#)

步骤 3：下载并构建代码

本主题提供了引入数据流（创建器）和处理此数据（使用者）的样本股票交易的示例实施代码。

下载并构建代码

1. 将源代码从 <https://github.com/aws-samples/amazon-kinesis-learning> GitHub 存储库下载到您的计算机上。
2. 按照提供的目录结构，使用源代码在您的 IDE 中创建一个项目。
3. 将以下库添加到该项目中：
 - Amazon Kinesis 客户端库 (KCL)
 - Amazon SDK
 - 阿帕奇 HttpCore
 - 阿帕奇 HttpClient
 - Apache Commons Lang
 - Apache Commons Logging
 - Guava (适用于 Java 的 Google 核心库)
 - Jackson Annotations
 - Jackson Core
 - Jackson Databind
 - Jackson Dataformat : CBOR
 - Joda Time
4. 根据您的 IDE，项目可能会自动构建。如果未自动构建项目，请使用适合您的 IDE 的步骤构建项目。

如果已成功完成这些步骤，则可进入下一节 [the section called “第 4 步：实施创建器” \(p. 23\)](#)。

后续步骤

[\(p. 23\)](#)

第 4 步：实施创建器

此教程使用股票市场交易监控的实际场景。以下准则简要说明了此场景如何映射到创建器及其支持的代码结构。

请参阅[源代码](#)并查看以下信息。

StockTrade 阶级

单次股票交易由一个 StockTrade 类实例表示。此实例包含一些属性，如股票代码、价格、股份数、交易类型（买入或卖出）以及唯一标识交易的 ID。将为您实现此类。

流记录

流是一个记录序列。记录是 JSON 格式的 StockTrade 实例序列化。例如：

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

StockTradeGenerator 阶级

StockTradeGenerator 有一个名为的方法 getRandomTrade()，每次调用时都会返回一个新的随机生成股票交易。将为您实现此类。

StockTradesWriter 阶级

生产者的 main 方法 StockTradesWriter 持续检索随机交易，然后通过执行以下任务将其发送到 Kinesis Data Streams：

1. 将数据流名称和区域名称作为输入读取。
2. 使用 KinesisAsyncClientBuilder 来设置区域、凭证和客户端配置。
3. 检查流是否存在且处于活动状态 (如果不是这样，它将退出并显示错误)。
4. 在连续循环中，会依次调用 StockTradeGenerator.getRandomTrade() 方法和 sendStockTrade 方法以便每 100 毫秒将交易发送到流一次。

sendStockTrade 类的 StockTradesWriter 方法具有以下代码：

```
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);
    }
}
```

```
    }  
}
```

请参阅以下代码细分：

- PutRecord API 需要一个字节数组，并且您需要将交易转换为 JSON 格式。此行代码将执行该操作：

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

- 您需要先创建新的 PutRecordRequest 实例（此示例中称为请求），然后才能发送交易。每个 request 均需要流名称、分区键和数据 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();
```

该示例使用股票行情自动收录器作为将记录映射到特定分片的分区键。实际上，每个分片应具有数百或数千个分区键，以便记录均匀地分布在流中。有关如何将数据添加到流的更多信息，请参阅 [将 Data Streams 添加到 Amazon Kinesis Data Streams \(p. 76\)](#)。

现在 request 已准备好发送到客户端（put 操作）：

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

- 错误检查和日志记录始终是有用的附加功能。此代码将记录错误条件：

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

围绕 put 操作添加 try/catch 块：

```
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);  
}
```

这是因为 Kinesis 数据流 put 操作可能因网络错误或数据流达到其吞吐量限额并受到限制而导致失败。建议您仔细考虑针对 put 操作的重试策略以避免数据丢失（如用作简单重试）。

- 状态日志记录很有用，但它是可选的：

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

此处显示的制作人使用 Kinesis Data Streams API 单条记录功能 `PutRecord`。实际上，如果单个创建者生成许多记录，则使用 `PutRecords` 的多记录功能并一次性发送批量记录通常会更有效。有关更多信息，请参阅 [将 Data Streams 写入 Amazon Kinesis Data Streams \(p. 76\)](#)：

运行创建器

1. 验证在 [步骤 2：创建 IAM 策略和用户 \(p. 20\)](#) 中检索到的访问密钥和私有密钥对是否保存到文件 `~/.aws/credentials` 中。
2. 使用以下参数运行 `StockTradeWriter` 类：

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

如果您在 `us-west-2` 之外的区域中创建流，则必须改为在此处指定该区域。

您应该可以看到类似于如下所示的输出内容：

```
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
```

您的股票交易现在正被 Kinesis Data Streams 采集。

后续步骤

[第 5 步：实施使用者 \(p. 26\)](#)

第 5 步：实施使用者

本教程中的使用者应用程序持续处理您数据流中的股票交易。然后，它输出每分钟买入和卖出最多的股票。该应用程序建立在 Kinesis 客户端库 (KCL) 之上，它可以完成消费类应用程序常见的大部分繁重工作。有关更多信息，请参阅 [使用 Kinesis 客户端库 \(p. 107\)](#)：

请参阅源代码并查看以下信息。

StockTradesProcessor 阶级

为您提供的使用者的主类，它将执行以下任务：

- 读取作为参数传递的应用程序名称、数据流名称和区域名称。
- 使用区域名称创建 `KinesisAsyncClient` 实例。
- 创建一个 `StockTradeRecordProcessorFactory` 实例，该实例提供由 `ShardRecordProcessor` 实例实施的 `StockTradeRecordProcessor` 的实例。

- 使用 `KinesisAsyncClient`、`StreamName`、`ApplicationName` 和 `StockTradeRecordProcessorFactory` 实例创建 `ConfigsBuilder` 实例。这对于创建具有默认值的所有配置非常有用。
- 使用 `ConfigsBuilder` 实例创建一个 KCL 计划程序（以前在 KCL 版本 1.x 中称为 KCL 工作线程）。
- 此计划程序为每个分片（已分配给此使用者实例）创建一个线程，以持续循环从数据量读取记录。之后，它调用 `StockTradeRecordProcessor` 实例以处理收到的每批记录。

StockTradeRecordProcessor 阶级

`StockTradeRecordProcessor` 实例的实施，该实例反过来将实施五个必需方法：`initialize`、`processRecords`、`leaseLost`、`shardEnded` 和 `shutdownRequested`。

`initialize` 和 `shutdownRequested` 方法由 KCL 使用，旨在让记录处理器分别了解何时应准备好开始接收记录，以及何时应停止接收记录，因此该方法可以执行任何特定于应用程序的设置和终止任务。`leaseLost` 和 `shardEnded` 用于实施当租约丢失或处理达到分片末尾时需执行的操作的任何逻辑。在此示例中，我们只记录指示这些事件的消息。

将为您提供这些方法的代码。`processRecords` 方法中进行的主要处理，该处理反过来对每条记录使用 `processRecord`。后一个方法作为大体为空的框架代码提供给您，以便您在下一步骤中实施，届时将更详细地对其进行说明。

另外要注意的是对 `processRecord` 的支持方法 `reportStats` 和 `resetStats` 的实施，二者在初始源代码中为空。

已为您实施 `processRecords` 方法，并执行了以下步骤：

- 对于传入的每条记录，它会对调用 `processRecord`。
- 如果自上一次报告以来已过去至少 1 分钟，请调用 `reportStats()`（它将打印出最新统计数据），然后调用 `resetStats()`（它将清除统计数据以便下一个间隔仅包含新记录）。
- 设置下一次报告时间。
- 如果自上一检查点以来已过去至少 1 分钟，请调用 `checkpoint()`。
- 设置下一次检查点操作时间。

此方法使用 60 秒间隔作为报告和检查点操作比率。有关检查点的更多信息，请参阅[使用 Kinesis 客户端库](#)。

StockStats 阶级

此类提供一段时间内针对最热门股票的数据保留和统计数据跟踪。此代码已提供给您并包含以下方法：

- `addStockTrade(StockTrade)`：将给定的 `StockTrade` 注入正在使用的统计数据。
- `toString()`：以格式化字符串形式返回统计数据。

此类跟踪最热门股票的方式是，保留每只股票的总交易数的连续计数和最大计数。每当股票交易达成时，它都会更新这些计数。

将代码添加到 `StockTradeRecordProcessor` 类的方法，如以下步骤中所示。

实施使用者

1. 通过实例化大小正确的 `processRecord` 对象并将记录数据添加到该对象来实施 `StockTrade` 方法，并在出现问题时记录警告。

```
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. 实施简单的 `reportStats` 方法。可随时将输出格式修改为适合您的首选格式。

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

3. 实施 `resetStats` 方法，这将创建新的 `stockStats` 实例。

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

4. 实施 `ShardRecordProcessor` 接口所需的以下方法

```
@Override  
public void leaseLost(LeaseLostInput leaseLostInput) {  
    log.info("Lost lease, so terminating.");  
}  
  
@Override  
public void shardEnded(ShardEndedInput shardEndedInput) {  
    try {  
        log.info("Reached shard end checkpointing.");  
        shardEndedInput.checkpointer().checkpoint();  
    } catch (ShutdownException | InvalidStateException e) {  
        log.error("Exception while checkpointing at shard end. Giving up.", e);  
    }  
}  
  
@Override  
public void shutdownRequested(ShutdownRequestedInput shutdownRequestedInput) {  
    log.info("Scheduler is shutting down, checkpointing.");  
    checkpoint(shutdownRequestedInput.checkpointer());  
}  
  
private void checkpoint(RecordProcessorCheckpointer checkpointer) {  
    log.info("Checkpointing shard " + kinesisShardId);  
    try {  
        checkpointer.checkpoint();  
    } catch (ShutdownException se) {  
        // Ignore checkpoint if the processor instance has been shutdown (fail over).  
        log.info("Caught shutdown exception, skipping checkpoint.", se);  
    } catch (ThrottlingException e) {  
        // Skip checkpoint when throttled. In practice, consider a backoff and retry  
        policy.  
        log.error("Caught throttling exception, skipping checkpoint.", e);  
    } catch (InvalidStateException e) {  
        // This indicates an issue with the DynamoDB table (check for table,  
        provisioned IOPS).  
        log.error("Cannot save checkpoint to the DynamoDB table used by the Amazon  
        Kinesis Client Library.", e);  
    }  
}
```

运行使用者

1. 运行您在 [\(p. 23\)](#) 中编写的创建者以将模拟股票交易记录引入流中。
2. 验证先前 (创建 IAM 用户时) 检索的访问密钥和key pair 是否保存在文件中 ~/.aws/credentials。
3. 使用以下参数运行 StockTradesProcessor 类：

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

请注意，如果您在 us-west-2 之外的区域中创建流，则必须改为在此处指定该区域。

- 1 分钟后，您应看到类似以下内容的输出，并且输出在此后每分钟刷新一次：

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

后续步骤

[第 6 步：\(可选\) 扩展使用者 \(p. 29\)](#)

第 6 步：(可选) 扩展使用者

此可选部分演示如何针对更为复杂的场景扩展使用者代码。

如果要了解每分钟的最大销售订单数，可以修改三个位置的 StockStats 类以适应此新的优先级。

扩展使用者

1. 添加新实例变量：

```
// 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. 将以下代码添加到 addStockTrade：

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

3. 修改 toString 方法以打印其他信息：

```
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);  
}
```

如果您现在运行使用者（请记住同时运行创建器），则应看到类似于以下内容的输出：

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

后续步骤

[第 7 步：收尾 \(p. 30\)](#)

第 7 步：收尾

由于您要付费使用 Kinesis 数据流，因此请确保在完成后将其和相应的 Amazon DynamoDB 表删除。即使您不发送和获取记录，活动流也会产生象征性的费用。这是因为活动流将持续“侦听”传入记录和获取记录的请求，这将耗用资源。

删除流和表

1. 关闭您可能仍在运行的任何创建者和使用者。
2. 打开 Kinesis 控制台，网址为：<https://console.aws.amazon.com/kinesis>。
3. 选择为此应用程序创建的流 (StockTradeStream)。
4. 选择 Delete Stream (删除流)。
5. 从 <https://console.aws.amazon.com/dynamodb/> 打开 DynamoDB 控制台。
6. 删除 StockTradesProcessor 表。

摘要

近乎实时处理大量数据不需要编写任何功能强大的代码或开发大型基础设施。它就像编写逻辑来处理少量数据（例如写入 `processRecord(Record)`）一样简单，但使用 Kinesis Data Streams 进行扩展以使其适用于大量流数据。你不必担心你的处理会如何扩展，因为 Kinesis Data Streams 会为你处理。你所要做的就是将你的直播记录发送到 Kinesis Data Streams，然后编写处理收到的每条新记录的逻辑。

以下是针对此应用程序的一些可能的改进。

跨所有分片进行聚合

当前，通过聚合单个分片中单个工作线程收到的数据记录来获得统计数据。（一个分片不能同时由单个应用程序中的多个工作线程处理。）当然，当您扩展并具有多个分片时，可能希望跨所有分片聚合。可通过部署管道架构完成此操作。在该架构中，每个工作线程的输出都注入具有单个分片的另一个流，分片由聚合第一个阶段输出的工作线程处理。由于来自第一个阶段的数据是有限的（每分片每分钟一个示例），因此一个分片即可轻松处理它。

缩放处理

当流进行扩展以包含多个分片（因为多个创建器正在发送数据）时，扩展处理的方式是添加更多工作程序。您可以在 Amazon EC2 实例中运行工作程序并使用 Auto Scaling 组。

使用连接器连接亚马逊 s3/Dynamodb/Amazon Redshift/Storm

随着数据流的持续处理，其输出可以发送到其他目的地。Amazon 提供[用于将 Kinesis Data Streams 与其他 Amazon 服务和第三方工具集成的连接器](#)。

教程：使用 KPL 和 KCL 1.x 处理实时股票数据

本教程的场景包括将股票交易提取到数据流中，然后编写一个简单的 Amazon Kinesis Data Streams 应用程序来对数据流进行计算。您将学习如何向 Kinesis Data Streams 发送记录流，以及如何实现近乎实时地使用和处理记录的应用程序。

Important

创建直播后，由于 Kinesis Data Streams 不符合 Amazon 免费套餐的资格，您的账户将收取 Kinesis Data Streams 的使用费。消费者应用程序启动后，使用 Amazon DynamoDB 也会产生象征性费用。消费者应用程序使用 DynamoDB 来跟踪处理状态。完成此应用程序后，删除您的 Amazon 资源以停止产生费用。有关更多信息，请参阅[第 7 步：收尾 \(p. 42\)](#)：

代码不访问实际股票市场数据，而是模拟股票交易流。它通过使用随机股票交易生成器（将截至 2015 年 2 月市值排名前 25 位的股票的实际市场数据作为起始点）来执行此操作。如果您有权访问实时的股票交易流，则可能有兴趣从该流派生有用且及时的统计数据。例如，您可能希望执行滑动窗口分析，从而确定前 5 分钟内购买的最热门股票。或者，您可能希望在销售订单过大（即具有过多股份）时收到通知。可以扩展此系列代码以提供此类功能。

您可以在台式机或笔记本电脑上完成本教程中的步骤，并在同一台计算机或任何支持已定义要求的平台（例如 Amazon Elastic Compute Cloud (Amazon EC2)）上运行生产者和使用者的代码。

显示的示例使用美国西部（俄勒冈）区域，但它们适用于任何[支持 Kinesis Data Streams 的 Amazon 区域](#)。

任务

- [先决条件 \(p. 31\)](#)
- [步骤 1：创建数据流 \(p. 32\)](#)
- [步骤 2：创建 IAM 策略和用户 \(p. 33\)](#)
- [步骤 3：下载和构建实施代码 \(p. 36\)](#)
- [第 4 步：实施创建器 \(p. 37\)](#)
- [第 5 步：实施使用者 \(p. 39\)](#)
- [第 6 步：\(可选\) 扩展使用者 \(p. 42\)](#)
- [第 7 步：收尾 \(p. 42\)](#)

先决条件

以下是完成[教程：使用 KPL 和 KCL 1.x 处理实时股票数据 \(p. 31\)](#)的要求。

Amazon Web Services 技

在开始之前，请确保熟悉[Amazon Kinesis Data Streams 术语和概念 \(p. 3\)](#)中讨论的概念，特别是流、分片、创建者和使用者。参阅[安装和配置 Amazon CLI \(p. 12\)](#)也很有帮助。

您需要 Amazon 账户和 Web 浏览器才能访问 Amazon Web Services Management Console。

要访问控制台，请使用您的 IAM 用户名和密码[Amazon Web Services Management Console](#)从 [IAM 登录页面登录](#)。IAM 让您以安全地控制对您的 Amazon Web Services 账户中 Amazon Web Services 和资源的访问。有关控制台和编程证书的详细信息，请参阅中的[了解和获取您的安全证书Amazon Web Services 一般参考](#)。

有关 IAM 和安全密钥设置说明的更多信息，请参阅[创建 IAM 用户](#)。

系统软件要求

用于运行应用程序的系统必须已安装 Java 7 或更高版本。要下载和安装最新 Java 开发工具包 (JDK)，请转到 [Oracle 的 Java SE 安装站点](#)。

如果您具有 Java IDE（如 [Eclipse](#)），则可打开源代码，然后编辑、构建并运行它。

您需要最新的 [Amazon SDK for Java](#) 版本。如果你使用 Eclipse 作为 IDE，你可以改为[安装 EclipseAmazon 工具包](#)。

消费者应用程序需要 Kinesis 客户端库 (KCL) 版本 1.2.1 或更高版本，您可以从 [Kinesis GitHub s 客户端库 \(Java\)](#) 获取。

后续步骤

[步骤 1：创建数据流 \(p. 32\)](#)

步骤 1：创建数据流

在 [教程：使用 KPL 和 KCL 1.x 处理实时股票数据 \(p. 31\)](#) 的第一步中，创建后续步骤中将用到的流。

创建流

1. 登录到 Amazon Web Services Management Console，然后通过以下网址打开 Kinesis 控制台：<https://console.aws.amazon.com/kinesisvideo/home>。
2. 在导航窗格中，选择 Data Streams (数据流)。
3. 在导航栏中，展开区域选择器并选择一个区域。
4. 选择 Create Kinesis stream (创建 Kinesis 流)。
5. 输入流的名称（例如，**StockTradeStream**）。
6. 在分片数量中输入 **1**，但保留 Estimate the number of shards you'll need (估计您需要的分片数量) 为折叠状态。
7. 选择 Create Kinesis stream (创建 Kinesis 流)。

在 Kinesis 直播列表页面上，直播的状态为创建直播CREATING时。当流可以使用时，状态会更改为 ACTIVE。选择流的名称。在显示的页面中，Details (详细信息) 选项卡会显示您的流配置摘要。Monitoring (监控) 部分显示流的监控信息。

有关分片的其他信息

当您在教程之外开始使用 Kinesis Data Streams 时，可能需要更仔细地规划直播创建过程。在配置分片时，您应规划预计最大需求。以此方案为例，美国股票市场某一天（东部时间）的交易流量峰值以及需求估计值应该从这一天的时间中采样。随后，您可以选择配置最大预计需求，或扩大或缩小流以响应需求波动。

分片是吞吐容量的单位。在创建 Kinesis 直播页面上，展开估算所需的分片数量。根据以下准则输入平均记录大小、每秒写入的最大记录数以及使用应用程序数量：

平均记录大小

您的记录的计算平均大小的估计值。如果您不知道此值，请使用估计的最大记录大小作为此值。

最大写入记录数

考虑提供数据的实体的数量以及每个实体每秒生成的记录的大约数量。例如，如果要从 20 台交易服务器获取股票交易数据，并且每台服务器每秒生成 250 次交易，则每秒的交易（记录）总数为 5000。

使用的应用程序数

应用程序的数量，这些应用程序单独从流进行读取以采用不同的方式处理流并生成不同的输出。每个应用程序可具有在不同计算机上运行（即在群集中运行）的多个实例，以便能跟进大容量流。

如果显示的估计分片数量超过当前分片数量限制，则可能需要先提交提高限制的请求，然后才能创建具有此分片数量的流。要申请提高分区限制，请使用 [Kinesis Data Streams 限制表单](#)。有关数据流和分片的更多信息，请参阅[创建和管理流 \(p. 59\)](#)。

后续步骤

[步骤 2：创建 IAM 策略和用户 \(p. 33\)](#)

步骤 2：创建 IAM 策略和用户

Amazon 规定使用细粒度权限来控制对不同资源的访问权限的安全最佳实践。Amazon Identity and Access Management(IAM) 可让用户和用户权限 Amazon。IAM 策略明确列出了允许的操作以及这些操作适用于的资源。

以下是 Kinesis Data Streams 制作者和使用者通常需要的最低权限。

创建者

操作	资源	目的
DescribeStream, DescribeStreamSummary, DescribeStreamConsumer	Kinesis 数据流	在尝试写入记录之前，创建者会检查流是否存在且处于活动状态。
SubscribeToShard, RegisterStreamConsumer	Kinesis 数据流	订阅 Kinesis 数据流分片并注册一个消费者。
PutRecord, PutRecords	Kinesis 数据流	将记录写入 Kinesis Data Streams。

使用者

操作	资源	目的
DescribeStream	Kinesis 数据流	在尝试读取记录前，使用者需检查流是否存在并处于活动状态。
GetRecords, GetShardIterator	Kinesis 数据流	从 Kinesis Data Streams 分区读取记录。
CreateTable, DescribeTable, GetItem, PutItem, Scan, UpdateItem	Amazon DynamoDB 表	如果使用者是使用 Kinesis 客户端库 (KCL) 开发的，则它需要处于活动状态。第一个使用者已开始创建表。
DeleteItem	Amazon DynamoDB 表	适用于使用者在 Kinesis Data Streams 分区上执行拆分/合并。
PutMetricData	亚马逊 CloudWatch 日志	KCL 还会将指标上传到 CloudWatch，这对于监控应用程序有用。

对于此应用程序，您可以创建单个 IAM 策略来授予上述所有权限。实际上，您可能需要考虑创建两个策略，一个策略适用于创建器，另一个策略适用于使用者。

创建 IAM policy

1. 找到新流的 Amazon 资源名称 (ARN)。您可以在“详细信息”选项卡的顶部找到以 Stream ARN 形式列出的此 ARN。ARN 格式如下所示：

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

区域

区域代码，例如 us-west-2。有关更多信息，请参阅[区域和可用区域概念](#)。

账户

Amazon 账户 ID，如[账户设置](#)中所示。

name

[步骤 1：创建数据流 \(p. 32\)](#) 中的流名称，即 StockTradeStream。

2. 确定使用者要使用的 DynamoDB 表的 ARN (并由第一个使用者实例创建)。它必须采用以下格式：

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

区域和账户来自上一步骤中的相同位置，但这一次，名称为使用者应用程序创建和使用的表的名称。使用者所使用的 KCL 将应用程序名称用作表名称。使用 StockTradesProcessor，它是稍后使用的应用程序名称。

3. 在 IAM 控制台的策略 (<https://console.aws.amazon.com/iam/home#policies>) 中，选择创建策略。如果这是您第一次使用 IAM 策略，请选择“开始，创建策略”。
4. 在 Policy Generator 旁，选择 Select。
5. 选择 Amazon Kinesis 作为 Amazon 服务。
6. 选择 DescribeStream、GetShardIterator、GetRecords、PutRecord 和 PutRecords 作为允许的操作。
7. 输入您在步骤 1 中创建的 ARN。
8. 对以下各项使用 Add Statement (添加语句)：

Amazon 服务	操作	ARN
Amazon DynamoDB	CreateTable, DeleteItem, DescribeTable, GetItem, PutItem, Scan, UpdateItem	您在步骤 2 中创建的 ARN
亚马逊 CloudWatch	PutMetricData	*

在不需要指定 ARN 时使用的星号 (*)。在这种情况下，这是因为没有可调用 PutMetricData 操作 CloudWatch 的特定资源。

9. 选择 Next Step。
10. 将 Policy Name (策略名称) 更改为 StockTradeStreamPolicy，审阅代码，然后选择 Create Policy (创建策略)。

生成的策略文档应类似于以下内容：

```
{
```

```
"Version": "2012-10-17",
"Statement": [
  {
    "Sid": "Stmt123",
    "Effect": "Allow",
    "Action": [
      "kinesis:DescribeStream",
      "kinesis:PutRecord",
      "kinesis:PutRecords",
      "kinesis:GetShardIterator",
      "kinesis:GetRecords",
      "kinesis:ListShards",
      "kinesis:DescribeStreamSummary",
      "kinesis:RegisterStreamConsumer"
    ],
    "Resource": [
      "arn:aws:kinesis:us-west-2:123:stream/StockTradeStream"
    ]
  },
  {
    "Sid": "Stmt234",
    "Effect": "Allow",
    "Action": [
      "kinesis:SubscribeToShard",
      "kinesis:DescribeStreamConsumer"
    ],
    "Resource": [
      "arn:aws:kinesis:us-west-2:123:stream/StockTradeStream/*"
    ]
  },
  {
    "Sid": "Stmt456",
    "Effect": "Allow",
    "Action": [
      "dynamodb:*"
    ],
    "Resource": [
      "arn:aws:dynamodb:us-west-2:123:table/StockTradesProcessor"
    ]
  },
  {
    "Sid": "Stmt789",
    "Effect": "Allow",
    "Action": [
      "cloudwatch:PutMetricData"
    ],
    "Resource": [
      "*"
    ]
  }
]
```

创建 IAM 用户

1. 通过以下网址打开 IAM 控制台：<https://console.aws.amazon.com/iam/>。
2. 在 Users (用户) 页面上，选择 Add user (添加用户)。
3. 对于 User name，键入 StockTradeStreamUser。
4. 对于 Access type (访问类型)，选择 Programmatic access (编程访问)，然后选择 Next: Permissions (下一步: 权限)。
5. 选择 Attach existing policies directly (直接附上现有策略)。
6. 按名称搜索您创建的策略。选中策略名称左侧的框，然后选择 Next: Review (下一步: 审核)。

7. 查看详细信息和摘要，然后选择 Create user (创建用户)。
8. 复制 Access key ID (访问密钥 ID)，并将其私下保存。在 Secret access key (私有访问密钥) 下面选择 Show (显示)，然后也将该密钥私下保存。
9. 将访问密钥和私有密钥粘贴到一个只有您可以访问的位于安全位置的本地文件中。对于此应用程序，请创建名为 `~/.aws/credentials` (具有严格权限) 的文件。该文件应采用以下格式：

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

将 IAM 策略附加到用户

1. 在 IAM 控制台中，打开[策略](#)并选择策略操作。
2. 选择 StockTradeStreamPolicy 和 Attach (附加)。
3. 选择 StockTradeStreamUser 和 Attach Policy (附加策略)。

后续步骤

[步骤 3：下载和构建实施代码 \(p. 36\)](#)

步骤 3：下载和构建实施代码

已提供 [the section called “教程：使用 KPL 和 KCL 1.x 处理实时股票数据” \(p. 31\)](#) 的框架代码。它包含用于股票交易流引入 (创建器) 和数据处理 (使用者) 的存根实施。以下过程演示如何完成实施。

下载和构建实施代码

1. 将[源代码](#)下载到计算机上。
2. 按照提供的目录结构，使用源代码在您喜爱的 IDE 中创建一个项目。
3. 将以下库添加到该项目中：
 - Amazon Kinesis 客户端库 (KCL)
 - Amazon SDK
 - 阿帕奇 HttpCore
 - 阿帕奇 HttpClient
 - Apache Commons Lang
 - Apache Commons Logging
 - Guava (适用于 Java 的 Google 核心库)
 - Jackson Annotations
 - Jackson Core
 - Jackson Databind
 - Jackson Dataformat : CBOR
 - Joda Time
4. 根据您的 IDE，项目可能会自动构建。如果未自动构建项目，请使用适合您的 IDE 的步骤构建项目。

如果已成功完成这些步骤，则可进入下一节 [the section called “第 4 步：实施创建器” \(p. 37\)](#)。如果您的构建在任何阶段出错，请调查并纠正错误，然后再继续。

后续步骤

[\(p. 37\)](#)

第 4 步：实施创建器

[教程：使用 KPL 和 KCL 1.x 处理实时股票数据 \(p. 31\)](#) 中的应用程序使用实际股票市场交易监控场景。以下准则简要说明了此场景如何映射到创建器和支持的代码结构。

请参阅源代码并查看以下信息。

StockTrade 阶级

单次股票交易由一个 StockTrade 类实例表示。此实例包含一些属性，如股票代码、价格、股份数、交易类型（买入或卖出）以及唯一标识交易的 ID。将为您实现此类。

流记录

流是一个记录序列。记录是 JSON 格式的 StockTrade 实例序列化。例如：

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

StockTradeGenerator 阶级

StockTradeGenerator 包含一个名为 getRandomTrade() 的方法，当调用此方法时，它将返回一个随机生成的新股票交易。将为您实现此类。

StockTradesWriter 阶级

生产者的 main 方法 StockTradesWriter 持续检索随机交易，然后通过执行以下任务将其发送到 Kinesis Data Streams：

1. 将流名称和区域名称作为输入读取。
2. 创建一个 AmazonKinesisClientBuilder。
3. 使用客户端生成器来设置区域、凭证和客户端配置。
4. 使用客户端生成器构建一个 AmazonKinesis 客户端。
5. 检查流是否存在且处于活动状态 (如果不是这样，它将退出并显示错误)。
6. 在连续循环中，会依次调用 StockTradeGenerator.getRandomTrade() 方法和 sendStockTrade 方法以便每 100 毫秒将交易发送到流一次。

sendStockTrade 类的 StockTradesWriter 方法具有以下代码：

```
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);
}
}
```

请参阅以下代码细分：

- PutRecord API 需要一个字节数组，并且您需要将 trade 转换为 JSON 格式。此行代码将执行该操作：

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

- 您需要先创建新的 PutRecordRequest 实例（此示例中称为 putRecord），然后才能发送交易：

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

每个 PutRecord 调用均需要流名称、分区键和数据 Blob。以下代码使用 putRecord 对象的 setXxxx() 方法来填充该对象中的这些字段：

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

该示例使用股票行情自动收录器作为将记录映射到特定分片的分区键。实际上，每个分片应具有数百或数千个分区键，以便记录均匀地分布在流中。有关如何将数据添加到流的更多信息，请参阅 [向流添加数据 \(p. 86\)](#)。

现在 putRecord 已准备好发送到客户端（put 操作）：

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

- 错误检查和日志记录始终是有用的附加功能。此代码将记录错误条件：

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

围绕 put 操作添加 try/catch 块：

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

这是因为 Kinesis Data Streams put 操作可能由于网络错误而失败，或者由于数据流达到其吞吐量限制并受到限制。建议您仔细考虑针对 put 操作的重试策略以避免数据丢失（用作简单重试）。

- 状态日志记录很有用，但它是可选的：

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

此处显示的制作人使用 Kinesis Data Streams API 的单条记录功能 `PutRecord`。实际上，如果单个创建者生成许多记录，则使用 `PutRecords` 的多记录功能并一次性发送批量记录通常会更有效。有关更多信息，请参阅 [向流添加数据 \(p. 86\)](#)：

运行创建器

1. 验证先前（创建 IAM 用户时）检索的访问密钥和 key pair 是否保存在文件中 `~/.aws/credentials`。
2. 使用以下参数运行 `StockTradeWriter` 类：

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

如果您在 `us-west-2` 之外的区域中创建流，则必须改为在此处指定该区域。

您应该可以看到类似于如下所示的输出内容：

```
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
```

你的股票交易流现在正被 Kinesis Data Streams 采集。

后续步骤

[第 5 步：实施使用者 \(p. 39\)](#)

第 5 步：实施使用者

[教程：使用 KPL 和 KCL 1.x 处理实时股票数据 \(p. 31\)](#) 中的使用者应用程序持续处理您在 [\(p. 37\)](#) 中创建的股票交易流。然后，它输出每分钟买入和卖出最多的股票。该应用程序建立在 Kinesis 客户端库 (KCL) 之上，它可以完成消费类应用程序常见的大部分繁重工作。有关更多信息，请参阅 [开发 KCL 1.x 使用者 \(p. 117\)](#)：

请参阅源代码并查看以下信息。

StockTradesProcessor 阶级

为您提供的使用者的主类，它将执行以下任务：

- 读取作为参数传递的应用程序名称、流名称和区域名称。
- 从 `~/.aws/credentials` 读取凭证。
- 创建一个 `RecordProcessorFactory` 实例，该实例提供由 `RecordProcessor` 实例实施的 `StockTradeRecordProcessor` 的实例。
- 使用 `RecordProcessorFactory` 实例和标准配置（包括流名称、凭证和应用程序名称）创建 KCL 工作程序。

- 工作线程为每个分片（分配给此使用者实例）创建一个新线程，该分片会持续循环读取 Kinesis Data Streams 中的记录。之后，它调用 `RecordProcessor` 实例以处理收到的每批记录。

StockTradeRecordProcessor 阶级

`RecordProcessor` 实例的实施，该实例反过来将实施三个必需方法：`initialize`、`processRecords` 和 `shutdown`。

顾名思义，Kinesis Client Library 使用 `initialize` 和 `shutdown` 分别让记录处理器知道何时应该准备好开始接收记录以及预计何时停止接收记录，这样它就可以执行任何特定于应用程序的设置和终止任务。将为您提供这些方法的代码。`processRecords` 方法中进行的主要处理，该处理反过来对每条记录使用 `processRecord`。后一个方法主要作为空框架代码提供给您，以便您在下一步骤中实施，届时将进一步对其进行说明。

另外要注意的是 `processRecord` 的支持方法 `reportStats` 和 `resetStats` 的实施，二者在初始源代码中为空。

已为您实施 `processRecords` 方法，并执行了以下步骤：

- 对于传入的每条记录，对其调用 `processRecord`。
- 如果自上一次报告以来已过去至少 1 分钟，请调用 `reportStats()`（它将打印出最新统计数据），然后调用 `resetStats()`（它将清除统计数据以便下一个间隔仅包含新记录）。
- 设置下一次报告时间。
- 如果自上一检查点以来已过去至少 1 分钟，请调用 `checkpoint()`。
- 设置下一次检查点操作时间。

此方法使用 60 秒间隔作为报告和检查点操作比率。有关检查点操作的更多信息，请参阅 [有关使用者的附加信息 \(p. 41\)](#)。

StockStats 阶级

此类提供一段时间内针对最热门股票的数据保留和统计数据跟踪。此代码已提供给您并包含以下方法：

- `addStockTrade(StockTrade)`：将给定的 `StockTrade` 注入正在使用的统计数据。
- `toString()`：以格式化字符串形式返回统计数据。

此类跟踪最热门股票的方式是，保留每只股票的总交易数的连续计数和最大计数。每当股票交易达成时，它都会更新这些计数。

将代码添加到 `StockTradeRecordProcessor` 类的方法，如以下步骤中所示。

实施使用者

1. 通过实例化大小正确的 `processRecord` 对象并将记录数据添加到该对象来实施 `StockTrade` 方法，并在出现问题时记录警告。

```
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. 实施简单的 `reportStats` 方法。可随时将输出格式修改为您的首选格式。

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

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

3. 最后，实施 `resetStats` 方法，这将创建新的 `stockStats` 实例。

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

运行使用者

1. 运行您在 [\(p. 37\)](#) 中编写的创建者以将模拟股票交易记录引入流中。
2. 验证先前（创建 IAM 用户时）检索的访问密钥和 key pair 是否保存在文件中 `~/.aws/credentials`。
3. 使用以下参数运行 `StockTradesProcessor` 类：

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

请注意，如果您在 `us-west-2` 之外的区域中创建流，则必须改为在此处指定该区域。

- 1 分钟后，您应看到类似以下内容的输出，并且输出在此后每分钟刷新一次：

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

有关使用者的附加信息

如果您熟悉在和其他地方讨论的 Kinesis 客户端库的[开发 KCL 1.x 使用者 \(p. 117\)](#)优势，您可能会想知道为什么要在哪里使用它。尽管您仅使用单个分片流和单个使用者实例对其进行处理，但使用 KCL 实现使用者仍然更容易。将创建器部分中的代码实施步骤与使用者部分中的进行对比，您会发现实施使用者相对来说容易一些。这在很大程度上要归功于 KCL 提供的服务。

在此应用程序中，您专注于实施可处理单条记录的记录处理器类。您不必担心如何从 Kinesis Data Streams 获取记录；每当有新记录可用时，KCL 都会提取记录并调用记录处理器。此外，不必担心分片和使用者实例的数量。如果已扩展流，则不必重写应用程序以处理多个分片或多个使用者实例。

术语检查点操作是指记录到流中目前已使用和处理的记录所在的点，这样一来，当应用程序发生崩溃时，系统将从该点读取流，而不是从头开始读取流。检查点操作主题、各种设计模式及其最佳实践不在本章讨论范围之内。但是，生产环境中可能会涉及上述内容。

正如您在中学到的那样 [\(p. 37\)](#)，Kinesis Data Streams API 中的 `put` 操作使用分区键作为输入。Kinesis Data Streams 使用分区键作为将记录拆分到多个分片的机制（当流中有多个分区时）。相同的分区键将始终路由到同一个分片。这使得能够基于以下假设来设计用于处理特定分片的使用者：具有相同分区键的记录只会发送给该使用者，具有相同分区键的任何记录都不会在任何其他使用者处结束。因此，使用者的工作程序可聚合具有相同分区键的所有记录而不用担心丢失所需的数据。

在此应用程序中，使用者对记录的处理并不密集，因此您可以使用一个分片并在与 KCL 线程相同的线程中进行处理。但在实际应用中，请先考虑增加分片数量。在某些情况下，您可能需要将处理切换到其他线程或需要使用线程池（如果您的记录处理是集中的）。这样，KCL 可以更快地获取新记录，而其他线程可以 `parallel` 处理记录。多线程设计并不是无关紧要的，应使用先进技术来实现，因此增加分片计数通常是最有效、最轻松的扩展方法。

后续步骤

[第 6 步：\(可选\) 扩展使用者 \(p. 42\)](#)

第 6 步：(可选) 扩展使用者

[教程：使用 KPL 和 KCL 1.x 处理实时股票数据 \(p. 31\)](#) 中的应用程序可能已足以达到您的目的。此可选部分演示如何针对更为复杂的场景扩展使用者代码。

如果要了解每分钟的最大销售订单数，可以修改三个位置的 `StockStats` 类以适应此新的优先级。

扩展使用者

1. 添加新实例变量：

```
// 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. 将以下代码添加到 `addStockTrade`：

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

3. 修改 `toString` 方法以打印其他信息：

```
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);
}
```

如果您现在运行使用者（请记住同时运行创建器），则应看到类似于以下内容的输出：

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

后续步骤

[第 7 步：收尾 \(p. 42\)](#)

第 7 步：收尾

由于您要付费使用 Kinesis 数据流，因此请确保在完成后将其和相应的 Amazon DynamoDB 表删除。即使您不发送和获取记录，活动流也会产生象征性的费用。这是因为活动流将持续“侦听”传入记录和获取记录的请求，这将耗用资源。

删除流和表

1. 关闭您可能仍在运行的任何创建者和使用者。
2. 打开 Kinesis 控制台，网址为：<https://console.aws.amazon.com/kinesis>。
3. 选择为此应用程序创建的流 (StockTradeStream)。
4. 选择 Delete Stream (删除流)。
5. 从 <https://console.aws.amazon.com/dynamodb/> 打开 DynamoDB 控制台。
6. 删除 StockTradesProcessor 表。

摘要

近乎实时处理大量数据不需要编写任何功能强大的代码或开发大型基础设施。它就像编写逻辑来处理少量数据（例如写入 `processRecord(Record)`）一样简单，但使用 Kinesis Data Streams 进行扩展以使其适用于大量流数据。你不必担心你的处理会如何扩展，因为 Kinesis Data Streams 会为你处理。你所要做的就是将你的直播记录发送到 Kinesis Data Streams，然后编写处理收到的每条新记录的逻辑。

以下是针对此应用程序的一些可能的改进。

跨所有分片进行聚合

当前，通过聚合单个分片中单个工作线程收到的数据记录来获得统计数据。（一个分片不能同时由单个应用程序中的多个工作线程处理。）当然，当您扩展并具有多个分片时，可能希望跨所有分片聚合。可通过部署管道架构完成此操作。在该架构中，每个工作线程的输出都注入具有单个分片的另一个流，分片由聚合第一个阶段输出的工作线程处理。由于来自第一个阶段的数据是有限的（每分片每分钟一个示例），因此一个分片即可轻松处理它。

缩放处理

当流进行扩展以包含多个分片（因为多个创建器正在发送数据）时，扩展处理的方式是添加更多工作程序。您可以在 Amazon EC2 实例中运行工作程序并使用 Auto Scaling 组。

使用连接器连接亚马逊 S3/Dynamodb/Amazon Redshift/Storm

随着数据流的持续处理，其输出可以发送到其他目的地。Amazon 提供[用于将 Kinesis Data Streams 与其他 Amazon 服务和第三方工具集成的连接器](#)。

后续步骤

- 有关使用 Kinesis Data Streams API 操作的更多信息，将[Amazon Kinesis Data Streams API 与结合使用来开发创建者](#)[Amazon SDK for Java \(p. 86\)](#)，请参阅[使用 Amazon SDK for Java 开发具有共享吞吐量的自定义使用者 \(p. 139\)](#)、和[创建和管理流 \(p. 59\)](#)。
- 有关 Kinesis 客户端库的更多信息，请参阅[开发 KCL 1.x 使用者 \(p. 117\)](#)。
- 有关如何优化应用程序的更多信息，请参阅[高级主题 \(p. 163\)](#)。

教程：使用适用于 Flink 应用程序的 Kinesis 数据分析实时股票数据

本教程的场景包括将股票交易纳入数据流，并编写一个简单的 [Amazon Kinesis 数据分析](#) 应用程序，在数据流上执行计算。您将学习如何向 Kinesis Data Streams 发送记录流，以及如何实现近乎实时地使用和处理记录的应用程序。

借助适用于 Flink 应用程序的 Kinesis 数据分析，您可以使用 Java 或 Scala 来处理和分析流数据。该服务使您能够针对流源编写和运行 Java 或 Scala 代码，以执行时间序列分析、提供实时仪表板并创建实时指标。

您可以使用基于 [Apache Flink](#) 的开源库在 Kinesis 数据分析中构建 Flink 应用程序。Apache Flink 是处理数据流的常用框架和引擎。

Important

在您创建两个数据流和一个应用程序后，您的账户会因使用 Kinesis Data Streams 和 Kinesis Data Analytics 而产生名义费用，因为它们没有资格享受免费套餐。Amazon 使用完此应用程序后，删除您的 Amazon 资源以停止产生费用。

代码不访问实际股票市场数据，而是模拟股票交易流。它通过使用随机股票交易生成器来实现这一点。如果您有权访问实时的股票交易流，则可能有兴趣从该流派生有用且及时的统计数据。例如，您可能希望执行滑动窗口分析，从而确定前 5 分钟内购买的最热门股票。或者，您可能希望在销售订单过大（即具有过多股份）时收到通知。可以扩展此系列代码以提供此类功能。

所示示例使用美国西部（俄勒冈）区域，但它们适用于任何[支持 Kinesis Data Analytics 的 Amazon 区域](#)。

任务

- [完成练习的先决条件 \(p. 44\)](#)
- [步骤 1：设置 Amazon 账户并创建管理员用户 \(p. 44\)](#)
- [步骤 2：设置 Amazon Command Line Interface \(Amazon CLI\) \(p. 46\)](#)
- [第 3 步：为 Flink 应用程序创建并运行 Kinesis 数据分析 \(p. 47\)](#)

完成练习的先决条件

要完成本指南中的步骤，您必须满足以下条件：

- [Java 开发工具包](#) (JDK) 版本 8。设置 JAVA_HOME 环境变量，使其指向您的 JDK 安装位置。
- 我们建议您使用开发环境（如 [Eclipse Java Neon](#) 或 [IntelliJ Idea](#)）来开发和编译您的应用程序。
- [Git 客户端](#)。如果尚未安装 Git 客户端，请安装它。
- [Apache Maven 编译器插件](#)。Maven 必须位于您的有效路径中。要测试您的 Apache Maven 安装，请输入以下内容：

```
$ mvn -version
```

要开始，请转到[步骤 1：设置 Amazon 账户并创建管理员用户 \(p. 44\)](#)。

步骤 1：设置 Amazon 账户并创建管理员用户

在您首次使用 Flink 应用程序的 Amazon Kinesis 数据分析之前，请完成以下任务：

1. [注册 Amazon \(p. 44\)](#)
2. [创建 IAM 用户 \(p. 45\)](#)

注册 Amazon

当您注册亚马逊网络服务 (Amazon) 时，您的 Amazon 账户将自动注册中的 Amazon 所有服务，包括亚马逊 Kinesis Data Analytics。您只需为使用的服务付费。

使用 Kinesis 数据分析，您只需为使用的资源付费。如果您是新的 Amazon 客户，则可以免费开始使用 Kinesis 数据分析。有关更多信息，请参阅 [Amazon Free Tier](#)（免费套餐）。

如果您已有 Amazon 账户，请跳到下一个任务。如果您没有 Amazon 帐户，请按照以下步骤创建一个帐户。

创建 Amazon 账户

1. 打开 <https://portal.aws.amazon.com/billing/signup>。
2. 按照屏幕上的说明进行操作。

在注册时，您将接到一通电话，要求您使用电话键盘输入一个验证码。

当您注册 Amazon Web Services 账户时，系统将会创建一个 Amazon Web Services 账户根用户。根用户有权访问该账户中的所有 Amazon Web Services 和资源。作为安全最佳实践，请 [为管理用户分配管理访问权限](#)，并且只使用根用户执行 [需要根用户访问权限的任务](#)。

记下您的 Amazon 账户 ID，因为下一个任务需要它。

创建 IAM 用户

中的 Amazon 服务（例如 Amazon Kinesis 数据分析）要求您在访问时提供证书。这样，服务才能确定您是否有权访问该服务所拥有的资源。Amazon Web Services Management Console 要求您输入密码。

您可以为您的 Amazon 账户创建访问密钥以访问 Amazon Command Line Interface (Amazon CLI) 或 API。但是，我们不建议使用您的 Amazon 账户的凭证访问 Amazon。相反，我们建议您使用 Amazon Identity and Access Management (IAM)。创建 IAM 用户，将该用户添加到具有管理权限的 IAM 组，然后向您创建的 IAM 用户授予管理权限。您随后便可以使用一个特殊的 URL 和该 IAM 用户的凭证访问 Amazon。

如果您注册了 IAM 用户 Amazon，但尚未为自己创建 IAM 用户，则可以使用 IAM 控制台创建一个。

本指南中的入门练习假定您拥有具有管理员权限的用户 (adminuser)。请按照以下过程在您的账户中创建 adminuser。

为管理员创建组

1. 登录 Amazon Web Services Management Console，然后通过以下网址打开 IAM 控制台：<https://console.aws.amazon.com/iam/>。
2. 在导航窗格中，选择 Groups (组)，然后选择 Create New Group (创建新组)。
3. 对于组名，输入组的名称，例如 **Administrators**，然后选择 下一步。
4. 在策略列表中，选中 AdministratorAccess 策略旁边的复选框。您可以使用 Filter (筛选) 菜单和 Search (搜索) 框来筛选策略列表。
5. 选择 Next Step (下一步)，然后选择 Create Group (创建组)。

您的新组列在 Group Name 下方。

要为自己创建 IAM 用户，请将其添加到管理员群组，然后创建密码

1. 在导航窗格中，选择 Users，然后选择 Add user。
2. 在 User name (用户名) 框中，输入一个用户名。
3. 选择编程访问 Amazon 和管理控制台访问权限。
4. 选择 Next: Permissions (下一步: 权限)。
5. 选中 Administrators 组旁的复选框。然后选择 Next: Review。
6. 选择 Create user。

以新 IAM 用户身份登录

1. 注销 Amazon Web Services Management Console。
2. 使用下面的 URL 格式登录控制台：

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

`aws_account_number` 是您的 Amazon 账户 ID，不含任何连字符。#####Amazon####
1234-5678-9012### `aws_account_number` #### 123456789012 有关如何查找您的账号的信息，请参阅 IAM 用户指南中的[您的 Amazon 账户 ID 及其别名](#)。

3. 输入您刚创建的 IAM 用户名和密码。登录后，导航栏将显示 `your_user_name @ your_aws_account_id`。

Note

如果您不希望您的登录页面 URL 包含 Amazon 账户 ID，可以创建账户别名。

创建或删除账户别名

1. 通过以下网址打开 IAM 控制台：<https://console.aws.amazon.com/iam/>。
2. 在导航窗格上，选择 Dashboard。
3. 查找 IAM 用户登录链接。
4. 要创建别名，请选择 Customize (自定义)。输入要用于别名的名称，然后选择 Yes, Create (是，创建)。
5. 要删除别名，请选择 Customize，然后选择 Yes, Delete。登录 URL 会恢复使用 Amazon 账户 ID。

要在创建账户别名后登录，请使用以下 URL：

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

要为您的账户验证 IAM 用户的登录链接，请打开 IAM 控制台并在控制面板的 IAM 用户登录链接下进行检查。

有关 IAM 的更多信息，请参阅以下文档：

- [Amazon Identity and Access Management \(IAM\)](#)
- [入门](#)
- [IAM 用户指南](#)

下一个步骤

[步骤 2：设置 Amazon Command Line Interface \(Amazon CLI\) \(p. 46\)](#)

步骤 2：设置 Amazon Command Line Interface (Amazon CLI)

在此步骤中，您将下载并配置为与 Flink Amazon CLI 应用程序的 Amazon Kinesis 数据分析一起使用。

Note

本指南中的入门练习假定您使用账户中的管理员凭证 (adminuser) 来执行这些操作。

Note

如果您已安装 Amazon CLI，您可能需要升级以获得最新的功能。有关更多信息，请参阅《Amazon Command Line Interface 用户指南》[Amazon 中的“安装命令行界面”](#)。要检查 Amazon CLI 的版本，请运行以下命令：

```
aws --version
```

本教程中的练习需要以下 Amazon CLI 版本或更高版本：

```
aws-cli/1.16.63
```

设置 Amazon CLI

1. 下载并配置 Amazon CLI。有关说明，请参阅《Amazon Command Line Interface 用户指南》中的以下主题：
 - [安装 Amazon Command Line Interface](#)
 - [配置 Amazon CLI](#)
2. 在 Amazon CLI 配置文件中为管理员用户添加一个命名配置文件。在执行 Amazon CLI 命令时，您将使用此配置文件。有关命名配置文件的更多信息，请参阅《Amazon Command Line Interface 用户指南》中的[命名配置文件](#)。

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

有关可用 Amazon 区域的列表，请参阅中的[Amazon 区域和终端节点 Amazon Web Services 一般参考](#)。

3. 在命令提示符处输入以下帮助命令来验证设置：

```
aws help
```

设置 Amazon 帐户后，您可以尝试下一个练习，在该练习中配置示例应用程序并测试端到端设置。Amazon CLI

下一个步骤

[第 3 步：为 Flink 应用程序创建并运行 Kinesis 数据分析 \(p. 47\)](#)

第 3 步：为 Flink 应用程序创建并运行 Kinesis 数据分析

在本练习中，您将使用数据流作为源和接收器创建 Kinesis 数据分析 Flink 应用程序。

本节包含以下步骤：

- [创建两个亚马逊 Kinesis 数据流 \(p. 47\)](#)
- [将示例记录写入输入流 \(p. 48\)](#)
- [下载并检查 Apache Flink 流式处理 Java 代码 \(p. 49\)](#)
- [编译应用程序代码 \(p. 49\)](#)
- [上传 Apache Flink 流式处理 Java 代码 \(p. 51\)](#)
- [创建和运行 Kinesis 数据分析应用程序 \(p. 52\)](#)

创建两个亚马逊 Kinesis 数据流

在本练习中为 Flink 创建 Kinesis 数据分析应用程序之前，请创建两个 Kinesis 数据流（和）ExampleInputStream。ExampleOutputStream 您的应用程序将这些数据流用于应用程序源和目标流。

您可以使用 Amazon Kinesis 控制台或以下 Amazon CLI 命令创建这些直播。有关控制台说明，请参阅[创建和更新数据流](#)。

创建数据流 (Amazon CLI)

1. 要创建第一个直播 (ExampleInputStream)，请使用以下 Amazon Kinesis create-stream Amazon CLI 命令。

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

2. 要创建应用程序用来写入输出的第二个流，请运行同一命令（将流名称更改为 ExampleOutputStream）。

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

将示例记录写入输入流

在本节中，您使用 Python 脚本将示例记录写入流，以供应用程序处理。

Note

此部分需要 [Amazon SDK for Python \(Boto\)](#)。

1. 使用以下内容创建名为 stock.py 的文件：

```
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. 在本教程的后面部分，您运行 `stock.py` 脚本，以将数据发送到应用程序。

```
$ python stock.py
```

下载并检查 Apache Flink 流式处理 Java 代码

此示例的 Java 应用程序代码可从中获得 GitHub。要下载应用程序代码，请执行以下操作：

1. 使用以下命令克隆远程存储库：

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

2. 导航到 `GettingStarted` 目录。

应用程序代码位于 `CustomSinkStreamingJob.java` 和 `CloudWatchLogSink.java` 文件中。请注意有关应用程序代码的以下信息：

- 该应用程序使用 Kinesis 源从源流中读取数据。以下代码片段创建了 Kinesis 水槽：

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

编译应用程序代码

在本节中，您使用 Apache Maven 编译器创建应用程序的 Java 代码。有关安装 Apache Maven 和 Java 开发工具包 (JDK) 的信息，请参阅 [完成练习的先决条件 \(p. 44\)](#)。

您的 Java 应用程序需要以下组件：

- 一个 [项目对象模型 \(pom.xml\)](#) 文件。该文件包含有关应用程序配置和依赖关系的信息，包括 Flink 应用程序库的 Kinesis 数据分析。
- 它是一种 `main` 方法，其中包含应用程序的逻辑。

Note

要将 Kinesis 连接器用于以下应用程序，您需要下载连接器的源代码并按照 [Apache Flink](#) 文档中的说明进行构建。

创建并编译应用程序代码

1. 在您的开发环境中创建 Java/Maven 应用程序。有关创建应用程序的信息，请参阅有关开发环境的文档：
 - [创建您的第一个 Java 项目 \(Eclipse Java Neon\)](#)
 - [创建、运行和打包您的第一个 Java 应用程序 \(IntelliJ Idea\)](#)
2. 将以下代码用于名为 `StreamingJob.java` 的文件。

```
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;
    }
}
```

- 此文件包含 `main` 方法，它定义应用程序的功能。
- 您的应用程序使用 `StreamExecutionEnvironment` 对象创建源和接收连接器以访问外部资源。
- 该应用程序将使用静态属性创建源和接收连接器。要使用动态应用程序属性，请使用 `createSourceFromApplicationProperties` 和 `createSinkFromApplicationProperties` 方法以创建连接器。这些方法读取应用程序的属性来配置连接器。

- 使用命令行 Maven 工具。在包含 pom.xml 文件的目录中通过运行以下命令创建您的 JAR 文件：

设置开发环境。有关详细信息，请参阅您的开发环境文档。

4. 如果编译时出错，请验证 JAVA_HOME 环境变量设置正确。

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

在本节中，您将创建一个亚马逊简单存储服务 (Amazon S3) 存储桶并上传您的应用程序代码。

1. 通过以下网址打开 Amazon S3 控制台：<https://console.aws.amazon.com/s3/>。

2. 选择 Create bucket (创建存储桶)。
3. 在“存储段名称”字段 **ka-app-code-*<username>*** 中输入。将后缀 (如您的用户名) 添加到存储桶名称, 以使其具有全局唯一性。选择下一步。
4. 在配置选项步骤中, 让设置保持原样, 然后选择下一步。
5. 在设置权限步骤中, 让设置保持原样, 然后选择下一步。
6. 选择创建桶。
7. 在 Amazon S3 控制台中, 选择 ka-app-code- *<username>* 存储桶, 然后选择上传。
8. 在选择文件步骤中, 选择添加文件。导航到您在上一步中创建的 java-getting-started-1.0.jar 文件。选择下一步。
9. 在设置权限步骤中, 让设置保持原样。选择下一步。
10. 在设置属性步骤中, 让设置保持原样。请选择 Upload (上传)。

您的应用程序代码现在存储在 Amazon S3 存储桶中, 您的应用程序可以在其中访问它。

创建和运行 Kinesis 数据分析应用程序

您可以使用控制台或 Flink 创建和运行适用于 Flink 的 Kinesis 数据分析应用程序。Amazon CLI

Note

当您使用控制台创建应用程序时, 系统会为您创建您的 Amazon Identity and Access Management (IAM) 和 Amazon CloudWatch Logs 资源。当您使用 Amazon CLI 创建应用程序时, 您可以单独创建这些资源。

主题

- [创建并运行应用程序 \(控制台 \) \(p. 52\)](#)
- [创建并运行应用程序 \(Amazon CLI\) \(p. 55\)](#)

创建并运行应用程序 (控制台)

按照以下步骤, 使用控制台创建、配置、更新和运行应用程序。

创建 应用程序

1. 打开 Kinesis 控制台, 网址为: <https://console.aws.amazon.com/kinesis>。
2. 在亚马逊 Kinesis 控制面板上, 选择创建分析应用程序。
3. 在 Kinesis Analytics – 创建应用程序页面上, 提供应用程序详细信息, 如下所示:
 - 对于 Application name (应用程序名称), 输入 **MyApplication**。
 - 对于描述, 输入 **My java test app**。
 - 对于 Runtime (运行时), 请选择 Apache Flink 1.6。
4. 对于访问权限, 请选择 Create / update IAM role (创建/更新 IAM 角色) **kinesis-analytics-MyApplication-us-west-2**。
5. 选择创建应用程序。

Note

当您使用控制台创建 Kinesis Data Analytics for Flink 应用程序时, 您可以选择为应用程序创建 IAM 角色和策略。您的应用程序使用此角色和策略访问其从属资源。这些 IAM 资源使用您的应用程序名称和区域命名, 如下所示:

- 策略: **kinesis-analytics-service-MyApplication-us-west-2**
- 角色: **kinesis-analytics-MyApplication-us-west-2**

编辑 IAM 策略

编辑 IAM 策略以添加访问 Kinesis 数据流的权限。

1. 通过以下网址打开 IAM 控制台：<https://console.aws.amazon.com/iam/>。
2. 选择策略。选择控制台在上一部分中为您创建的 **kinesis-analytics-service-MyApplication-us-west-2** 策略。
3. 在 Summary (摘要) 页面上，选择 Edit policy (编辑策略)。请选择 JSON 选项卡。
4. 将以下策略示例中突出显示的部分添加到策略中。将示例账户 ID (**012345678901**) 替换为您的账户 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"  
    }  
  ]  
}
```

配置应用程序

1. 在MyApplication页面上，选择配置。
2. 在 Configure application (配置应用程序) 页面上，提供 Code location (代码位置)：
 - 对于亚马逊 S3 存储桶，输入 **ka-app-code-*<username>***。
 - 在 Amazon S3 对象的路径中，输入 **java-getting-started-1.0.jar**。
3. 在 Access to application resources (对应用程序的访问权限) 下，对于 Access permissions (访问权限)，选择 Create / update IAM role (创建/更新 IAM 角色) **kinesis-analytics-MyApplication-us-west-2**。
4. 在 Properties (属性) 下，对于 Group ID (组 ID)，输入 **ProducerConfigProperties**。
5. 输入以下应用程序属性和值：

键	值
flink.inputstream.initpos	LATEST
aws:region	us-west-2
AggregationEnabled	false

6. 在 Monitoring (监控) 下，确保 Monitoring metrics level (监控指标级别) 设置为 Application (应用程序)。
7. 要CloudWatch登录，请选中“启用”复选框。
8. 选择更新。

Note

当您选择启用CloudWatch日志记录时，Kinesis Data Analytics 会为您创建日志组和日志流。这些资源的名称如下所示：

- 日志组：/aws/kinesis-analytics/MyApplication
- 日志流：kinesis-analytics-log-stream

运行应用程序

1. 在MyApplication页面上，选择“运行”。确认该操作。
2. 当应用程序正在运行时，请刷新页面。控制台将显示 Application graph (应用程序图表)。

停止应用程序

在MyApplication页面上，选择“停止”。确认该操作。

更新应用程序

使用控制台，您可以更新应用程序设置，例如应用程序属性、监控设置，或应用程序 JAR 文件的位置和文件名。如果您需要更新应用程序代码，也可以从 Amazon S3 存储桶重新加载应用程序 JAR。

在 MyApplication 页面上，选择配置。更新应用程序设置，然后选择更新。

创建并运行应用程序 (Amazon CLI)

在本节中，您将使用 Amazon CLI 来创建和运行 Kinesis 数据分析应用程序。Flink 应用程序的 Kinesis 数据分析使用该 `kinesisanalyticsv2` Amazon CLI 命令来创建 Kinesis 数据分析应用程序并与之交互。

创建权限策略

首先，使用两个语句创建权限策略：一个语句授予对源流执行 read 操作的权限，另一个语句授予对接收器流执行 write 操作的权限。然后，您将该策略附加到 IAM 角色（将在下一节中创建）。因此，当 Kinesis Data Analytics 担任该角色时，该服务具有从源流读取和写入汇流的必要权限。

使用以下代码创建 `KAReadSourceStreamWriteSinkStream` 权限策略。`username` 替换为用于创建用于存储应用程序代码的 Amazon S3 存储桶的用户名。将 Amazon 资源名称 (ARN) 中的账户 ID (`012345678901`) 替换为您的账户 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"
    }
  ]
}
```

有关创建权限策略的 step-by-step 说明，请参阅 IAM 用户指南中的[教程：创建并附加您的第一个客户托管策略](#)。

Note

要访问其他 Amazon 服务，您可以使用 Amazon SDK for Java。Kinesis Data Analytics 会自动将 SDK 所需的证书设置为与您的应用程序关联的服务执行 IAM 角色的证书。无需执行其他步骤。

创建 IAM 角色

在本节中，您将创建一个 IAM 角色，Kinesis Data Analytics for Flink 应用程序可以假定该角色来读取源流并写入汇流。

没有权限，Kinesis 数据分析无法访问您的直播。您可以通过 IAM 角色授予这些权限。每个 IAM 角色都有两个附加的策略。信任策略授予 Kinesis Data Analytics 担任该角色的权限，权限策略决定 Kinesis Data Analytics 在担任该角色后可以做什么。

您将在上一部分中创建的权限策略附加到此角色。

创建 IAM 角色

1. 通过以下网址打开 IAM 控制台：<https://console.aws.amazon.com/iam/>。
2. 在导航窗格中，选择 Roles (角色) 和 Create Role (创建角色)。
3. 在“选择可信身份类型”下，选择“Amazon 服务”。在 Choose the service that will use this role (选择将使用此角色的服务) 下，选择 Kinesis。在选择您的使用案例下，选择 Kinesis Analytics。

选择 Next: Permissions (下一步: 权限)。

4. 在 Attach permissions policies 页面上，选择 Next: Review。在创建角色后，您可以附加权限策略。
5. 在创建角色页面上 **KA-stream-rw-role**，输入角色名称。选择 Create role (创建角色)。

现在，您已经创建了一个名为的新 IAM 角色 KA-stream-rw-role。接下来，您更新角色的信任和权限策略。

6. 将权限策略附加到角色。

Note

在本练习中，Kinesis Data Analytics 扮演了这个角色，既要从 Kinesis 数据流（源）读取数据，又要将输出写入另一个 Kinesis 数据流。因此，您附加在上一步 ([the section called “创建权限策略” \(p. 55\)](#)) 中创建的策略。

- a. 在 Summary (摘要) 页上，选择 Permissions (权限) 选项卡。
- b. 选择附加策略。
- c. 在搜索框中，输入 **KAReadSourceStreamWriteSinkStream** (您在上一部分中创建的策略)。
- d. 选择 KA ReadInputStreamWriteOutputStream 策略，然后选择“附加策略”。

现在，您已经创建了应用程序用来访问资源的服务执行角色。记下新角色的 ARN。

有关创建角色的 step-by-step 说明，请参阅 [IAM 用户指南中的创建 IAM 角色 \(控制台\)](#)。

创建 Kinesis 数据分析应用程序

1. 将以下 JSON 代码保存到名为 create_request.json 的文件中。将示例角色 ARN 替换为您之前创建的角色 ARN。将存储桶 ARN 后缀 (*username*) 替换为在前一部分中选择的后缀。将服务执行角色中的示例账户 ID (**012345678901**) 替换为您的账户 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. 使用上述请求执行 [CreateApplication](#) 操作来创建应用程序：

```
aws kinesisanalyticstv2 create-application --cli-input-json file://create_request.json
```

应用程序现已创建。您在下一步中启动应用程序。

启动应用程序

在本节中，您使用 [StartApplication](#) 操作来启动应用程序。

启动应用程序

1. 将以下 JSON 代码保存到名为 `start_request.json` 的文件中。

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

2. 使用上述请求执行 [StartApplication](#) 操作来启动应用程序：

```
aws kinesisanalyticstv2 start-application --cli-input-json file://start_request.json
```

应用程序正在运行。您可以在亚马逊CloudWatch控制台上查看 Kinesis 数据分析指标，以验证应用程序是否正常运行。

停止应用程序

在本节中，您使用 [StopApplication](#) 操作来停止应用程序。

停止应用程序

1. 将以下 JSON 代码保存到名为 `stop_request.json` 的文件中。

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

2. 使用下面的请求执行 [StopApplication](#) 操作来停止应用程序：

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

应用程序现已停止。

教程：使用AmazonLambda Kinesis Data Streams

在本教程中，您将创建 Lambda 函数来处理 Kinesis 数据流中的事件。在此示例场景中，自定义应用程序会将记录写入 Kinesis 数据流。Amazon 然后，Lambda 会轮询此数据流，如果检测到新的数据记录，则调用您的 Lambda 函数。Amazon 然后，Lambda 会代入您在创建 Lambda 函数时指定的执行角色来执行该 Lambda 函数。

如需详细的分步说明，请参阅[教程：使用AmazonLambda Kinesis](#)。

Note

本教程假设您对 Lambda 基本操作和 Lambda 控制台有一定了解。如果您尚不了解，请按照中的说明进行操作。[入门AmazonLambda](#)来创建你的第一个 Lambda 函数。

Amazon Kinesis 的流数据解决方案

这些区域有：Amazon 适用于 Amazon Kinesis 的流数据解决方案自动配置 Amazon 轻松捕获、存储、处理和交交流数据所需的服务。该解决方案提供了多种解决使用多种流数据使用案例的选项 Amazon 包括 Kinesis Data Streams 在内的服务 Amazon Lambda、Amazon API Gateway 和 Amazon Kinesis Data Analytics。

每个解决方案包含以下组件：

- 一个 Amazon CloudFormation 以便部署完整示例。
- 一个 CloudWatch 显示应用程序指标的仪表板。
- CloudWatch 关于最相关的应用程序指标的警报。
- 所有必要的 IAM 角色和策略。

可在此处找到解决方案：[Amazon Kinesis 的流数据解决方案](#)

创建和管理流

Amazon Kinesis Data Streams 实时摄取大量数据，持久存储数据，并使数据可供使用。Kinesis Data Streams 存储的数据单位是数据记录。数据流 表示一组数据记录。数据流中的数据记录将分发到分片中。

分片具有流中的一系列数据记录。它是 Kinesis 数据流的基本吞吐量单位。在按需和预置容量模式下，分区支持每秒 1 MB/s 和 1000 条记录的写入，每秒 2 MB/s 的读取。分片限制确保了可预测的性能，使设计和操作高度可靠的数据流工作流程变得更加容易。

主题

- [选择数据流容量模式 \(p. 59\)](#)
- [通过Amazon管理控制台创建直播 \(p. 61\)](#)
- [通过 API 创建直播 \(p. 62\)](#)
- [更新流 \(p. 63\)](#)
- [列出流 \(p. 64\)](#)
- [列表分片 \(p. 65\)](#)
- [删除流 \(p. 68\)](#)
- [对流进行重新分片 \(p. 68\)](#)
- [更改数据保留期 \(p. 72\)](#)
- [在 Amazon Kinesis Data Streams 中为流添加标签 \(p. 72\)](#)

选择数据流容量模式

主题

- [什么是数据流容量模式？ \(p. 59\)](#)
- [按需模式 \(p. 60\)](#)
- [预置模式 \(p. 60\)](#)
- [在容量模式之间切换 \(p. 61\)](#)

什么是数据流容量模式？

容量模式决定如何管理数据流的容量以及如何向您收取数据流使用费。在 Amazon Kinesis Data Streams 中，您可以为数据流选择按需模式和预置。

- 按需-按需模式下的数据流无需容量规划，可自动扩展以处理每分钟千兆字节的写入和读取吞吐量。在按需模式下，Kinesis Data Streams 会自动管理分片以提供必要的吞吐量。
- 预配置-对于采用预置模式的数据流，您必须指定数据流的分片数量。数据流的总容量是其分片容量的总和。您可以根据需要增加或减少数据流中的分片数量。

您可以使用 Kinesis Data Streams PutRecord 和 PutRecords API 在按需和预置容量模式下将数据写入数据流。为了检索数据，两种容量模式都支持使用 GetRecords API 的默认使用者和使用 SubscribeToShard API 的增强型扇出 (EFO) 使用者。

按需模式和预配模式均支持所有 Kinesis Data Streams 功能，包括保留模式、加密、监控指标等。Kinesis Data Streams 在按需和预置容量模式下均提供高耐久性和可用性。

按需模式

按需模式下的数据流无需容量规划，可自动扩展以处理每分钟千兆字节的写入和读取吞吐量。按需模式可简化以低延迟摄取和存储大量数据，因为它无需配置和管理服务器、存储或吞吐量。您每天可以摄取数十亿条记录，而不会产生任何运营开销。

按需模式非常适合满足高度可变和不可预测的应用程序流量的需求。您不再需要为峰值容量预置这些工作负载，这可能会由于利用率低而导致成本增加。按需模式适用于流量模式不可预测且变化很大的工作负载。

在按需容量模式下，您按数据流中写入和读取的 GB 数据。您无需指定应用程序预计将执行多少读取和写入吞吐量。Kinesis Data Streams 会根据工作负载的增减来即时做出调整。有关更多信息，请参阅 [Amazon Kinesis Data Streams 定价](#)。

您可以使用 Kinesis Data Streams 控制台、API 或 CLI 命令在按需模式下创建新的数据流。

按需模式下的数据流最多可容纳过去 30 天观察到的峰值写入吞吐量的两倍。当数据流的写入吞吐量达到新的峰值时，Kinesis Data Streams 会自动扩展数据流的容量。例如，如果您的数据流的写入吞吐量介于 10 MB/s 到 40 MB/s 之间，那么 Kinesis Data Streams 可确保您可以轻松突增至之前的峰值吞吐量的两倍，即 80 MB/s。如果同一数据流维持 50 MB/s 的新峰值吞吐量，Kinesis Data Streams 可确保有足够的容量来提取 100 MB/s 的写入吞吐量。但是，如果您的流量在 15 分钟持续时间内增加到前一峰值的两倍以上，则可能会出现写入限制。您需要重试这些受限的请求。

按需模式下的数据流的总读取容量与写入吞吐量成比例增加。这有助于确保消费类应用程序始终具有足够的读取吞吐量来实时处理传入的数据。与使用 GetRecords API 读取数据相比，您获得的写入吞吐量至少是其两倍。我们建议您将一个消费者应用程序与 GetRecord API 一起使用，这样当应用程序需要从停机中恢复时，它就有足够的空间来 catch。对于需要添加多个消费类应用程序的场景，建议您使用 Kinesis Data Streams 的增强扇出功能。增强型 Fan-Out 支持使用 SubscribeToShard API 向数据流添加多达 20 个消费类应用程序，每个消费类应用程序都有专用的吞吐量。

处理读取和写入吞吐量异常

使用按需容量模式（与预置容量模式相同），您必须为每条记录指定分区键才能将数据写入数据流。Kinesis Data Streams 使用您的分区键在分区之间分配数据。Kinesis Data Streams 监控每个分片的流量。当每个分片的传入流量超过 500 KB/s 时，它会在 15 分钟内分割分片。父分片的哈希键值在子分片之间均匀地重新分配。

如果您的传入流量超过之前峰值的两倍，则即使您的数据均匀分布在分区上，您也可能在大约 15 分钟内遇到读取或写入异常。我们建议您重试所有此类请求，以便将所有记录正确存储在 Kinesis Data Streams 中。

如果您使用的分区键导致数据分布不均匀，并且分配给特定分区的记录超出其限制，则可能会遇到读写异常。在按需模式下，除非单个分区键超过分片的 1 MB/s 吞吐量和每秒 1000 条记录的限制，否则数据流会自动适应以处理不均匀的数据分布模式。

在按需模式下，Kinesis Data Streams 会在检测到流量增加时均匀分配分片。但是，它不会检测和隔离将更多传入流量带到特定分区的哈希密钥。如果您使用高度不均匀的分区键，则可能会继续收到写入异常。对于此类使用案例，我们建议您使用支持细分片的预置容量模式。

预置模式

在预置模式下，创建数据流后，您可以使用或 [UpdateShardCount](#) API 动态地向上或向下扩展 Amazon Web Services Management Console 容量。您可以在 Kinesis Data Streams 生成器或使用者应用程序向数据流写入数据或从数据流中读取数据时进行更新。

预置模式适用于容量要求易于预测的可预测流量。如果您想精细控制数据在分片间的分布方式，则可以使用预配置模式。

在预置模式下，您必须指定数据流的分片数量。要在预配模式下确定数据流的大小，您需要以下输入值：

- 写入流的数据记录的平均大小，以千字节 (KB) 为单位，四舍五入到最接近的 1 KB (average_data_size_in_KB)。
- 每秒写入流和从流读取的数据记录数 (records_per_second)。
- 使用者的数量，即 Kinesis Data Streams 应用程序，可同时独立于数据流消耗数据 (number_of_consumers)。
- 以 KB 为单位的传入写入带宽 (incoming_write_bandwidth_in_KB)，等于 average_data_size_in_KB 乘以 records_per_second。
- 以 KB 为单位的传出读取带宽 (outgoing_read_bandwidth_in_KB)，等于 incoming_write_bandwidth_in_KB 乘以 number_of_consumers。

可以使用以下公式中的输入值，计算流需要的分片数量 (number_of_shards)。

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

如果您未将数据流配置为处理峰值吞吐量，则在预配置模式下仍可能遇到读取和写入吞吐量异常。在这种情况下，您必须手动扩展数据流以适应您的数据流量。

如果您使用的分区键导致数据分布不均匀，并且分配给分区的记录超出其限制，则也可能会遇到读写异常。要在预配模式下解决此问题，请识别此类分片并手动拆分它们以更好地容纳您的流量。有关更多信息，请参阅[重新分片流](#)。

在容量模式之间切换

您可以将数据流的容量模式从按需切换到预配置，或从预配置切换到按需。对于 Amazon 账户中的每个数据流，您可以在 24 小时内两次在按需和预配置容量模式之间切换。

在数据流的容量模式之间切换不会对使用该数据流的应用程序造成任何中断。您可以继续写入和读取该数据流。当您在容量模式之间切换（从按需到预配或从预配到按需）时，直播的状态设置为更新。必须等待数据流状态变为“活动”，然后才能再次修改其属性。

当您从预置容量模式切换到按需容量模式时，您的数据流最初会保留过渡之前的分片数量，从那时起，Kinesis Data Streams 会监控您的数据流量并根据您的写入吞吐量扩展此按需数据流的分片数量。

当您从按需模式切换到预配置模式时，您的数据流最初还会保留过渡之前的分片数量，但是从现在起，您有责任监控和调整该数据流的分区数量，以正确适应您的写入吞吐量。

通过 Amazon 管理控制台创建直播

您可以使用 Kinesis Data Streams 控制台、Kinesis Data Streams API 或 Amazon Command Line Interface (Amazon CLI) 创建直播。

使用控制台创建数据流

1. 登录到 Amazon Web Services Management Console，然后通过以下网址打开 Kinesis 控制台：<https://console.aws.amazon.com/kinesisvideo/home>。
2. 在导航栏中，展开区域选择器并选择一个区域。
3. 选择创建数据流。
4. 在创建 Kinesis 直播页面上，输入数据流的名称，然后选择按需或预置容量模式。默认情况下，按需模式处于选中状态。有关更多信息，请参阅[选择数据流容量模式 \(p. 59\)](#)：

在按需模式下，您可以选择“创建 Kinesis 流”来创建数据流。在预配模式下，您必须指定所需的分片数量，然后选择 Create Kinesis 直播。

在 Kinesis stream (Kinesis 流)页面上，当流处于创建中时，流的 Status (状态) 为 Creating (正在创建)。当流可以使用时，Status (状态) 会更改为 Active (有效)。

5. 选择流的名称。Stream Details (流详细信息) 页面显示了流配置摘要以及监控信息。

使用 Kinesis Data Streams API 创建流

- 有关使用 Kinesis Data Streams API 创建数据流的信息，请参阅[通过 API 创建直播 \(p. 62\)](#)。

使用 Amazon CLI 创建流

- 有关使用 Amazon CLI 创建流的信息，请参阅 [create-stream](#) 命令。

通过 API 创建直播

使用以下步骤创建 Kinesis 数据流。

构建 Kinesis Data Streams 客户端

必须先构建客户端对象，然后才能使用 Kinesis 数据流。以下 Java 代码实例化一个客户端生成器，并使用它来设置区域、凭据和客户端配置。然后，它会构建一个客户端对象。

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

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

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

有关更多信息，请参阅中的 [Kinesis Data Streams 区域和端点](#)和[Amazon Web Services 一般参考](#)。

创建流

现在，您已经创建了 Kinesis Data Streams 客户端，您可以创建要使用的流，您可以使用 Kinesis Data Streams 控制台或以编程方式完成此操作。要以编程方式创建流，请实例化一个>CreateStreamRequest对象并指定该流的名称以及（如果您想使用预配置模式）该流要使用的分片数量。

- 按需：

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

- 已配置：

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

流名称用于标识流。该名称的范围限于应用程序使用的Amazon帐户。它还受区域限制。也就是说，两个不同Amazon账户中的两个流可以有相同的名称，同一个Amazon账户中的两个直播可以有相同的名称，但两个不同区域中的两个流可以有相同的名称，但不能有同一个账户和同一区域中的两个流。

流的吞吐量是分片数量的函数；配置更大的吞吐量需要更多分片。分片越多，直播的Amazon费用也会增加。有关计算适合您的应用程序的分片数量的更多信息，请参阅[选择数据流容量模式 \(p. 59\)](#)。

在配置 `createStreamRequest` 对象之后，通过对客户端调用 `createStream` 方法来创建流。在调用 `createStream` 之后，应等待流达到 `ACTIVE` 状态，然后再对流执行任何操作。要查看流的状态，请调用 `describeStream` 方法。但是，如果流不存在，`describeStream` 将引发异常。因此，请将 `describeStream` 调用包括在 `try/catch` 块中。

```
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" );
}
```

更新流

您可以使用 Kinesis Data Streams 控制台、Kinesis Data Streams API 或者，更新直播的详细信息Amazon CLI。

Note

您可以为现有流或最近刚创建的流启用服务器端加密。

使用控制台更新数据流

1. 通过 <https://console.aws.amazon.com/kinesis/> 打开Amazon Kinesis 主机。
2. 在导航栏中，展开区域选择器并选择一个区域。
3. 在列表中选择流的名称。Stream Details (流详细信息) 页面显示流配置摘要和监控信息。
4. 要在数据流的按需容量模式和预置容量模式之间切换，请在配置选项卡中选择编辑容量模式。有关更多信息，请参阅[选择数据流容量模式 \(p. 59\)](#)：

Important

对于 Amazon 账户中的每个数据流，您可以在 24 小时内按需模式和预配模式之间切换两次。

5. 对于配置模式的数据流，要编辑分片数量，请在“配置”选项卡中选择“编辑已配置的分片”，然后输入新的分区数量。
6. 要启用数据记录的服务器端加密，请在“服务器端加密”部分中选择“编辑”。选择 KMS 密钥用作加密的主密钥，或者使用由 Kinesis 管理的默认主密钥 aws/kinesis。如果您为流启用加密，并使用自己的 Amazon KMS 主密钥，请确保您的创建器和使用应用程序有权访问您使用的 Amazon KMS 主密钥。要将权限分配给应用程序以访问用户生成的 Amazon KMS 密钥，请参阅[the section called “使用用户生成的 KMS 主密钥的权限” \(p. 197\)](#)。
7. 要编辑数据保留期，请选择 Data retention period 部分中的 Edit，然后输入新的数据保留期。
8. 如果您在账户上启用了自定义指标，请在分片级别指标部分选择编辑，然后为您的直播指定指标。有关更多信息，请参阅[the section called “使用 CloudWatch 监控服务” \(p. 168\)](#)：

使用 API 更新流

要使用 API 更新流详细信息，请参阅以下方法：

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

使用 Amazon CLI 更新流

有关使用 Amazon CLI 更新流的信息，请参阅 [Kinesis CLI 参考](#)。

列出流

如上一节所述，直播的范围限于与用于实例化 Kinesis Data Streams 客户端的 Amazon 凭据关联的 Amazon 账户，也限于为客户指定的区域。一个 Amazon 账户可以同时有多个活跃流。您可以在 Kinesis Data Streams 控制台中列出您的直播，也可以通过编程方式列出您的直播。本部分中的代码显示如何列出您的 Amazon 账户的所有直播。

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

此代码示例首先创建 ListStreamsRequest 的一个新实例，然后调用其 setLimit 方法来指定对 listStreams 的每次调用应返回最多 20 个流。如果您未指定值，Kinesis Data Streams 会返回小于或等于账户中数字的流数量。setLimit 此代码之后将 listStreamsRequest 传递到客户端的 listStreams 方法。返回值 listStreams 存储在 ListStreamsResult 对象中。此代码对此对象调用 getStreamNames

方法，并将返回的流名称存储在 `streamNames` 列表中。请注意，Kinesis Data Streams 返回的流可能少于指定限制值，即使账户和区域中的流数量多于指定限制。要确保检索所有流，请使用下一代代码示例中描述的 `getHasMoreStreams` 方法。

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

此代码对 `getHasMoreStreams` 调用 `listStreamsRequest` 方法，以检查是否有超出在对 `listStreams` 的初始调用中返回的流数量的可用流。如果有，则此代码将使用在对 `setExclusiveStartStreamName` 的上一调用中返回的最后一个流的名称调用 `listStreams` 方法。`setExclusiveStartStreamName` 方法将导致对 `listStreams` 的下一调用在该流之后开始。该调用返回的一组流名称之后将添加到 `streamNames` 列表。此过程将继续，直到所有流名称已收集到列表中。

`listStreams` 返回的流可能处于下列状态之一：

- CREATING
- ACTIVE
- UPDATING
- DELETING

您可使用 `describeStream` 方法查看流的状态（如上一节 [通过 API 创建直播 \(p. 62\)](#) 中所示）。

列表分片

一个数据流可以有一个或多个分片。有两种方法可以从数据流中列出（或检索）分片。

主题

- [ListShards API-推荐 \(p. 65\)](#)
- [DescribeStream API-已过时 \(p. 67\)](#)

ListShards API-推荐

从数据流中列出或检索分片的推荐方法是使用 [ListShards API](#)。以下示例显示如何获取数据流中的分片列表。有关本示例中使用的主要操作以及您可以为该操作设置的所有参数的完整描述，请参见[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());
}
}
```

要运行上一个代码示例，您可以使用类似于下文的 POM 文件。

```
<?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>
```

ListShards通过 API，您可以使用[ShardFilter](#)参数筛选 API 的响应。一次只能指定一个过滤器。

如果您在调用 ListShards API 时使用ShardFilter参数，则Type是必需的属性，必须指定。如果您指定AT_TRIM_HORIZONFROM_TRIM_HORIZON、或AT_LATEST类型，则无需指定ShardId或Timestamp可选属性。

如果您指定AFTER_SHARD_ID类型，则还必须提供可选属ShardId性的值。该ShardId属性的功能与ListShards API 的ExclusiveStartShardId参数相同。指定ShardId属性后，响应包括以分片开头的分片，分片的 ID 紧随您ShardId提供的分片。

如果您指定AT_TIMESTAMP或FROM_TIMESTAMP_ID类型，则还必须提供可选属Timestamp性的值。如果您指定AT_TIMESTAMP类型，则将返回在提供的时间戳处打开的所有分片。如果您指定FROM_TIMESTAMP类型，则将返回从提供的时间戳到 TIP 的所有分片。

Important

DescribeStreamSummary 而且 ListShard API 提供了一种更具可扩展性的方式来检索有关数据流的信息。更具体地说，DescribeStream API 的配额可能会导致限制。有关更多信息，请参阅[配额和限制 \(p. 6\)](#)：另请注意，DescribeStream 配额在与您 Amazon 账户中的所有数据流交互的所有应用程序之间共享。另一方面，ListShards API 的配额是特定于单个数据流的。因此，您不仅可以 ListShards 通过 API 获得更高的 TPS，而且随着您创建更多数据流，操作规模也会更大。我们建议您迁移所有调用 DescribeStream API 的生产者和使用者，改为调用 DescribeStreamSummary 和 ListShard API。为了识别这些生产者和消费者，我们建议使用 Athena 解析 CloudTrail 日志，因为在 API 调用中捕获 KPL 和 KCL 的用户代理。

```
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
```

我们还建议重新配置调用 API 的 Amazon Lambda 和 Kinesis Data Firehose 与调用 DescribeStream API 的 Kinesis Data Streams 集成，以便这些集成改为调用 DescribeStreamSummary 和 ListShards。具体而言，对于 Amazon Lambda，您必须更新事件源映射。对于 Kinesis Data Firehose，必须更新相应的 IAM 权限，使其包含 ListShards IAM 权限。

DescribeStream API-已过时

Important

以下信息描述了一种目前不推荐使用的通过 DescribeStream API 从数据流中检索分片的方法。目前强烈建议您使用 ListShards API 来检索构成数据流的分片。

describeStream 方法返回的响应对象使您能够检索有关组成流的分片的信息。要检索分片，请对此对象调用 getShards 方法。此方法可能不会通过一次调用返回流中的所有分片。在以下代码中，我们查看 getHasMoreShards 上的 getStreamDescription 方法以了解是否有未返回的其他分片。如果有，也就是说，如果此方法返回 true，则我们将继续循环调用 getShards，以将返回的新的分片批次添加到我们的分片列表中。循环将在 getHasMoreShards 返回 false 时退出；即，所有分片均已返回。请注意，getShards 不会返回处于 EXPIRED 状态的分片。有关分片状态（包括 EXPIRED 状态）的更多信息，请参阅[分片之后的数据路由、数据保留和分片状态 \(p. 71\)](#)。

```
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 );
```

删除流

您可以使用 Kinesis Data Streams 控制台删除数据流，也可以通过编程方式删除数据流。要以编程方式删除流，请使用 `DeleteStreamRequest`，如以下代码所示。

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

删除流之前，请关闭流上运行的任何应用程序。如果应用程序尝试对已删除的流进行操作，则将收到 `ResourceNotFound` 异常。此外，如果您随后创建名称与之前的流相同的新流，并且在之前的流上运行的应用程序仍在运行，则这些应用程序可能会尝试与新流交互（就像新流是之前的流一样），这将产生无法预测的结果。

对流进行重新分片

Important

您可以使用 [UpdateShardCount](#) API 对直播进行重新分片。否则，您可以按照此处的说明继续执行拆分和合并。

Amazon Kinesis Data Streams 支持重新分片，这使您可以调整数据流中的分区数量，以适应流中数据流速的变化。重新分片被视为高级操作。如果你是 Kinesis Data Streams 的新手，请在熟悉 Kinesis Data Streams 的所有其他方面之后返回本主题。

这里有两种类型的重新分片操作：分片拆分和分片合并。在分片拆分中，可将一个分片拆分为两个分片。在分片合并中，可将两个分片组合成一个分片。重新分片始终是成对进行的，也就是说，无法在一次操作中拆分为两个以上的分片，并且无法在一次操作中合并两个以上的分片。重新分片操作涉及的分片或分片对称称为父分片。从重新分片操作中生成的分片或分片对称称为子分片。

拆分将增加流中分片的数量，从而增加流的数据容量。由于按分片收费，因此拆分将增加流的费用。同样，合并会减少流中的分片数量，从而降低数据流的数据容量和成本。

重新分片一般由不同于创建者（放置）应用程序和使用者（获取）应用程序的管理应用程序执行。此类管理应用程序根据亚马逊提供的指标 CloudWatch 或从制作人和消费者那里收集的指标来监控直播的整体表现。管理应用程序还需要比使用者或生产者更广泛的 IAM 权限，因为使用者和生产者通常不需要访问用于重新分片的 API。有关 Kinesis Data Streams 的 IAM 权限的更多信息，请参阅[使用 IAM 控制对亚马逊 Kinesis 数据流资源的访问权限 \(p. 201\)](#)。

有关重新分片的更多信息，请参阅[如何更改 Kinesis Data Streams 中打开的分片数量？](#)

主题

- [重新分片策略 \(p. 68\)](#)
- [拆分分片 \(p. 69\)](#)
- [合并两个分片 \(p. 70\)](#)
- [重新分片后 \(p. 70\)](#)

重新分片策略

在 Amazon Kinesis Data Streams 中重新分片的目的是使您的数据流能够适应数据流速率的变化。拆分分片将增加流的容量（和费用）。合并分片将减少流的费用（和容量）。

重新分片的一种方法可能是拆分流中的每个分片，这将使数据流的容量增加一倍。但是，这提供的容量可能比您实际需要的要多，从而产生不必要的费用。

您还可以使用指标来确定哪些是热分片还是冷分片，即分片接收的数据比预期多得多或少得多。之后您可选择性地拆分热分片以增加面向这些分片的哈希键的容量。同样，您可合并冷分片以更好地利用其未使用的容量。

您可以从 Kinesis Data Streams 发布的亚马逊 CloudWatch 指标中获取直播的一些性能数据。但是，您也可收集您自己的一些关于流的指标。一种方式是记录由您的数据记录的分区键生成的哈希键值。记住，您在向流添加记录时指定了分区键。

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

Kinesis [Data Streams 使用 MD5](#) 根据分区键计算哈希键。由于您为记录指定了分区键，因此可使用 MD5 计算该记录的哈希键值并将此值记录下来。

您还可记录您的数据记录分配到的分片的 ID。分片 ID 是通过使用 `getShardId` 对象（由 `putRecordResults` 方法返回）和对 `putRecords` 对象（由 `putRecordResult` 方法返回）的 `putRecord` 方法提供的。

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

利用分片 ID 和哈希键值，您可确定将接收最多流量或最少流量的分片和哈希键。您之后可使用重新分片操作来增加或减少容量（视这些键的情况而定）。

拆分分片

要拆分 Amazon Kinesis Data Streams 中的分片，您需要指定如何将父分区的哈希键值重新分配到子分片。当您向流添加数据记录时，将基于哈希键值将数据记录分配给分片。哈希键值是在向流添加数据记录时为数据记录指定的分区键的 [MD5](#) 哈希。具有相同分区键的数据记录也具有相同的哈希键值。

指定分片的可能的哈希键值构成一组有序连续非负整数。此可能的哈希键值范围是通过以下命令指定的：

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

在拆分分片时，您指定此范围内的一个值。此哈希键值和所有较高的哈希键值将分配到其中一个子分片。所有较小的哈希键值将分配到另一子分片。

以下代码演示在所有子分片之间均匀地重新分配哈希键的分片拆分操作，基本上是将父分片一分为二。这只是一种可能的父分片划分方式。例如，可以拆分分片，以便父分片中下层三分之一的键分配给一个子分片，上层三分之二的键分配给另一子分片。但是，在许多应用程序中，将分片一分为二是一种很有效的方法。

此代码假定 `myStreamName` 包含您的流名称，对象变量 `shard` 包含要拆分的分片。首先实例化一个新的 `splitShardRequest` 对象并设置流名称和分片 ID。

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

确定位于分片中最低值和最高值的中间的哈希键值。这是将包含父分片中上半层哈希键的子分片的起始哈希键值。在 `setNewStartingHashKey` 方法中指定此值。您只需要指定此值。Kinesis Data Streams 会自动将低于此值的哈希键分发给通过拆分创建的另一个子分片。最后一步是在 Kinesis Data Streams 客户端上调用该 `splitShard` 方法。

```
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);
```

[等待流再次变为活动状态 \(p. 71\)](#)中演示了此过程之后的第一步。

合并两个分片

分片合并操作使用两个指定分片并将它们合并为一个分片。在合并后，单个子分片将收到两个父分片所包含的所有哈希键值的数据。

分片相邻

要合并两个分片，分片必须相邻。如果两个分片的哈希键范围联合构成一个无间断的连续集，则认为两个分片相邻。例如，假设您有两个分片，一个分片的哈希键范围为 276...381，另一个分片的哈希键范围为 382...454。您可以将这两个分片合并为一个分片，其哈希键范围为 276...454。

另举一例，假设您有两个分片，一个分片的哈希键范围为 276...381，另一个分片的哈希键范围为 455...560。您无法合并这两个分片，因为这两个分片之间有一个或多个哈希键位于 382...454 范围的分片。

流中所有OPEN分片的集合（作为一个组）始终涵盖 MD5 哈希键值的整个范围。有关分片状态的更多信息，例如CLOSED —请参见[分片之后的数据路由、数据保留和分片状态 \(p. 71\)](#)。

要标识作为合并候选的分片，您应筛选出处于 CLOSED 状态的所有分片。（CLOSED即不OPEN是）的分片的结尾序列号为null。您可使用以下命令测试此结束序列号：

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

在筛选出已关闭分片后，按每个分片支持的最高哈希键值对剩余分片进行排序。您可使用以下命令检索此值：

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

如果两个分片在这个经过筛选和排序的列表中相邻，则可合并它们。

合并操作的代码

以下代码可合并两个分片。此代码假定 myStreamName 包含您的流名称并且对象变量 shard1 和 shard2 包含要合并的两个相邻分片。

为进行合并操作，首先实例化一个新的 mergeShardsRequest 对象。使用 setStreamName 方法指定流名称。然后使用 setShardToMerge 和 setAdjacentShardToMerge 方法指定要合并的两个分片。最后，在 Kinesis Data Streams 客户端上调用该mergeShards方法来执行该操作。

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

[等待流再次变为活动状态 \(p. 71\)](#)中演示了此过程之后的第一步。

重新分片后

在 Amazon Kinesis Data Streams 中执行任何类型的重新分片程序之后，在恢复正常记录处理之前，都需要执行其他步骤和注意事项。以下各节介绍了这些过程。

主题

- [等待流再次变为活动状态 \(p. 71\)](#)
- [分片之后的数据路由、数据保留和分片状态 \(p. 71\)](#)

等待流再次变为活动状态

在调用重新分片操作 `splitShard` 或 `mergeShards` 之后，您需要等待流再次变为活动状态。要使用的代码与您在[创建流 \(p. 62\)](#)之后等待流变为活动状态时使用的代码相同。代码如下所示：

```
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" );
}
```

分片之后的数据路由、数据保留和分片状态

Kinesis Data Streams 是一种实时数据流服务，也就是说，你的应用程序应该假设数据在流中的分片中持续流动。当您重新分片时，流至父分片的数据记录将基于数据记录分区键映射到的哈希键值重新路由至子分片。但是，重新分片前位于父分片中的任何数据记录将仍位于这些父分片中。换句话说，父分片在重新分片发生后不会消失。它们与重新分片之前包含的数据一起保留。可使用 Kinesis Data Streams API 中的[getShardIterator](#)和[getRecords \(p. 139\)](#)操作或通过 Kinesis 客户端库访问父分区中的数据记录。

Note

数据记录在当前保留期内添加到流中后即可访问。不论在该期间内对流中的分片进行了任何更改，都是如此。有关流的保留期的更多信息，请参阅[更改数据保留期 \(p. 72\)](#)。

在重新分片过程中，父分片将从 OPEN 状态过渡到 CLOSED 状态再过渡到 EXPIRED 状态。

- OPEN：在重新分片操作之前，父分片处于 OPEN 状态，这意味着数据记录可添加到分片中并且可从分片进行检索。

- **CLOSED**：在重新分片操作之后，父分片将过渡到 **CLOSED** 状态。这意味着无法再向此分片添加数据记录。原本应该已添加到此分片的数据记录现在将改为添加到子分片。但是，数据记录在有限时间内仍可从此分片进行检索。
- **EXPIRED**：在流的保留期过期之后，父分片中的所有数据记录将会过期，不再可供访问。此时，父分片自身将过渡到 **EXPIRED** 状态。用于枚举流中的分片的 `getStreamDescription().getShards` 调用不包括返回的分片列表中的 **EXPIRED** 分片。有关流的保留期的更多信息，请参阅[更改数据保留期 \(p. 72\)](#)。

在进行重新分片并且流再次处于 **ACTIVE** 状态之后，您可立即开始读取子分片中的数据。但是，在重新分片后保留的父分片可能仍包含您尚未读取并且已在重新分片前添加到流中的数据。如果您在读取完父分片中的所有数据之前读取子分片中的数据，则可不按数据记录的序列号指定的顺序读取特定哈希键的数据。因此，假定数据顺序很重要，您在重新分片后应始终继续读取父分片中的数据直到读取完。并且只有在这之后才应开始读取子分片中的数据。如果 `getRecordsResult.getNextShardIterator` 返回 `null`，则这指示您已读取完父分片中的所有数据。如果您使用 Kinesis 客户端库读取数据，则即使发生重新分片，该库也会确保您按顺序接收数据。

更改数据保留期

Amazon Kinesis Data Streams 支持更改数据流的数据记录保留期。Kinesis 数据流是数据记录的有序序列，旨在实时写入和读取。因此，数据记录临时存储在您的流的分片中。从添加记录开始，到记录不再可供访问为止的时间段称为保留期。默认情况下，Kinesis 数据流存储的记录从 24 小时到最多 8760 小时（365 天）不等。

您可以通过 Kinesis Data Streams 控制台或使用[IncreaseStreamRetentionPeriod](#)和[DecreaseStreamRetentionPeriod](#)操作更新保留期。使用 Kinesis Data Streams 控制台，您可以同时批量编辑多个数据流的保留期。您可以使用[IncreaseStreamRetentionPeriod](#)操作或 Kinesis Data Streams 控制台将保留期延长至最长 8760 小时（365 天）。您可以使用[DecreaseStreamRetentionPeriod](#)操作或 Kinesis Data Streams 控制台将保留期缩短至至少 24 小时。两个操作的请求语法均包括流名称和保留期（以小时为单位）。最后，您可以通过调用[DescribeStream](#)操作来检查流的当前保留期。

以下是使用 Amazon CLI 更改保留期的示例：

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

在延长保留期后的几分钟内，Kinesis Data Streams 停止在旧保留期内使记录不可访问。例如，将保留期从 24 小时更改为 48 小时意味着，在 23 小时 55 分钟之前添加到流中的记录在 24 小时之后仍可用。

在缩短保留期后，Kinesis Data Streams 几乎会立即使比新保留期更早的记录不可访问。因此，在调用[DecreaseStreamRetentionPeriod](#)操作时务必小心谨慎。

设置数据保留期以确保在出现问题时，您的使用者可以在数据过期之前读取数据。您应该仔细考虑所有可能的情况，例如记录处理逻辑出现问题，或者下游依赖关系长时间断开。将保留期视为安全网，可留出更多时间供您的数据使用者恢复。使用保留期 API 操作，您可以主动设置此项，或者积极响应操作事件。

对于保留期设置为 24 小时以上的流，将收取额外费用。有关更多信息，请参阅[Amazon Kinesis 数据流定价](#)。

在 Amazon Kinesis Data Streams 中为流添加标签

您可以将自己的元数据分配给您在 Amazon Kinesis Data Streams 中以下形式创建的流：标签。标签是您为流定义的键值对。使用标签是管理 Amazon 资源和组织数据（包括账单数据）的一种简单却强有力的方式。

目录

- [有关标签的基本知识 \(p. 73\)](#)
- [使用标签跟踪成本 \(p. 73\)](#)
- [标签限制 \(p. 73\)](#)
- [使用 Kinesis Data Streams 控制台为流添加标签 \(p. 74\)](#)
- [使用 Amazon CLI 为流添加标签 \(p. 74\)](#)
- [使用 Kinesis Data Streams API 为流添加标签 \(p. 75\)](#)

有关标签的基本知识

您可以使用 Kinesis Data Streams 控制台，Amazon CLI 或 Kinesis Data Streams API 完成以下任务：

- 向流添加标签
- 列出流的标签
- 从流中删除标签

您可以使用标签对流进行分类。例如，您可以按用途、所有者或环境对流进行分类。由于您定义每个标签的键和值，因此您可以创建一组自定义类别来满足您的特定需求。例如，您可以定义一组标签来帮助您按所有者和关联应用程序跟踪流。以下几个标签示例：

- 项目：项目名称
- 拥有者：名称
- 目的：负载测试
- 应用程序：应用程序名称
- ：Environment 生产

使用标签跟踪成本

您可以使用标签对 Amazon 成本进行分类和跟踪。当您将标签应用到 Amazon 资源，包括直播，Amazon 按标签汇总的使用率和成本。您可以设置代表业务类别（例如成本中心、应用程序名称或所有者）的标签，以便整理多种服务的成本。有关更多信息，请参阅 Amazon Billing 用户指南中的[对自定义账单报告使用成本分配标签](#)。

标签限制

以下限制适用于标签。

基本限制

- 每个资源（流）的最大标签数是 50。
- 标签键和值区分大小写。
- 无法更改或编辑已删除的流的标签。

标签键限制

- 每个标签键必须是唯一的。如果您添加的标签具有已使用的键，则您的新标签将覆盖现有键值对。
- 标签键不能以 `aws:` 开头，因为此前缀将预留以供 Amazon 使用。Amazon 将代表您创建以此前缀开头的标签，但您不能编辑或删除这些标签。
- 标签键的长度必须介于 1 和 128 个 Unicode 字符之间。

- 标签键必须由以下字符组成：Unicode 字母、数字、空格和以下特殊字符：_ . / = + - @.

标签值限制

- 标签值的长度必须介于 0 和 255 个 Unicode 字符之间。
- 标签值可以为空。否则，它们必须由以下字符组成：Unicode 字母、数字、空格和以下任意特殊字符：_ . / = + - @.

使用 Kinesis Data Streams 控制台为流添加标签

您可以使用 Kinesis Data Streams 控制台添加、列出和删除标签。

查看流的标签

1. 打开 Kinesis Data Streams 控制台。在导航栏中，展开区域选择器并选择一个区域。
2. 在 Stream List (流列表) 页面上，选择一个流。
3. 在 Stream Details (流详细信息) 页面上，单击 Tags (标签) 选项卡。

向流添加标签

1. 打开 Kinesis Data Streams 控制台。在导航栏中，展开区域选择器并选择一个区域。
2. 在 Stream List (流列表) 页面上，选择一个流。
3. 在 Stream Details (流详细信息) 页面上，单击 Tags (标签) 选项卡。
4. 在 Key (键) 字段中指定标签键，(可选) 在 Value (值) 字段中指定标签值，然后单击 Add Tag (添加标签)。

如果 Add Tag (添加标签) 按钮未启用，即表示您指定的标签键或标签值不满足标签限制。有关更多信息，请参阅 [标签限制 \(p. 73\)](#)。

5. 要在 Tags (标签) 选项卡上的列表中查看您的新标签，请单击刷新图标。

从流中删除标签

1. 打开 Kinesis Data Streams 控制台。在导航栏中，展开区域选择器并选择一个区域。
2. 在“Stream List (流列表)”页面上，选择一个流。
3. 在“Stream Details (流详细信息)”页面上，单击 Tags (标签) 选项卡，然后单击标签的 Remove (删除) 图标。
4. 在 Delete Tag (删除标签) 对话框中，单击 Yes, Delete (是，删除)。

使用 Amazon CLI 为流添加标签

您可以使用 Amazon CLI 添加、列出和删除标签。有关示例，请参阅以下文档。

[add-tags-to流式传输](#)

为指定的流添加或更新标签。

[list-tags-for流式传输](#)

列出指定流的标签。

[remove-tags-from流式传输](#)

从指定的流中删除标签。

使用 Kinesis Data Streams API 为流添加标签

您可以使用 Kinesis Data Streams API 添加、列出和删除标签。有关示例，请参阅以下文档：

[AddTagsToStream](#)

为指定的流添加或更新标签。

[ListTagsForStream](#)

列出指定流的标签。

[RemoveTagsFromStream](#)

从指定的流中删除标签。

将 Data Streams 与 Amazon Kinesis Data Streams 集成

Data Streams 是 Amazon Kinesis Data Streams 的应用程序。您可以使用 Amazon SDK for Java 和 Kinesis Data Streams 编写器。

如果您不熟悉 Kinesis Data Streams，请先熟悉和了解的概念 [什么是 Amazon Kinesis Data Streams？ \(p. 1\)](#) 和术语 [Amazon Kinesis Data Streams 入门 \(p. 12\)](#)。

Important

Kinesis Data Streams 支持更改数据流的数据记录保留期。有关更多信息，请参阅 [更改数据保留期 \(p. 72\)](#)：

要将数据放入流，您必须指定流的名称、分区键和要添加到流的数据 Blob。分区键用来确定数据记录将添加到流中的哪个分片。

分片中的所有数据将发送至正在处理分片的同一个工作程序。使用哪个分区键取决于您的应用程序逻辑。通常，分区键的数量应比分片数量多得多。这是因为分区键用来确定如何将数据记录映射到特定分片。如果您有足够的分区键，数据可以在流中的分片间均匀分布。

目录

- [使用 Amazon Kinesis 生产者库开发生产者 \(p. 76\)](#)
- [将 Amazon Kinesis Data Streams API 与结合使用来开发创建者 Amazon SDK for Java \(p. 86\)](#)
- [使用 Amazon Kinesis Data Streams 与 Kinesis Streams \(p. 90\)](#)
- [使用其他 Amazon 服务写入 Kinesis 数据流 \(p. 98\)](#)
- [使用第三方集成 \(p. 100\)](#)
- [将 Amazon Kinesis Data Streams 创建器故障 \(p. 101\)](#)
- [针对 Kinesis Data Streams 的高级主题 \(p. 103\)](#)

使用 Amazon Kinesis 生产者库开发生产者

Amazon Kinesis Data Streams 创建器是一种将用户数据记录放入 Kinesis 数据流（也称为数据摄取）的应用程序。Kinesis 生产者库 (KPL) 简化了生产者应用程序开发，允许开发人员实现对 Kinesis 数据流的高写入吞吐量。

您可以通过亚马逊监控 KPL CloudWatch。有关更多信息，请参阅 [使用 Amazon CloudWatch 监控 Kinesis 生产者库 \(p. 191\)](#)。

目录

- [KPL 的作用 \(p. 77\)](#)
- [使用 KPL 的优势 \(p. 77\)](#)
- [何时不使用 KPL \(p. 78\)](#)
- [安装 KPL \(p. 78\)](#)
- [过渡到适用于 Kinesis Producer Library 的 Amazon Trust Services \(ATS\) 证书 \(p. 78\)](#)
- [KPL 支持的平台 \(p. 78\)](#)
- [KPL 重要概念 \(p. 79\)](#)
- [将 KPL 与生产者代码集成 \(p. 80\)](#)

- [使用 KPL 写入您的 Kinesis 数据流 \(p. 81\)](#)
- [配置 Kinesis 生产者库 \(p. 82\)](#)
- [使用者取消聚合 \(p. 83\)](#)
- [将 KPL 与 Kinesis Data Firehose 一起使用 \(p. 85\)](#)
- [将 KPL 与 Glue 架构注册表一起使用 \(p. 85\)](#)
- [KPL 代理配置 \(p. 85\)](#)

Note

建议您升级到最新的 KPL 版本。KPL 会定期更新新版本，其中包括最新的依赖项和安全补丁、错误修复和向后兼容的新功能。有关更多信息，请参阅 <https://github.com/aws/aws-kinesis-producer-lib>。

KPL 的作用

KPL 是一个 easy-to-use 高度可配置的库，可帮助您写入 Kinesis 数据流。它充当您的生产者应用程序代码和 Kinesis Data Streams API 操作之间的中介。KPL 执行以下主要任务：

- 使用自动和可配置的重试机制写入一个或多个 Kinesis 数据流
- 收集记录并使用 PutRecords 根据请求将多条记录写入多个分片
- 聚合用户记录以增加负载大小并提高吞吐量
- 与 Kinesis 客户端库 (KCL) 无缝集成，以解聚消费者上的批量记录
- 代表您提交亚马逊 CloudWatch 指标，以提供生产者绩效的可见性

请注意，KPL 不同于 SDK 中提供的 Kinesis [AmazonData Streams](#) API。Kinesis Data Streams API 可帮助您管理 Kinesis Data Streams 的许多方面（包括创建流、重新分片以及放置和获取记录），而 KPL 则提供了专门用于摄取数据的抽象层。有关 Kinesis Data Streams API 的信息，请参阅 [Amazon Kinesis API 参考](#)。

使用 KPL 的优势

以下列表显示了使用 KPL 开发 Kinesis Data Streams 生成器的一些主要优势。

KPL 可以在同步或异步用例中使用。除非存在使用同步操作的具体原因，否则建议您使用异步接口的较高性能。有关这两种使用案例和示例代码的更多信息，请参阅 [使用 KPL 写入您的 Kinesis 数据流 \(p. 81\)](#)。

性能优势

KPL 可以帮助建立高性能的生产者。假设您的 Amazon EC2 实例充当代理，用于收集来自数百或数千个低功耗设备的 100 字节事件并将记录写入 Kinesis 数据流的情况。这些 EC2 实例均须将每秒数以千计的事件写入您的数据流。要实现所需的吞吐量，创建器必须实施复杂逻辑（例如，批处理或多线程处理）及重试逻辑并在使用者端取消记录聚合。KPL 为您执行所有这些任务。

使用者端易于使用

对于使用 Java 中的 KCL 的消费者端开发者来说，KPL 无需额外努力即可集成。当 KCL 检索由多个 KPL 用户记录组成的聚合 Kinesis Data Streams 记录时，它会自动调用 KPL 提取单个用户记录，然后将其返回给用户。

对于不使用 KCL 而是 GetRecords 直接使用 API 操作的消费者端开发人员，可以使用 KPL Java 库在将单个用户记录返回给用户之前提取这些记录。

创建器监控

您可以使用亚马逊 CloudWatch 和 KPL 收集、监控和分析您的 Kinesis Data Streams 生成器。KPL 代表您向 CloudWatch 其发送吞吐量、错误和其他指标，并且可配置为在流、分片或生产者级别进行监控。

异步架构

由于 KPL 可能会在将记录发送到 Kinesis Data Streams 之前对其进行缓冲，因此它不会强制调用应用程序阻塞并等待确认记录已到达服务器后才能继续执行。将记录放入 KPL 的调用总是立即返回，不会等待记录发送或服务器收到响应。取而代之的是创建一个 Future 对象，该对象将在以后接收将记录发送到 Kinesis Data Streams 的结果。这与 Amazon SDK 中的异步客户端的行为相同。

何时不使用 KPL

KPL 可能会产生额外的处理延迟，最高可达库 RecordMaxBufferedTime 内（用户可配置）。RecordMaxBufferedTime 值越大，产生的包装效率和性能就越高。无法容忍这种额外延迟的应用程序可能需要直接使用 Amazon SDK。有关在 Kinesis Data Streams 中使用 Amazon SDK 的更多信息，请参阅[将 Amazon Kinesis Data Streams API 与结合使用来开发创建者 Amazon SDK for Java \(p. 86\)](#)。有关 KPLRecordMaxBufferedTime 和其他用户可配置属性的更多信息，请参阅[配置 Kinesis 生产者库 \(p. 82\)](#)。

安装 KPL

亚马逊提供适用于 macOS、Windows 和最新的 Linux 发行版的 C++ Kinesis Producer Library (KPL) 的预建二进制文件（有关支持的平台详情，请参阅下一节）。这些二进制文件打包在 Java .jar 文件中，如果您使用 Maven 安装程序包，将自动调用和使用这些二进制文件。要查找 KPL 和 KCL 的最新版本，请使用以下 Maven 搜索链接：

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

Linux 二进制文件已采用 GNU 编译器集合 (GCC) 进行编译并已静态链接到 Linux 上的 libstdc++。这些二进制文件应适用于包含 glibc 2.5 版或更高版本的任何 64 位 Linux 发行版。

旧版 Linux 发行版的用户可以使用随源代码一起提供的构建说明构建 KPL GitHub。要从中下载 KPL GitHub，请参阅 [Kinesis 制作者库](#)。

过渡到适用于 Kinesis Producer Library 的 Amazon Trust Services (ATS) 证书

太平洋标准时间 2018 年 2 月 9 日上午 9:00，Amazon Kinesis Data Streams 安装了 ATS 证书。要继续使用 Kinesis Producer 库 (KPL) 向 Kinesis Data Streams 写入记录，必须将 KPL 的安装升级到 [0.12.6 或更高版本](#)。此更改会影响所有 Amazon 区域。

有关向 ATS 迁移的信息，请参阅[如何为 Amazon 迁移到自己的证书颁发机构做好准备](#)。

如果您遇到问题并需要技术支持，[请在 Amazon 支持中心创建案例](#)。

KPL 支持的平台

Kinesis 生产者库 (KPL) 以 C++ 编写，作为主用户进程的子进程运行。预编译的 64 位本机二进制文件与 Java 版本捆绑在一起并由 Java 包装程序管理。

无需在以下操作系统上安装任何其他库即可运行 Java 程序包：

- 内核为 2.6.18 (2006 年 9 月) 及更高版本的 Linux 发行版
- Apple OS X 10.9 及更高版本

- Windows Server 2008 及更高版本

Important

0.14.0 之前的所有 KPL 版本支持 Windows Server 2008 及更高版本。
KPL 0.14.0 及更高版本将不再支持该 Windows 平台。

请注意，KPL 仅为 64 位。

源代码

如果 KPL 安装中提供的二进制文件不足以满足您的环境，则 KPL 的核心将作为 C++ 模块编写。C++ 模块和 Java 接口的源代码根据亚马逊公共许可证发布，可在 [Kinesis Producer Library GitHub](#) 上找到。尽管 KPL 可以在任何最近有符合标准的 C++ 编译器和 JRE 的平台上使用，但亚马逊不正式支持任何不在支持平台列表中的平台。

KPL 重要概念

以下各节包含理解 Kinesis 生产者库 (KPL) 并从中受益所必需的概念和术语。

主题

- [记录 \(p. 79\)](#)
- [批处理 \(p. 79\)](#)
- [聚合 \(p. 80\)](#)
- [集合 \(p. 80\)](#)

记录

在本指南中，我们区分了 KPL 用户记录和 Kinesis Data Streams 记录。当我们使用不带限定符的术语记录时，我们指的是 KPL 用户记录。当我们提及 Kinesis Data Streams 记录时，我们明确地说 Kinesis Data Streams 记录。

KPL 用户记录是一组对用户具有特殊意义的数据。示例包括表示网站上的 UI 事件的 JSON Blob 或 Web 服务器中的日志条目。

Kinesis Data Streams 记录是 Kinesis Data Streams 服务 API 定义 `Record` 的数据结构的一个实例。它包含一个分区键、序列号和数据 Blob。

批处理

批处理 指对多个项执行单个操作而不是对每个单独的项重复执行操作。

在这种情况下，“物品”是一条记录，操作会将其发送到 Kinesis Data Streams。在非批处理情况下，您可以将每条记录放在单独的 Kinesis Data Streams 记录中，然后发出一个 HTTP 请求将其发送到 Kinesis Data Streams。利用批处理，每个 HTTP 请求可携带多条记录而不仅仅是一条记录。

KPL 支持两种类型的批处理：

- 聚合 — 在单个 Kinesis Data Streams 记录中存储多个记录。
- 收集-使用 API 操作 `PutRecords` 将多个 Kinesis Data Streams 记录发送到 Kinesis 数据流中的一个或多个分片。

这两种类型的 KPL 批处理设计为共存，可以相互独立开启或关闭。默认情况下，两种批处理将同时启用。

聚合

聚合是指在 Kinesis Data Streams 记录中存储多个记录。聚合允许客户增加每个 API 调用发送的记录数目，这将有效增加创建器吞吐量。

Kinesis Data Streams 分片每秒最多支持 1,000 条 Kinesis Data Streams 记录，或 1 MB 的吞吐量。Kinesis Data Streams 每秒记录数限制将客户与小于 1 KB 的记录绑定。记录聚合允许客户将多个记录合并为单个 Kinesis Data Streams 记录。这使客户能够提高其每分片吞吐量。

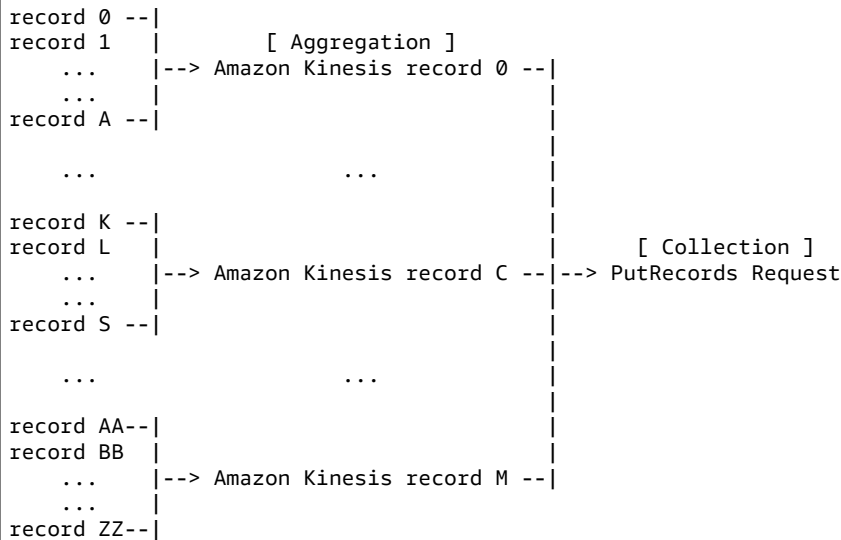
考虑以下情况：区域 us-east-1 中的一个分片当前正在以每秒 1000 条记录的恒速运行，其中每条记录的大小为 512 个字节。通过 KPL 聚合，您只能将 1,000 条记录打包成 10 条 Kinesis Data Streams 记录，从而将 RPS 减少到 10 条（每条 50 KB）。

集合

收集是指批处理多个 Kinesis Data Streams 记录，然后通过调用 API 操作在单个 HTTP 请求中发送它们 PutRecords，而不是在自己的 HTTP 请求中发送每条 Kinesis Data Streams 记录。

与不使用集合相比，这将增加吞吐量，因为它减少了发出多个单独 HTTP 请求的开销。实际上，PutRecords 本身专用于实现此目的。

收集与聚合的不同之处在于，它处理的是 Kinesis Data Streams 记录组。正在收集的 Kinesis Data Streams 记录仍然可以包含来自用户的多条记录。可通过如下方式来可视化此关系：



将 KPL 与生产者代码集成

Kinesis 生产者库 (KPL) 在单独的进程中运行，并使用 IPC 与您的父用户进程通信。此架构有时称为[微服务](#)，出于两个主要原因选择该架构：

1) 即使 KPL 崩溃，你的用户进程也不会崩溃

您的进程可能有与 Kinesis Data Streams 无关的任务，即使 KPL 崩溃，也可以继续运行。您的父用户进程也可以重新启动 KPL 并恢复到完全正常运行的状态（此功能在官方包装器中）。

例如，向 Kinesis Data Streams 发送指标的 Web 服务器；即使 Kinesis Data Streams 部分已停止工作，该服务器仍可以继续提供页面。因此，由于 KPL 中的错误导致整个服务器崩溃会导致不必要的中断。

2) 可支持任意客户端

始终存在使用官方支持的语言之外的语言的客户。这些客户还应该能够轻松使用 KPL。

推荐的使用矩阵

以下用法矩阵列举了不同用户的推荐设置，并建议您是否以及如何使用 KPL。请记住，如果启用聚合，则还必须使用取消聚合来提取使用者端的记录。

创建器端语言	使用者端语言	KCL 版本	检查点逻辑	是否能使用 KPL ?	警告
除 Java 之外的任何语言	*	*	*	否	不适用
Java	Java	直接使用 Java 软件开发工具包	不适用	是	如果使用聚合，您必须在 GetRecords 调用后使用提供的取消聚合库。
Java	除 Java 之外的任何语言	直接使用软件开发工具包	不适用	是	必须禁用聚合。
Java	Java	1.3.x	不适用	是	必须禁用聚合。
Java	Java	1.4.x	调用不带任何参数的检查点	是	None (无)
Java	Java	1.4.x	调用带明确序列号的检查点	是	禁用聚合或更改代码以使用扩展的序列号进行检查点操作。
Java	除 Java 之外的任何语言	1.3.x + 多语言守护进程 + 特定于语言的包装程序	不适用	是	必须禁用聚合。

使用 KPL 写入您的 Kinesis 数据流

以下部分说明了正在运行的示例代码，从可能最简单的“基本要素”创建者到完全异步的代码。

Barebones 创建器代码

以下代码是写入最小工作创建器所需的全部代码。Kinesis 生产者库 (KPL) 用户记录在后台处理。

```
// 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 ...
```

同步响应结果

在前面的示例中，代码没有检查 KPL 用户记录是否成功。KPL 执行解释失败所需的任何重试。但如果您要检查结果，可使用从 `addUserRecord` 返回的 `Future` 对象检查结果，如以下示例中所示（显示上一示例是为了提供上下文信息）：

```
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
        }
    }
}
```

异步响应结果

上一个示例对 `get()` 对象调用 `Future`，这会阻止执行。如果您不想阻止执行，则可使用异步回调，如以下示例所示：

```
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);
}
```

配置 Kinesis 生产者库

虽然默认设置应适用于大多数使用案例，但您可能想更改部分默认设置以定制 `KinesisProducer` 的行为来满足您的需求。为此，可将 `KinesisProducerConfiguration` 类的实例传递给 `KinesisProducer` 构造函数，例如：

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

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

您也可从属性文件中加载配置：

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

您可替换用户进程可访问的任何路径和文件名。此外，您可在通过此方式创建的 `KinesisProducerConfiguration` 实例上调用 `set` 方法来自定义 `config`。

属性文件应使用参数的名称来指定参数 `PascalCase`。这些名称将与 `KinesisProducerConfiguration` 类中的设置方法中使用的名称匹配。例如：

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

有关配置参数使用规则和值限制的更多信息，请参阅[上的示例配置属性文件 GitHub](#)。

请注意，初始化 `KinesisProducer` 后，更改已使用的 `KinesisProducerConfiguration` 实例不会产生进一步的影响。`KinesisProducer` 当前不支持动态重新配置。

使用者取消聚合

从版本 1.4.0 开始，KCL 支持自动解聚合 KPL 用户记录。更新 KCL 后，使用先前版本的 KCL 编写的消费者应用程序代码无需任何修改即可编译。但是，如果在生产者端使用 KPL 聚合，则存在一个涉及检查点的微妙之处：聚合记录中的所有子记录都具有相同的序列号，因此，如果您需要区分子记录，则必须使用检查点存储额外数据。此额外数据称作子序列号。

从 KCL 的先前版本迁移

您无需更改现有调用来同时执行检查点操作与聚合。仍然可以保证您可以检索成功存储在 Kinesis Data Streams 中的所有记录。KCL 现在提供了两个新的检查点操作来支持特定的用例，如下所述。

如果您的现有代码是在 KPL 支持之前为 KCL 编写的，并且您的检查点操作是在不带参数的情况下调用的，则等同于对批次中最后一个 KPL 用户记录的序列号进行检查点操作。如果调用带序列号字符串的检查点操作，则它等同于对批处理的指定序列号以及隐式子序列号 0（零）进行检查点操作。

调用 `checkpoint()` 不带任何参数的新 KCL 检查点操作在语义上等同于检查批处理中最后一次 `Record` 调用的序列号以及隐式子序列号 0（零）。

调用新 `checkpoint(Record record)` 的 KCL 检查点操作在语义上等同于对给定 `Record` 序列号和隐式子序列号 0（零）进行检查点操作。如果 `Record` 调用实际为 `UserRecord`，则对 `UserRecord` 序列号和子序列号进行检查点操作。

调用新的 KCL 检查点操作会 `checkpoint(String sequenceNumber, long subSequenceNumber)` 明确检查给定的序列号和给定的子序列号。

在任何情况下，将检查点存储在 Amazon DynamoDB 检查点表中后，即使应用程序崩溃并重新启动，KCL 也可以正确恢复检索记录。如果在序列中包含多条记录，则检索会从具有已进行检查点操作的最新序列号的记录中的下一个子序列号记录开始。如果最新检查点包括上一条序列号记录的最新子序列号，则检索会从具有下一个序列号的记录开始。

以下部分将讨论需要避免跳过和重复记录的使用者的序列和子序列检查点操作的详细信息。如果在停止并重新启动使用器的记录处理时跳过（或重复）记录并不重要，则可运行您的现有代码而无需进行修改。

用于 KPL 解聚合的 KCL 扩展

如前所述，KPL 解聚可能涉及子序列检查点。为了便于使用子序列检查点，已在 KCL 中添加了一个 `UserRecord` 类：

```
public class UserRecord extends Record {
    public long getSubSequenceNumber() {
        /* ... */
    }
    @Override
    public int hashCode() {
        /* contract-satisfying implementation */
    }
    @Override
    public boolean equals(Object obj) {
        /* contract-satisfying implementation */
    }
}
```

现在使用的是此类而不是 `Record`。这不会破坏现有代码，因为它是 `Record` 的子类。`UserRecord` 类同时表示实际子记录 and 标准非聚合记录。非聚合记录可被视为刚好具有一条子记录的聚合记录。

此外，将两个新的操作添加到 `IRecordProcessorCheckpoint`：

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

要开始使用子序列号检查点操作，您可执行以下转换。更改以下形式代码：

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

新的形式代码：

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

建议您使用 `checkpoint(Record record)` 形式来进行子序列检查点操作。不过，如果您已以字符串形式存储 `sequenceNumbers` 以用于检查点操作，则您现在也应存储 `subSequenceNumber`，如以下示例所示：

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

从 `Record` 到 `UserRecord` 的转换始终成功实施，因为实施始终在后台使用 `UserRecord`。除非需要对序列号进行算术运算，否则建议不要采用此方法。

在处理 KPL 用户记录时，KCL 将子序列号作为每行的额外字段写入 Amazon DynamoDB。以前版本的 `KCLATL_SEQUENCE_NUMBER` 用于在恢复检查点时提取记录。当前支持 KPL 的 `KCLATL_SEQUENCE_NUMBER` 改为使用。在检索带已进行检查点操作的序列号的记录时，会检查已进行检查点操作的子序列号，而且会根据需要删除子记录（如果上一条子记录为已进行检查点操作的记录，则可能删除所有子记录）。同样，非聚合记录可被视为具有单条子记录的聚合记录，因此相同的算法同时适用于聚合和非聚合记录。

GetRecords直接使用

您也可以选择不使用 KCL，而是 GetRecords 直接调用 API 操作来检索 Kinesis Data Streams 记录。要将这些检索到的记录解压缩到您的原始 KPL 用户记录中，请在中调用以下静态操作之一 UserRecord.java：

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

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

第一个操作使用 0 的默认值 startingHashKey (零) 和 $2^{128} - 1$ 的默认值 endingHashKey。

所有这些操作都会将给定的 Kinesis Data Streams 记录列表解聚为 KPL 用户记录列表。任何其显式哈希键或分区键超出 startingHashKey (包括) 和 (包括) 范围的 KPL 用户记录都将从返回的记录列表中丢弃。endingHashKey

将 KPL 与 Kinesis Data Firehose 一起使用

如果您使用 Kinesis 生产者库 (KPL) 将数据写入 Kinesis 数据流，则可以使用聚合来合并写入该 Kinesis 数据流的记录。如果您随后使用该数据流作为 Kinesis Data Firehose 传输流的来源，Kinesis Data Firehose 会在将记录传输到目的地之前对其进行分解。如果您将传输流配置为转换数据，Kinesis Data Firehose 会在将记录传送到之前解聚这些记录 Amazon Lambda。有关更多信息，请参阅 [使用 Kinesis Data Streams 写入 Kinesis Data Firehose](#)。

将 KPL 与 Glue 架构注册表一起使用

您可以将您的 Kinesis 数据流与 Amazon Glue 架构注册表集成。Amazon Glue 架构注册表允许您集中发现、控制和演变架构，同时确保生成的数据由注册架构持续验证。架构定义了数据记录的结构和格式。架构是用于可靠数据发布、使用或存储的版本化规范。Amazon Glue Schema Registry 使您能够改善流媒体应用程序中的 end-to-end 数据质量和数据治理。有关更多信息，请参阅 [Amazon Glue 架构注册表](#)。设置这种集成的方法之一是通过 Java 中的 KPL 和 Kinesis Client (KPL) 库。

Important

目前，只有使用 Java 实现的 KPL 生成器的 Kinesis Data Streams 支持 Kinesis 数据流和 Amazon Glue 架构注册表集成。不提供多语言支持。

有关如何使用 KPL 设置 Kinesis Data Streams 与架构注册表集成的详细说明，请参阅 [用例中的“使用 KPL/KCL 库与数据交互”部分：将 Amazon Kinesis Data Streams 与 Amazon Glue 架构注册表集成](#)。

KPL 代理配置

对于无法直接连接到互联网的应用程序，所有 Amazon SDK 客户端都支持使用 HTTP 或 HTTPS 代理。在典型的企业环境中，所有出站网络流量都必须通过代理服务器。如果您的应用程序使用 Kinesis Producer 库 (KPL) Amazon 在使用代理服务器的环境中收集和发送数据，则您的应用程序将需要 KPL 代理配置。KPL 是基于 Kinesis SDK 构建的高级库。它被分为一个原生进程和一个封装器。本机进程执行处理和发送记录的所有任务，而封装器管理本机进程并与之通信。有关更多信息，请参阅 [使用 Amazon Kinesis 生产者库实现高效可靠的生产器](#)。

包装器使用 Java 编写，本机进程使用 Kinesis SDK 用 C++ 编写。KPL 版本 0.14.7 及更高版本现在支持 Java 包装器中的代理配置，它可以将所有代理配置传递给本机进程。有关更多信息，请参阅 <https://github.com/aws-labs/amazon-kinesis-producer/releases/tag/v0.14.7>。

您可以使用以下代码将代理配置添加到 KPL 应用程序中。

```
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);
```

将 Amazon Kinesis Data Streams API 与结合使用来开发创建者 Amazon SDK for Java

您可使用 Amazon Kinesis Data Streams API 与结合使用来开发创建者。Amazon SDK for Java。如果您是首次使用 Kinesis Data Streams，请先熟悉一下中介绍的概念和术语。[什么是 Amazon Kinesis Data Streams？ \(p. 1\)](#)和[Amazon Kinesis Data Streams 入门 \(p. 12\)](#)。

这些示例讨论 [Kinesis Data Streams API](#) 然后使用 [Amazon 适用于 Java 的开发工具包](#) 向流添加（放置）数据。但是，在大多数使用案例中，您应会更喜欢使用 Kinesis Data Streams KPL 库。有关更多信息，请参阅 [使用 Amazon Kinesis 生产者库开发生产者 \(p. 76\)](#)。

本章中的 Java 示例代码演示如何执行基本的 Kinesis Data Streams API 操作，并按照操作类型从逻辑上进行划分。这些示例并非可直接用于生产的代码，因为它们不会检查所有可能的异常，或者不会考虑到所有可能的安全或性能问题。此外，您可以调用 [Kinesis Data Streams API](#) 使用其他编程语言。有关所有可用的更多信息 Amazon 开发工具包，请参阅 [使用 Amazon Web Services 开发](#)。

每个任务都有先决条件；例如，您在创建流之后才能向流中添加数据，而创建流需要先创建一个客户端。有关更多信息，请参阅 [创建和管理流 \(p. 59\)](#)。

主题

- [向流添加数据 \(p. 86\)](#)
- [使用与数据进行交互 Amazon 架 Glue 注册表 \(p. 90\)](#)

向流添加数据

在创建流之后，您可以记录的形式向其中添加数据。记录是一种数据结构，其中包含要处理的数据（采用数据 Blob 形式）。在将数据存储到记录中之后，Kinesis Data Streams 不会以任何形式检查、解释或更改数据。每个记录还有一个关联的序列号和分区键。

Kinesis Data Streams API 中有两种不同的向流添加数据的操作：[PutRecords](#)和[PutRecord](#)。

[PutRecords](#) 操作将按 HTTP 请求向您的流发送多个记录，并且单个 [PutRecord](#) 操作一次可向您的流发送多个记录（每个记录需要单独的 HTTP 请求）。对于大多数应用程序，您应会更喜欢使用 [PutRecords](#)，因为这将使每个数据创建者实现更高的吞吐量。有关每种操作的更多信息，请参阅以下各小节。

主题

- [使用添加多个记录 PutRecords \(p. 87\)](#)
- [使用添加单一记录 PutRecord \(p. 89\)](#)

请始终记住，在您的源应用程序使用 Kinesis Data Streams API 向流添加数据时，很可能存在一个或多个同时处理离开流的数据的使用者应用程序。有关使用者使用 Kinesis Data Streams API 获取数据的信息，请参阅 [从流中获取数据 \(p. 139\)](#)。

Important

[更改数据保留期 \(p. 72\)](#)

使用添加多个记录PutRecords

这些区域有：[PutRecords](#)操作在一次请求中向 Kinesis Data Streams 发送多个记录。通过使用PutRecords，创建器可在向 Kinesis Data Streams 发送数据时实现更高的吞吐量。EARDSPutRecords请求最多可以支持 500 条记录。请求中的每一个记录最大可以为 1 MB，整个请求的上限为 5 MB，包括分区键。和单一样PutRecord下面描述的操作，PutRecords使用序列号和分区键。但是，PutRecord 参数 SequenceNumberForOrdering 未包含在 PutRecords 调用中。PutRecords 操作将尝试按请求的自然顺序处理所有记录。

每个数据记录都有一个唯一的序列号。此序列号在您调用之后由 Kinesis Data Streams 分配。client.putRecords向流添加数据记录。同一分区键的序列号通常会随时间变化增加；PutRecords 请求之间的时间段越长，序列号变得越大。

Note

序列号不能用作相同流中的数据集的索引。为了在逻辑上分隔数据集，请使用分区键或者为每个数据集创建单独的流。

PutRecords 请求可包含具有不同分区键的记录。请求的应用范围是一个流；每个请求可包含分区键和记录的任何组合，直到达到请求限制。使用许多不同的分区键对具有许多不同分片的流进行的请求一般快于使用少量分区键对少量分片进行的请求。分区键的数量应远大于分片的数量以减少延迟并最大程度提高吞吐量。

PutRecords示例

以下代码创建 100 个使用连续分区键的数据记录并将其放入名为 DataStream 的流中。

```
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);
```

PutRecords 响应包含响应 Records 的数组。响应数组中的每个记录按自然顺序（从请求和响应的顶部到底部）直接与请求数组中的一个记录关联。响应 Records 数组包含的记录数量始终与请求数组相同。

处理使用时的失败PutRecords

默认情况下，请求内的单个记录的失败不会中止对 PutRecords 请求中后续记录的处理。这意味着，响应 Records 数组包含处理成功和不成功的记录。您必须删除处理不成功的记录并在后续调用中包括它们。

成功的记录包括 SequenceNumber 和 ShardID 值，而不成功的记录包含 ErrorCode 和 ErrorMessage 值。ErrorCode 参数反映了错误类型，可能为下列值之

一：ProvisionedThroughputExceededException 或 InternalFailure。ErrorMessage 提供有关 ProvisionedThroughputExceededException 异常的更多详细信息，包括账户 ID、流名称和已阻止的记录的片 ID。以下示例在 PutRecords 请求中有三个记录。第二个记录失败并反映在响应中。

Example PutRecords请求语法

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

Example PutRecords响应语法

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

处理不成功的请求可包含在后续 PutRecords 请求中。首先，查看 putRecordsResult 中的 FailedRecordCount 参数以确认请求中是否存在失败的记录。如果存在，则应将具有 putRecordsEntry (不是 ErrorCode) 的每个 null 添加到后续请求中。有关此类处理程序的示例，请参阅以下代码。

Example PutRecords失败处理程

```
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);
}
```

使用添加单一记录PutRecord

对 [PutRecord](#) 的每次调用对一个记录起作用。应首选 [PutRecords](#) 中描述的 [使用添加多个记录 PutRecords \(p. 87\)](#) 操作，除非您的应用程序明确需要每一请求始终发送一条记录，或因某种其他原因无法使用 [PutRecords](#)。

每个数据记录都有一个唯一的序列号。此序列号在您调用之后由 Kinesis Data Streams 分配。client.putRecord向流添加数据记录。同一分区键的序列号通常会随时间变化增加；PutRecord 请求之间的时间段越长，序列号变得越大。

在快速连续进行放置时，不保证返回的序列号会递增，因为放置操作的发生对于 Kinesis Data Streams 实际上是同步进行的。要确保相同分区键的序列号严格递增，请使用 [SequenceNumberForOrdering](#) 参数，如 [PutRecord示例 \(p. 89\)](#) 代码示例中所示。

你是否使用SequenceNumberForOrdering，Kinesis Data Streams 通过GetRecords呼叫严格按序列号排序。

Note

序列号不能用作相同流中的数据集的索引。为了在逻辑上分隔数据集，请使用分区键或者为每个数据集创建单独的流。

分区键用于对流中的数据进行分组。数据记录将基于其分区键分配给流中的分片。具体来说，Kinesis Data Streams 将分区键作为将分区键（和关联数据）映射到特定分片的哈希函数的输入。

作为此哈希机制的结果，具有相同分区键的所有数据记录将映射到流中的同一分片。但是，如果分区键的数量超出分片的数量，则一些分片必定会包含具有不同分区键的记录。从设计的角度看，要确保您的所有分片得到充分利用，分片的数量（由 setShardCount 的 CreateStreamRequest 方法指定）应远少于唯一分区键的数量，并且流至单一分区键的数据量应远少于分片容量。

PutRecord示例

以下代码创建跨两个分区键分配的 10 条数据记录，并将它们放入名为 myStreamName 的流中。

```
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();  
}
```

上一个代码示例使用 `setSequenceNumberForOrdering` 来确保每个分区键内的顺序严格递增。要有效地使用此参数，请将 `SequenceNumberForOrdering` 当前记录的 (记录) `n` 到前一条记录的序列号 (记录) `n-1`。要获取已添加到流的记录的序列号，请对 `putRecord` 的结果调用 `getSequenceNumber`。

`SequenceNumberForOrdering` 参数可确保相同分区键的序列号严格递增。`SequenceNumberForOrdering` 不提供跨多个分区键的记录排序。

使用与数据进行交互Amazon架 Glue 注册表

您可以将 Kinesis 数据流与 Amazon Glue 模式注册表。这些区域有：Amazon 使用 Glue 架构注册表，您可以集中发现、控制和演变架构，同时确保通过注册表持续验证创建的数据。架构定义了数据记录的结构和格式。架构是用于可靠数据发布、使用或存储的版本化规范。这些区域有：Amazon Glue 模式注册表使您能够改进 end-to-end 流媒体应用程序中的数据质量和数据治理。有关更多信息，请参阅 [Amazon 架 Glue 注册表](#)。设置此集成的方法之一是通过 `PutRecords` 和 `PutRecord` Kinesis Data Streams API 可在 Amazon Java 开发工具包。

有关如何使用将 Kinesis Data Streams 与架构注册表集成的详细说明 `PutRecords` 和 `PutRecord` Kinesis Data Streams API，请参阅中的“使用 Kinesis Data Streams API 与数据交互”部分 [使用案例：将 Amazon Kinesis Data Streams 与 Amazon 架 Glue 注册表](#)。

使用 Amazon Kinesis Data Streams s Kinesis Streams

Kinesis Agent 是一款独立的 Java 软件应用程序，它提供了一种收集数据并将其发送到 Kinesis Data Streams 的简便方法。此代理持续监控一组文件，并将新数据发送到您的流。代理处理文件轮换、检查点操作并在失败时重试。它以可靠及时的简单方法提供您的所有数据。它还会发布亚马逊 CloudWatch 指标，以帮助您更好地监控和解决直播过程。

默认情况下，会基于换行符 ('\n') 分析每个文件中的记录。但是，也可以将代理配置为分析多行记录 (请参阅 [代理配置设置 \(p. 92\)](#))。

您可以在基于 Linux 的服务器环境 (如 Web 服务器、日志服务器和数据库服务器) 上安装此代理。在安装代理后，通过指定要监控的日志文件和 名称来配置代理。在配置好代理之后，代理将持续从这些文件中收集数据并以可靠的方式将数据发送到流。

主题

- [先决条件 \(p. 90\)](#)
- [下载并安装代理 \(p. 91\)](#)
- [配置并启动代理 \(p. 91\)](#)
- [代理配置设置 \(p. 92\)](#)
- [监控多个文件目录并写入多个流 \(p. 94\)](#)
- [使用代理预处理数据 \(p. 94\)](#)
- [代理 CLI 命令 \(p. 97\)](#)

先决条件

- 您的操作系统必须是 2015.09 或更高版本的亚马逊 Linux AMI，或者是红帽企业 Linux 版本 7 或更高版本。

- ## 下载并安装代理

配置并启动代理

用文件或创建新文件的次数不能超过一次。代理使用文件创建时间戳来确定要跟踪并送入您的流中的文件；如果每秒创建新文件或轮换使用文件的次数超过一次，代理将无法正确区分这些文件。

```
{
  "flows": [
    {
      "filePattern": "/tmp/app.log*",
      "kinesisStream": "yourkinesisstream"
    }
  ]
}
```

2. 手动启动代理：

```
sudo service aws-kinesis-agent start
```

3. (可选) 将代理配置为在系统启动时启动：

```
sudo chkconfig aws-kinesis-agent on
```

现在，代理作为系统服务在后台运行。它会持续监控指定的文件，并将数据发送到指定的流。代理活动记录在 `/var/log/aws-kinesis-agent/aws-kinesis-agent.log` 中。

代理配置设置

代理支持两个必需的配置设置，即 `filePattern` 和 `kinesisStream`，以及可用于其他功能的可选配置设置。您可以在 `/etc/aws-kinesis/agent.json` 中指定必需配置和可选配置。

只要您更改配置文件，就必须使用以下命令停止再启动代理：

```
sudo service aws-kinesis-agent stop
sudo service aws-kinesis-agent start
```

或者，您也可以使用以下命令：

```
sudo service aws-kinesis-agent restart
```

一般的设置配置如下。

配置设置	描述
<code>assumeRoleARN</code>	由该用户担任的角色的 ARN。有关更多信息，请参阅《IAM 用户指南中的使用 IAM 角色委托跨 Amazon 账户的访问权限》。
<code>assumeRoleExternalId</code>	确定谁可以担任该角色的可选标识符。有关更多信息，请参阅《IAM 用户指南》中的 如何使用外部 ID 。
<code>awsAccessKeyId</code>	Amazon 覆盖默认凭据的访问密钥 ID。此设置优先于所有其他凭证提供程序。
<code>awsSecretAccessKey</code>	Amazon 覆盖默认凭证的密钥。此设置优先于所有其他凭证提供程序。
<code>cloudwatch.emitMetrics</code>	CloudWatch 如果设置为 <code>true</code> ，则允许代理向其发送指标。 默认值：True
<code>cloudwatch.endpoint</code>	的区域终端节点 CloudWatch。

配置设置	描述
	默认值：monitoring.us-east-1.amazonaws.com
kinesis.endpoint	Kinesis Data Streams 的区域终端节点。 默认值：kinesis.us-east-1.amazonaws.com

流配置设置如下。

配置设置	描述
dataProcessingOptions	在将每个被分析的记录发送到流之前应用于这些记录的处理选项的列表。这些处理选项按指定的顺序执行。有关更多信息，请参阅 使用代理预处理数据 (p. 94) ：
kinesisStream	[必需] 流名称。
filePattern	[必需] 必须由代理监控的文件的 Glob。与此模式匹配的任何文件都会自动由代理挑选出来并进行监控。对于与此模式匹配的所有文件，必须向 aws-kinesis-agent-user 授予读取权限。对于包含这些文件的目录，必须向 aws-kinesis-agent-user 授予读取和执行权限。
initialPosition	开始解析文件的初始位置。有效值为 START_OF_FILE 和 END_OF_FILE。 默认值：END_OF_FILE
maxBufferAgeMillis	代理在将数据发送到流之前缓冲数据的最长时间（以毫秒计）。 值范围：1000 到 900000（1 秒到 15 分钟） 默认值：60000（1 分钟）
maxBufferSizeBytes	代理在将数据发送到流之前缓冲的数据的最大大小（以字节计）。 值范围：1 到 4194304 (4 MB) 默认值：4194304 (4 MB)
maxBufferSizeRecords	代理在将数据发送到流之前缓冲数据的最大记录数。 值范围：1 到 500 默认值：500
minTimeBetweenFilePolls	代理检查和分析受监控文件中是否有新数据的时间间隔（以毫秒计）。 值范围：1 或更大值 默认值：100
multiLineStartPattern	用于标识记录开始的模式。记录由与模式匹配的行以及与模式不匹配的任何以下行组成。有效值为正则表达式。默认情况下，日志文件中的每一个新行都被解析为一条记录。
partitionKeyOption	生成分区键的方法。有效值为 RANDOM（随机生成的整数）和 DETERMINISTIC（基于数据计算出来的哈希值）。 默认值：RANDOM

配置设置	描述
skipHeaderLines	代理从受监控文件开头跳过分析的行数。 值范围：0 或更大值 默认值：0 (零)
truncatedRecordTerminationChar	当记录大小超过 Kinesis Data Streams 记录大小限制时，代理用来截断已解析的记录字符串。(1000 KB) 默认值：'\n' (换行符)

监控多个文件目录并写入多个流

通过指定多个流程配置设置，您可以配置代理以监控多个文件目录并将数据发送到多个流。在以下配置示例中，代理监控两个文件目录并将数据分别发送到 Kinesis 流和 Kinesis Data Firehose 传输流。请注意，您可以为 Kinesis Data Streams 和 Kinesis Data Firehose 指定不同的端点，这样您的 Kinesis 直播和 Kinesis Data Firehose 传输流就不必位于同一区域。

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

有关在 Kinesis Data Firehose 中使用代理的更多详细信息，请参阅使用 Kinesis Agent 写入亚马逊 Kinesis Data Firehose。

使用代理预处理数据

代理可以预处理从受监控文件分析的记录，然后再将这些记录发送到流。通过将 `dataProcessingOptions` 配置设置添加到您的文件流可以启用此功能。可以添加一个或多个处理选项，这些选项将按指定的顺序执行。

代理支持下面列出的处理选项。由于此代理是开源的，您可以进一步开发和扩展其处理选项。你可以从 [Kinesis Agent 下载代理](#)。

处理选项

SINGLELINE

通过移除换行符和首尾的空格，将多行记录转换为单行记录。

```
{
  "optionName": "SINGLELINE"
}
```

CSVTOJSON

将记录从分隔符分隔的格式转换为 JSON 格式。

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

customFieldNames

[必需] 在每个 JSON 键值对中用作键的字段名称。例如，如果您指定 ["f1", "f2"]，则记录“v1, v2”将转换为 {"f1": "v1", "f2": "v2"}。

delimiter

在记录中用作分隔符的字符串。默认为逗号 (,)。

LOGTOJSON

将记录从日志格式转换为 JSON 格式。支持的日志格式为 Apache Common Log、Apache Combined Log、Apache Error Log 和 RFC3164 Syslog。

```
{
  "optionName": "LOGTOJSON",
  "logFormat": "logformat",
  "matchPattern": "yourregexpattern",
  "customFieldNames": [ "field1", "field2", ... ]
}
```

logFormat

[必需] 日志条目格式。以下是可能的值：

- COMMONAPACHELOG— Apache 通用日志格式。默认情况下每个日志条目都为以下模式：“%{host} %{ident} %{authuser} [%{datetime}] \"%{request}\" %{response} %{bytes}”。
- COMBINEDAPACHELOG— Apache 组合日志格式。默认情况下每个日志条目都为以下模式：“%{host} %{ident} %{authuser} [%{datetime}] \"%{request}\" %{response} %{bytes} %{referrer} %{agent}”。
- APACHEERRORLOG— Apache 错误日志格式。默认情况下每个日志条目都为以下模式：“[%{timestamp}] [%{module}:%{severity}] [pid %{processid}:tid %{threadid}] [client: %{client}] %{message}”。
- SYSLOG— RFC3164 系统日志格式。默认情况下每个日志条目都为以下模式：“%{timestamp} %{hostname} %{program}[%{processid}]: %{message}”。

matchPattern

用于从日志条目中提取值的正则表达式模式。如果您的日志条目不属于其中一种预定义日志格式，则将使用此设置。如果使用此设置，您还必须指定 customFieldNames。

customFieldNames

在每个 JSON 键值对中用作键的自定义字段名称。您可以使用此设置定义从 matchPattern 中提取的值的字段名称，或覆盖预定义日志格式的默认字段名称。

Example : LOGTOJSON 配置

下面是一个转换为 JSON 格式的 Apache 通用日志条目的 LOGTOJSON 配置示例：

```
{
  "optionName": "LOGTOJSON",
  "logFormat": "COMMONAPACHELOG"
}
```

转换前：

```
64.242.88.10 - - [07/Mar/2004:16:10:02 -0800] "GET /mailman/listinfo/hsdivision HTTP/1.1"
200 6291
```

转换后：

```
{
  "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：包含自定义字段的 LOGTOJSON 配置

下面是 LOGTOJSON 配置的另一个示例：

```
{
  "optionName": "LOGTOJSON",
  "logFormat": "COMMONAPACHELOG",
  "customFieldNames": ["f1", "f2", "f3", "f4", "f5", "f6", "f7"]
}
```

使用此配置设置时，上一个示例中的同一个 Apache 通用日志条目将如下转换为 JSON 格式：

```
{
  "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：转换 Apache 通用日志条目

以下流配置将一个 Apache 通用日志条目转换为 JSON 格式的单行记录：

```
{
  "flows": [
    {
      "filePattern": "/tmp/app.log*",
      "kinesisStream": "my-stream",
      "dataProcessingOptions": [
        {
          "optionName": "LOGTOJSON",
          "logFormat": "COMMONAPACHELOG"
        }
      ]
    }
  ]
}
```

Example：转换多行记录

以下流配置分析第一行以“[SEQUENCE=”开头的多行记录。每个记录先转换为一个单行记录。然后，将基于制表分隔符从记录中提取值。提取的值映射到指定的 customFieldNames 值，从而形成 JSON 格式的单行记录。

```
{
```

```

    "flows": [
      {
        "filePattern": "/tmp/app.log*",
        "kinesisStream": "my-stream",
        "multilineStartPattern": "\\[SEQUENCE=",
        "dataProcessingOptions": [
          {
            "optionName": "SINGLELINE"
          },
          {
            "optionName": "CSVTOJSON",
            "customFieldNames": [ "field1", "field2", "field3" ],
            "delimiter": "\\t"
          }
        ]
      }
    ]
  }
}

```

Example : 具有匹配模式的 LOGTOJSON 配置

下面是一个转换为 JSON 格式的 Apache 通用日志条目的 LOGTOJSON 配置示例，其中省略了最后一个字段 (bytes)：

```

{
  "optionName": "LOGTOJSON",
  "logFormat": "COMMONAPACHELOG",
  "matchPattern": "^(\\d\\.\\d+\\.\\d+) (\\S+) (\\S+) \\[([\\w:/]+\\s[+\\-]\\d{4})\\] \\\"(.+?)\\\" (\\d{3})",
  "customFieldNames": ["host", "ident", "authuser", "datetime", "request", "response"]
}

```

转换前：

```
123.45.67.89 - - [27/Oct/2000:09:27:09 -0400] "GET /java/javaResources.html HTTP/1.0" 200
```

转换后：

```
{"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"}
```

代理 CLI 命令

系统启动时自动启动代理：

```
sudo chkconfig aws-kinesis-agent on
```

检查代理的状态：

```
sudo service aws-kinesis-agent status
```

停止代理：

```
sudo service aws-kinesis-agent stop
```

从此位置读取代理的日志文件：

```
/var/log/aws-kinesis-agent/aws-kinesis-agent.log
```

卸载代理：

```
sudo yum remove aws-kinesis-agent
```

使用其他Amazon服务写入 Kinesis 数据流

以下是可以直接与 Kinesis Data Streams 集成以将数据写入 Kinesis Data Streams 的其他Amazon服务列表：

主题

- [Amazon Amplify \(p. 98\)](#)
- [Amazon Aurora \(p. 98\)](#)
- [Amazon CloudFront \(p. 98\)](#)
- [亚马逊CloudWatch日志 \(p. 99\)](#)
- [Amazon Connect \(p. 99\)](#)
- [Amazon Database Migration Service \(p. 99\)](#)
- [Amazon DynamoDB \(p. 99\)](#)
- [Amazon EventBridge \(p. 99\)](#)
- [Amazon IoT Core \(p. 99\)](#)
- [Amazon Relational Database Service \(p. 100\)](#)
- [Amazon Pinpoint \(p. 100\)](#)
- [亚马逊量子账本数据库 \(p. 100\)](#)

Amazon Amplify

您可以使用 Amazon Kinesis 数据流轻松地使用 Amazon Amplify 构建的移动应用程序流式传输数据，以进行实时处理。然后，您可以构建实时仪表板，捕获异常并生成警报，提出建议，并做出其他实时业务或运营决策。您还可以将数据轻松发送到其他服务中，如 Amazon Simple Storage Service、Amazon DynamoDB 和 Amazon Redshift。

有关更多信息，请参阅 Amazon Amplify 开发者中心中的 [“使用亚马逊 Kinesis”](#)。

Amazon Aurora

您可以使用亚马逊 Kinesis 数据流来监控您的亚马逊 Aurora 数据库集群上的活动。使用数据库活动流，您的 Aurora 数据库集群将活动实时推送到 Amazon Kinesis 数据流。然后，您可以构建用于合规性管理的应用程序，这些应用程序使用这些活动，对其进行审核并生成警报。您也可以使用亚马逊 Kinesis Data Firehose 来存储数据。

有关更多信息，请参阅 Amazon Aurora 开发人员指南中的 [数据库活动流](#)。

Amazon CloudFront

您可以将 Amazon Kinesis 数据流与 CloudFront 实时日志结合使用，实时获取有关向分配发出的请求的信息。然后，您可以构建自己的 [Kinesis 数据流使用者](#)，或者使用亚马逊 Kinesis Data Firehose 将日志数据发

送到亚马逊简单存储服务 (Amazon S3)、亚马逊 Redshift、亚马逊OpenSearch服务 (OpenSearch服务) 或第三方日志处理服务。

有关更多信息，请参阅《亚马逊CloudFront开发者指南》中的[实时日志](#)。

亚马逊CloudWatch日志

您可以使用CloudWatch订阅来获取来自 Amazon Logs CloudWatch 的日志事件的实时源并将其传输到 Amazon Kinesis 数据流进行处理、分析并加载到其他系统。

有关更多信息，请参阅 Amazon Log CloudWatch s 用户指南中的通过订阅实时处理日志[数据](#)。

Amazon Connect

您可以使用 Kinesis 数据流从 Amazon Connect 实例实时导出联系记录和代理事件。然后，您可以对联系人进行实时分析，并构建处理数据的消费者应用程序。

然后，您可以构建消费者应用程序来实时处理和分析数据。例如，使用联系人记录和客户档案数据，您可以使用最新信息保留源系统数据，例如 CRM 和营销自动化工具。up-to-date使用代理事件数据，您可以创建显示代理信息和事件的仪表板，并触发特定代理活动的自定义通知。

有关更多信息，请参阅 Amazon Connect 管理员指南中的[实例数据流、设置实时导出和代理事件流](#)。

Amazon Database Migration Service

您可以使用将数据迁移Amazon Database Migration Service到亚马逊 Kinesis 数据流。然后，您可以构建用于实时处理数据记录的消费者应用程序。您还可以轻松地将数据向下游发送到其他服务，例如亚马逊简单存储服务、亚马逊 DynamoDB 和亚马逊 Redshift。

有关更多信息，请参阅《Amazon Database Migration Service用户指南》中的[“使用 Kinesis 数据流”](#)。

Amazon DynamoDB

您可以使用 Amazon Kinesis Data Streams 捕获 Amazon DynamoDB 的更改。Kinesis Data Streams 在任何 DynamoDB 表中捕获物品级修改并将其复制到 Kinesis 数据流。您的消费者应用程序可以访问此数据流，实时查看商品级别的更改，并将这些更改传送到下游或根据内容采取行动。

有关更多信息，请参阅《亚马逊 DynamoDB 开发者指南》中的 Kinesis Data Streams 如何与 Dynam [oDB](#) 配合使用。

Amazon EventBridge

使用 Kinesis 数据流，您可以将 Amazon API 调用[事件](#)发送EventBridge到数据流、构建消费者应用程序和处理大量数据。您还可以在 Pipes 中使用 Kinesis Data Streams 作为 Pipes 中的EventBridge目标，并在可选的筛选和丰富后将记录传送到来自可用源之一的流中。

有关更多信息，请参阅亚马逊EventBridge用户指南中的[将事件发送到亚马逊 Kinesis 直播](#)和[EventBridge管道](#)。

Amazon IoT Core

您可以使用 Amazon IoT 规则操作从 IoT Core 中的 MQTT Amazon T 消息中实时写入数据。然后，您可以构建用于处理数据、分析其内容和生成警报的应用程序，并将其传送到分析应用程序或其他Amazon服务中，

有关更多信息，请参阅《Amazon 物联网核心开发者指南》中的 [Kinesis 数据流](#)。

Amazon Relational Database Service

您可以使用亚马逊 Kinesis 数据流来监控您的亚马逊 RDS 实例上的活动。使用数据库活动流，Amazon RDS 将活动实时推送到亚马逊 Kinesis 数据流。然后，您可以构建用于合规性管理的应用程序，这些应用程序使用这些活动，对其进行审核并生成警报。您也可以使用亚马逊 Kinesis Data Firehose 来存储数据。

有关更多信息，请参阅 Amazon RDS 开发人员指南中的 [数据库活动流](#)。

Amazon Pinpoint

您可以将 Amazon Pinpoint 设置为将事件数据发送到亚马逊 Kinesis 数据流。Amazon Pinpoint 可以发送活动、旅程以及交易电子邮件和短信的事件数据。然后，您可以将数据采集到分析应用程序中，或者构建自己的消费者应用程序，根据事件内容采取行动。

有关更多信息，请参阅《亚马逊 Pinpoint 开发者指南》中的 [直播活动](#)。

亚马逊量子账本数据库

您可以在 QLDB 中创建一个数据流，以捕获提交给您的日记的所有文档修订版，并将这些数据实时传输到 Amazon Kinesis 数据流。QLDB 流是从账本日志流向 Kinesis 数据流资源的连续数据流。然后，您可以使用 Kinesis 流媒体平台或 Kinesis 客户端库来使用您的直播、处理数据记录和分析数据内容。QLDB 流以三种类型的记录将您的数据写入 Kinesis 数据流：control、block summary 和 revision details。

有关更多信息，请参阅 Amazon QLDB 开发者指南中的 [直播](#)。

使用第三方集成

您可以使用以下与 Kinesis 数据流集成的第三方选项之一将数据写入 Kinesis Data Streams：

主题

- [Apache Flink \(p. 100\)](#)
- [Fluentd \(p. 100\)](#)
- [Debezium \(p. 101\)](#)
- [甲骨文 GoldenGate \(p. 101\)](#)
- [Kafka Connect \(p. 101\)](#)
- [Adobe \(p. 101\)](#)
- [Striim \(p. 101\)](#)

Apache Flink

Apache Flink 是一个框架和分布式处理引擎，用于对无界和有界数据流进行有状态计算。有关从 Apache Flink 写入 Kinesis Data Streams 的更多信息，请参阅 [Amazon Kinesis Data Streams 连接器](#)。

Fluentd

Fluentd 是用于统一日志层的开源数据收集器。有关从 Fluentd 写入 Kinesis Data Streams 的更多信息，有关更多信息，请参阅 [使用 Kinesis 进行流处理](#)。

Debezium

Debezium 是一个用于采集变更数据的开源分布式平台。有关从 Debezium 写入 Kinesis Data Streams 的更多信息，请参阅[将 MySQL 数据更改流式传输到 Amazon Kinesis](#)。

甲骨文 GoldenGate

Oracle GoldenGate 是一种软件产品，允许您将数据从一个数据库复制、筛选和转换到另一个数据库。有关从 Oracle 写入 Kinesis Data Streams 的更多信息 GoldenGate，请参阅[使用 Oracle 将数据复制到 Kinesis 数据流 GoldenGate](#)。

Kafka Connect

Kafka Connect 是一款可扩展、可靠地在 Apache Kafka 和其他系统之间传输数据的工具。有关将数据从 Apache Kafka 写入 Kinesis Data Streams 的更多信息，请参阅[Kinesis kafka 连接器](#)。

Adobe

Adobe Experience Platform 使组织能够集中和标准化来自任何系统的客户数据。然后，它应用数据科学和机器学习来显著改善丰富的个性化体验的设计和交付。有关将数据从 Adobe Experience Platform 写入 Kinesis Data Streams 的更多信息。请参阅[如何创建 Amazon Kinesis 连接](#)。

Striim

Striim 是一个完整的内存平台，用于实时收集、筛选、转换、丰富、聚合、分析和提供数据。end-to-end 有关如何从 Striim 向 Kinesis Data Streams 写入数据的更多信息，请参阅[Kinesis Writer](#)。

将 Amazon Kinesis Data Streams 创建器故障

下列章节提供使用 Amazon Kinesis Data Streams 创建器时可能发现的一些常见问题的解决方案。

- [创建器应用程序的写入速率比预期的慢 \(p. 101\)](#)
- [未授权的 KMS 主密钥权限错误 \(p. 102\)](#)
- [面向生产者的常见问题、问题和故障排除想法 \(p. 102\)](#)

创建器应用程序的写入速率比预期的慢

写入吞吐量低于预期的最常见原因如下所示。

- [超过服务限制 \(p. 101\)](#)
- [创建器优化 \(p. 102\)](#)

超过服务限制

要查明是否超过了服务限制，请检查您的创建器是否从服务引发了吞吐量异常，并验证哪些 API 操作受限制。请记住，根据调用会有不同的限制，具体请参阅[配额和限制 \(p. 6\)](#)。例如，除了最广为人知的对读写操作的分片级别限制之外，还存在以下流级别的限制：

- [CreateStream](#)
- [DeleteStream](#)

- [ListStreams](#)
- [GetShard迭代器](#)
- [MergeShards](#)
- [DescribeStream](#)
- [DescribeStream摘要](#)

操作 `CreateStream`、`DeleteStream`、`ListStreams`、`GetShardIterator` 和 `MergeShards` 限制为每秒 5 个调用。`DescribeStream` 操作限制为每秒 10 个调用。`DescribeStreamSummary` 操作限制为每秒 20 个调用。

如果这些调用不存在问题，请确保您选择了允许在所有分片之间均匀分配 `put` 操作的分区键，并且没有某个特殊分区键无意中达到了服务限制而其他分区键则未达到限制。这要求您衡量高峰吞吐量并考虑流中的分片数量。有关管理流的详细信息，请参阅[创建和管理流 \(p. 59\)](#)。

Tip

请记住，在使用单记录操作 [PutRecord](#) 时，吞吐量限制计算会向上舍入到最接近的 KB 值，而多记录操作 [PutRecords](#) 对各个调用中记录的累加和进行舍入。例如，带有 600 个 1.1 KB 大小记录的 `PutRecords` 请求不会受到限制。

创建器优化

在您开始优化创建器之前，有一些关键任务需要完成。首先，根据记录大小和每秒记录数，确定您需要的高峰吞吐量。接下来，排除流容量作为限制因素 ([超过服务限制 \(p. 101\)](#)) 的可能性。如果您已排除了流容量，对于两种常见类型的创建器，请使用以下故障排除提示和优化指南。

大型创建器

大型创建器通常从本地服务器或 Amazon EC2 实例运行。需要由大型创建器提供较高吞吐量的客户通常会关注每记录延迟。处理延迟的策略包括以下内容：如果客户可以微批处理/缓冲区记录，请使用[Kinesis 创建者库](#)（具有高级聚合逻辑），多记录操作 [PutRecords](#)，或者在使用单记录操作之前将记录聚合到较大的文件中 [PutRecord](#)。如果无法批处理/缓冲，则使用多个线程来同时写入 Kinesis Data Streams 服务。Amazon SDK for Java 和其他开发工具包中包括异步客户端，可以用非常少的代码完成此任务。

小型创建器

小型创建器通常是移动应用程序、IoT 设备或 Web 客户端。如果是移动应用程序，我们建议使用 `PutRecords` 操作或者 Kinesis 记录器中 Amazon 移动开发工具包。有关更多信息，请参阅 [Amazon Mobile SDK for Android 入门指南](#) [Amazon Mobile SDK for iOS 入门指南](#)。移动应用程序自身必须处理断续连接，并且需要某种类型的批量 `put`，例如 `PutRecords`。如果由于某些原因而无法批处理，请参阅上面的大型创建器信息。如果您的创建器是浏览器，则生成的数据量通常非常小。不过，您将 `put` 操作放在了应用程序的关键路径上，我们建议不要这样做。

未授权的 KMS 主密钥权限错误

当创建器应用程序在 KMS 主密钥上写入加密流但没有权限时，会发生此错误。要将权限分配给应用程序以访问 KMS 密钥，请参阅[使用以下密钥策略 Amazon 自杀](#)和[将 IAM 策略用于 Amazon 自杀](#)。

面向生产者的常见问题、问题和故障排除想法

- [为什么我的 Kinesis 数据流返回 500 个内部服务器错误？](#)
- [如何解决从 Flink 写入 Kinesis Data Streams 时的超时错误？](#)
- [如何解决 Kinesis Data Streams 中的限制错误？](#)
- [为什么我的 Kinesis 数据流限制？](#)
- [如何使用 KPL 将数据记录放入 Kinesis 数据流？](#)

针对 Kinesis Data Streams 的高级主题

本节讨论如何优化您的 Amazon Kinesis Data Streams 生产者。

主题

- [KPL 重试和速率限制 \(p. 103\)](#)
- [使用 KPL 聚合时的注意事项 \(p. 103\)](#)

KPL 重试和速率限制

当您使用 KPL 添加 Kinesis Producer Library (KPL) 用户记录时 `addUserRecord()` 操作时，将指定一个时间戳并将记录添加到缓冲区，其截止日期由 `RecordMaxBufferedTime` 配置参数。此时间戳/截止日期组合将设置缓冲区优先级。记录基于以下标准从缓冲区进行刷新：

- 缓冲区优先级
- 聚合配置
- 集合配置

影响缓冲区行为的聚合和集合配置参数如下所示：

- `AggregationMaxCount`
- `AggregationMaxSize`
- `CollectionMaxCount`
- `CollectionMaxSize`

然后，刷新的记录将作为 Amazon Kinesis 数据流记录使用 Kinesis Data Streams API 操作的调用来记录发送到您的 Kinesis 数据流 `PutRecords`。 `PutRecords` 操作将请求发送到有时完全失败或部分失败的流。失败的记录将自动添加回 KPL 缓冲区。基于这两个值中的较小值设置新的截止日期：

- 将当前 `RecordMaxBufferedTime` 配置分为两半
- 记录的 `time-to-live` 值

此策略允许将重试的 KPL 用户记录包含在后续的 Kinesis Data Streams API 调用中，从而在强制实施 Kinesis Data Streams 记录时提高吞吐量并降低复杂性。 `time-to-live` 值。不存在退避算法，这使得该策略成为相对积极的重试策略。由于重试次数过多而造成的垃圾邮件会受到速率限制的阻止，该内容将在下一部分中进行讨论。

速率限制

KPL 包括速率限制功能，该功能可限制从单个创建者发送的每个分片的吞吐量。使用令牌桶算法以及同时适用于 Kinesis Data Streams 记录和字节的单独存储桶来实施速率限制。每当对 Kinesis Data Streams 进行成功写入时都会将一个令牌（或多个令牌）添加到每个存储桶，直至达到特定阈值。此阈值是可配置的，但默认情况下设定的值将比实际分片限制高出 50%，这将允许单个创建器中的分片饱和。

您可降低此限制来减少因重试次数过多而造成的垃圾邮件。然而，最佳实践是每个创建器主动重试最大吞吐量，通过扩展流的容量并实施相应的分区键策略来处理已确定为过多的任何结果限制。

使用 KPL 聚合时的注意事项

尽管生成的 Amazon Kinesis Data Streams 记录的序列号方案保持不变，但是聚合的 Kinesis Producer Library (KPL) 用户记录的索引将从 0（零）开始。为了唯一标识您的 KPL 用户记录的数字，您的代码可以

忽略这一点，因为（将您的 KPL 用户记录的聚合到 Kinesis Data Streams）和随后的解合（将 Kinesis Data Streams 记录到您的 KPL 用户记录中）自动处理此问题。无论使用者使用的是 KCL 还是 Amazon 发工具包。要使用此聚合功能，您需要将 KPL 的 Java 部分拉入您的版本中（如果您的使用者是使用在 Amazon 发工具包）。

如果您打算将序列号用作 KPL 用户记录的唯一标识符，我们建议您使用遵守合约的 `public int hashCode()` 和 `public boolean equals(Object obj)` 中提供的操作 `Record` 和 `UserRecord` 以便对你的 KPL 用户记录进行比较。此外，如果您想要检查 KPL 用户记录的子序列号，则可将其转换为 `UserRecord` 实例并检索其子序列号。

有关更多信息，请参阅 [使用者取消聚合 \(p. 83\)](#)。

从 Amazon Kinesis Data Streams

使用者是一种处理 Kinesis 数据流中的所有数据的应用程序。当使用者使用增强型扇出功能时，它会获取其自己的 2 MB/秒的读取吞吐量配额，从而允许多个使用者并行读取相同流中的数据，而不必与其他使用者争用读取吞吐量。要使用分片的增强型扇出功能，请参阅[开发具有专用吞吐量的自定义使用者（增强扇出功能）](#) (p. 143)。

默认情况下，流中的每个分片均提供 2 MB/秒的读取吞吐量。此吞吐量跨正在从某给定分片进行读取的所有使用器获取分片。换言之，每个分片的默认 2 MB/秒的吞吐量是固定的，即使有多个使用者正在从分片中进行读取。要使用分片的此默认吞吐量，请参阅[开发具有共享吞吐量的自定义使用者](#) (p. 116)。

下表将默认吞吐量与增强型扇出功能进行比较。消息传播延迟定义为使用负载分配 API（如 `PutRecord` 和 `PutRecords`）发送的有效负载通过消耗负载的 API（如 `GetRecords` 和 `SubscribeToShard`）到达消费者应用程序所花费的时间（以毫秒为单位 `SubscribeToShard`）。GetRecords

特性	没有增强型扇出功能的未注册使用者	具有增强型扇出功能的注册使用者
分片读取吞吐量	每个分片的 2MB/秒总吞吐量是固定的。如果有多个使用器正在从同一分片进行读取，则它们将全部共享此吞吐量。它们从分片中接收的吞吐量总和不会超出 2 MB/秒。	随着使用器注册进行扩展以使用增强型扇出功能。注册为使用增强型扇出功能的每个使用者均接收其自己的每个分片的读取吞吐量，最多 2MB/秒，独立于其他使用者。
消息传播延迟	平均约 200 毫秒（如果您有一个从流中读取的使用者）。如果您有五个使用者，则这个平均值高达约 1000 毫秒。	通常情况下，平均为 70 毫秒，无论您是拥有一个使用者，还是五个使用者。
费用	不适用	存在数据检索费用和使用器分片小时费用。有关更多信息，请参阅 Amazon Kinesis 数据流定价 。
记录传输模型	使用通过 HTTP 提取模型 GetRecords 。	Kinesis Data Streams 使用 HTTP/2 将记录推送给你 SubscribeToShard 。

主题

- [在 Kinesis 控制台中使用数据查看器](#) (p. 106)
- [使用 Amazon Lambda 开发使用器](#) (p. 106)
- [使用 AAmazon Kinesis Data Analytics](#) (p. 106)
- [使用 Amazon Kinesis Data Firehose](#) (p. 106)
- [使用 Kinesis 客户端库](#) (p. 107)
- [开发具有共享吞吐量的自定义使用者](#) (p. 116)
- [开发具有专用吞吐量的自定义使用者（增强扇出功能）](#) (p. 143)
- [将使用者从 KCL 1.x 迁移到 KCL 2.x](#) (p. 150)
- [使用其他 Amazon 服务](#) (p. 158)
- [使用第三方集成](#) (p. 159)
- [排查 Kinesis Data Streams 使用者的问题](#) (p. 160)
- [Amazon Kinesis Data Streams 使用者的高级主题](#) (p. 163)

在 Kinesis 控制台中使用数据查看器

Kinesis 管理控制台中的数据查看器使您无需开发消费者应用程序，即可查看数据流的指定分片内的数据记录。若要使用 Data Viewer，请按照下列步骤操作：

1. 登录到 Amazon Web Services Management Console，然后通过以下网址打开 Kinesis 控制台：<https://console.aws.amazon.com/kinesisvideo/home>。
2. 选择要使用数据查看器查看其记录的活动数据流，然后选择数据查看器选项卡。
3. 在所选活动数据流的数据查看器选项卡中，选择要查看其记录的分片，选择起始位置，然后单击获取记录。您可以为以下值之一：
 - 在序列号处：显示从序列号字段中指定的序列号表示的位置开始的记录。
 - 序列号之后：在序列号字段中指定的序列号所表示的位置之后显示记录。
 - 在时间戳处：显示时间戳字段中指定的时间戳所表示的位置的记录。
 - 修剪地平线：显示分区中最后一条未修剪记录的记录，这是分片中最古老的数据记录。
 - 最新：在分区的最新记录之后显示记录，这样您就可以始终读取分片中的最新数据。

然后，生成的与指定分片 ID 和起始位置相匹配的数据记录将显示在控制台的记录表中。一次最多显示 50 条记录。要查看下一组记录，请单击“下一步”按钮。

4. 单击任何单个记录，在单独的窗口中以原始数据或 JSON 格式查看该记录的有效负载。

请注意，当您单击 Data Viewer 中的“获取记录”或“下一步”按钮时，这会调用 GetRecordsAPI，这适用于每秒 5 笔交易的 GetRecordsAPI 限制。

使用 Amazon Lambda 开发使用器

您可以使用 Amazon Lambda 函数来处理数据流中的记录。Amazon Lambda 是一项计算服务，可让您无需预配置或管理服务器即可运行代码。它只在需要时才执行您的代码并自动缩放，从每天几个请求到每秒数千个请求。您只需按使用的计算时间付费。代码不运行时不会产生任何费用。借助 Amazon Lambda，您几乎可以为任何类型的应用程序或后端服务运行代码，并且不必进行任何管理。它在可用性高的计算基础设施上运行您的代码，执行计算资源的所有管理工作，其中包括服务器和操作系统维护、容量预置和自动扩展、代码监控和记录。有关更多信息，请参阅[Amazon Lambda 与 Amazon Kinesis 使用](#)。

有关故障排除信息，请参阅[为什么 Kinesis Data Streams 触发器无法调用我的 Lambda 函数？](#)

使用 Amazon Kinesis Data Analytics

您可以使用 Amazon Kinesis Data Analytics 应用程序使用 SQL、Java 或 Scala 处理和分析 Kinesis 流中的数据。Kinesis Data Analytics 应用程序可以使用参考源丰富数据、随着时间的推移聚合数据，或者使用机器学习来发现数据异常。然后，您可以将分析结果写入另一个 Kinesis 流、Kinesis Data Firehose 传输流或 Lambda 函数。如需更多信息，请参阅适用于 [SQL 应用程序的 Kinesis Data Analytics 开发人员指南](#) 或 [Flink 应用程序的 Kinesis Data Analytics 开发人员指南](#)。

使用 Amazon Kinesis Data Firehose

您可以使用 Kinesis Data Firehose 来读取和处理 Kinesis 直播中的记录。Kinesis Data Firehose 是一项完全托管的服务，用于将实时流数据传输到 Amazon S3、Amazon Redshift、Amazon OpenSearch Services 和 Splunk 等目标。Kinesis Data Firehose 还支持由受支持的第三方服务提供商（包括 Datadog、MongoDB 和

New Relic) 拥有的任何一个或多个自定义 HTTP 端点。您还可以将 Kinesis Data Firehose 配置为在将数据传输到目的地之前转换数据记录并转换记录格式。有关更多信息, 请参阅[使用 Kinesis Data Streams 写入 Kinesis Data Firehose](#)。

使用 Kinesis 客户端库

开发可以处理来自 KDS 数据流的数据的自定义消费者应用程序的方法之一是使用 Kinesis 客户端库 (KCL)。

主题

- [Kinesis 客户端库是什么? \(p. 107\)](#)
- [KCL 可用的版本 \(p. 108\)](#)
- [KCL 概念 \(p. 108\)](#)
- [使用租赁表跟踪 KCL 消费者应用程序处理的碎片 \(p. 109\)](#)
- [针对 Java 消费者应用程序使用相同的 KCL 2.x 处理多个数据流 \(p. 114\)](#)
- [将 Kinesis 客户端库与 AmazonGlue 架构注册 \(p. 116\)](#)

Note

对于 KCL 1.x 和 KCL 2.x, 建议您根据使用情况升级到最新的 KCL 1.x 版本或 KCL 2.x 版本。KCL 1.x 和 KCL 2.x 都会定期更新新版本, 其中包括最新的依赖项和安全补丁、错误修复以及向后兼容的新功能。有关更多信息, 请参阅 <https://github.com/awslabs/amazon-kinesis-client/> 发布。

Kinesis 客户端库是什么?

KCL 通过处理与分布式计算相关的许多复杂任务来帮助您使用和处理来自 Kinesis 数据流的数据。其中包括跨多个使用者应用程序实例执行负载均衡、响应使用者应用程序实例故障、对已处理记录执行检查点操作以及对重新分片 KCL 负责所有这些子任务, 以便您可以专注于编写自定义记录处理逻辑。

KCL 与 Kinesis Data Streams API 不同, 后者可以在 Amazon 开发工具包。Kinesis Data Streams API 可帮助您管理 Kinesis Data Streams 的许多方面, 包括创建流、重新分片以及放置和获取记录。KCL 围绕所有这些子任务提供了一层抽象, 特别是为了让您专注于消费者应用程序的自定义数据处理逻辑。有关 Kinesis Data Streams API 的信息, 请参阅[Amazon Kinesis API 参考](#)。

Important

KCL 是一个 Java 库。使用名为 MultiLang 守护程序。此守护程序是基于 Java 的, 当您使用除 Java 之外的 KCL 语言时, 会在后台运行。例如, 如果您安装了适用于 Python 的 KCL 并完全在 Python 中编写使用者应用程序, 则您仍需要在您的系统中安装 Java, 因此 MultiLang 守护程序。此外, MultiLangDaemon 有一些默认设置, 您可能需要根据自己的使用案例自定义这些设置, 例如 Amazon 它连接到的区域。有关的更多信息 MultiLang 上守护程序 GitHub 请参阅[KCL MultiLang 守护程序](#)。

KCL 充当您的记录处理逻辑和 Kinesis Data Streams 之间的中介。KCL 执行以下任务:

- 连接到数据流
- 枚举数据流中的分片
- 使用租约与其工作人员协调分片关联
- 为其管理的每个分片实例化记录处理器
- 从数据流中提取数据记录
- 将记录推送到对应的记录处理器
- 对已处理记录进行检查点操作

- 当工作线程实例计数发生变化或重新分片数据流时（分片被拆分或合并）时，平衡分片工作人员关联（租赁）

KCL 可用的版本

目前，您可以使用以下支持版本之一来构建自定义使用者应用程序：

- KCL 1.x
有关更多信息，请参阅 [开发 KCL 1.x 使用者 \(p. 117\)](#)
- KCL 2.x
有关更多信息，请参阅 [开发 KCL 2.x 使用者 \(p. 128\)](#)

您可以使用 KCL 1.x 或 KCL 2.x 来构建使用共享吞吐量的消费者应用程序。有关更多信息，请参阅 [使用 KCL 开发具有共享吞吐量的自定义使用者 \(p. 117\)](#)。

要构建使用专用吞吐量（增强的扇出消费者）的消费者应用程序，您只能使用 KCL 2.x。有关更多信息，请参阅 [开发具有专用吞吐量的自定义使用者（增强扇出功能） \(p. 143\)](#)。

有关 KCL 1.x 和 KCL 2.x 之间的区别的信息，以及有关如何从 KCL 1.x 迁移到 KCL 2.x 的说明，请参阅[将使用者从 KCL 1.x 迁移到 KCL 2.x \(p. 150\)](#)。

KCL 概念

- KCL 使用者应用— 使用 KCL 自定义构建并旨在读取和处理数据流中记录的应用程序。
- 消费者应用实例-KCL 消费者应用程序通常是分布式的，一个或多个应用程序实例同时运行，以便协调故障并动态负载均衡数据记录处理。
- 工作线程— KCL 消费者应用程序实例用于开始处理数据的高级别类。

Important

每个 KCL 消费者应用程序实例都有一个 Worker。

工作人员初始化并监督各种任务，包括同步分片和租赁信息、跟踪分片分配以及处理分片中的数据。工作程序为 KCL 提供了使用者应用程序的配置信息，例如将记录此 KCL 使用者应用程序要处理的数据流的名词以及 Amazon 访问此数据流所需的凭据。工作人员还启动该特定的 KCL 消费者应用程序实例，将数据记录从数据流传送到记录处理器。

Important

在 KCL 1.x 中，这个课被称为工作线程。有关更多信息（这些是 Java KCL 存储库），请参阅<https://github.com/aws-labs/amazon-kinesis-client/blob/v1.x/SRC/Main/java/com/亚马逊/服务/Kinesis/客户端库/Lib/工作人员/工作人.Java>。在 KCL 2.x 中，这个课被称为计划程序。KCL 2.x 中的调度程序的用途与 KCL 1.x 中的工作人员的目的相同。有关 KCL 2.x 中的调度器类的更多信息，请参阅<https://github.com/aws-labs/amazon-kinesis-client/blob/master/amazon-kinesis-client/src/main/java/software/amazon/kinesis/coordinator/Scheduler.java>。

- 租赁— 定义工作器和分片之间绑定的数据。分布式 KCL 消费者应用程序使用租赁将数据记录处理跨工作人员队列进行分区。在任何给定时间，每个数据记录分片都由 leaseKey 变量。

默认情况下，工作人员可以持有一个或多个租约（取决于 maxLeasesFor 工作线程同时提交。

Important

每个工作人员都会争夺持有数据流中所有可用分片的所有可用租约。但是在任何时候只有一名员工可以成功持有每份租约。

例如，如果您有一个具有工作器 A 的消费者应用程序实例 A 正在处理包含 4 个分片的数据流，则工作器 A 可以同时保存分片 1、2、3 和 4 的租约。但是如果你有两个消费者应用程序实例：A 和 B 使用工作器 A 和工作器 B，并且这些实例正在处理包含 4 个分片、工作器 A 和工作器 B 的数据流不能同时持有分片 1 的租约。一名工作人员持有特定分片的租约，直到准备停止处理此分片的数据记录或失败为止。当一名工作人员停止持有租约时，另一名员工接管并持有租约。

有关更多信息（这些是 Java KCL 存储库），请参阅<https://github.com/awslabs/amazon-kinesis-client/blob/v1.x/SRC/main/java/com/亚马逊/服务/Kinesis/租赁/impl/租赁.java>对于 KCL 1.x 和<https://github.com/awslabs/amazon-kinesis-client/blob/master/amazon-kinesis-client/src/main/java/software/amazon/kinesis/leases/Lease.java>对于 KCL 2.x。

- 租赁表—一个独特的 Amazon DynamoDB 表，用于跟踪 KCL 消费者应用程序工作人员租用和处理的 KDS 数据流中的分片。在运行 KCL 使用者应用程序时，租赁表必须与数据流中的最新分片信息保持同步（在工作程序内部和所有工作线程之间）。有关更多信息，请参阅 [使用租赁表跟踪 KCL 消费者应用程序处理的碎片](#) (p. 109)。
- 录制处理器—定义 KCL 消费者应用程序如何处理从数据流中获取的数据的逻辑。在运行时，KCL 消费者应用程序实例化一个工作器，此工作程序为其持有租约的每个分片实例化一个记录处理器。

使用租赁表跟踪 KCL 消费者应用程序处理的碎片

主题

- [什么是租赁表](#) (p. 109)
- [吞吐量](#) (p. 110)
- [租赁表如何与 KDS 数据流中的分片同步](#) (p. 110)

什么是租赁表

对于每个 Amazon Kinesis Data Streams 应用程序，KCL 都使用唯一的租赁表（存储在 Amazon DynamoDB 表中）来跟踪 KDS 数据流中由 KCL 消费者应用程序的工作人员租用和处理的碎片。

Important

KCL 使用者应用程序的名称来创建此使用者应用程序使用的租赁表的名称，因此每个使用者应用程序名称都必须是唯一的。

您可以使用[Amazon DynamoDB 控制台](#)在消费者应用程序运行时。

如果您的 KCL 使用者应用程序的租赁表在应用程序启动时不存在，则其中一个工作程序会为此应用程序创建租赁表。

Important

您的账户将被收取与 DynamoDB 表关联的费用（除开与 Kinesis Data Streams）本身关联的费用。

租赁表中的每行表示使用者应用程序的工作程序正在处理的碎片。如果你的 KCL 消费者应用程序只处理一个数据流，那么 leaseKey 这是租赁表的哈希键是分片 ID。如果您是[针对 Java 消费者应用程序使用相同的 KCL 2.x 处理多个数据流](#) (p. 114)，那么 leaseKey 的结构看起来像这样：account-id:StreamName:streamCreationTimestamp:ShardId。例如，111111111:multiStreamTest-1:12345:shardId-000000000336。

除了分片 ID 以外，每行还包含以下数据：

- 检查点：分片的最新检查点序号。此值在数据流中的所有分片中都是唯一的。

- `checkpointSubSequence`数量：使用 Kinesis Producer Library 的聚合功能时，这是一个扩展检查点它跟踪 Kinesis 记录中的单个用户记录。
- `leaseCounter`：用于租赁版本控制，以便工作程序可以检测其租赁已由其他工作程序获取。
- `leaseKey`：租赁的唯一标识符。每个租赁特定于数据流中的一个分片，一次由一个工作程序持有。
- `leaseOwner`：持有此租赁的工作程序。
- `ownerSwitchesSince`检查点：自上次写入检查点以来，此租赁更改了工作程序的次数。
- `parentShardId`：用于确保在开始子分片的处理之前已经完全处理了父分片。这可以确保记录按照放入流中的相同顺序处理。
- `hashrange`：由 `PeriodicShardSyncManager` 运行定期同步以查找租赁表中缺少的分片，并根据需要为它们创建租约。

Note

这些数据出现在从 KCL 1.14 和 KCL 2.3 开始的每个分片的租赁表中。有关 [更多信息](#) `PeriodicShardSyncManager` 以及租赁和分片之间的定期同步，请参阅 [租赁表如何与 KDS 数据流中的分片同步 \(p. 110\)](#)。

- 孩子碎片：由 `LeaseCleanupManager` 以查看子分片的处理状态，并决定是否可以从租赁表中删除父分片。

Note

这些数据出现在从 KCL 1.14 和 KCL 2.3 开始的每个分片的租赁表中。

- `ShardID`：分片的 ID。

Note

这些数据仅出现在租赁表中，如果你是 [针对 Java 消费者应用程序使用相同的 KCL 2.x 处理多个数据流 \(p. 114\)](#)。这仅在 KCL 2.x 中对于 Java 提供支持，从 KCL 2.3 开始，对于 Java 及更高版本。

- `stream name`数据流的标识符采用以下格式：`account-id:StreamName:streamCreationTimestamp`。

Note

这些数据仅出现在租赁表中，如果你是 [针对 Java 消费者应用程序使用相同的 KCL 2.x 处理多个数据流 \(p. 114\)](#)。这仅在 KCL 2.x 中对于 Java 提供支持，从 KCL 2.3 开始，对于 Java 及更高版本。

吞吐量

如果您的 Amazon Kinesis Data Streams 应用程序收到了预置吞吐量异常，您应增加 DynamoDB 表的预置吞吐量。KCL 会创建预置吞吐量为 10 次读取/秒和 10 次写入/秒的表，但这对于您的应用程序可能不够。例如，如果您的 Amazon Kinesis Data Streams 应用程序频繁执行检查点操作或对由许多分片组成的流执行操作，您可能需要更多吞吐量。

有关 DynamoDB 中预配置吞吐量的信息，请参阅 [读/写容量模式](#) 和 [处理表和数据](#) 中的 Amazon DynamoDB 开发人员指南。

租赁表如何与 KDS 数据流中的分片同步

KCL 消费者应用程序中的工作人员使用租赁处理来自给定数据流的分片。有关哪些员工在任何给定时间租赁什么分片的信息存储在租赁表中。在 KCL 使用者应用程序运行期间，租用表必须与数据流中的最新分片信息保持同步。KCL 将租赁表与从 Kinesis Data Streams 服务获取的分片信息同步在使用者应用程序引导期间（无论是在初始化或重新启动使用者应用程序时）以及在处理的分片到达结束时（重新分片）。换句话说，工作人员或 KCL 使用者应用程序应用程序将与他们在初始使用者应用程序引导期间以及每当使用者应用程序遇到数据流重新分片事件时所处理的数据流同步。

主题

- [KCL 1.0-1.13 和 KCL 2.0-2.2 中的同步 \(p. 111\)](#)
- [在 KCL 2.x 中进行同步，从 KCL 2.3 开始及以后 \(p. 111\)](#)
- [KCL 1.x 中的同步，从 KCL 1.14 及以后开始 \(p. 112\)](#)

KCL 1.0-1.13 和 KCL 2.0-2.2 中的同步

在 KCL 1.0-1.13 和 KCL 2.0-2.2 中，在消费者应用程序的引导期间以及在每个数 Kinesis Data Streams 重新分片事件期间，KCL 通过调用 `ListShards` 或者 `DescribeStream` 发现 API。在上面列出的所有 KCL 版本中，KCL 使用者应用程序的每个工作程序都完成以下步骤，以便在消费者应用程序的引导期间和每个流重新分片事件中执行租赁/分片同步过程：

- 获取正在处理的流的数据的所有分片
- 从租赁表中获取所有分片租约
- 筛选出租赁表中没有租约的每个打开的分片
- 迭代所有找到的打开分片以及每个没有打开父级的打开分片：
 - 遍历层次结构树的祖先路径，以确定分片是否为后代。如果正在处理祖先分片（租赁表中存在祖先分片的租赁条目），或者是否应该处理祖先分片（例如，如果初始位置为 `TRIM_HORIZON` 要么 `AT_TIMESTAMP`）
 - 如果上下文中的开放分片是后代，KCL 会根据初始位置检查分片并可能需要为其父级创建租约

在 KCL 2.x 中进行同步，从 KCL 2.3 开始及以后

从 KCL 2.x (KCL 2.3) 及更高版本开始，库现在支持对同步过程进行以下更改。这些租赁/分片同步更改显著减少了 KCL 消费者应用程序对 Kinesis Data Streams 服务发出的 API 调用数量，并优化了 KCL 消费者应用程序中的租赁管理。

- 在应用程序引导过程中，如果租赁表为空，KCL 会使用 `ListShardAPI` 的过滤选项（`ShardFilter` 可选请求参数），用于仅针对在指定的时间打开的分片的快照检索和创建租赁 `ShardFilter` 参数。这些区域有：`ShardFilter` 参数使您能够过滤掉 `ListShardsAPI`。唯一必需的财产 `ShardFilter` 参数为 `Type`。KCL 使用 `Typefilter` 属性及其有效值的以下有效值，用于识别并返回可能需要新租用的打开分片的快照：
 - `AT_TRIM_HORIZON`-回复包括所有打开的碎片 `TRIM_HORIZON`。
 - `AT_LATEST`-响应仅包括当前打开的数据流分片。
 - `AT_TIMESTAMP`-响应包括开始时间戳小于或等于给定时间戳且结束时间戳大于或等于给定时间戳或仍然打开的所有分片。

`ShardFilter` 用于为空租赁表创建租赁，以便为指定的分片快照初始化租赁 `RetrievalConfig#initialPositionInStreamExtended`。

有关 `ShardFilter` 的更多信息，请参阅 https://docs.amazonaws.cn/kinesis/latest/APIReference/API_ShardFilter.html。

- 由一名当选的工作人员负责人执行租赁/分片同步以使租赁表与数据流中的最新分片保持最新状态，而不是执行租赁/分片同步的所有工作人员，而是执行租赁/分片同步。
- KCL 2.3 使用 `ChildShards` 返回的参数 `GetRecords` 和 `SubscribeToShard` 用于执行租赁/分片同步的 `API_SHARD_END` 对于封闭的分片，允许 KCL 工作人员只能为已完成处理的分片的子分片创建租约。对于在整个消费者应用程序中共享，这种租赁/分片同步的优化使用 `ChildShards` 的参数 `GetRecordsAPI`。对于专用吞吐量（增强的扇出）消费者应用程序，这种租赁/分片同步的优化使用 `ChildShards` 的参数 `SubscribeToShardAPI`。有关更多信息，请参阅 [GetRecords](#)、[SubscribeTo分片](#)，和 [ChildShard](#)。
- 随着上述变化，KCL 的行为正从所有员工了解所有现有碎片的模式转变为工人只学习每个员工拥有的碎片的子级碎片的模式。因此，除了在消费者应用程序引导和重新分片事件期间发生的同步之外，KCL 现在还在执行额外的定期分片/租赁扫描，以便识别租赁表中的任何潜在漏洞（换句话说，了解所有新分片），以确

保完整正在处理数据流的哈希范围，并根据需要为它们创建租赁。PeriodicShardSyncManager是负责运行定期租赁/分片扫描的组件。

有关 的更多信息PeriodicShardSyncManager在 KCL 2.3 中，请参阅<https://github.com/awslabs/amazon-kinesis-client/blob/master/amazon-kinesis-client/src/main/java/软件/亚马逊/kinesis/租赁/LeaseManagementConfig.java#L201-L213>。

在 KCL 2.3 中，可以配置新的配置选项PeriodicShardSyncManager在LeaseManagementConfig：

名称	默认值	说明
leasesRecoveryAuditorExecutionFrequency	120000 (2分钟)	在租赁表中扫描部分租约的审计员作业的频率（以毫秒为单位）。如果审计员检测到流的租约中有任何漏洞，那么它将根据以下方式触发分片同步leasesRecoveryAuditorIn
leasesRecoveryAuditorInconsistencyConfidenceThreshold	3	定期审计员作业的可信度阈值，用于确定租赁表中数据流的租约是否不一致。如果审计员多次为数据流连续发现同一组不一致之处，那么它将触发分片同步。

NewCloudWatch现在还可以发出指标来监控PeriodicShardSyncManager. 有关更多信息，请参阅 [周期 SHARDSYNC 管理器 \(p. 186\)](#)。

- 包括优化HierarchicalShardSyncer只为一层碎片创建租约。

KCL 1.x 中的同步，从 KCL 1.14 及以后开始

从 KCL 1.x (KCL 1.14) 及更高版本开始，库现在支持对同步过程进行以下更改。这些租赁/分片同步更改显著减少了 KCL 消费者应用程序对 Kinesis Data Streams 服务发出的 API 调用数量，并优化了 KCL 消费者应用程序中的租赁管理。

- 在应用程序引导过程中，如果租赁表为空，KCL 会使用ListShardAPI 的过滤选项（ShardFilter可选请求参数），用于仅针对在指定的时间打开的分片的快照检索和创建租赁ShardFilter参数。这些区域有：ShardFilter参数使您能够过滤掉ListShardsAPI。唯一必需的财产ShardFilter参数为Type。KCL 使用Typefilter 属性及其有效值的以下有效值，用于识别并返回可能需要新租用的打开分片的快照：
 - AT_TRIM_HORIZON-回复包括所有打开的碎片TRIM_HORIZON.
 - AT_LATEST-响应仅包括当前打开的数据流分片。
 - AT_TIMESTAMP-响应包括开始时间戳小于或等于给定时间戳且结束时间戳大于或等于给定时间戳或仍然打开的所有分片。

ShardFilter用于为空租赁表创建租赁，以便为指定的分片快照初始化租赁KinesisClientLibConfiguration#initialPositionInStreamExtended.

有关 ShardFilter 的更多信息，请参阅 https://docs.amazonaws.cn/kinesis/latest/APIReference/API_ShardFilter.html。

- 由一名当选的工作人员负责人执行租赁/分片同步以使租赁表与数据流中的最新分片保持最新状态，而不是执行租赁/分片同步的所有工作人员，而是执行租赁/分片同步。
- KCL 1.14 使用ChildShards返回的参数GetRecords和SubscribeToShard用于执行租赁/分片同步的 APISHARD_END对于封闭的分片，允许 KCL 工作人员只能为已完成处理的分片的子分片创建租约。有关更多信息，请参阅 [GetRecords](#)和[ChildShard](#)。
- 随着上述变化，KCL 的行为正从所有员工了解所有现有碎片的模式转变为工人只学习每个员工拥有的碎片的子级碎片的模式。因此，除了在消费者应用程序引导和重新分片事件期间发生的同步之外，KCL 现在还执行额外的定期分片/租赁扫描，以便识别租赁表中的任何潜在漏洞（换句话说，了解所有新分片），以确保完整正在处理数据流的哈希范围，并根据需要为它们创建租赁。PeriodicShardSyncManager是负责运行定期租赁/分片扫描的组件。

何时KinesisClientLibConfiguration#shardSyncStrategyType设置为ShardSyncStrategyType.SHARD_END、PeriodicShardSync leasesRecoveryAuditorInconsistencyConfidenceThreshold用于确定租赁表中包含漏洞的连续扫描次数的阈值，之后要强制执行分片同步。何时KinesisClientLibConfiguration#shardSyncStrategyType设置为ShardSyncStrategyType.PERIODIC、leasesRecoveryAuditorInconsistencyConfidenceThreshold已忽略。

有关的更多信息PeriodicShardSyncManager在 KCL 1.14 中，请参阅<https://github.com/aws/aws-kinesis-client-java/blob/v1.x/src/main/java/com/amazonaws/services/kinesis/client/lib/PeriodicShardSyncManager.java#L987-L999>。

在 KCL 1.14 中，新的配置选项可用于配置PeriodicShardSyncManager在LeaseManagementConfig：

名称	默认值	说明
leasesRecoveryAuditorInconsistencyConfidenceThreshold	0.1	定期审计员作业的可信度阈值，用于确定租赁表中数据流的租约是否不一致。如果审计员多次为数据流连续发现同一组不一致之处，那么它将触发分片同步。

NewCloudWatch现在还可以发出指标来监控PeriodicShardSyncManager。有关更多信息，请参阅 [周期 SHARDSYNC 管理器 \(p. 186\)](#)。

- KCL 1.14 现在还支持延期租约清理。通过以下方式异步删除租赁LeaseCleanupManager到达后SHARD_END，当分片已超过数据流的保留期之后过期或者因重新分片操作而被关闭时。

新的配置选项如下：LeaseCleanupManager。

名称	默认值	说明
leaseCleanupIntervalMillis	1分钟	运行租赁清理线程的时间间隔。
completedLeaseCleanupIntervalMillis	5分钟	检查租赁是否完成的时间间隔。
garbageLeaseCleanupIntervalMillis	30分钟	检查租约是否为垃圾的时间间隔（即

名称	默认值	说明
		已超过数据流的保留期限) 与否。

- 包括优化KinesisShardSyncer只为一层碎片创建租约。

针对 Java 消费者应用程序使用相同的 KCL 2.x 处理多个数据流

本节介绍了 KCL 2.x for Java 中的以下更改，这些更改使您能够创建可以同时处理多个数据流的 KCL 使用者应用程序。

Important

只有针对 Java 的 KCL 2.x 支持多流处理，对于 Java 及更高版本，从 KCL 2.3 开始。
任何其他可以实现 KCL 2.x 的语言都不支持多流处理。
KCL 1.x 的任何版本都不支持多流处理。

- MultistreamTracker 界面

要构建可以同时处理多个流的消费者应用程序，必须实现一个名为 [MultistreamTracker](#)。此界面包括 `streamConfigList` 方法，返回要由 KCL 使用者应用程序处理的数据流及其配置的列表。请注意，正在处理的数据流可以在使用者应用程序运行时期进行更改。`streamConfigList` 定期调用，以了解要处理的数据流的变化。

这些区域有：`streamConfigList` 填充 [StreamConfig](#) 列表。

```
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;
}
```

请注意，`StreamIdentifier` 和 `InitialPositionInStreamExtended` 是必填字段，而 `consumerArn` 为可选项。您必须提供 `consumerArn` 仅当您使用 KCL 2.x 来实施增强型扇出消费者应用程序时才能使用。

有关的更多信息 `StreamIdentifier` 请参阅 <https://github.com/aws-labs/amazon-kinesis-client/blob/0c5042dadf794fe988438436252a5a8fe70b6b0b/amazon-kinesis-client/src/main/java/软件/亚马逊/kinesis/常见/StreamIdentifier.java#L29>。您可以为 `StreamIdentifier` 来自序列化的流标识符。序列化的流标识符应采用以下格式：`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还包括在租赁表中删除旧流的租赁的策略

(formerStreamsLeasesDeletionStrategy)。请注意，在消费者应用程序运行时期无法更改策略。有关更多信息，请参阅。<https://github.com/aws-labs/amazon-kinesis-client/blob/0c5042dadf794fe988438436252a5a8fe70b6b0b/amazon-kinesis-client/src/main/java/软件/亚马逊/kinesis/处理器/FormerStreamsLeasesDeletionStrategy.java>

- [ConfigsBuilder](#)是一个应用程序范围的类，您可以用它来指定在构建 KCL 消费者应用程序时要使用的所有 KCL 2.x 配置设置。ConfigsBuilder课程现在支持MultistreamTracker接口。你可以初始化ConfigsBuilder要么是使用要从中消耗记录的一个数据流的名称：

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

或者你可以初始化ConfigsBuilder和MultiStreamTracker如果你想实现一个同时处理多个流的 KCL 消费者应用程序。

```
* 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;
}
```

- 随着为 KCL 消费者应用程序实施了多流支持，应用程序的租赁表中的每一行现在都包含该应用程序处理的多个数据流的分片 ID 和流名称。
- 当实施对 KCL 消费者应用程序的多流支持时，leaseKey 将采用以下结构：account-id:StreamName:streamCreationTimestamp:ShardId. 例如，111111111:multiStreamTest-1:12345:shardId-000000000336.

Important

当您现有的 KCL 消费者应用程序配置为仅处理一个数据流时，leaseKey（即租赁表的哈希键）是分片 ID。如果您重新配置此现有的 KCL 消费者应用程序以处理多个数据流，它会打破租赁表，因为对于多流支持，leaseKey 结构必须如下所示：account-id:StreamName:StreamCreationTimestamp:ShardId.

将 Kinesis 客户端库与AmazonGlue 架构注册

您可以将 Kinesis 数据流与AmazonGlue 模式注册表。这些区域有：AmazonGlue 架构注册表允许您集中发现、控制和演变架构，同时确保已注册架构持续验证生成的数据。架构定义了数据记录的结构和格式。架构是用于可靠数据发布、使用或存储的版本化规范。这些区域有：AmazonGlue 模式注册表使您能够改进end-to-end流媒体应用程序中的数据质量和数据治理。有关更多信息，请参阅。[AmazonGlue 架构注册](#). 设置此集成的方法之一是通过 Java 中的 KCL。

Important

目前，Kinesis Data StreamsAmazon只有使用在 Java 中实现的 KCL 2.3 消费者的 Kinesis 数据流才支持 Glue 模式注册表集成。不提供多语言支持。不支持 KCL 1.0 消费者。不支持 KCL 2.3 之前的 KCL 2.x 消费者。

有关如何使用 KCL 将 Kinesis Data Streams 与模式注册表集成的详细说明，请参阅中的“使用 KPL/KCL 库与数据交互”部分[使用案例：将 Amazon Kinesis Data Streams 与AmazonGlue 架构注册](#).

开发具有共享吞吐量的自定义使用者

如果您在从 Kinesis Data Streams 接收数据时不需要专用吞吐量，并且在 200 毫秒下不需要读取传播延迟，则可以构建使用器应用程序，如以下主题所述。

主题

- [使用 KCL 开发具有共享吞吐量的自定义使用者 \(p. 117\)](#)
- [使用 Amazon SDK for Java 开发具有共享吞吐量的自定义使用者 \(p. 139\)](#)

有关使用专用吞吐量构建可从 Kinesis 数据流接收记录的使用器的信息，请参阅。[开发具有专用吞吐量的自定义使用者（增强扇出功能） \(p. 143\)](#).

使用 KCL 开发具有共享吞吐量的自定义使用者

开发具有共享吞吐量的自定义消费者应用程序的方法之一是使用 Kinesis 客户端库 (KCL)。

主题

- [开发 KCL 1.x 使用者 \(p. 117\)](#)
- [开发 KCL 2.x 使用者 \(p. 128\)](#)

开发 KCL 1.x 使用者

可以使用 Kinesis 客户端库 (KCL) 为 Amazon Kinesis Data Streams 开发使用者应用程序。有关 KCL 的更多信息，请参阅[Kinesis 客户端库是什么？ \(p. 107\)](#)。

目录

- [在 Java 中开发 Kinesis 客户端库用户 \(p. 117\)](#)
- [在 Node.js 中开发 Kinesis Client Library 使用者 \(p. 121\)](#)
- [在 .NET 中开发 Kinesis Client Library 使用者 \(p. 124\)](#)
- [在 Python 中开发 Kinesis 客户端库用户 \(p. 126\)](#)
- [在 Ruby 中开发 Kinesis 客户端库用户 \(p. 128\)](#)

在 Java 中开发 Kinesis 客户端库用户

可以使用 Kinesis Client Library (KCL) 构建处理 Kinesis 数据流中数据的应用程序。Kinesis Client Library 有多种语言版本。本主题将讨论 Java。要查看 Javadoc 参考，请参阅[Amazon 班级的 Javadoc 主题 AmazonKinesis 客户端](#)。

从下载 Java KCL GitHub，转到[Kinesis 客户端库 \(Java\)](#)。要在 Apache Maven 上找到 Java KCL，请转至[KCL 搜索结果](#)页。从下载 Java KCL 消费者应用程序的示例代码 GitHub，转到[KCL for Java 示例项目](#)上的页面 GitHub。

该示例应用程序使用 [Apache Commons Logging](#)。可以使用 configure 文件中定义的静态 AmazonKinesisApplicationSample.java 方法更改日志记录配置。有关如何将 Apache Commons 日志记录与 Log4j 结合使用的更多信息和 AmazonJava 应用程序，请参阅[使用 Log4j 日志记录](#)中的 Amazon SDK for Java 开发人员指南。

在 Java 中实现 KCL 使用者应用程序时，您必须完成下列任务：

任务

- [实施 IRecordProcessor 方法 \(p. 117\)](#)
- [为 I 实现类工厂 RecordProcessor 接口 \(p. 119\)](#)
- [创建工作线程 \(p. 120\)](#)
- [修改配置属性 \(p. 120\)](#)
- [迁移到版本 2 的记录处理器接口 \(p. 121\)](#)

实施 IRecordProcessor 方法

KCL 目前支持两个版本 IRecordProcessor 接口：原始接口与 KCL 的第一个版本一起可用；而版本 2 从 KCL 1.5.0 版开始可用。这两个接口都完全受支持。您的选择取决于您的特定方案要求。要查看所有区别，请参阅您在本地构建的 Javadocs 或源代码。以下各节概述了开始使用的最低实施要求。

IRecordProcessor 版本

- [原始接口 \(版本 1\) \(p. 118\)](#)

- [更新后的接口 \(版本 2\) \(p. 119\)](#)

原始接口 (版本 1)

原始 IRecordProcessor 接口 (package com.amazonaws.services.kinesis.clientlibrary.interfaces) 公开了下列记录处理器方法，您的使用者必须实施这些方法。该示例提供了可用作起点的实现 (请参阅 AmazonKinesisApplicationSampleRecordProcessor.java)。

```
public void initialize(String shardId)
public void processRecords(List<Record> records, IRecordProcessorCheckpointier checkpointier)
public void shutdown(IRecordProcessorCheckpointier checkpointier, ShutdownReason reason)
```

initialize

KCL 称 initialize 方法在实例化记录处理器时，将特定分片 ID 作为参数传递。此记录处理器仅处理此分片，并且通常情况下反过来说也成立 (此分片仅由此记录处理器处理)。但是，您的使用者应该考虑数据记录可能会经过多次处理的情况。Kinesis Data Streams 有至少一次语义，即分片中的每个数据记录至少会由使用者中的工作程序处理一次。有关特定分片可能由多个工作程序进行处理的情况的更多信息，请参阅[重新分片、扩展和并行处理 \(p. 164\)](#)。

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

processRecords

KCL 称 processRecords 方法，并传递来自自由指定的分片的数据记录的列表。initialize(shardId) 方法。记录处理器根据使用者的语义处理这些记录中的数据。例如，工作程序可能执行数据转换，然后将结果存储在 Amazon Simple Storage Service (Amazon S3) 存储桶中。

```
public void processRecords(List<Record> records, IRecordProcessorCheckpointier
    checkpointier)
```

除了数据本身之外，记录还包含一个序号和一个分区键。工作程序可在处理数据时使用这些值。例如，工作线程可选择 S3 存储桶，并在其中根据分区键的值存储数据。Record 类公开了以下方法，这些方法提供对记录的数据、序列号和分区键的访问。

```
record.getData()
record.getSequenceNumber()
record.getPartitionKey()
```

在该示例中，私有方法 processRecordsWithRetries 具有显示工作程序如何访问记录的数据、序号和分区键的代码。

Kinesis Data Streams 需要记录处理器来跟踪已在分片中处理的记录。KCL 通过传递检查指针来为您执行此跟踪 (IRecordProcessorCheckpointier) 到 processRecords。记录处理器调用 checkpoint 方法以向 KCL 通知记录处理分片中的记录的进度。如果工作线程失败，KCL 将使用此信息在已知的上一个已处理记录处重新启动对分片的处理。

对于拆分或合并操作，在原始分片的处理器调用之前，KCL 不会开始处理新分片。checkpoint 表示原始分片的所有处理都已完成。

如果你没有传递参数，KCL 假定调用 checkpoint 表示所有记录都已处理，一直处理到传递到记录处理器的最后一个记录。因此，记录处理器只应在已处理传递到它的列表中的所有记录后才调用 checkpoint。记录处理器不需要在每次调用 checkpoint 时调用 processRecords。例如，处理器可以在每第三次调用 checkpoint 时调用 processRecords。您可以选择性地为某个记录的确切序号指定为 checkpoint 的参数。在本例中，KCL 将假定所有记录都已处理，一直到达该记录。

在该示例中，私有方法 `checkpoint` 展示了如何使用适当的异常处理和重试逻辑调用 `IRecordProcessorCheckpointInterface.checkpoint`。

KCL 依靠 `processRecords` 以处理由处理数据记录引起的任何异常。如果从中抛出异常 `processRecords`，KCL 将跳过在异常发生前已传递的数据记录。也就是说，这些记录不会重新发送到引发异常的记录处理器或使用者中的任何其他记录处理器。

`shutdown`

KCL 称 `shutdown` 方法无论是在处理结束时（关闭原因是 `TERMINATE`）或工作程序不再响应（关闭原因为 `ZOMBIE`）。

```
public void shutdown(IRecordProcessorCheckpointInterface checkpointer, ShutdownReason reason)
```

处理操作在记录处理器不再从分片中接收任何记录时结束，因为分片已被拆分或合并，或者流已删除。

KCL 还通过了 `IRecordProcessorCheckpointInterface` 接口到 `shutdown`。如果关闭原因为 `TERMINATE`，则记录处理器应完成处理任何数据记录，然后对此接口调用 `checkpoint` 方法。

更新后的接口（版本 2）

更新后的 `IRecordProcessor` 接口 (package `com.amazonaws.services.kinesis.clientlibrary.interfaces.v2`) 公开了下列记录处理器方法，您的使用者必须实施这些方法：

```
void initialize(InitializationInput initializationInput)
void processRecords(ProcessRecordsInput processRecordsInput)
void shutdown(ShutdownInput shutdownInput)
```

原始版本的接口中的所有参数可通过容器对象上的 `get` 方法进行访问。例如，要检索 `processRecords()` 中的记录的列表，可使用 `processRecordsInput.getRecords()`。

自此接口的版本 2（KCL 1.5.0 及更高版本）起，除了原始接口提供的输入之外，以下新输入也可用：

起始序列号

在传递给 `InitializationInput` 运算的 `initialize()` 对象中，将提供给记录处理器实例的记录的起始序列号。这是由之前处理同一分片的记录处理器实例进行最近一次检查点操作的序列号。此序列号在您的应用程序需要此信息时提供。

待进行检查点操作的序列号

在传递给 `initialize()` 运算的 `InitializationInput` 对象中，在上一个记录处理器实例停止前可能无法提交的待进行检查点操作的序列号（如果有）。

为 I 实现类工厂 `RecordProcessor` 接口

您还需要为实现记录处理器方法的类实现一个工厂。当使用者实例化工作程序时，它将传递对此工厂的引用。

以下示例使用原始记录处理器接口在文件 `AmazonKinesisApplicationSampleRecordProcessorFactory.java` 中实现工厂类。如果您希望此工厂类创建版本 2 记录处理器，请使用程序包名称 `com.amazonaws.services.kinesis.clientlibrary.interfaces.v2`。

```
public class SampleRecordProcessorFactory implements IRecordProcessorFactory {
    /**
     * Constructor.
```

```
*/
public SampleRecordProcessorFactory() {
    super();
}
/**
 * {@inheritDoc}
 */
@Override
public IRecordProcessor createProcessor() {
    return new SampleRecordProcessor();
}
}
```

创建工作线程

正如中讨论的[实施 IRecordProcessor 方法 \(p. 117\)](#)，有两个版本的 KCL 记录处理器接口可供选择，这将影响您创建工作线程的方式。原始记录处理器接口使用以下代码结构创建工作线程：

```
final KinesisClientLibConfiguration config = new KinesisClientLibConfiguration(...)
final IRecordProcessorFactory recordProcessorFactory = new RecordProcessorFactory();
final Worker worker = new Worker(recordProcessorFactory, config);
```

当使用版本 2 的记录处理器接口时，您可使用 Worker.Builder 创建工作线程而无需担心要使用的构造函数以及参数的顺序。更新后的记录处理器接口使用以下代码结构创建工作线程：

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

修改配置属性

该示例提供了配置属性的默认值。工作程序的此配置数据随后将整合到 KinesisClientLibConfiguration 对象中。此对象和对 IRecordProcessor 的类工厂的引用将传入用于实例化工作程序的调用。您可借助 Java 属性文件（请参阅 AmazonKinesisApplicationSample.java）用您自己的值覆盖任何这些属性。

Application Name

KCL 需要一个应用程序名称，该名称在您的应用程序中以及同一区域的 Amazon DynamoDB 表中是唯一的。KCL 通过以下方法使用应用程序名称配置值：

- 假定所有与此应用程序名称关联的工作程序在同一数据流上合作。这些工作程序可被分配到多个实例上。如果运行同一应用程序代码的其他实例，但使用不同的应用程序名称，则 KCL 会将第二个实例视为在同一数据流运行的完全独立的应用程序。
- KCL 利用应用程序名称创建 DynamoDB 表并使用该表保留应用程序的状态信息（如检查点和工作程序-分片映射）。每个应用程序都有自己的 DynamoDB 表。有关更多信息，请参阅[使用租赁表跟踪 KCL 消费者应用程序处理的碎片 \(p. 109\)](#)。

设置凭证

你必须让你 Amazon 凭证提供程序链中的凭证提供程序之一提供给凭证。例如，如果您正在 EC2 实例上运行您的使用者，我们建议您使用 IAM 角色启动实例。Amazon 反映与此 IAM 角色关联的权限的凭证通过实例的元数据提供给实例上的应用程序。使用这种方式管理在 EC2 实例上运行的使用者的凭证最安全。

示例应用程序首先尝试从实例元数据中检索 IAM 凭证：

```
credentialsProvider = new InstanceProfileCredentialsProvider();
```

如果示例应用程序无法从实例元数据中获取凭证，它会尝试从属性文件中检索凭证：

```
credentialsProvider = new ClasspathPropertiesFileCredentialsProvider();
```

有关实例元数据的更多信息，请参阅[实例元数据](#)中的适用于 Linux 实例的 Amazon EC2 用户指南。

将工作程序 ID 用于多个实例

示例初始化代码通过使用本地计算机的名称并附加一个全局唯一的标识符为工作程序创建 ID (workerId)，如以下代码段所示。此方法支持使用者应用程序的多个实例在单台计算机上运行的方案。

```
String workerId = InetAddress.getLocalHost().getCanonicalHostName() + ":" +  
    UUID.randomUUID();
```

迁移到版本 2 的记录处理器接口

如果要迁移使用原始接口的代码，则除了上述步骤之外，还需要执行以下步骤：

1. 更改您的记录处理器类以导入版本 2 记录处理器接口：

```
import com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor;
```

2. 更改对输入的引用以在容器对象上使用 get 方法。例如，在 shutdown() 运算中，将“checkpointer”更改为“shutdownInput.getCheckpointner()”。
3. 更改您的记录处理器工厂类以导入版本 2 记录处理器工厂接口：

```
import  
    com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessorFactory;
```

4. 更改工作线程的结构以使用 Worker.Builder。例如：

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

在 Node.js 中开发 Kinesis Client Library 使用者

可以使用 Kinesis Client Library (KCL) 构建处理 Kinesis 数据流中数据的应用程序。Kinesis Client Library 有多种语言版本。本主题将讨论 Node.js。

KCL 是一个 Java 库；使用名为 MultiLang 守护程序。此守护程序是基于 Java 的，当您使用除 Java 之外的 KCL 语言时，会在后台运行。因此，如果您安装了适用于 Node.js 的 KCL 并完全在 Node.js 中编写使用者应用程序，则由于 MultiLang 守护程序。此外，MultiLangDaemon 有一些默认设置，您可能需要根据自己的使用案例自定义这些设置，例如，Amazon 它连接到的区域。有关的更多信息 MultiLang 开守护程序 GitHub，转到[KCL MultiLang 守护程序](#)页。

要下载 Node.js KCL GitHub，转到[Kinesis Client Library \(Node.js\)](#)。

示例代码下载

Node.js 中有两个代码示例可用于 KCL：

- [basic-sample](#)

在下列节中用于阐释使用 Node.js 构建 KCL 使用者应用程序的基础知识。

- [click-stream-sample](#)

稍微复杂一些，使用了现实世界的情景，适合在您熟悉基本示例代码之后采用。此示例在这里不做讨论，但它有一个包含更多信息的自述文件。

在 Node.js 中实现 KCL 使用者应用程序时，您必须完成下列任务：

任务

- [实现记录处理器 \(p. 122\)](#)
- [修改配置属性 \(p. 123\)](#)

实现记录处理器

使用适用于 Node.js 的 KCL 的可能最简单的使用者必须实现 `recordProcessor` 函数，该函数反过来包含函数 `initialize`、`processRecords` 和 `shutdown`。该示例提供了可用作起点的实现（请参阅 `sample_kcl_app.js`）。

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

`initialize`

KCL 称 `initialize` 在记录处理器启动时函数。此记录处理器只处理作为 `initializeInput.shardId` 传递的分片 ID，并且通常情况下反过来说也成立（此分片只能由此记录处理器处理）。但是，您的使用者应该考虑数据记录可能会经过多次处理的情况。这是因为 Kinesis Data Streams 有至少一次语义，即分片中的每个数据记录至少会由使用者中的工作程序处理一次。有关特定分片可能由多个工作线程处理的情况的更多信息，请参阅 [重新分片、扩展和并行处理 \(p. 164\)](#)。

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

`processRecords`

KCL 将使用包含一个数据记录的列表（该记录从指定到 `initialize` 函数。您实现的记录处理器根据您的使用者的语义处理这些记录中的数据。例如，工作程序可能执行数据转换，然后将结果存储在 Amazon Simple Storage Service (Amazon S3) 存储桶中。

```
processRecords: function(processRecordsInput, completeCallback)
```

除了数据本身之外，记录还包含工作程序在处理数据时可使用的序号和分区键。例如，工作线程可选择 S3 存储桶，并在其中根据分区键的值存储数据。`record` 词典公开了以下键-值对来访问记录的数据、序号和分区键：

```
record.data  
record.sequenceNumber  
record.partitionKey
```

请注意，数据是 Base64 编码的。

在该基本示例中，函数 `processRecords` 具有显示工作程序如何访问记录的数据、序号和分区键的代码。

Kinesis Data Streams 需要记录处理器来跟踪已在分片中处理的记录。KCL 用 `checkpoint` 对象传递为 `processRecordsInput.checkpointer`。你的记录处理器调用 `checkpointer.checkpoint` 函数以向

告知 KCL 处理分片中的记录的进度。如果工作程序失败，将在您重新启动分片的处理时使用此信息，以便在上一个已知的已处理记录处继续处理。

对于拆分或合并操作，在原始分片的处理器调用之前，KCL 不会开始处理新分片。checkpoint表示原始分片的所有处理都已完成。

如果你没有将序列号传递给checkpoint函数，KCL 假定调用checkpoint表示所有记录都已处理，一直处理到传递到记录处理器的最后一个记录。因此，记录处理器应该调用checkpoint 仅限在处理传递到它的列表中的所有记录后。记录处理器不需要在每次调用 checkpoint 时调用 processRecords。例如，处理器可以在每第三次调用时调用 checkpoint，或调用记录处理器外部的某个事件（如您已实现的自定义确认/验证服务）。

您可以选择性地为某个记录的确切序号指定为 checkpoint 的参数。在本例中，KCL 将假定所有记录都已处理，一直到达该记录。

基本示例应用程序显示了对 checkpointer.checkpoint 函数最简单的调用。您此时可以在该函数中为您的使用者添加您需要的其他检查点逻辑。

shutdown

KCL 称shutdown处理结束时可以是函数 (shutdownInput.reason是TERMINATE) 或者工作人员不再响应 (shutdownInput.reason是ZOMBIE)。

```
shutdown: function(shutdownInput, completeCallback)
```

处理操作在记录处理器不再从分片中接收任何记录时结束，因为分片已被拆分或合并，或者流已删除。

KCL 还通过了shutdownInput.checkpointer反对shutdown. 如果关闭原因为 TERMINATE，则应确保记录处理器已完成处理任何数据记录，然后对此接口调用 checkpoint 函数。

修改配置属性

该示例提供了配置属性的默认值。您可使用自己的值覆盖任何这些属性（请参阅基本示例中的 sample.properties）。

Application Name

KCL 需要一个应用程序名称，该名称在您的应用程序中以及同一区域的 Amazon DynamoDB 表中是唯一的。KCL 通过以下方法使用应用程序名称配置值：

- 假定所有与此应用程序名称关联的工作程序在同一数据流上合作。这些工作程序可被分配到多个实例上。如果运行同一应用程序代码的其他实例，但使用不同的应用程序名称，则 KCL 会将第二个实例视为在同一数据流运行的完全独立的应用程序。
- KCL 利用应用程序名称创建 DynamoDB 表并使用该表保留应用程序的状态信息（如检查点和工作程序-分片映射）。每个应用程序都有自己的 DynamoDB 表。有关更多信息，请参阅 [使用租赁表跟踪 KCL 消费者应用程序处理的碎片 \(p. 109\)](#)。

设置凭证

你必须让你Amazon凭证提供程序链中的凭证提供程序之一提供给凭证。可以使用 AWSCredentialsProvider 属性设置凭证提供程序。sample.properties 文件必须向[默认凭证提供程序链](#)中的凭证提供程序之一提供您的凭证。如果您正在 Amazon EC2 实例上运行您的使用者，我们建议您使用 IAM 角色配置实例。Amazon反映与此 IAM 角色关联的权限的凭证通过实例的元数据提供给实例上的应用程序。使用这种方式管理在 EC2 实例上运行的使用者应用程序的凭证最安全。

下面的示例将 KCL 配置为处理名为的 Kinesis 数据流：kclnodejssample使用中提供的记录处理器sample_kcl_app.js：

```
# The Node.js executable script
```

```
executableName = node sample_kcl_app.js
# The name of an Amazon Kinesis stream to process
streamName = kclnodejssample
# Unique KCL application name
applicationName = kclnodejssample
# Use default Amazon credentials provider chain
AWSCredentialsProvider = DefaultAWSCredentialsProviderChain
# Read from the beginning of the stream
initialPositionInStream = TRIM_HORIZON
```

在 .NET 中开发 Kinesis Client Library 使用者

可以使用 Kinesis Client Library (KCL) 构建处理 Kinesis 数据流中数据的应用程序。Kinesis Client Library 有多种语言版本。本主题将讨论 .NET。

KCL 是一个 Java 库；使用名为 MultiLang 守护程序。此守护程序是基于 Java 的，当您使用除 Java 之外的 KCL 语言时，会在后台运行。因此，如果您安装了适用于 .NET 的 KCL 并完全在 .NET 中编写使用者应用程序，则由于 MultiLang 守护程序。此外，MultiLangDaemon 有一些默认设置，您可能需要根据自己的使用案例自定义这些设置，例如，Amazon 它连接到的区域。有关的更多信息 MultiLang 守护程序 GitHub，转到 [KCL MultiLang 守护程序](#) 页。

从中下载 .NET KCL GitHub，转到 [Kinesis 客户端库 \(.NET\)](#)。要下载适用于 .NET KCL 使用者应用程序的示例代码，请转至 [.NET 示例消费者项目的 KCL](#) 上的页面 GitHub。

在 .NET 中实现 KCL 使用者应用程序时，您必须完成下列任务：

任务

- [实施 IRecordProcessor 类方法 \(p. 124\)](#)
- [修改配置属性 \(p. 125\)](#)

实施 IRecordProcessor 类方法

使用者必须实现适用于 IRecordProcessor 的以下方法。示例使用者提供了可用作起点的实现（请参阅 SampleRecordProcessor 中的 SampleConsumer/AmazonKinesisSampleConsumer.cs 类）。

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

Initialize

在实例化记录处理器时调用此方法，并将特定分片 ID 传入 input 参数 (input.ShardId)。此记录处理器只处理此分片，并且通常情况下反过来说也成立（此分片只能由此记录处理器处理）。但是，您的使用者应该考虑数据记录可能会经过多次处理的情况。这是因为 Kinesis Data Streams 有至少一次语义，即分片中的每个数据记录至少会由使用者中的工作程序处理一次。有关特定分片可能由多个工作线程处理的情况的更多信息，请参阅 [重新分片、扩展和并行处理 \(p. 164\)](#)。

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

ProcessRecords

KCL 调用此方法，并将数据记录的列表传入 input 参数 (input.Records) 来自指定的分片 Initialize 方法。您实现的记录处理器根据您的使用者的语义处理这些记录中的数据。例如，工作程序可能执行数据转换，然后将结果存储在 Amazon Simple Storage Service (Amazon S3) 存储桶中。

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

除了数据本身之外，记录还包含一个序号和一个分区键。工作程序可在处理数据时使用这些值。例如，工作线程可选择 S3 存储桶，并在其中根据分区键的值存储数据。Record 类公开了以下代理来访问记录的数据、序号和分区键：

```
byte[] Record.Data  
string Record.SequenceNumber  
string Record.PartitionKey
```

在该示例中，方法 ProcessRecordsWithRetries 具有显示工作程序如何访问记录的数据、序号和分区键的代码。

Kinesis Data Streams 需要记录处理器来跟踪已在分片中处理的记录。KCL 通过传递一个来为您执行此跟踪。Checkpoint 反对 ProcessRecords(input.Checkpointer)。记录处理器调用 Checkpointer.Checkpoint 方法以向告知 KCL 处理分片中的记录的进度。如果工作线程失败，KCL 将使用此信息在已知的上一个已处理记录处重新启动对分片的处理。

对于拆分或合并操作，在原始分片的处理器调用之前，KCL 不会开始处理新分片。Checkpoint.Checkpoint 表示原始分片的所有处理都已完成。

如果你没有传递参数，KCL 假定调用 Checkpointer.Checkpoint 表示所有记录都已处理，一直处理到传递到记录处理器的最后一个记录。因此，记录处理器只应在已处理传递到它的列表中的所有记录后才调用 Checkpointer.Checkpoint。记录处理器不需要在每次调用 Checkpointer.Checkpoint 时调用 ProcessRecords。例如，处理器在每第三次或第四次调用时调用 Checkpointer.Checkpoint。您可以选择性地为某个记录的确切序号指定为 Checkpointer.Checkpoint 的参数。在本例中，KCL 将假定所有记录都已处理，一直到达该记录。

在该示例中，私有方法 Checkpoint(Checkpointer checkpointer) 展示了如何使用适当的异常处理和重试逻辑调用 Checkpointer.Checkpoint 方法。

KCL 适用于 .NET 的处理异常的方式不同于其他 KCL 语言库，后者不处理因处理数据记录而引起的任何异常。用户代码中未捕获的任何异常都将使程序崩溃。

关闭

KCL 称 Shutdown 方法无论是在处理结束时（关闭原因是 TERMINATE）或工作程序不再响应（关闭）input.Reason 值是 ZOMBIE）。

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

处理操作在记录处理器不再从分片中接收任何记录时结束，因为分片已被拆分或合并，或者流已删除。

KCL 还通过了 Checkpointer 反对 shutdown。如果关闭原因为 TERMINATE，则记录处理器应完成处理任何数据记录，然后对此接口调用 checkpoint 方法。

修改配置属性

示例使用者提供了配置属性的默认值。您可使用自己的值覆盖任何这些属性（请参阅 SampleConsumer/kcl.properties）。

Application Name

KCL 需要一个应用程序名称，该名称在您的应用程序中以及同一区域的 Amazon DynamoDB 表中是唯一的。KCL 通过以下方法使用应用程序名称配置值：

- 假定所有与此应用程序名称关联的工作程序在同一数据流上合作。这些工作程序可被分配到多个实例上。如果运行同一应用程序代码的其他实例，但使用不同的应用程序名称，则 KCL 会将第二个实例视为在同一数据流运行的完全独立的应用程序。
- KCL 利用应用程序名称创建 DynamoDB 表并使用该表保留应用程序的状态信息（如检查点和工作程序-分片映射）。每个应用程序都有自己的 DynamoDB 表。有关更多信息，请参阅 [使用租赁表跟踪 KCL 消费者应用程序处理的碎片 \(p. 109\)](#)。

设置凭证

你必须让你Amazon凭证提供程序链中的凭证提供程序之一提供给凭证。可以使用 `AWSCredentialsProvider` 属性设置凭证提供程序。[sample.properties](#) 必须向[默认凭证提供程序链](#)中的凭证提供程序之一提供您的凭证。如果您正在 EC2 实例上运行您的使用者应用程序，我们建议您使用 IAM 角色配置实例。Amazon反映与此 IAM 角色关联的权限的凭证通过实例的元数据提供给实例上的应用程序。使用这种方式管理在 EC2 实例上运行的使用者的凭证最安全。

该示例的属性文件将配置 KCL 以使用中提供的记录处理器处理名为 “words” 的 Kinesis 数据流。AmazonKinesisSampleConsumer.cs。

在 Python 中开发 Kinesis 客户端库用户

可以使用 Kinesis Client Library (KCL) 构建处理 Kinesis 数据流中数据的应用程序。Kinesis Client Library 有多种语言版本。本主题将讨论 Python。

KCL 是一个 Java 库；使用名为MultiLang守护程序。此守护程序是基于 Java 的，当您使用除 Java 之外的 KCL 语言时，会在后台运行。因此，如果您安装了适用于 Python 的 KCL 并完全在 Python 中编写使用者应用程序，则由于MultiLang守护程序。此外，MultiLangDaemon 有一些默认设置，您可能需要根据自己的使用案例自定义这些设置，例如，Amazon它连接到的区域。有关的更多信息MultiLang开守护程序GitHub，转到[KCLMultiLang守护程序](#)页。

从下载 Python KCLGitHub，转到[Kinesis Client Library \(Python\)](#)。要下载适用于 Python KCL 使用者应用程序的示例代码，请转至[KCL for Python 示例项目](#)上的页面GitHub。

在 Python 中实现 KCL 使用者应用程序时，您必须完成下列任务：

任务

- [实施RecordProcessor类方法 \(p. 126\)](#)
- [修改配置属性 \(p. 127\)](#)

实施RecordProcessor类方法

RecordProcess 类必须扩展 RecordProcessorBase 以实现以下方法。该示例提供了可用作起点的实现（请参阅 `sample_kclpy_app.py`）。

```
def initialize(self, shard_id)
def process_records(self, records, checkpoint)
def shutdown(self, checkpoint, reason)
```

initialize

KCL 称initialize方法在实例化记录处理器时，将特定分片 ID 作为参数传递。此记录处理器只处理此分片，并且通常情况下反过来说也成立（此分片只能由此记录处理器处理）。但是，您的使用者应该考虑数据记录可能会经过多次处理的情况。这是因为 Kinesis Data Streams 有至少一次语义，即分片中的每个数据记录至少会由使用者中的工作程序处理一次。有关特定分片可能由多个工作程序进行处理的情况的更多信息，请参阅[重新分片、扩展和并行处理 \(p. 164\)](#)。

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

process_records

KCL 将调用此方法，并传递来自指定分片的数据记录的列表。initialize方法。您实现的记录处理器根据您的使用者的语义处理这些记录中的数据。例如，工作程序可能执行数据转换，然后将结果存储在 Amazon Simple Storage Service (Amazon S3) 存储桶中。

```
def process_records(self, records, checkpoint)
```

除了数据本身之外，记录还包含一个序号和一个分区键。工作程序可在处理数据时使用这些值。例如，工作线程可选择 S3 存储桶，并在其中根据分区键的值存储数据。record 词典公开了以下键-值对来访问记录的数据、序号和分区键：

```
record.get('data')
record.get('sequenceNumber')
record.get('partitionKey')
```

请注意，数据是 Base64 编码的。

在该示例中，方法 process_records 具有显示工作程序如何访问记录的数据、序号和分区键的代码。

Kinesis Data Streams 需要记录处理器来跟踪已在分片中处理的记录。KCL 通过传递一个来为您执行此跟踪。Checkpoint 反对 process_records。记录处理器调用 checkpoint 方法以向 KCL 通知记录处理分片中的记录的进度。如果工作线程失败，KCL 将使用此信息在已知的上一个已处理记录处重新启动对分片的处理。

对于拆分或合并操作，在原始分片的处理器调用之前，KCL 不会开始处理新分片。checkpoint 表示原始分片的所有处理都已完成。

如果你没有传递参数，KCL 假定调用 checkpoint 表示所有记录都已处理，一直处理到传递到记录处理器的最后一个记录。因此，记录处理器只应在已处理传递到它的列表中的所有记录后才调用 checkpoint。记录处理器不需要在每次调用 checkpoint 时调用 process_records。例如，处理器可在每第三次调用时调用 checkpoint。您可以选择性地为某个记录的确切序号指定为 checkpoint 的参数。在本例中，KCL 将假定所有记录都已处理，一直到达该记录。

在该示例中，私有方法 checkpoint 展示了如何使用适当的异常处理和重试逻辑调用 Checkpointer.checkpoint 方法。

KCL 依靠 process_records 以处理由处理数据记录引起的任何异常。如果从中抛出异常 process_records，KCL 将跳过已传递到的数据记录 process_records 在例外之前。也就是说，这些记录不会重新发送到引发异常的记录处理器或使用者的任何其他记录处理器。

shutdown

KCL 称 shutdown 方法无论是在处理结束时（关闭原因是 TERMINATE）或工作程序不再响应（关闭）reason 是 ZOMBIE）。

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

处理操作在记录处理器不再从分片中接收任何记录时结束，因为分片已被拆分或合并，或者流已删除。

KCL 还通过了 Checkpointer 反对 shutdown。如果关闭 reason 是 TERMINATE，则记录处理器应完成处理任何数据记录，然后对此接口调用 checkpoint 方法。

修改配置属性

该示例提供了配置属性的默认值。您可使用自己的值覆盖任何这些属性（请参阅 sample.properties）。

Application Name

KCL 需要一个应用程序名称，该名称在您的各个应用程序中以及同一区域的 Amazon DynamoDB 表中是唯一的。KCL 通过以下方法使用应用程序名称配置值：

- 假定与此应用程序名称关联的所有工作线程在同一个流上一起运行。这些工作线程可分布在多个实例上。如果运行同一应用程序代码的其他实例，但使用不同的应用程序名称，则 KCL 会将第二个实例视为在同一数据流运行的完全独立的应用程序。
- KCL 利用应用程序名称创建 DynamoDB 表并使用该表保留应用程序的状态信息（如检查点和工作程序-分片映射）。每个应用程序都有自己的 DynamoDB 表。有关更多信息，请参阅 [使用租赁表跟踪 KCL 消费者应用程序处理的碎片 \(p. 109\)](#)。

设置凭证

你必须让你Amazon凭证提供程序链中的凭证提供程序之一提供给凭证。可以使用 `AWSCredentialsProvider` 属性设置凭证提供程序。[sample.properties](#) 必须向[默认凭证提供程序链](#)中的凭证提供程序之一提供您的凭证。如果您正在 Amazon EC2 实例上运行您的使用者应用程序，我们建议您使用 IAM 角色配置实例。Amazon反映与此 IAM 角色关联的权限的凭证通过实例的元数据提供给实例上的应用程序。使用这种方式管理在 EC2 实例上运行的使用者应用程序的凭证最安全。

该示例的属性文件将配置 KCL 以使用中提供的记录处理器处理名为 “words” 的 Kinesis 数据流。`sample_kclpy_app.py`.

在 Ruby 中开发 Kinesis 客户端库用户

可以使用 Kinesis Client Library (KCL) 构建处理 Kinesis 数据流中数据的应用程序。Kinesis Client Library 有多种语言版本。本主题将讨论 Ruby。

KCL 是一个 Java 库；使用名为 MultiLang 守护程序。此守护程序是基于 Java 的，当您使用除 Java 之外的 KCL 语言时，会在后台运行。因此，如果您安装了适用于 Ruby 的 KCL 并完全在 Ruby 中编写使用者应用程序，则由于 MultiLang 守护程序。此外，MultiLangDaemon 有一些默认设置，您可能需要根据自己的使用案例自定义这些设置，例如，Amazon它连接到的区域。有关的更多信息MultiLang开守护程序GitHub，转到[KCLMultiLang守护程序](#)页。

从下载 Ruby KCLGitHub，转到[Kinesis 客户端库 \(Ruby\)](#)。要下载适用于 Ruby KCL 使用者应用程序的示例代码，请转至[KCL for Ruby 示例项目](#)上的页面GitHub。

有关 KCL Ruby 支持库的更多信息，请参阅[KCL Ruby Gems 文档](#)。

开发 KCL 2.x 使用者

本主题将说明如何使用 2.0 版本的 Kinesis 客户端库 (KCL)。有关 KCL 的更多信息，请参阅中提供的概述。[使用 Kinesis 客户端库 1.x 开发使用者](#)。

目录

- [在 Java 中开发 Kinesis 客户端库用户 \(p. 128\)](#)
- [在 Python 中开发 Kinesis 客户端库用户 \(p. 134\)](#)

在 Java 中开发 Kinesis 客户端库用户

以下代码显示 `ProcessorFactory` 和 `RecordProcessor` 在 Java 中的实施示例。如果要利用增强型扇出功能，请参阅[利用使用增强型扇出功能的用户](#)。

```
/*
 * 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 Amazon SDK
 * to publish data.
 * Before running this program you must first create a Kinesis stream through the Amazon
 * console or Amazon 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 {
        kinesysClient.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);
        }
    }
}

```

```
}
```

在 Python 中开发 Kinesis 客户端库用户

可以使用 Kinesis 客户端库 (KCL) 构建处理 Kinesis 数据流中数据的应用程序。Kinesis 客户端库有多种语言版本。本主题将讨论 Python。

KCL 是一个 Java 库；使用名为 MultiLang 守护程序。此守护程序是基于 Java 的，当您使用除 Java 之外的 KCL 语言时，会在后台运行。因此，如果您安装了适用于 Python 的 KCL 并完全在 Python 中编写使用者应用程序，则仍需要在您的系统中安装 Java，因为 MultiLang 守护程序。此外，MultiLang 守护程序有一些默认设置，您可能需要根据自己的使用案例自定义这些设置，例如，Amazon 它连接到的区域。有关的更多信息 MultiLang 守护程序 GitHub，转到 [KCL MultiLang 守护程序](#) 页。

从下载 Python KCL GitHub，转到 [Kinesis 客户端库 \(Python\)](#)。要下载适用于 Python KCL 使用者应用程序的示例代码，请转至 [KCL for Python 示例项目](#) “打开” 页 GitHub。

在 Python 中实现 KCL 使用者应用程序时，您必须完成下列任务：

任务

- [实施 RecordProcessor 类方法 \(p. 126\)](#)
- [修改配置属性 \(p. 127\)](#)

实施 RecordProcessor 类方法

RecordProcess 类必须扩展 RecordProcessorBase 类以实现以下方法：

```
initialize  
process_records  
shutdown_requested
```

此示例提供了可用作起点的实现。

```
#!/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()
```

修改配置属性

该示例提供了配置属性的默认值，如以下脚本所示。您可使用自己的值覆盖任何这些属性。

```
# 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
```

Application Name

KCL 需要一个应用程序名称，该名称在您的各个应用程序中以及同一区域的 Amazon DynamoDB 表中是唯一的。KCL 通过以下方法使用应用程序名称配置值：

- 假定与此应用程序名称关联的所有工作线程在同一个流上一起运行。这些工作线程可分布在多个实例中。如果运行同一应用程序代码的其他实例，但使用不同的应用程序名称，则 KCL 会将第二个实例视为在同一数据流运行的完全独立的应用程序。
- KCL 使用应用程序名称创建 DynamoDB 表并使用该表保留应用程序的状态信息（如检查点和工作程序-分片映射）。每个应用程序都有自己的 DynamoDB 表。有关更多信息，请参阅 [使用租赁表跟踪 KCL 消费者应用程序处理的碎片 \(p. 109\)](#)。

凭证

你必须做你的 Amazon 中的凭证提供程序之一可用于 [默认凭证提供商链](#)。可以使用 `AWSCredentialsProvider` 属性设置凭证提供程序。如果您在 Amazon EC2 实例上运行您的使用者应用程序，我们建议您使用 IAM 角色配置实例。Amazon 反映与此 IAM 角色关联的权限的凭证通过实例元数据提供给实例上的应用程序。使用这种方式管理在 EC2 实例上运行的使用者应用程序的凭证最安全。

使用 Amazon SDK for Java 开发具有共享吞吐量的自定义使用者

开发自定义 Kinesis Data Streams 使用者并在整个过程中共享的方法之一是使用 Amazon Kinesis Data Streams API。本部分介绍在 Amazon SDK for Java 开发工具包中使用 Kinesis Streams API 本节中的 Java 示例代码演示了如何执行基本的 KDS API 操作，并按操作类型在逻辑上进行了划分。

这些示例并非可直接用于生产的代码。它们不会检查所有可能的异常，或者不会考虑到所有可能的安全或性能问题。

您可以使用其他不同的编程语言调用 Kinesis Data Streams API。有关所有可用 Amazon 软件开发工具包的更多信息，请参阅 [使用 Amazon Web Services 开始开发](#)。

Important

开发自定义 Kinesis Data Streams 客户端库 (KCL) KCL 通过处理与分布式计算相关的许多复杂任务，帮助您使用和处理 Kinesis 数据流中的数据。有关更多信息，请参阅 [使用 KCL 开发具有共享吞吐量的自定义使用者](#)。

主题

- [从流中获取数据 \(p. 139\)](#)
- [使用分片迭代器 \(p. 140\)](#)
- [使用 GetRecords \(p. 141\)](#)
- [适应重新分片 \(p. 142\)](#)
- [使用 Amazon Glue 架构注册表与数据交互 \(p. 142\)](#)

从流中获取数据

Kinesis Data Streams API 包括 `getShardIterator` 和 `getRecords` 方法，您可以调用这些方法从数据流检索记录。这是拉取模型，您的代码直接从数据流的分片中提取数据记录。

Important

我们建议您使用 KCL 提供的记录处理器支持从数据流中检索记录。这是推送模型，您可以在其中实现处理数据的代码。KCL 从数据流中检索数据记录并将其传送到您的应用程序代码。此外，KCL 还提供故障转移、恢复和负载均衡功能。有关更多信息，请参阅 [使用 KCL 开发具有共享吞吐量的自定义使用者](#)。

但是 Kinesis Streams Streams API 例如，实现用于监控或调试数据流的自定义工具。

Important

Kinesis Data Streams 支持更改数据流的数据记录保留期。有关更多信息，请参阅[更改数据保留期 \(p. 72\)](#)：

使用分片迭代器

可从流中按分片检索记录。对于每个分片以及您从分片中检索的每批记录，您必须获取分片迭代器。可在 `getRecordsRequest` 对象中使用分片迭代器来指定要从中检索记录的分片。与分片迭代器关联的类型决定了应在分片中检索记录的起点（有关更多信息，请参阅此部分中后面的内容）。您需要先检索分片（[DescribeStream API-已过时 \(p. 67\)](#)中已讨论），然后才能使用分片迭代器。

使用 `getShardIterator` 方法获取初始分片迭代器。使用 `getNextShardIterator` 对象（由 `getRecordsResult` 方法返回）的 `getRecords` 方法为其他记录批次获取分片迭代器。分片迭代器的有效时间为 5 分钟。如果使用有效期内的分片迭代器，则将获得一个新的迭代器。每个分片迭代器在 5 分钟内一直有效，即使使用过也是如此。

要获取初始分片迭代器，请实例化 `GetShardIteratorRequest` 并将其传递给 `getShardIterator` 方法。要配置请求，请指定流和分片 ID。Amazon[列出流 \(p. 64\)](#) 有关如何获取流中分片的信息，请参阅[DescribeStream API-已过时 \(p. 67\)](#)。

```
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();
```

此示例代码在获取初始分片迭代器时将 `TRIM_HORIZON` 指定为迭代器类型。这种迭代器类型意味着应从添加到分片的第一条记录开始返回记录，而不是从最近添加的记录（也称为提示）开始。以下是可能的迭代器类型：

- `AT_SEQUENCE_NUMBER`
- `AFTER_SEQUENCE_NUMBER`
- `AT_TIMESTAMP`
- `TRIM_HORIZON`
- `LATEST`

[ShardIteratorType](#)

部分迭代器类型除了需要指定类型之外，还需要指定序列号；例如：

```
getShardIteratorRequest.setShardIteratorType("AT_SEQUENCE_NUMBER");
getShardIteratorRequest.setStartingSequenceNumber(specialSequenceNumber);
```

使用 `getRecords` 获取记录之后，可通过调用记录的 `getSequenceNumber` 方法来获取记录的序列号。

```
record.getSequenceNumber()
```

此外，将记录添加到数据流的代码可通过对 `getSequenceNumber` 的结果调用 `putRecord` 获取已添加记录的序列号。

```
lastSequenceNumber = putRecordResult.getSequenceNumber();
```

您可使用序列号确保记录的顺序严格递增。有关更多信息，请参阅 [PutRecord 示例 \(p. 89\)](#) 中的代码示例。

使用 GetRecords

获取分片迭代器之后，请实例化 `GetRecordsRequest` 对象。使用 `setShardIterator` 方法为请求指定迭代器。

(可选) 您还可使用 `setLimit` 方法设置要检索的记录的数量。`getRecords` 返回的记录数量始终等于或小于此限制。如果您未指定此限制，则 `getRecords` 返回 10 MB 的检索记录。以下示例代码将此限制设置为 25 个记录。

如果未返回任何记录，则意味着此分片中当前没有分片迭代器引用的序列号对应的可用数据记录。在这种情况下，您的应用程序应等待一段适合流数据源的时间。然后尝试使用对 `getRecords` 的上一调用返回的分片迭代器再次从分片获取数据。

将 `getRecordsRequest` 传递给 `getRecords` 方法并捕获返回的值作为 `getRecordsResult` 对象。要获取数据记录，请对 `getRecords` 对象调用 `getRecordsResult` 方法。

```
GetRecordsRequest getRecordsRequest = new GetRecordsRequest();
getRecordsRequest.setShardIterator(shardIterator);
getRecordsRequest.setLimit(25);

GetRecordsResult getRecordsResult = client.getRecords(getRecordsRequest);
List<Record> records = getRecordsResult.getRecords();
```

要准备对 `getRecords` 的另一次调用，请通过 `getRecordsResult` 获取下一分片迭代器。

```
shardIterator = getRecordsResult.getNextShardIterator();
```

为获得最佳效果，请在对 `getRecords` 的各次调用之间停止至少 1 秒 (1000 毫秒) 以免超出 `getRecords` 频率限制。

```
try {
    Thread.sleep(1000);
}
catch (InterruptedException e) {}
```

通常，您应循环调用 `getRecords`，甚至当您在测试方案中检索单一记录时也是如此。对 `getRecords` 的单一调用可能返回空的记录列表，即使分片包含更多具有之后的序列号的记录也是如此。出现此情况时，将返回 `NextShardIterator`，同时空记录列表将引用分片中之后的序列号，并且后续的 `getRecords` 调用最终将返回记录。以下示例演示循环的使用。

示例：getRecords

以下代码示例反映了此节中的 `getRecords` 顶端，包括循环发出调用。

```
// 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();  
}
```

如果您使用的是 Kinesis 客户端库，它可能会在返回数据之前进行多次调用。这种行为是设计使然，并不表示 KCL 或您的数据有问题。

适应重新分片

如果 `getRecordsResult.getNextShardIterator` 返回 `null`，则表示发生了涉及该分片的分片拆分或合并。此分片现在处于 `CLOSED` 状态，您已经读取了该分片的所有可用数据记录。

在这种情况下，您可以使用 `getRecordsResult.childShards` 来了解正在处理的分区的新子分区，这些分片是由拆分或合并创建的。[ChildShard](#)

在拆分中，两个新分片的 `parentShardId` 都与您之前处理的分片的分片 ID 相同。这两个分片的 `adjacentParentShardId` 值为 `null`。

在合并中，合并创建的一个新分片的 `parentShardId` 等于父分片之一的分片 ID，并且 `adjacentParentShardId` 等于另一父分片的分片 ID。您的应用程序已读取这些分片之一中的所有数据。这是 `getRecordsResult.getNextShardIterator` 返回 `null` 的分片。如果数据顺序对于您的应用程序很重要，则应确保它在读取合并创建的子分片中的任何新数据之前，还读取另一父分片中的所有数据。

如果您使用多个处理器从流检索数据（假定一个分片一个处理器），并且出现分片拆分或合并时，您应增加或减少处理器数量以适应分片数量的变化。

有关重新分片的更多信息，包括有关分片状态（如 `CLOSED`）的讨论，请参阅 [对流进行重新分片 \(p. 68\)](#)。

使用 Amazon Glue 架构注册表与数据交互

您可以将 Kinesis 数据流与 Glue 架构注册表集成。Amazon Glue 架构，注册表允许您集中发现、控制和演变架构。架构定义了数据记录的结构和格式。架构是用于可靠数据发布、使用或存储的版本化规范。Amazon Glue Schema Registry 使您能够改善流媒体应用程序中的 end-to-end 数据质量和数据治理。[Amazon](#) 设置这种集成的方法之一是通过 Amazon Java SDK 中提供的 `GetRecords` Kinesis Data Streams API。

有关如何使用 Kinesis Data Streams API 设置 `KinesisDataStreamGetRecords` 与架构注册表集成的详细说明，请参阅[用例：将 Amazon Kinesis Data Streams 与 Amazon Glue 架构注册表集成中的“使用 Kinesis 数据流 API 与数据交互”部分](#)。

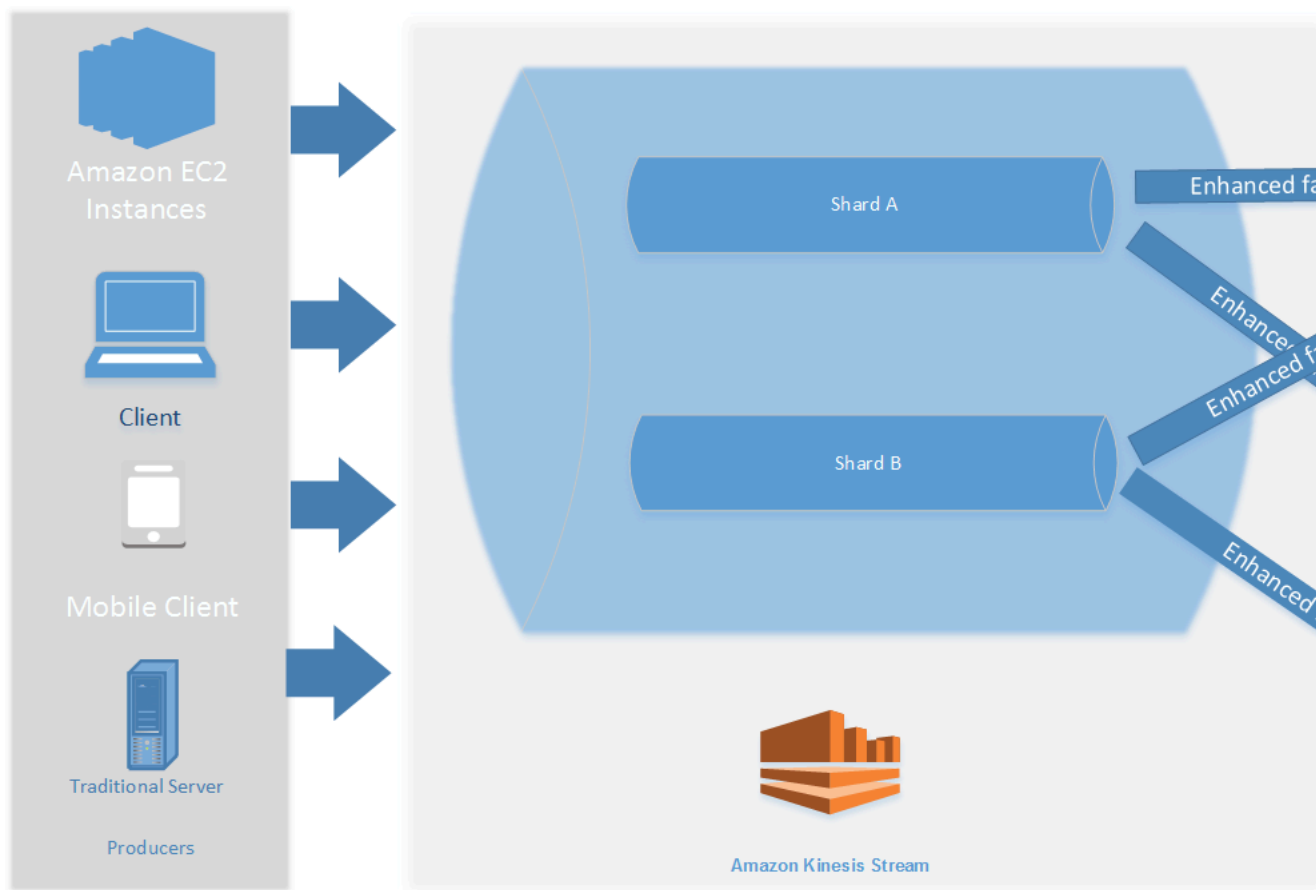
开发具有专用吞吐量的自定义使用者（增强扇出功能）

在 Amazon Kinesis Data Streams 中，您可以构建使用名为的功能的使用者增强的扇出。利用此功能，使用者可以从流中接收记录，其数据吞吐量高达每分片 2 MB/秒。此吞吐量是专用的，这意味着，使用增强型扇出功能的使用者不必与接收流中数据的其他使用者争夺。Kinesis Data Streams 将流中的数据记录推送到使用增强型扇出功能的使用者。因此，这些使用者无需轮询数据。

Important

您可以为每个流注册多达 20 个使用者，以便使用增强型扇出功能。

下图显示的是增强型扇出功能架构。如果使用 2.0 版或更高版本的 Amazon Kinesis 客户端库 (KCL) 构建使用者，则 KCL 会将使用者设置为使用增强型扇出功能接收来自流的所有分片的数据。如果使用 API 构建使用者，则可订阅单个分片。



此图显示以下内容：

- 一个具有两个分片的流。
- 使用增强型扇出功能接收流中数据的两个使用者：使用者 X 和使用者 Y。这两个使用者均已订阅流的所有分片和所有记录。如果使用 2.0 版或更高版本的 KCL 构建使用者，则 KCL 将自动为使用者订阅流的所有分片。另一方面，如果使用 API 构建使用者，则可订阅单个分片。
- 表示使用者用于接收流中数据的增强型扇出功能管道的箭头。增强型扇出功能管道每分片提供高达 2 MB/秒 数据，独立于任何其他管道或使用者的总数量。

主题

- [使用 KCL 2.x 开发具有增强型扇出功能的使用者 \(p. 144\)](#)
- [使用 Kinesis Data Streams API 开发具有增强型扇出功能的使用者 \(p. 148\)](#)
- [使用 Amazon Web Services Management Console 管理具有增强扇出功能的使用者 \(p. 150\)](#)

使用 KCL 2.x 开发具有增强型扇出功能的使用者

使用者增强的扇出在 Amazon Kinesis Data Streams 中可接收每分片每秒专用吞吐量高达 2 MB 数据的数据流中的记录。此类使用者不必与接收流中数据的其他使用者争夺。有关更多信息，请参阅 [开发具有专用吞吐量的自定义使用者 \(增强扇出功能\) \(p. 143\)](#)。

可以使用 2.0 版或更高版本的 Kinesis 客户端库 (KCL) 开发使用增强型扇出功能接收流中数据的应用程序。KCL 会自动为您的应用程序订阅流的所有分片，并确保使用者应用程序的读取吞吐量的值为每分片 2 MB/ 秒。如果要在不开启增强型扇出功能的情况下使用 KCL，请参阅[使用 Kinesis 客户端库 2.0 开发使用者](#)。

主题

- [使用 KCL 2.x 在 Java 中开发具有增强扇出功能的使用者 \(p. 144\)](#)

使用 KCL 2.x 在 Java 中开发具有增强扇出功能的使用者

可以使用 2.0 版或更高版本的 Kinesis 客户端库 (KCL) 在 Amazon Kinesis Data Streams 中开发应用程序，以使用增强型扇出功能接收流中数据的数据。以下代码显示 ProcessorFactory 和 RecordProcessor 在 Java 中的实施示例。

建议您使用 KinesisClientUtil 创建 KinesisAsyncClient，并在 KinesisAsyncClient 中配置 maxConcurrency。

Important

Amazon Kinesis 客户端可能会看到延迟大幅增加，除非您将 KinesisAsyncClient 配置为具有足够高的 maxConcurrency，以允许所有租期以及额外使用 KinesisAsyncClient。

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 * Licensed under the Amazon Software License (the "License").
 * You may not use this file except in compliance with the License.
 * A copy of the License is located at
 *
 * http://aws.amazon.com/asl/
 *
 * or in the "license" file accompanying this file. This file is distributed
 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
 * express or implied. See the License for the specific language governing
 * permissions and limitations under the License.
 */

/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 * Licensed under the Apache License, Version 2.0 (the "License").
 * You may not use this file except in compliance with the License.
 * A copy of the License is located at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * or in the "license" file accompanying this file. This file is distributed
 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
```

```
* express or implied. See the License for the specific language governing
* permissions and limitations under the license.
*/

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.UUID;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.ScheduledFuture;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

import org.apache.commons.lang3.ObjectUtils;
import org.apache.commons.lang3.RandomStringUtils;
import org.apache.commons.lang3.RandomUtils;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.slf4j.MDC;

import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cloudwatch.CloudWatchAsyncClient;
import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;
import software.amazon.awssdk.services.kinesis.KinesisAsyncClient;
import software.amazon.awssdk.services.kinesis.model.PutRecordRequest;
import software.amazon.kinesis.common.ConfigsBuilder;
import software.amazon.kinesis.common.KinesisClientUtil;
import software.amazon.kinesis.coordinator.Scheduler;
import software.amazon.kinesis.exceptions.InvalidStateException;
import software.amazon.kinesis.exceptions.ShutdownException;
import software.amazon.kinesis.lifecycle.events.InitializationInput;
import software.amazon.kinesis.lifecycle.events.LeaseLostInput;
import software.amazon.kinesis.lifecycle.events.ProcessRecordsInput;
import software.amazon.kinesis.lifecycle.events.ShardEndedInput;
import software.amazon.kinesis.lifecycle.events.ShutdownRequestedInput;
import software.amazon.kinesis.processor.ShardRecordProcessor;
import software.amazon.kinesis.processor.ShardRecordProcessorFactory;

public class SampleSingle {

    private static final Logger log = LoggerFactory.getLogger(SampleSingle.class);

    public static void main(String... args) {
        if (args.length < 1) {
            log.error("At a minimum, the stream name is required as the first argument. The
Region may be specified as the second argument.");
            System.exit(1);
        }

        String streamName = args[0];
        String region = null;
        if (args.length > 1) {
            region = args[1];
        }

        new SampleSingle(streamName, region).run();
    }

    private final String streamName;
    private final Region region;
    private final KinesisAsyncClient kinesisClient;
```

```
private SampleSingle(String streamName, String region) {
    this.streamName = streamName;
    this.region = Region.of(ObjectUtils.firstNonNull(region, "us-east-2"));
    this.kinesisClient =
KinesisClientUtil.createKinesisAsyncClient(KinesisAsyncClient.builder().region(this.region));
}

private void run() {
    ScheduledExecutorService producerExecutor =
Executors.newSingleThreadScheduledExecutor();
    ScheduledFuture<?> producerFuture =
producerExecutor.scheduleAtFixedRate(this::publishRecord, 10, 1, TimeUnit.SECONDS);

    DynamoDbAsyncClient dynamoClient =
DynamoDbAsyncClient.builder().region(region).build();
    CloudWatchAsyncClient cloudWatchClient =
CloudWatchAsyncClient.builder().region(region).build();
    ConfigsBuilder configsBuilder = new ConfigsBuilder(streamName, streamName,
kinesisClient, dynamoClient, cloudWatchClient, UUID.randomUUID().toString(), new
SampleRecordProcessorFactory());

    Scheduler scheduler = new Scheduler(
        configsBuilder.checkpointConfig(),
        configsBuilder.coordinatorConfig(),
        configsBuilder.leaseManagementConfig(),
        configsBuilder.lifecycleConfig(),
        configsBuilder.metricsConfig(),
        configsBuilder.processorConfig(),
        configsBuilder.retrievalConfig()
    );

    Thread schedulerThread = new Thread(scheduler);
    schedulerThread.setDaemon(true);
    schedulerThread.start();

    System.out.println("Press enter to shutdown");
    BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
    try {
        reader.readLine();
    } catch (IOException ioex) {
        log.error("Caught exception while waiting for confirm. Shutting down.", ioex);
    }

    log.info("Cancelling producer, and shutting down executor.");
    producerFuture.cancel(true);
    producerExecutor.shutdownNow();

    Future<Boolean> gracefulShutdownFuture = scheduler.startGracefulShutdown();
    log.info("Waiting up to 20 seconds for shutdown to complete.");
    try {
        gracefulShutdownFuture.get(20, TimeUnit.SECONDS);
    } catch (InterruptedException e) {
        log.info("Interrupted while waiting for graceful shutdown. Continuing.");
    } catch (ExecutionException e) {
        log.error("Exception while executing graceful shutdown.", e);
    } catch (TimeoutException e) {
        log.error("Timeout while waiting for shutdown. Scheduler may not have
exited.");
    }
    log.info("Completed, shutting down now.");
}

private void publishRecord() {
    PutRecordRequest request = PutRecordRequest.builder()
        .partitionKey(RandomStringUtils.randomAlphabetic(5, 20))
        .streamName(streamName)
```

```
        .data(SdkBytes.fromByteArray(RandomUtils.nextBytes(10)))
        .build();
    try {
        kinesisisClient.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);
    }
}
}
```

使用 Kinesis Data Streams API 开发具有增强型扇出功能的使用者

增强了扇出是一种 Amazon Kinesis Data Streams 功能，使用者利用此功能能够接收数据流（其中每分片每秒的专用吞吐量高达 2 MB 数据）中的记录。使用增强型扇出功能的使用者不必与接收流中数据的其他使用者争夺。有关更多信息，请参阅 [开发具有专用吞吐量的自定义使用者（增强扇出功能）](#) (p. 143)。

可以使用 API 操作构建在 Kinesis Data Streams 中使用增强型扇出功能的使用者。

使用 Kinesis Data Streams API 注册使用增强型扇出功能的使用者

1. 调用 [RegisterStreamConsumer](#) 将您的应用程序注册为使用增强型扇出功能的使用者。Kinesis Data Streams 为使用者生成一个 Amazon 资源名称 (ARN) 并在响应中返回此名称。
2. 要开始侦听特定分片，请将调用中的使用者 ARN 传递给 [SubscribeToShard](#)。之后，Kinesis Data Streams 会以类型事件的形式将记录从分片中的记录推送到您。[SubscribeToShardEvent](#) 通过 HTTP/2 连接。此连接将保持打开状态长达 5 分钟。如果要在通过调用 [SubscribeToShard](#) 返回的 future 正常或异常完成之后继续接收分片中的记录，请再次调用 [SubscribeToShard](#)。

Note

[SubscribeToShard](#) 当前分片结束时，API 还会返回当前分片的子分片列表。

3. 要取消注册使用增强型扇出功能的使用者，请调用 [DeregisterStreamConsumer](#)。

以下代码是一个示例，演示如何为使用者订阅分片、定期续订订阅以及处理事件。

```
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);
    }
}
```

如果`event.ContinuationSequenceNumber`回报`null`，它表示发生了涉及该分片的分片拆分或合并。这个碎片现在在CLOSED状态，并且您已经从该分片中读取了所有可用的数据记录。在这种情况下，根据上面的例子，您可以使用`event.childShards`了解通过拆分或合并创建的正在处理的分片的新子分片。有关更多信息，请参阅 [ChildShard](#)。

使用与数据交互Amazon架 Glue 注册表

您可以将 Kinesis 数据流与AmazonGlue 模式注册表。这些区域有：Amazon使用 Glue 架构注册表可以集中发现、控制和演变架构，同时确保已注册架构持续验证生成的数据。架构定义了数据记录的结构和格式。

架构是用于可靠数据发布、使用或存储的版本化规范。这些区域有：AmazonGlue 模式注册表使你能够改进 end-to-end 流媒体应用程序中的数据质量和数据治理。有关更多信息，请参阅 [Amazon 架 Glue 注册表](#)。设置此集成的方法之一是通过 GetRecords 在中提供 Kinesis Data Streams API AmazonJava 开发工具包。

有关如何使用设置 Kinesis Data Streams 与架构注册表集成的详细说明 GetRecords 请参阅中的“使用 Kinesis Data Streams API 与数据交互”部分 [使用案例：将 Amazon Kinesis Data Streams 与 Amazon 架 Glue 注册表](#)。

使用 Amazon Web Services Management Console 管理具有增强扇出功能的使用者

使用者增强的扇出在 Amazon Kinesis Data Streams 中可接收每分片每秒专用吞吐量高达 2 MB 数据的数据流中的记录。有关更多信息，请参阅 [开发具有专用吞吐量的自定义使用者（增强扇出功能）](#) (p. 143)。

可以使用 Amazon Web Services Management Console 查看已向特定流注册以使用增强型扇出功能的所有使用者的列表。对于每个此类使用者，都可以查看诸如 ARN、状态、创建日期和监控指标等详细信息。

在控制台上查看已注册为使用增强型扇出功能的使用者、其状态、创建日期和指标

1. 登录到 Amazon Web Services Management Console，然后通过以下网址打开 Kinesis 控制台：<https://console.aws.amazon.com/kinesisvideo/home>。
2. 在导航窗格中，选择 Data Streams (数据流)。
3. 选择 Kinesis Data Streams 以查看其详细信息。
4. 在流的详细信息页面上，选择 Enhanced fan-out (增强型扇出功能) 选项卡。
5. 选择使用者以查看其名称、状态和注册日期。

取消注册使用者

1. 打开 Kinesis 控制台，网址为：<https://console.aws.amazon.com/kinesis>。
2. 在导航窗格中，选择 Data Streams (数据流)。
3. 选择 Kinesis Data Streams 以查看其详细信息。
4. 在流的详细信息页面上，选择 Enhanced fan-out (增强型扇出功能) 选项卡。
5. 选中要取消注册的每个使用者名称左侧的复选框。
6. 选择 Deregister consumer (取消注册使用者)。

将使用者从 KCL 1.x 迁移到 KCL 2.x

本主题介绍 Kinesis 客户端库 (KCL) 的版本 1.x 和 2.x 之间的区别。它还向您展示如何将使用者从 KCL 的版本 1.x 迁移到版本 2.x。在迁移您的客户端后，它将从最后一个检查点位置开始处理记录。

KCL 的版本 2.0 引入了以下接口更改：

KCL 接口更改

KCL 1.x 接口	KCL 2.0 接口
com.amazonaws.services.kinesis.clientlibrary.interfaces.v1.IRecordProcessor	software.amazon.kinesis.processor.ShardRecordProcessor
com.amazonaws.services.kinesis.clientlibrary.interfaces.v1.IRecordProcessorFactory	software.amazon.kinesis.processor.ShardRecordProcessorFactory
com.amazonaws.services.kinesis.clientlibrary.interfaces.v1.IShutdownNotificationAware	software.amazon.kinesis.processor.ShardRecordProcessorFactory

主题

- [迁移记录处理器 \(p. 151\)](#)
- [迁移记录处理器工厂 \(p. 154\)](#)
- [迁移工作程序 \(p. 155\)](#)
- [配置 Amazon Kinesis 客户端 \(p. 156\)](#)
- [闲置时间删除 \(p. 158\)](#)
- [客户端配置删除 \(p. 158\)](#)

迁移记录处理器

以下示例显示了为 KCL 1.x 实施的记录处理器：

```
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();
    }
}
```

迁移记录处理器类

1. 将接口从

`com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor`
和
`com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IShutdownNotificationAware`
更改为 `software.amazon.kinesis.processor.ShardRecordProcessor`，如下所示：

```
// 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. 更新 import 和 initialize 方法的 processRecords 语句。

```
// 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. 使用以下新方法替换 shutdown 方法：leaseLost、shardEnded 和 shutdownRequested。

```
// @Override
// public void shutdownRequested(IRecordProcessorCheckpoint checkpoint) {
//     //
//     // This is moved to shardEnded(...)
//     //
//     try {
//         checkpoint.checkpoint();
//     } catch (ShutdownException | InvalidStateException e) {
//         //
//         // Swallow exception
//         //
//         e.printStackTrace();
//     }
// }

@Override
public void leaseLost(LeaseLostInput leaseLostInput) {

}

@Override
public void shardEnded(ShardEndedInput shardEndedInput) {
    try {
        shardEndedInput.checkpointer().checkpoint();
    } catch (ShutdownException | InvalidStateException e) {
        //
        // Swallow the exception
        //
        e.printStackTrace();
    }
}
```

```

    }

    // @Override
    // public void shutdownRequested(IRecordProcessorCheckpoint checkpoint) {
    //     //
    //     // This is moved to shutdownRequested(ShutdownRequestedInput)
    //     //
    //     try {
    //         checkpoint.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();
        }
    }
}

```

下面是记录处理器类的更新版本。

```

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();
    }
}
}
```

迁移记录处理器工厂

记录处理器工厂负责在获得租赁时创建记录处理器。下面是 KCL 1.x 工厂的示例。

```
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();
    }
}
```

迁移记录处理器工厂

1. 将已实施的接口从 `com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessorFactory` 更改为 `software.amazon.kinesis.processor.ShardRecordProcessorFactory`，如下所示。

```
// 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. 更改 `createProcessor` 的返回签名。

```
// public IRecordProcessor createProcessor() {
public ShardRecordProcessor shardRecordProcessor() {
```

下面是 2.0 中的记录处理器工厂的示例：

```
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();
    }
}
```

迁移工作程序

在 2.0 版本的 KCL 中，名为 Scheduler 中，替换 Worker 类。下面是 KCL 1.x 工作程序的示例。

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

迁移工作程序

1. 将 Worker 类的 import 语句更改为 Scheduler 和 ConfigsBuilder 类的导入语句。

```
// import com.amazonaws.services.kinesis.clientlibrary.lib.worker.Worker;
import software.amazon.kinesis.coordinator.Scheduler;
import software.amazon.kinesis.common.ConfigsBuilder;
```

2. 创建 ConfigsBuilder 和 Scheduler，如下示例所示。

建议您使用 KinesisClientUtil 创建 KinesisAsyncClient，并在 KinesisAsyncClient 中配置 maxConcurrency。

Important

Amazon Kinesis 户端可能会看到延迟大幅增加，除非您将 KinesisAsyncClient 配置为具有足够高的 maxConcurrency，以允许所有租期以及额外使用 KinesisAsyncClient。

```
import java.util.UUID;

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;
import software.amazon.awssdk.services.cloudwatch.CloudWatchAsyncClient;
import software.amazon.awssdk.services.kinesis.KinesisAsyncClient;
import software.amazon.kinesis.common.ConfigsBuilder;
import software.amazon.kinesis.common.KinesisClientUtil;
import software.amazon.kinesis.coordinator.Scheduler;

...

Region region = Region.AP_NORTHEAST_2;
KinesisAsyncClient kinesisClient =
    KinesisClientUtil.createKinesisAsyncClient(KinesisAsyncClient.builder().region(region));
DynamoDbAsyncClient dynamoClient =
    DynamoDbAsyncClient.builder().region(region).build();
```

```
CloudWatchAsyncClient cloudWatchClient =
    CloudWatchAsyncClient.builder().region(region).build();

ConfigsBuilder configsBuilder = new ConfigsBuilder(streamName, applicationName,
    kinesisClient, dynamoClient, cloudWatchClient, UUID.randomUUID().toString(), new
    SampleRecordProcessorFactory());

Scheduler scheduler = new Scheduler(
    configsBuilder.checkpointConfig(),
    configsBuilder.coordinatorConfig(),
    configsBuilder.leaseManagementConfig(),
    configsBuilder.lifecycleConfig(),
    configsBuilder.metricsConfig(),
    configsBuilder.processorConfig(),
    configsBuilder.retrievalConfig()
);
```

配置 Amazon Kinesis 客户端

使用 2.0 版本的 Kinesis 客户端库，客户端的配置已多单个配置类 (KinesisClientLibConfiguration) 到六个配置类。下表描述了迁移。

配置字段及其新类

原始字段	新配置类	描述
applicationName	ConfigsBuilder	此 KCL 应用程序的名称。用作 tableName 和 consumerName 的默认名称。
tableName	ConfigsBuilder	允许覆盖用于 Amazon DynamoDB 租赁表的表名称。
streamName	ConfigsBuilder	此应用程序从其中处理记录的流的名称。
kinesisEndpoint	ConfigsBuilder	此选项已删除。请参阅“客户端配置删除”。
dynamoDBEndpoint	ConfigsBuilder	此选项已删除。请参阅“客户端配置删除”。
initialPositionInStreamExtention	PollingConfig	分片中 KCL 开始获取记录的位置，从应用程序的初始运行开始。
kinesisCredentials	ConfigsBuilder	此选项已删除。请参阅“客户端配置删除”。
dynamoDBCredentials	ConfigsBuilder	此选项已删除。请参阅“客户端配置删除”。
cloudWatchCredentials	ConfigsBuilder	此选项已删除。请参阅“客户端配置删除”。
failoverTimeMillis	LeaseManagementConfig	您可以将租赁所有者视为已失败之前必须经过的毫秒数。
workerIdentifier	ConfigsBuilder	表示应用程序处理器的这种实例化的唯一标识符。此值必须唯一。
shardSyncIntervalMillis	LeaseManagementConfig	轮询同步调用之间的时间。
maxRecords	PollingConfig	允许设置 Kinesis 返回的最大记录数。
idleTimeBetweenReadsInMillis	CoordinatorConfig	此选项已删除。请参阅“闲置时间删除”。
callProcessRecordsEvenIfEmpty	ProcessorConfig	如果设置，即使在 Kinesis 中未提供任何记录时也将调用记录处理器。
parentShardPollIntervalInMillis	CoordinatorConfig	记录处理器应轮询多少时间才能查看是否已完成父分片。

原始字段	新配置类	描述
cleanupLeasesUponShutdown	LeaseManagementConfig	如果设置，只要子租赁已开始处理，即可删除租赁。
ignoreUnexpectedChildShards	LeaseManagementConfig	如果设置，将忽略具有打开的分片的子分片。这主要用于 DynamoDB Streams。
kinesisClientConfig	ConfigsBuilder	此选项已删除。请参阅“客户端配置删除”。
dynamoDBClientConfig	ConfigsBuilder	此选项已删除。请参阅“客户端配置删除”。
cloudWatchClientConfig	ConfigsBuilder	此选项已删除。请参阅“客户端配置删除”。
taskBackoffTimeMillis	LifecycleConfig	等待重试失败任务的时间。
metricsBufferTimeMillis	MetricsConfig	控件CloudWatch指标发布。
metricsMaxQueueSize	MetricsConfig	控件CloudWatch指标发布。
metricsLevel	MetricsConfig	控件CloudWatch指标发布。
metricsEnabledDimensions	MetricsConfig	控件CloudWatch指标发布。
validateSequenceNumbersBeforeCheckpointing	ConfigsBuilder	此选项已删除。请参阅“检查点序列号验证”。
regionName	ConfigsBuilder	此选项已删除。请参阅“客户端配置删除”。
maxLeasesForWorker	LeaseManagementConfig	应用程序的单个实例应接受的最大租赁数量。
maxLeasesToStealAtOneTime	LeaseManagementConfig	应用程序一次应尝试窃取的最大租赁数量。
initialLeaseTableReadWriteCapacity	LeaseManagementConfig	如果Kinesis 客户端库需要创建新的 DynamoDB 租赁表，DynamoDB 将读取使用的 IOP。
initialLeaseTableWriteCapacity	LeaseManagementConfig	如果Kinesis 客户端库需要创建新的 DynamoDB 租赁表，DynamoDB 将读取使用的 IOP。
initialPositionInStream	StreamManagementConfig	应用程序应在流中开始的初始位置。此值仅在创建初始租赁时使用。
skipShardSyncAtWorkerInitialization	CoordinatorConfig	如果租赁表包含现有租赁，请禁用同步的分片数据。TODO：KinesisEco-438
shardPrioritization	CoordinatorConfig	要使用的分片优先级。
shutdownGraceMillis	不适用	此选项已删除。请参阅MultiLang删除。
timeoutInSeconds	不适用	此选项已删除。请参阅MultiLang删除。
retryGetRecordsInSeconds	RollingConfig	配置之间的延迟GetRecords尝试失败。
maxGetRecordsThreads	RollingConfig	用于的线程池大小GetRecords。
maxLeaseRenewalThreads	LeaseManagementConfig	控制租赁续订线程池的大小。您的应用程序可以容纳的租赁越多，此池应该就越大。
recordsFetcherFactory	RollingConfig	允许替换用于创建从流中检索的提取程序的工厂。
logWarningForTaskAfterMillis	LifecycleConfig	任务尚未完成的情况下在记录警告之前要等待的时长。
listShardsBackoffTimeMillis	RollingConfig	发生故障时在调用 ListShards 之间要等待的时间（以毫秒为单位）。

原始字段	新配置类	描述
maxListShardsRetryAttempts	AttemptedConfig	listShards 在放弃之前重试的最长时间。

闲置时间删除

在 1.x 版本的 KCL 中，idleTimeBetweenReadsInMillis 相当于两个数量：

- 任务分派检查之间的时间量。您现在可以通过设置 CoordinatorConfig#shardConsumerDispatchPollIntervalMillis 来在任务之间配置此时间。
- 从 Kinesis Data Streams 中未返回任何记录时的睡眠时间量。在版本 2.0 中，带增强型扇出功能的记录是从其各自的检索器中推送的。分片使用器上的活动仅发生在推送的请求到达时。

客户端配置删除

在 2.0 版本中，KCL 不再创建客户端。这取决于用户提供有效的客户端。进行此更改后，已删除控制客户端创建的所有配置参数。如果需要这些参数，您可以在向 ConfigsBuilder 提供客户端之前在客户端上设置它们。

已删除字段	等效配置
kinesisEndpoint	使用以下首选终端节点配置开发工具包 KinesisAsyncClient : KinesisAsyncClient.builder().endpointOverride(URI.create("<kinesis endpoint>")).build()。
dynamoDBEndpoint	使用以下首选终端节点配置开发工具包 DynamoDbAsyncClient : DynamoDbAsyncClient.builder().endpointOverride(URI.create("<dynamodb endpoint>")).build()。
kinesisClientConfiguration	使用以下所需配置来配置开发工具包 KinesisAsyncClient : KinesisAsyncClient.builder().overrideConfiguration(<your configuration>).build()。
dynamoDBClientConfiguration	使用以下所需配置来配置开发工具包 DynamoDbAsyncClient : DynamoDbAsyncClient.builder().overrideConfiguration(<your configuration>).build()。
cloudWatchClientConfiguration	使用以下所需配置来配置开发工具包 CloudWatchAsyncClient : CloudWatchAsyncClient.builder().overrideConfiguration(<your configuration>).build()。
regionName	使用首选区域配置开发工具包。这对所有开发工具包客户端均相同。例如：KinesisAsyncClient.builder().region(Region.US_WEST_2).build()。

使用其他 Amazon 服务

以下是可以直接与 Kinesis Data Streams 集成以读取 Kinesis Data Streams 数据的其他 Amazon 服务列表：

主题

- [EMR \(p. 159\)](#)
- [使用亚马逊 EventBridge 管道 \(p. 159\)](#)

- [使用 Amazon Glue \(p. 159\)](#)
- [Amazon Redshift \(p. 159\)](#)

EMR

Amazon EMR 集群可以使用 Hadoop 生态系统中的熟悉工具（如 Hive、Pig、Hadoop Streaming API 和 Cascading）直接读取和处理 Amazon Kinesis 流。MapReduce 您还可以将 Amazon Kinesis 中的实时数据与正在运行的集群中 Amazon S3、Amazon DynamoDB 和 HDFS 上的现有数据进行连接。您可以直接将 Amazon EMR 中的数据加载到 Amazon S3 或 DynamoDB 来进行后处理。

有关更多信息，请参阅 [Amazon EMR 版本指南中的 Amazon Kinesis](#)。

使用亚马逊 EventBridge 管道

Amazon EventBridge 管道将 Kinesis Data Streams 作为源。Amazon EventBridge 管道通过可选 point-to-point 的转换、筛选和丰富步骤，帮助您在活动制作者和消费者之间创建集成。您可以使用 EventBridge Pipes 接收 Kinesis 数据流中的记录，也可以选择对这些记录进行筛选或增强，然后将其发送到一个可用目的地（包括 Kinesis Data Streams）进行处理。

有关更多信息，请参阅《[亚马逊 EventBridge 发布指南](#)》中的 [Amazon Kinesis 直播作为来源](#)。

使用 Amazon Glue

使用 Amazon Glue 与 Amazon Kinesis Data Streams 创建连续运行并使用来自 Amazon Kinesis Data Streams 的串流提取、转换和加载（ETL）任务，例如 Amazon Kinesis Data Analytics 这些任务会清理并转换数据，然后将结果加载到 Amazon S3 数据湖或 JDBC 数据存储中。

有关更多信息，请参阅 Amazon Glue 发行指南 Amazon Glue [中的流式传输 ETL 任务](#)。

Amazon Redshift

Amazon Redshift 支持从 Amazon Kinesis Data Streams 串流 Amazon Redshift 串流摄取功能可以以低延迟、高速度的方式将流数据摄取到 Amazon Redshift 实体化视图中。Amazon Redshift 串流摄取不再需要将数据暂存 Amazon S3 Redshift 之前，将数据暂存到 Amazon Redshift。

有关更多信息，请参阅《[Amazon Redshift 发行指南](#)》中的 [流媒体采集](#)。

使用第三方集成

您可以使用以下与 Kinesis 数据流集成的第三方选项之一从 Kinesis Data Streams 读取数据：

主题

- [Apache Flink \(p. 160\)](#)
- [Adobe 体验平台 \(p. 160\)](#)
- [Apache Druid \(p. 160\)](#)
- [Apache Spark \(p. 160\)](#)
- [Databricks \(p. 160\)](#)
- [Kafka \(p. 160\)](#)
- [Kinesis \(p. 160\)](#)

- [塔伦德 \(p. 160\)](#)

Apache Flink

Apache Flink 是一个热门的框架和分布式处理引擎，用于对无界和有界数据流进行有状态计算。有关使用 Apache Flink 使用 Kinesis Data Streams 的更多信息，请参阅 [Amazon Kinesis Data Streams 连接器](#)。

Adobe 体验平台

Adobe Experience Platform 使组织能够集中和标准化来自任何系统的客户数据。然后，它应用数据科学和机器学习来显著改善丰富的个性化体验的设计和交付。有关使用 Adobe Experience 平台使用 Kinesis Data Streams 的更多信息，请参阅 [Amazon Kinesis 摄取](#)。

Apache Duid

Druid 是一个高性能的实时分析数据库，可大规模和负载下提供有关流媒体和批处理数据的亚秒级查询。有关使用 Apache Druid 提取 Kinesis Data Streams 的更多信息，请参阅 [Amazon Kinesis 摄取](#)。

Apache Spark

Apache Spark 是用于大规模数据处理的统一分析引擎。它提供 Java、Scala、Python 和 R 中的高级 API，以及支持通用执行图的优化引擎。有关使用 Spark Streaming 使用 Kinesis Data Streams 的更多信息，请参阅 [Spark Streaming + Kinesis 集成](#)。

Databricks

Databricks 是一个基于云的平台，为数据工程、数据科学和机器学习提供协作环境。有关使用 Databricks 使用 Kinesis Data Streams 的更多信息，请参阅 [适用于结构化流的 Kinesis 连接器](#)。

Kafka

Confluent Platform 建立在 Kafka 之上，并提供其他特性和功能，帮助企业构建和管理实时数据管道和流媒体应用程序。有关使用 Confluent 平台使用 Kinesis Data Streams 的更多信息，请参阅 [适用于 Confluent 平台的 Amazon Kinesis Source Connect](#) or。

Kinesumer

Kinesumer 是一个 Go 客户端，为亚马逊 Kinesis Data Streams 实现了客户端分布式消费者组客户端。有关更多信息，请参阅 [Kinesumer Github 存储库](#)。

塔伦德

Talend 是一款数据集成和管理软件，允许用户以可扩展和高效的方式收集、转换和连接来自各种来源的数据。有关使用 Talend 使用 Kinesis Data Streams 的更多信息，请参阅 [将 talend 连接到 Amazon Kinesis 直播](#)。

排查 Kinesis Data Streams 使用者的问题

以下部分为您在与 Amazon Kinesis Data Streams 消费者合作时可能发现的一些常见问题提供了解决方案。

- [使用 Kinesis 客户端库时会跳过某些 Kinesis 数据流记录 \(p. 161\)](#)
- [属于同一分片的记录通过不同的记录处理器同时处理 \(p. 161\)](#)
- [使用者应用程序的读取速率比预期的慢 \(p. 161\)](#)
- [GetRecords即使流中有数据，也会返回空记录数组 \(p. 162\)](#)
- [分片迭代器意外过期 \(p. 162\)](#)
- [使用者记录处理滞后 \(p. 162\)](#)
- [未授权的 KMS 主密钥权限错误 \(p. 163\)](#)
- [消费者常见问题、疑问和疑难解答建议 \(p. 163\)](#)

使用 Kinesis 客户端库时会跳过某些 Kinesis 数据流记录

跳过记录的最常见原因是未处理从 `processRecords` 引发的异常。Kinesis 客户端库 (KCL) 依赖您的 `processRecords` 代码来处理因处理数据记录而产生的任何异常。抛出的任何异常都会被 `processRecords` 被 KCL 吸收。为避免在反复出现的故障时无限重试，KCL 不会重新发送异常时处理的批次记录。然后，KCL 在不重新启动记录处理器的情况下调用 `processRecords` 用下一批数据记录。这有效地导致使用者应用程序观察到跳过的记录。要防止跳过记录，请适当处理 `processRecords` 中的所有异常。

属于同一分片的记录通过不同的记录处理器同时处理

对于任何正在运行的 Kinesis 客户端库 (KCL) 应用程序，一个分片只有一个所有者。但是，多个记录处理器可以临时处理同一分片。对于断开网络连接的工作器实例，KCL 假设在故障转移时间到期后，无法访问的工作线程不再处理记录，并指示其他工作线程实例接管。在一段很短的时间内，新的记录处理器和来自无法联系的工作程序的记录处理器可能都会处理来自同一个分片的数据。

您应该根据您的应用程序设置适当的故障转移时间。对于低延迟应用程序，默认值 10 秒可以代表您可以等待的最长时间。但是，在一些情况下，如您预计会有一些连接问题（如跨地理区域打电话）并且连接可能较频繁地中断时，这个数字设置可能就过低了。

您的应用程序应预料到并处理这种情况，尤其是因为网络连接通常会恢复到之前无法访问的工作程序。如果一个记录处理器让另一个记录处理器接手其分片，它必须处理以下两种情况才能顺利执行关闭：

1. 当前对的调用完成后 `processRecords`，KCL 调用记录处理器上的关机方法，关机原因为“ZOMBIE”。您的记录处理器应视情况清除所有资源然后退出。
2. 当您尝试从“僵尸”工作人员那里检查点时，KCL 会投掷 `ShutdownException`。收到此异常后，您的代码应完全退出当前方法。

有关更多信息，请参阅[处理重复记录 \(p. 165\)](#)：

使用者应用程序的读取速率比预期的慢

读取吞吐量低于预期的最常见原因如下：

1. 多个使用者应用程序的总读取量超过每个分片的限制。有关更多信息，请参阅[配额和限制 \(p. 6\)](#)：在这种情况下，增加 Kinesis 数据流中的分片数量。
2. 指定 `GetRecords` 每次呼叫最大次数的限制可能已配置为低值。如果您正在使用 KCL，您可能已经使用一个较低的 `maxRecords` 属性值配置了工作程序。一般来说，我们推荐对此属性使用系统默认值。
3. 出于很多可能的原因，您的 `processRecords` 调用内的逻辑花费的时间可能超过预期；该逻辑可能是 CPU 使用率高、I/O 阻止或同步出现瓶颈。要测试是否如此，请对空的记录处理器进行测试运行并比较读取吞吐量。有关如何跟踪传入数据的信息，请参阅[重新分片、扩展和并行处理 \(p. 164\)](#)。

如果您只有一个使用器应用程序，那么读取速率比放入速率至少高两倍始终是可能的。这是因为您每秒最多可以写入 1,000 条记录进行写入，最大总数据写入速率为每秒 1 MB（包括分区键）。每个打开的分片每秒最多可支持 5 个事务进行读取，最大总数据读取速率为每秒 2 MB。请注意，每次读取（GetRecords 调用）都将获取一批记录。GetRecords 返回的数据的大小因分片的使用率而异。GetRecords 可以返回的最大数据大小为 10 MB。如果某个调用返回了该限制，在接下来 5 分钟内进行的后续调用将引发 `ProvisionedThroughputExceededException`。

GetRecords即使流中有数据，也会返回空记录数组

使用或获取记录是一种拉取模型。开发人员应该连续循环调用 [GetRecords](#)，不会退缩。每个 GetRecords 调用还将返回一个 `ShardIterator` 值，该值必须在循环的下一个迭代中使用。

GetRecords 操作不会阻止。相反，它将立即返回一些相关数据记录或一个空的 `Records` 元素。在两种情况下，将返回空的 `Records` 元素：

1. 目前分片中沒有更多数据。
2. `ShardIterator` 指向的分片部分附近没有数据。

后一种情况很微妙，但却是避免在检索数据时搜寻时间无止境（延迟）的一种必要的设计折衷。因此，流使用应用程序应循环并调用 `GetRecords`，并且理所当然地处理空记录。

在生产场景中，仅当 `NextShardIterator` 值为 `NULL` 时，才应退出连续循环。当 `NextShardIterator` 为 `NULL` 时，这意味着当前分片已关闭，`ShardIterator` 值的指向应越过最后一条记录。如果使用应用程序从不调用 `SplitShard` 或 `MergeShards`，分片将保持打开状态，并且对 `GetRecords` 的调用从不返回为 `NextShardIterator` 的 `NULL` 值。

如果您使用 Kinesis 客户端库 (KCL)，则上述消费模式会被抽象出来。这包括自动处理一组动态变化的分片。使用 KCL，开发人员仅提供处理传入记录的逻辑。这是可能的，因为该库会为您连续调用 `GetRecords`。

分片迭代器意外过期

每个 `GetRecords` 请求返回新的分片迭代器（作为 `NextShardIterator`），然后您可以将其用于下一个 `GetRecords` 请求（作为 `ShardIterator`）。此分片迭代器一般来说不会在您使用前过期。不过，您可能会发现，由于您超过 5 分钟没有调用 `GetRecords`，或者您执行了使用器应用程序的重新启动操作，该分片迭代器会过期。

如果分片迭代器在您使用之前立即过期，这可能表明 Kinesis 使用的 DynamoDB 表没有足够的容量来存储租约数据。如果您的分片数量很多，则很可能发生这种情况。要解决此问题，请增加分配给分片表的写入容量。有关更多信息，请参阅[使用租赁表跟踪 KCL 消费者应用程序处理的碎片 \(p. 109\)](#)：

使用者记录处理滞后

对于大多数使用案例，使用者应用程序从流中读取最新数据。在特定情况下，使用者读取可能会滞后，而您可能并不希望出现这种情况。在确定您的使用者读取滞后多久后，请查看使用者滞后的最常见原因。

从 `GetRecords.IteratorAgeMilliseconds` 指标开始，该指标跟踪流中所有分片和使用者的读取位置。请注意，如果迭代器的使用寿命超过保留期的 50%（默认情况下为 24 小时，最多可配置为 365 天），则存在因记录过期而丢失数据的风险。一种快速的权宜之计是增加保留期。这会在您进一步对问题进行故障排除时防止重要数据丢失。有关更多信息，请参阅[使用 Amazon CloudWatch 监控 Amazon Kinesis Data Streams \(p. 168\)](#)：接下来，使用 Kinesis Client Library (KCL) 发出的自定义 CloudWatch 指标，确定您的消费者应用程序从每个分片读取的内容落后了多远。`MillisBehindLatest` 有关更多信息，请参阅[使用 Amazon CloudWatch 监控 Kinesis 客户端库 \(p. 182\)](#)：

下面是使用者滞后的最常见原因：

- `GetRecords.IteratorAgeMilliseconds` 或 `MillisBehindLatest` 突然发生大幅提升，通常表明临时性问题，例如下游应用程序的 API 操作失败。如果任何一个指标持续指示此行为，您应该调查这些突然增长的原因。
- 这些指标的逐步增大表明，由于处理记录的速度不够快，使用者无法与流保持同步。此行为最常见的根本原因是没有足够的物理资源，或者记录处理逻辑没有随着流吞吐量的增大而进行扩展。您可以通过查看 KCL 发出的与 `processTask` 操作相关的其他自定义 CloudWatch 指标来验证此行为，包括 `RecordProcessor.processRecords.TimeSuccess`、和 `RecordsProcessed`
 - 如果您发现与吞吐量上升相关的 `processRecords.Time` 指标发生增长，则应该分析记录处理逻辑，以确定为什么逻辑没有随吞吐量的增长而扩展。
 - 如果您发现与吞吐量上升无关的 `processRecords.Time` 值发生增长，请检查您是否在关键路径中执行了任何阻塞性调用，这通常会导致记录处理速度下降。替代方法是通过增加分片数来提高并行度。最后，请确认需求高峰期间在底层处理节点上，您有足够数量的物理资源（内存、CPU 使用率等）。

未授权的 KMS 主密钥权限错误

当使用者应用程序从加密流读取但没有 KMS 主密钥的权限时，会发生此错误。要向应用程序分配访问 KMS 密钥的权限，请参阅在 KMS 中[使用密钥策略](#)和在 [Amazon KMS 中使用 IAM 策略](#)。Amazon

消费者常见问题、疑问和疑难解答建议

- [为什么 Kinesis 数据流触发器无法调用我的 Lambda 函数？](#)
- [如何检测 and 解决 Kinesis 数据流中的 `ReadProvisionedThroughputExceeded` 异常？](#)
- [为什么我在使用 Kinesis 数据流时会遇到高延迟问题？](#)
- [为什么我的 Kinesis 数据流返回 500 内部服务器错误？](#)
- [如何解决 Kinesis Data Streams 的 KCL 应用程序被阻塞或卡住的问题？](#)
- [我可以在同一个亚马逊 DynamoDB 表中使用不同的亚马逊 Kinesis 客户端库应用程序吗？](#)

Amazon Kinesis Data Streams 使用者的高级主题

了解如何优化您的 Amazon Kinesis Data Streams 使用者。

目录

- [低延迟处理 \(p. 163\)](#)
- [使用 Amazon Lambda 与 Kinesis 创建者库一起使用 \(p. 164\)](#)
- [重新分片、扩展和并行处理 \(p. 164\)](#)
- [处理重复记录 \(p. 165\)](#)
- [处理启动、关闭和限制 \(p. 166\)](#)

低延迟处理

传播延迟被定义为 end-to-end 从将记录写入到流到使用者应用程序读取该记录的延迟。此延迟会因各种因素而变化，但主要受使用者应用程序的轮询间隔影响。

对于大多数应用程序，我们建议针对每个应用程序每秒轮询每个分片一次。这使您能够具有并行处理流的多个使用者应用程序，而不会达到 Amazon Kinesis Data Streams 5 的限制。GetRecords 每秒调用。此外，若要处理大批量的数据，降低您的应用程序中的网络和其他下游延迟时往往更高效。

KCL 默认值被设置为遵循每 1 秒轮询一次的最佳实践。此默认值导致了通常少于 1 秒的平均传播延迟。

Kinesis Data Streams 记录在写入后便立即可被读取。有一些需要利用此延迟并且在流中的数据可用时立即需要使用它的使用案例。您可通过覆盖 KCL 默认设置来更频繁地进行轮询，从而显著降低传播延迟，如下示例所示。

Java KCL 配置代码：

```
kinesisClientLibConfiguration = new
    KinesisClientLibConfiguration(applicationName,
        streamName,
        credentialsProvider,

workerId).withInitialPositionInStream(initialPositionInStream).withIdleTimeBetweenReadsInMillis(250);
```

Python 和 Ruby KCL 的属性文件设置：

```
idleTimeBetweenReadsInMillis = 250
```

Note

因为 Kinesis Data Streams 的限制为 5GetRecords每个分片每秒调用，将idleTimeBetweenReadsInMillis低于 200ms 的属性可能会导致您的应用程序观察ProvisionedThroughputExceededException例外。如果这类异常过多，则可能导致指数退避，并因此导致处理过程中出现重大的意外延迟。如果您将此属性设置为 200 ms 或更大并且具有多个正在处理的应用程序，则会遇到类似的限制。

使用Amazon Lambda与 Kinesis 创建者库一起使用

这些区域有：[Kinesis 创建者库](#)(KPL) 将较小的用户格式化记录聚合为较大的记录（最大为 1 MB），以更好地利用 Amazon Kinesis Data Streams 吞吐量。虽然 Java KCL 支持取消聚合这些记录，但在使用时，您需要使用特殊模块来取消聚合记录。Amazon Lambda作为你的直播的消费者。您可以从以下处获取必要的项目代码和说明：[GitHub在Kinesis 生产者库分解模块AmazonLambda](#)。本项目中的组件使您能够在其中处理 KPL 序列化数据。Amazon Lambda，在 Java、Node.js 和 Python 中。这些组件可用作[多语言 KCL 应用程序](#)的一部分。

重新分片、扩展和并行处理

利用重新分片，您可以增加或减少流中的分片数量，以便适应流过流的数据的速率的变化。重新分片通常是由监控分片数据处理指标的管理应用程序执行的。尽管 KCL 本身不启动重新分片操作，但它旨在适应由于重新分片而生成的分片数量的变化。

正如中所述[使用租赁表跟踪 KCL 消费者应用程序处理的碎片 \(p. 109\)](#)，KCL 会使用 Amazon DynamoDB 表跟踪流中的分片。当由于重新分片而创建新的分片时，KCL 会发现新的分片并在该表中填充新行。工作程序将自动发现新的分片并创建处理器以处理来自分片的数据。KCL 还将在流中的分片分配到所有可用工作程序和记录处理器。

KCL 确保在重新分片之前存在于分片中的任何数据首先处理。在处理该数据后，新分片中的数据将发送到记录处理器。通过这种方式，KCL 将保留为特定分区键将数据记录添加到流的顺序。

例如：重新分片、扩展和并行处理

以下示例将演示 KCL 如何帮助您处理扩展和重新分片：

- 例如，如果您的应用程序正在 1 个 EC2 实例上运行，并且正在处理 1 个包含 4 个分片的 Kinesis Data Streams。这一个实例有 1 个 KCL 工作程序和 4 个记录处理器（每个分片有 1 个记录处理器）。这 4 个记录处理器在同一进程内并行运行。
- 接下来，如果您扩展应用程序以使用其他实例，您将有 2 个处理 1 个包含 4 个分片的流的实例。当 KCL 工作程序在第二个实例上启动时，它会与第一个实例进行负载均衡，以便让每个实例现在处理两个分片。

- 如果您随后决定将 4 个分片拆分为 5 个分片。KCL 再次跨实例协调处理：一个实例处理 3 个分片，另一个实例处理两个分片。类似协调在合并分片时出发生。

通常，在使用 KCL 时，您应确保实例的数量不超过分片的数量（故障待机除外）。每个分片正好由一个 KCL 工作程序处理并且正好有一个对应的记录处理器，因此您永远不需要多个实例来处理一个分片。但是，一个工作程序可处理任意数量的分片，因此分片的数量超过实例的数量没有关系。

要扩展您的应用程序中的处理，您应测试以下方法的组合：

- 增加实例大小（因为所有记录处理器都在进程内并行运行）
- 增加实例的数量，最多为开放分片的最大数量（因为分片可以单独处理）
- 增加分片的数量（这会提高并行机制的级别）

请注意，您可以使用 Auto Scaling 基于适当的指标自动扩展您的实例。有关更多信息，请参阅 [Amazon EC2 Auto Scaling 用户指南](#)。

当重新分片增加了流中的分片数时，记录处理器的数量的相应增加会增加托管处理器的 EC2 实例上的负载。如果实例为 Auto Scaling 组的一部分，并且负载增加得足够多，则 Auto Scaling 组会添加更多实例来处理增加的负载。您应在启动时配置用来启动的 Amazon Kinesis Data Streams 应用程序，以便让其他工作程序和记录处理器在新实例上立即激活。

有关重新分片的更多信息，请参阅[对流进行重新分片 \(p. 68\)](#)。

处理重复记录

有两个主要原因可能导致多次向您的 Amazon Kinesis Data Streams 应用程序提供记录：创建者重试和使用者的重试。您的应用程序必须预计并适当地应对多次处理单个记录的问题。

创建者重试

假设一个创建者在调用后遇到了与网络相关的超时 PutRecord，但在它可以从 Amazon Kinesis Data Streams 获取确认。创建者无法确定记录是否已传输到 Kinesis Data Streams。假定每个记录对应用程序都很重要，创建者应该已被写入以重试对相同数据的调用。如果两者的 PutRecord 已将相同数据的调用已成功提交到 Kinesis Data Streams，然后将有两个 Kinesis Data Streams 记录。尽管这两个记录具有相同的数据，但它们各具有唯一的序号。需要严格保证的应用程序应在记录中嵌入一个主键，以便在随后的处理过程中删除重复项。请注意，由创建者重试产生的重复项的数量通常低于由使用者重试产生的重复项的数量。

Note

如果您将 Amazon 开发工具包 PutRecord，默认值 [重试失败的配置 PutRecord 最多调用三次](#)。

使用者重试

使用者（数据处理应用程序）重试在记录处理器重新启动时发生。相同分片的记录处理器在以下情况下重新启动：

1. 工作程序意外终止
2. 已添加或删除工作程序实例
3. 已拆分或合并分片
4. 已部署应用程序

在所有这些情况下，shards-to-worker-to-record-processor 映射会持续更新以实现负载平衡处理。已迁移到其他实例的分片处理器将从上一个检查点开始重新启动处理记录。这导致了重复的记录处理，如以下示例所示。有关负载均衡的更多信息，请参阅[重新分片、扩展和并行处理 \(p. 164\)](#)。

例如：导致重新传输记录的使用者重试

在此示例中，您有一个持续从流中读取记录、将记录聚合到本地文件并将文件上传到 Amazon S3 的应用程序。为简便起见，假定只有 1 个分片和 1 个处理该分片的工作程序。考虑以下示例顺序的事件，假定上一个检查点位于记录编号 10000 处：

1. 工作程序从该分片中读取下一批记录，即从 10001 到 20000 的记录。
2. 工作程序随后将这批记录传递到关联的记录处理器。
3. 记录处理器聚合数据、创建 Amazon S3 文件并成功将文件上传到 Amazon S3。
4. 工作程序在新的检查点出现之前意外终止。
5. 应用程序、工作程序和记录处理器重新启动。
6. 工作程序现在开始从上次成功的检查点（在本案例中为 10001）开始读取。

因此，记录 10001-20000 使用了多次。

实现对使用者重试的弹性

尽管记录可被处理多次，但您的应用程序可能会带来副作用，就像记录只能处理一次一样（幂等处理）。有关此问题的解决方案因复杂性和准确性而异。如果最终数据的目标可以很好地处理重复，我们建议依靠最终目标来实现幂等处理。例如，与 [OpenSearch](#) 您可以使用版本控制和唯一 ID 的组合来防止重复的处理。

在上一节的示例应用程序中，它持续从流中读取记录、将记录聚合到本地文件并将文件上传到 Amazon S3。如图所示，记录 10001-20000 使用了多次，从而生成了具有相同数据的多个 Amazon S3 文件。减少此示例中的重复的一种方法是确保步骤 3 使用了以下方案：

1. 记录处理器每个 Amazon S3 文件使用固定数量的记录，如 5000。
2. 文件名使用以下架构：Amazon S3 前缀、分片 ID 和 First-Sequence-Num. 在本例中，它可以是类似于 sample-shard000001-10001 的形式。
3. 在上传 Amazon S3 文件后，通过指定进行检查点操作 Last-Sequence-Num. 在本例中，您将在记录编号 15000 处进行检查点操作。

利用此方案，即使记录被处理了多次，生成的 Amazon S3 文件也会具有相同的名称和数据。重试只会导致将相同的数据多次写入到相同的文件。

对于重新分片操作，分片中剩余的记录的数量可能少于您需要的固定数量。在这种情况下，您的 shutdown() 方法必须将文件刷新到 Amazon S3 并对上一个序号进行检查点操作。以上方案也与重新分片操作兼容。

处理启动、关闭和限制

以下是要融入到您的 Amazon Kinesis Data Streams 应用程序的一些其他注意事项。

目录

- [启动数据创建器和数据使用者 \(p. 166\)](#)
- [关闭 Amazon Kinesis Data Streams 应用程序 \(p. 167\)](#)
- [读取限制 \(p. 167\)](#)

启动数据创建器和数据使用者

默认情况下，KCL 从流的提示处开始读取记录，这是最近添加的记录。在此配置中，如果数据生成应用程序在任何接收记录的处理器运行前向流添加记录，那么记录处理器不会在启动后读取这些记录。

要更改记录处理器的行为以便它始终从流的开头处读取数据，请在 Amazon Kinesis Data Streams 应用程序的属性文件中设置以下值：

```
initialPositionInStream = TRIM_HORIZON
```

默认情况下，Amazon Kinesis Data Streams 将所有数据存储 24 小时。它还支持最多 7 天的延长保留期和长期保留长达 365 天。此时间范围称为保留期。将开始位置设置为 TRIM_HORIZON 时，将根据保留期的定义，对流中最早的数据启动记录处理器。即使使用 TRIM_HORIZON 设置，如果记录处理器在超过了保留期很长一段时间之后启动，流中的一些数据将不再可用。因此，您应始终让使用者应用程序读取流，并使用 CloudWatch 指标 `GetRecords.IteratorAgeMilliseconds` 进行监视，以确保应用程序与传入数据保持同步。

在某些情况下，对于记录处理器来说，错过流中的前几个记录没有关系。例如，您可能会在流中运行一些初始记录来测试流是否正在运行。end-to-end正如预期的。在此初步验证之后，您随后会启动您的工作程序并开始将生产数据放入流中。

有关 TRIM_HORIZON 设置的更多信息，请参阅[使用分片迭代器 \(p. 140\)](#)。

关闭 Amazon Kinesis Data Streams 应用程序

当您的 Amazon Kinesis Data Streams 应用程序已完成其预定任务时，您应通过终止运行该应用程序的 EC2 实例来关闭它。您可使用 [Amazon Web Services Management Console](#) 或 [Amazon CLI](#) 来终止实例。

关闭您的 Amazon Kinesis 数据流应用程序后，您应删除用于跟踪应用程序的状态的 Amazon DynamoDB 表。

读取限制

流的吞吐量在分片级别进行配置。每个分片的读取吞吐量为每秒 5 个事务，最大总数据读取速率为每秒 2 MB。如果某个应用程序 (或对相同的流执行操作的一组应用程序) 尝试以较快的速率从分片中获取数据，则 Kinesis Data Streams 将限制对应的 Get 操作。

在 Amazon Kinesis Data Streams 应用程序中，如果记录处理器处理数据的速度超过限制（例如在故障转移的情况下），则会发生限制。由于 KCL 管理应用程序与 Kinesis Data Streams 之间的交互，因此限制异常会在 KCL 代码中出现，而不会在应用程序代码中出现。但是，由于 KCL 会记录这些异常，因此您可以在日志中看到它们。

如果您发现您的应用程序一直受到限制，则应考虑增加流的分片的数量。

监控 Amazon Kinesis Data Streams

您可使用以下功能监控 Amazon Kinesis Data Streams 中的数据流：

- [CloudWatch 指标 \(p. 168\)](#)— Kinesis Data Streams 向亚马逊发送 CloudWatch 对每个流进行详细监控的自定义指标。
- [Kinesis 代理 \(p. 178\)](#)— Kinesis 代理发布自定义 CloudWatch 指标，以帮助评估代理是否按预期运行。
- [API 日志记录 \(p. 178\)](#)— 使用 Kinesis Data Streams Amazon CloudTrail 以记录 API 调用并将数据存储在 Amazon S3 存储桶中。
- [Kinesis 客户端库 \(p. 182\)](#)— Kinesis 客户端库 (KCL) 提供针对分片、工作程序和 KCL 应用程序的指标。
- [Kinesis 创建者库 \(p. 191\)](#)— Kinesis Producer Library (KPL) 提供针对分片、工作程序和 KPL 应用程序的指标。

有关常见监控问题、问题和故障排除的详细信息，请参阅以下内容：

- [我应该使用哪些指标来监控和解决 Kinesis Data Streams 问题？](#)
- [为什么选择 IteratorAgeMilliseconds Kinesis Data Streams 中的价值不断增长？](#)

使用 Amazon CloudWatch 监控 Amazon Kinesis Data Streams

Amazon Kinesis Data Streams 和 Amazon CloudWatch 已集成，因此您可以为您的收集、查看和分析 Kinesis 数据流的 CloudWatch 指标。例如，要跟踪分片使用情况，您可监控 IncomingBytes 和 OutgoingBytes 指标并将它们与流中分片数量进行比较。

将自动收集您为流配置的指标并将其推送到。CloudWatch 每分钟。指标会存档两周。两周后，数据会被丢弃。

下表介绍了针对 Kinesis 数据流的基本流级监控和增强的分片级监控。

类型	描述
基本 (流级)	每分钟自动发送流级数据是免费的。
增强 (分片级)	每分钟发送分片级数据需要额外付费。要获得此级别的数据，您必须使用 EnableEnhancedMonitoring operation。 有关定价的信息，请参阅 亚马逊 CloudWatch 产品页面 。

Amazon Kinesis Data Streams 维度和指标

Kinesis Data Streams 将指标发送给 CloudWatch 在两个级别分别：流级别和 (可选) 分片级别。流级别指标适用于一般条件下的最常见的监控使用案例。分片级别指标适用于特定的监控任务，通常与故障排除有关，并通过使用 [EnableEnhancedMonitoring](#) operation。

有关从中收集到的统计数据的说明 CloudWatch 指标，请参阅 [CloudWatch 统计数据](#) 中的亚马逊 CloudWatch 用户指南。

主题

- [基本的流级指标 \(p. 169\)](#)
- [增强的分片级指标 \(p. 174\)](#)
- [Amazon Kinesis Data Streams 指标的维度 \(p. 176\)](#)
- [推荐的 Amazon Kinesis Data Streams 指标 \(p. 176\)](#)

基本的流级指标

Amazon/Kinesis 命名空间包括以下流级指标。

Kinesis Data Streams 将这些流级别指标发送到 CloudWatch 每分钟。这些指标始终可用。

指标	描述
GetRecords.Bytes	在指定时段内测量的从 Kinesis 流中检索的字节数。统计数据 Minimum、Maximum 和 Average 表示指定时段内流的单个 GetRecords 操作中的字节数。 分片级别指标名称：OutgoingBytes 维度：StreamName 统计数据：Minimum、Maximum、Sum、Samples 单位：字节
GetRecords.IteratorAge	此指标已弃用。使用 GetRecords.IteratorAgeMilliseconds。
GetRecords.IteratorAgeMilliseconds	总而言之，最后一个记录的时代 GetRecords 在指定时段内测量的 Kinesis 流中的调用。存在时间是当前时间与最后一条 GetRecords 调用记录写入流的时间之差。Minimum 和 Maximum 统计数据可用于跟踪 Kinesis 使用者应用程序的进度。值为“零”表示正在读取的记录已完全与流匹配。 分片级别指标名称：IteratorAgeMilliseconds 维度：StreamName 统计数据：Minimum、Maximum、Average、Samples 单位：毫秒
GetRecords.Latency	在指定时段内测量的每个 GetRecords 操作所用的时间。 维度：StreamName 统计数据：Average、Maximum、Average 单位：毫秒
GetRecords.Records	在指定时段内测量的从分片中检索的记录数。统计数据 Minimum、Maximum 和 Average 表示指定时段内流的单个 GetRecords 操作中的记录数。 分片级别指标名称：OutgoingRecords 维度：StreamName

指标	描述
	统计数据：Minimum、Maximum、Sum、Samples 单位：计数
GetRecords.Success	在指定时段内测量的每个流中的成功 GetRecords 操作数。 维度：StreamName 统计数据：Average、Sum、Samples 单位：计数
IncomingBytes	在指定时段内成功放置到 Kinesis 流的字节数。该指标包含来自 PutRecord 和 PutRecords 的字节数。统计数据 Minimum、Maximum 和 Average 表示指定时段内流的单个 put 操作中的字节数。 分片级别指标名称：IncomingBytes 维度：StreamName 统计数据：Minimum、Maximum、Sum、Samples 单位：字节
IncomingRecords	在指定时段内成功放置到 Kinesis 流的记录数。该指标包含来自 PutRecord 和 PutRecords 的记录数。统计数据 Minimum、Maximum 和 Average 表示指定时段内流的单个 put 操作中的记录数。 分片级别指标名称：IncomingRecords 维度：StreamName 统计数据：Minimum、Maximum、Sum、Samples 单位：计数
PutRecord.Bytes	使用放置到 Kinesis 流的字节数 PutRecord 在指定时间段内的操作。 维度：StreamName 统计数据：Minimum、Maximum、Sum、Samples 单位：字节
PutRecord.Latency	在指定时段内测量的每个 PutRecord 操作所用的时间。 维度：StreamName 统计数据：Average、Maximum、Average 单位：毫秒

指标	描述
<code>PutRecord.Success</code>	成功的数量<code>PutRecord</code>在指定时段内测量的每个 Kinesis 流中的操作数。平均值反映了对流的成功写入的百分比。 维度：StreamName 统计数据：Average、Sum、Samples 单位：计数
<code>PutRecords.Bytes</code>	使用放置到 Kinesis 流的字节数<code>PutRecords</code>在指定时间段内的操作。 维度：StreamName 统计数据：Minimum、Maximum、Sum、Samples 单位：字节
<code>PutRecords.Latency</code>	在指定时段内测量的每个 <code>PutRecords</code> 操作所用的时间。 维度：StreamName 统计数据：Average、Maximum、Average 单位：毫秒
<code>PutRecords.Records</code>	此指标已弃用。使用 <code>PutRecords.SuccessfulRecords</code>。 维度：StreamName 统计数据：Minimum、Maximum、Sum、Samples 单位：计数
<code>PutRecords.Success</code>	的数量<code>PutRecords</code>在指定时段内测得的每个 Kinesis 流中至少有一个记录成功的操作。 维度：StreamName 统计数据：Average、Sum、Samples 单位：计数
<code>PutRecords.TotalRecords</code>	在<code>PutRecords</code>在指定时段内测量的每个 Kinesis 数据流的操作数。 维度：StreamName 统计数据：Minimum、Maximum、Sum、Samples 单位：计数

指标	描述
PutRecords.SuccessfulRecords	在一个成功的记录的数量PutRecords在指定时段内测量的每个 Kinesis 数据流的操作数。 维度：StreamName 统计数据：Minimum、Maximum、Sum、Samples 单位：计数
PutRecords.FailedRecords	因内部故障而被拒绝的记录数目。PutRecords在指定时段内测量的每个 Kinesis 数据流的操作数。偶尔会出现内部故障，应该重试。 维度：StreamName 统计数据：Minimum、Maximum、Sum、Samples 单位：计数
PutRecords.ThrottledRecords	由于存在限制而被拒绝的记录数PutRecords在指定时段内测量的每个 Kinesis 数据流的操作数。 维度：StreamName 统计数据：Minimum、Maximum、Sum、Samples 单位：计数
ReadProvisionedThroughputExceeded	在指定时段内针对流的受限的 GetRecords 调用数。此指标的最常用的统计数据为 Average。 当统计数据 Minimum 的值为 1 时，流的所有记录在指定时段内将受限。 当统计数据 Maximum 的值为 0 (零) 时，流的任何记录在指定时段内将不受限。 分片级别指标名称：ReadProvisionedThroughputExceeded 维度：StreamName 统计数据：Minimum、Maximum、Sum、Samples 单位：计数
SubscribeToShard.RateExceeded	当新订阅尝试失败时，将发出此指标，因为同一使用者已有活动订阅，或者超过了此操作允许的每秒调用数，也将发出此指标。 维度：StreamName、ConsumerName
SubscribeToShard.Success	此指标记录 SubscribeToShard 订阅是否已成功建立。订阅最多只能有效 5 分钟。因此，该指标至少每 5 分钟发送一次。 维度：StreamName、ConsumerName

指标	描述
<code>SubscribeToShardEvent.Bytes</code>	在指定时段内测量的从分片中接收的字节数。统计数据 <code>Minimum</code>、<code>Maximum</code> 和 <code>Average</code> 表示指定时段内单个事件中的已发布字节数。 分片级别指标名称：<code>OutgoingBytes</code> 维度：<code>StreamName</code>、<code>ConsumerName</code> 统计数据：<code>Minimum</code>、<code>Maximum</code>、<code>Sum</code>、<code>Samples</code> 单位：字节
<code>SubscribeToShardEvent.MillisBehind</code>	当前时间和最后一条记录之间的差异 <code>SubscribeToShard</code> 事件已写入直播。 维度：<code>StreamName</code>、<code>ConsumerName</code> 统计数据：<code>Minimum</code>、<code>Maximum</code>、<code>Average</code>、<code>Samples</code> 单位：毫秒
<code>SubscribeToShardEvent.Records</code>	在指定时段内测量的从分片中接收的记录数。统计数据 <code>Minimum</code>、<code>Maximum</code> 和 <code>Average</code> 表示指定时段内单个事件中的记录数。 分片级别指标名称：<code>OutgoingRecords</code> 维度：<code>StreamName</code>、<code>ConsumerName</code> 统计数据：<code>Minimum</code>、<code>Maximum</code>、<code>Sum</code>、<code>Samples</code> 单位：计数
<code>SubscribeToShardEvent.Success</code>	每次成功发布事件时都会发出此指标。只有在有活动订阅时，才会发出它。 维度：<code>StreamName</code>、<code>ConsumerName</code> 统计数据：<code>Minimum</code>、<code>Maximum</code>、<code>Sum</code>、<code>Samples</code> 单位：计数

指标	描述
WriteProvisionedThroughputExceeded	在指定时段内因流限制而被拒绝的记录数。该指标包含来自 PutRecord 和 PutRecords 操作的限制。此指标的最常用的统计数据为 Average。 当统计数据 Minimum 的值不为零时，流的记录在指定时段内将受限。 当统计数据 Maximum 的值为 0（零）时，流的任何记录在指定时段内将不受限。 分片级别指标名称：WriteProvisionedThroughputExceeded 维度：StreamName 统计数据：Minimum、Maximum、Sum、Samples 单位：计数

增强的分片级指标

Amazon/Kinesis 命名空间包括以下分片级指标。

Kinesis 将以下分片级指标发送至 CloudWatch 每分钟。每个指标维度创建 1 CloudWatch 指标，制造了大约 43,200PutMetricData 每月 API 调用数。默认情况下，这些指标未启用。Kinesis 发出的增强指标将收费。有关更多信息，请参阅 [亚马逊 CloudWatch 定价](#) 在标题下 Amazon CloudWatch 自定义指标。按每个月每个指标每个给定收费。

指标	描述
IncomingBytes	在指定时段内成功放置到分片的字节数。该指标包含来自 PutRecord 和 PutRecords 的字节数。统计数据 Minimum、Maximum 和 Average 表示指定时段内分片的单个 put 操作中的字节数。 流级别指标名称：IncomingBytes 维度：StreamName、SShardId 统计数据：Minimum、Maximum、Sum、Samples 单位：字节
IncomingRecords	在指定时段内成功放置到分片的记录数。该指标包含来自 PutRecord 和 PutRecords 的记录数。统计数据 Minimum、Maximum 和 Average 表示指定时段内分片的单个 put 操作中的记录数。 流级别指标名称：IncomingRecords 维度：StreamName、SShardId 统计数据：Minimum、Maximum、Sum、Samples 单位：计数

指标	描述
IteratorAgeMilliseconds	对某个分片进行的所有 GetRecords 调用中最后一条记录的存在时间，是在指定的时间段测量的。存在时间是当前时间与最后一条 GetRecords 调用记录写入流的时间之差。Minimum 和 Maximum 统计数据可用于跟踪 Kinesis 使用者应用程序的进度。值为 0 (零) 表示正在读取的记录已完全与流匹配。 流级别指标名称 : GetRecords.IteratorAgeMilliseconds 维度 : StreamName、SShardId 统计数据 : Minimum、Maximum、Average、Samples 单位 : 毫秒
OutgoingBytes	在指定时段内测量的从分片中检索的字节数。统计数据 Minimum、Maximum 和 Average 表示单个 GetRecords 操作中返回的字节数或指定时段内分片的单个 SubscribeToShard 事件中发布的字节数。 流级别指标名称 : GetRecords.Bytes 维度 : StreamName、SShardId 统计数据 : Minimum、Maximum、Sum、Samples 单位 : 字节
OutgoingRecords	在指定时段内测量的从分片中检索的记录数。统计数据 Minimum、Maximum 和 Average 表示单个 GetRecords 操作中返回的记录数或指定时段内分片的单个 SubscribeToShard 事件中发布的记录数。 流级别指标名称 : GetRecords.Records 维度 : StreamName、SShardId 统计数据 : Minimum、Maximum、Sum、Samples 单位 : 计数

指标	描述
ReadProvisionedThroughputExceeded	在指定时段内针对分片的受限的 GetRecords 调用数。此异常计数涵盖了以下限制的所有维度：每秒读取每个分片 5 次或每分片每秒 2 MB。此指标的最常用的统计数据为 Average。 当统计数据 Minimum 的值为 1 时，分片的所有记录在指定时段内将受限。 当统计数据 Maximum 的值为 0 (零) 时，分片的任何记录在指定时段内将不受限。 流级别指标名 称：ReadProvisionedThroughputExceeded 维度：StreamName、SShardId 统计数据：Minimum、Maximum、Sum、Samples 单位：计数
WriteProvisionedThroughputExceeded	在指定时段内因分片限制而被拒绝的记录数。此指标包括来自 PutRecord 和 PutRecords 操作的限制，并涵盖以下限制的所有维度：每分片每秒 1,000 条记录或每分片每秒 1 MB。此指标的最常用的统计数据为 Average。 当统计数据 Minimum 的值不为零时，分片的记录在指定时段内将受限。 当统计数据 Maximum 的值为 0 (零) 时，分片的任何记录在指定时段内将不受限。 流级别指标名 称：WriteProvisionedThroughputExceeded 维度：StreamName、SShardId 统计数据：Minimum、Maximum、Sum、Samples 单位：计数

Amazon Kinesis Data Streams 指标的维度

维度	描述
StreamName	Kinesis 流的名称。所有可用统计数据按 StreamName 进行筛选。

推荐的 Amazon Kinesis Data Streams 指标

Kinesis Data Streams 客户可能对几个 Amazon Kinesis 数据流指标特别感兴趣。以下列表提供了推荐的指标及其用法。

指标	使用说明
GetRecords.IteratorAgePercent	跟踪流中所有分片和用户的读取位置。如果某个迭代器的寿命超过了保留期的 50%（默认值为 24 小时，可配置为最高 7 天），则存在由于记录过期造成数据丢失的风险。我们建议您使用 CloudWatch 在最大值统计数据中提醒您，以便在此问题构成风险之前提醒您。要了解使用此指标的示例情景，请参阅 使用者记录处理滞后 (p. 162) 。
ReadProvisionedThroughputExceeded	当您的使用者端记录处理滞后时，有时难以确定瓶颈的位置。使用此指标可确定读取操作是否由于超出了读取吞吐量上限而受到限制。此指标的最常用的统计数据为 Average。
WriteProvisionedThroughputExceeded	这与 ReadProvisionedThroughputExceeded 指标的用途相同，但是此指标用于流的创建者 (put) 端。此指标的最常用的统计数据为 Average。
PutRecord.Success, PutRecords.Success	我们建议使用 CloudWatch 在平均值统计数据中提醒，以指示记录未能放入流中的时间。根据创建器所使用的对象选择一种或两种 put 类型。如果使用的是 Kinesis Producer Library (KPL)，请使用 PutRecords.Success。
GetRecords.Success	我们建议使用 CloudWatch 在平均值统计数据中提醒，以指示从流中读取记录失败的时间。

访问 Amazon CloudWatch Kinesis Data Streams 的指标

您可 Kinesis Data Streams 用 CloudWatch 控制台、命令行或 CloudWatch API。以下过程介绍如何使用这些不同的方式访问指标。

使用 访问指标 CloudWatch 控制台

1. 打开 CloudWatch 控制台<https://console.aws.amazon.com/cloudwatch/>。
2. 在导航栏中，选择一个区域。
3. 在导航窗格中，选择 Metrics (指标)。
4. 在 CloudWatch Metrics by Category (按类别显示的 CloudWatch 指标) 窗格中，选择 Kinesis Metrics (Kinesis 指标)。
5. 单击相关行可查看指定的 MetricName 和 StreamName 的统计数据。

注意：大多数控制台统计名称匹配相应 CloudWatch 上面列出的指标名称，但 Read Throughput (读取吞吐量)和 Write Throughput (写入吞吐量) 按照 5 分钟的时间间隔计算：Write Throughput (写入吞吐量)监控 IncomingBytesCloudWatch 指标、Read Throughput (读取吞吐量)显示器 GetRecords.Bytes。

6. (可选) 在图形窗格中，选择一个统计数据和时间段，然后创建 CloudWatch 使用这些设置发出警报。

使用 Amazon CLI 访问指标

使用 [list-metrics](#) 和 [get-metric-statistics](#) 命令。

使用 访问指标 CloudWatch CLI

使用 [mon-list-metrics](#) 和 [mon-get-stats](#) 命令。

使用 访问指标 CloudWatch API

使用 [ListMetrics](#) 和 [GetMetricStatistics](#) 操作。

使用 Amazon CloudWatch 监控 Kinesis Data Streams

代理发布自定义 CloudWatch 命名空间为 AmazonKinesisAgent。这些指标可帮助您评估代理是否按指定方式将数据提交到 Kinesis Data Streams stream、运行是否正常以及在数据创建器上是否使用适当数量的 CPU 和内存资源。记录数和发送的字节数等指标对于了解代理将数据提交到流的速率非常有用。当这些指标低于预期阈值一定的百分比或者降低为零时，可能表明存在配置问题、网络错误或代理运行状况问题。诸如主机上的 CPU 和内存消耗以及代理错误计数器等指标可用于指示数据创建器资源使用情况，并提供对潜在的配置或主机错误的深入分析。最后，代理还会记录服务异常，以帮助调查代理问题。这些指标在代理配置设置 `cloudwatch.endpoint` 指定的区域中报告。从多个 Kinesis 代理发布的 Cloudwatch 指标是聚合或合并的。有关代理配置的更多信息，请参阅[代理配置设置 \(p. 92\)](#)。

使用 CloudWatch 进行监控

Kinesis Data Streams 代理会向 CloudWatch 发送以下指标。

指标	描述
BytesSent	在指定时段内发送到 Kinesis Data Streams 的字节数。 单位：字节
RecordSendAttempts	在指定的时间范围内对 PutRecords 的一次调用中尝试的记录数（第一次，或者作为重试）。 单位：计数
RecordSendErrors	在指定时间范围内对 PutRecords 的一次调用中返回故障状态的记录数，包括重试。 单位：计数
ServiceErrors	在指定时间范围内产生服务错误（限制错误之外的其他错误）的 PutRecords 调用次数。 单位：计数

记录 Amazon Kinesis Data Streams API 调用 Amazon CloudTrail

Amazon Kinesis Data Streams 与 Amazon CloudTrail，提供用户、角色或用户执行操作的记录的服务。Amazon Kinesis Data Streams 中的服务。CloudTrail 将 Kinesis Data Streams 的所有 API 调用作为事件捕获。捕获的调用包括来自 Kinesis Data Streams 控制台的调用和对 Kinesis Data Streams API 操作的代码调用。如果您创建了跟踪，则可以使 CloudTrail 发送到 Amazon S3 存储桶的事件，包括 Kinesis Data Streams 的事件。如果您不配置跟踪，则仍可在 CloudTrail 在控制台事件记录。使用 CloudTrail 收集的信息，您可以确定向 Kinesis Data Streams 发出了什么请求、发出请求的 IP 地址、何人发出的请求、请求的发出时间以及其他详细信息。

要了解有关 CloudTrail 的更多信息（包括如何对其进行配置和启用），请参阅《Amazon CloudTrail 用户指南》<https://docs.amazonaws.cn/awscloudtrail/latest/userguide/>。

CloudTrail 中的 Kinesis Data Streams

在您创建 Amazon 账户时，将在该账户上启用 CloudTrail。当 Kinesis Data Streams 中发生受支持的事件活动时，该活动将记录在 CloudTrail 活动以及其他 Amazon 中的服务事件记录。您可以在 Amazon 账户中查看、搜索和下载最新事件。有关更多信息，请参阅[使用 CloudTrail 事件历史记录查看事件](#)。

要持续记录您的事件 Amazon 帐户（包括 Kinesis Data Streams）创建跟踪。一个踪迹启用 CloudTrail 将日志文件传送到 Amazon S3 存储桶。预设情况下，在控制台中创建跟踪时，此跟踪应用于所有 Amazon 区域。此跟踪记录在 Amazon 分区中记录所有区域中的事件，并将日志文件传送到您指定的 Simple Storage Service (Amazon S3) 存储桶。此外，您还可以配置其他 Amazon 进一步分析在中收集的事件数据并采取措施的服务。CloudTrail 日志。有关更多信息，请参阅下列内容：

- [创建跟踪概览](#)
- [CloudTrail 支持的服务和集成](#)
- [为 CloudTrail 配置 Amazon SNS 通知](#)
- [接收 CloudTrail 多个区域中的日志文件](#)和[接收 CloudTrail 多个账户中的日志文件](#)

Kinesis Data Streams 支持在中将以下操作记录为事件：CloudTrail 日志文件：

- [AddTagsToStream](#)
- [CreateStream](#)
- [DecreaseStreamRetentionPeriod](#)
- [DeleteStream](#)
- [DeregisterStreamConsumer](#)
- [DescribeStream](#)
- [DescribeStreamConsumer](#)
- [DisableEnhancedMonitoring](#)
- [EnableEnhancedMonitoring](#)
- [IncreaseStreamRetentionPeriod](#)
- [ListStreamConsumers](#)
- [ListStreams](#)
- [ListTagsForStream](#)
- [MergeShards](#)
- [RegisterStreamConsumer](#)
- [RemoveTagsFromStream](#)
- [SplitShard](#)
- [StartStreamEncryption](#)
- [StopStreamEncryption](#)
- [UpdateShardCount](#)

每个事件或日志条目都包含有关生成请求的人员信息。身份信息可帮助您确定以下内容：

- 请求是使用根用户凭证还是 Amazon Identity and Access Management (IAM) 用户凭证发出的。
- 请求是使用角色还是联合身份用户的临时安全凭证发出的。
- 请求是否由其它 Amazon 服务发出。

有关更多信息，请参阅 [CloudTrail userIdentity 元素](#)。

例如：Kinesis Data Streams 日志文件条目

跟踪是一种配置，可用于将事件作为日志文件传送到您指定的 Amazon S3 存储桶。CloudTrail 日志文件包含一个或多个日志条目。一个事件表示来自任何源的一个请求，包括有关所请求的操作、操作的日期和时间、请求参数等方面的信息。CloudTrail 日志文件不是公用 API 调用的有序堆栈跟踪，因此它们不会按任何特定顺序显示。

以下示例显示了 CloudTrail 演示的日志条目 CreateStream、DescribeStream、ListStreams、DeleteStream、SplitShard、和 MergeShards 行动。

```
{
  "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": "b7acfdc0-6ca9-4ee1-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"
  }
]
```

使用 Amazon CloudWatch 监控 Kinesis 客户端库

这些区域有：[Kinesis 客户端库\(KCL\)](#) 针对 Amazon Kinesis Data Streams 发布自定义亚马逊 CloudWatch 指标代表您使用 KCL 应用程序的名称作为命名空间。您可以通过导航至[CloudWatch 控制台](#)然后选择自定义指标。有关自定义指标的更多信息，请参阅[发布自定义指标](#)中的亚马逊 CloudWatch 用户指南。

对于上传到的指标，存在象征性的少许费用。CloudWatch 来自 KCL；具体而言，亚马逊 CloudWatch 自定义指标和亚马逊 CloudWatch API 请求将收取费用。有关更多信息，请参阅。[亚马逊 CloudWatch 定价](#)。

主题

- [指标和命名空间 \(p. 182\)](#)
- [指标级别和维度 \(p. 182\)](#)
- [指标配置 \(p. 183\)](#)
- [指标的列表 \(p. 183\)](#)

指标和命名空间

用于上传指标的命名空间是您在启动 KCL 时指定的应用程序名称。

指标级别和维度

可通过两个选项控制将上传到 CloudWatch 的指标：

指标级别

每个指标分配有一个独立的级别。当您设置指标报告级别后，单个级别低于报告级别的指标将不会发送到 CloudWatch。级别如下：NONE、SUMMARY 和 DETAILED。默认设置为 DETAILED 所有指标；也就是

说，所有指标都将发送到 CloudWatch。NONE 报告级别意味着不会发送任何指标。有关将哪些级别分配到哪些指标的信息，请参阅 [指标的列表 \(p. 183\)](#)。

启用的维度

每个 KCL 指标都有一个也会发送到 CloudWatch 的关联维度。在 KCL 2.x 中，如果 KCL 配置为处理单个数据流，则所有指标维度 (Operation、ShardId, 和 WorkerIdentifier) 默认处于启用状态。另外，在 KCL 2.x 中，如果 KCL 配置为处理单个数据流，Operation 无法禁用维度。在 KCL 2.x 中，如果 KCL 被配置为处理多个数据流，则所有指标维度 (Operation、ShardId、StreamId, 和 WorkerIdentifier) 默认处于启用状态。此外，在 KCL 2.x 中，如果将 KCL 配置为处理多个数据流，则 Operation 和 StreamId 无法禁用维度。StreamId 维度仅对每个分片指标有效。

在 KCL 1.x 中，只有 Operation 和 ShardId 默认处于启用状态，WorkerIdentifier 维度已禁用。在 KCL 1.x 中，Operation 无法禁用维度。

有关 的更多信息 CloudWatch 指标维度，请参阅 [维度](#) 亚马逊中的部分 CloudWatch 概念主题，在亚马逊 CloudWatch 用户指南。

当 WorkerIdentifier 启用维度后，如果每次特定的 KCL 工作程序重新启动时使用的是不同的工作程序 ID 属性值，则会将新的指标组和新的指标组。WorkerIdentifier 维度值将发送到 CloudWatch。如果你需要 WorkerIdentifier 维度值在特定的 KCL 工作程序重新启动时保持不变，则必须在每个工作程序初始化时显式指定相同的工作程序 ID 值。请注意，每个有效 KCL 工作程序的工作程序 ID 值在所有 KCL 工作程序中必须是唯一的。

指标配置

指标级别和启用的维度可使用 KinesisClientLibConfiguration 实例，该实例将在启动 KCL 应用程序时将传递给工作程序。在 MultiLangDaemon 案例，metricsLevel 和 metricsEnabledDimensions 属性可在用于启动的 .properties 文件中指定。MultiLangDaemon KCL 应用程序。

可向指标级别分配下列三个值之一：无、摘要或详细。启用的维度值必须是以逗号分隔的字符串，并具有允许的维度的列表。CloudWatch 指标。KCL 应用程序使用的维度为 Operation、ShardId, 和 WorkerIdentifier。

指标的列表

下表列出了按范围和操作分组的 KCL 指标。

主题

- [每个 KCL 应用程序指标 \(p. 183\)](#)
- [Per-Worker 指标 \(p. 187\)](#)
- [Per-Shard 指标 \(p. 189\)](#)

每个 KCL 应用程序指标

这些指标跨应用程序范围内的所有 KCL 工作程序聚合一起，就像亚马逊所定义的那样。CloudWatch 命名空间。

主题

- [InitializeTask \(p. 184\)](#)
- [ShutdownTask \(p. 184\)](#)
- [ShardSyncTask \(p. 185\)](#)
- [BlockOnParentTask \(p. 186\)](#)
- [周期 SHARDSYNC 管理器 \(p. 186\)](#)

- [MultitreMartacker \(p. 187\)](#)

InitializeTask

这些区域有：InitializeTask操作负责初始化 KCL 应用程序的记录处理器。此操作的逻辑包括从 Kinesis Data Stream 中获取分片迭代器以及初始化记录处理器。

指标	描述
KinesisDataFetcher.getIteratorSuccessCount	成功调用 GetShardIterator 每个 KCL 应用程序的操作。 指标级别：明细 单位：计数
KinesisDataFetcher.getIteratorTime	每次使用 GetShardIterator 对于给定 KCL 应用程序的操作。 指标级别：明细 单位：毫秒
RecordProcessor.initializeTime	记录处理器的初始化方法所花费的时间。 指标级别：摘要 单位：毫秒
成功	成功的记录处理程序初始化的数目。 指标级别：摘要 单位：计数
Time	KCL 工作程序初始化记录处理器所花费的时间。 指标级别：摘要 单位：毫秒

ShutdownTask

ShutdownTask 操作初始化分片处理的关闭顺序。这可能是因为分片被拆分或合并，或者当分片租赁从工作程序中丢失时。在这两种情况下，将调用记录处理程序 shutdown() 函数。在分片被拆分或合并的情况下也会发现新的分片，这将创建一个或两个新的分片。

指标	描述
CreateLease.Success	新的子分片在父分片关闭后成功添加到 KCL 应用程序 DynamoDB 表的次数。 指标级别：明细 单位：计数
CreateLease.Time	在 KCL 应用程序 DynamoDB 表中添加新的子分片信息所花费的时间。 指标级别：明细 单位：毫秒

指标	描述
UpdateLease.Success	记录处理程序关闭期间成功的最终检查点的数目。 指标级别：明细 单位：计数
UpdateLease.Time	记录处理程序关闭期间检查点操作所花费的时间。 指标级别：明细 单位：毫秒
RecordProcessor.shutdown	记录处理器的关闭方法所花费的时间。 指标级别：摘要 单位：毫秒
成功	成功的关闭任务的数目。 指标级别：摘要 单位：计数
Time	KCL 工作程序关闭任务所花费的时间。 指标级别：摘要 单位：毫秒

ShardSyncTask

这些区域有：ShardSyncTask 操作会发现对 Kinesis 数据流的分片信息所做的更改，因此 KCL 应用程序可处理新的分片。

指标	描述
CreateLease.Success	将新的分片信息添加到 KCL 应用程序 DynamoDB 表的成功尝试次数。 指标级别：明细 单位：计数
CreateLease.Time	在 KCL 应用程序 DynamoDB 表中添加新的分片信息所花费的时间。 指标级别：明细 单位：毫秒
成功	成功的分片同步操作的数目。 指标级别：摘要 单位：计数
Time	分片同步操作所花费的时间。 指标级别：摘要

指标	描述
	单位：毫秒

BlockOnParentTask

如果一个分片被拆分或与其他分片合并，则会创建新的子分片。这些区域有：BlockOnParentTask操作确保新分片的记录处理在 KCL 完全处理父分片之前不会开始。

指标	描述
成功	父分片完成的成功检查的数目。 指标级别：摘要 单位：计数
Time	父分片完成所花费的时间。 指标级别：摘要 单位：毫秒

周期 SHARDSYNC 管理器

这些区域有：PeriodicShardSyncManager负责检查 KCL 消费者应用程序正在处理的数据流，识别具有部分租用的数据流并将其移交以进行同步。

如果将 KCL 配置为处理单个数据流（然后是 NumStreamsToSync 和 NumStreamsWithPartialLeases 设置为 1）以及当 KCL 被配置为处理多个数据流时也是如此。

指标	描述
numstreamtoSync	数据流的数量（每Amazon账户）由包含部分租约且必须移交才能同步的消费者应用程序处理。 指标级别：摘要 单位：计数
带部分租赁的 NUMSTREAM	数据流的数量（每Amazon账户），消费者应用程序正在处理包含部分租约的内容。 指标级别：摘要 单位：计数
成功	次数PeriodicShardSyncManager能够成功识别消费者应用程序正在处理的数据流中的部分租赁。 指标级别：摘要 单位：计数
Time	的时间量（以毫秒为单位）PeriodicShardSyncManager需要检查使用者应用程序正在处理的数据流，以确定哪些数据流需要分片同步。 指标级别：摘要

指标	描述
	单位：毫秒

MultitreMartacker

这些区域有：MultistreamTracker接口使您能够构建可以同时处理多个数据流的 KCL 消费者应用程序。

指标	描述
已删除 Streams.Count	此时段删除的数据流的数量。 指标级别：摘要 单位：计数
ActiveStreams.Count	正在处理的活动数据流的数量。 指标级别：摘要 单位：计数
直播花费删除。COUNT	等待删除的数据流的数量，基于FormerStreamsLeasesDeletionStrategy。 指标级别：摘要 单位：计数

Per-Worker 指标

这些指标跨使用 Kinesis 数据流（例如 Amazon EC2 实例）中的数据的所有记录处理程序进行聚合。

主题

- [RenewAllLeases \(p. 187\)](#)
- [TakeLeases \(p. 188\)](#)

RenewAllLeases

RenewAllLeases 操作定期续订由特定工作程序实例拥有的分片租约。

指标	描述
RenewLease.Success	工作程序成功续订租约的数目。 指标级别：明细 单位：计数
RenewLease.Time	租约续订操作所花费的时间。 指标级别：明细 单位：毫秒
CurrentLeases	续订所有租约后由工作程序拥有的分片租约数。

指标	描述
	指标级别：摘要 单位：计数
LostLeases	在尝试续订由工作程序拥有的所有租约后丢失的分片租约数。 指标级别：摘要 单位：计数
成功	工作程序的成功的租约续订操作次数。 指标级别：摘要 单位：计数
Time	续订工作程序的所有租约所花费的时间。 指标级别：摘要 单位：毫秒

TakeLeases

这些区域有：TakeLeases操作平衡所有 KCL 工作程序之间的记录处理。如果当前 KCL 工作程序拥有的分片租约少于所需的分片租约，则会从已过载的另一个工作程序中提取分片租约。

指标	描述
ListLeases.Success	成功从 KCL 应用程序 DynamoDB 表中检索所有分片租约的次数。 指标级别：明细 单位：计数
ListLeases.Time	从 KCL 应用程序 DynamoDB 表中检索所花费的时间。 指标级别：明细 单位：毫秒
TakeLease.Success	工作程序成功从其他 KCL 工作程序中获得分片租约的次数。 指标级别：明细 单位：计数
TakeLease.Time	利用该工作程序提取的租约更新租约表所花费的时间。 指标级别：明细 单位：毫秒
NumWorkers	工作程序总数，由特定工作程序标识。 指标级别：摘要 单位：计数

指标	描述
NeededLeases	当前工作程序为平衡分片处理负载所需的分片租约数。 指标级别：明细 单位：计数
LeasesToTake	工作程序将尝试提取的租约的数目。 指标级别：明细 单位：计数
TakenLeases	工作程序已成功提取的租约的数目。 指标级别：摘要 单位：计数
TotalLeases	KCL 应用程序正在处理的分片的总数。 指标级别：明细 单位：计数
ExpiredLeases	未由任何工作程序处理的分片的总数，由特定工作程序标识。 指标级别：摘要 单位：计数
成功	TakeLeases 操作已成功完成的次数。 指标级别：摘要 单位：计数
Time	工作程序的 TakeLeases 操作所花费的时间。 指标级别：摘要 单位：毫秒

Per-Shard 指标

这些指标跨单个记录处理程序聚合一起。

ProcessTask

这些区域有：ProcessTask调用操作[GetRecords](#)使用当前迭代器位置以从流中检索记录并调用记录处理程序。processRecordsfunction.

指标	描述
KinesisDataFetcher.getRecordsSuccessfulCount	成功的数量GetRecords每个 Kinesis 数据流分片的操作。 指标级别：明细

指标	描述
	单位：计数
KinesisDataFetcher.getRecords	每次使用的时间GetRecordsKinesis Data Streams 分片的操作。 指标级别：明细 单位：毫秒
UpdateLease.Success	给定分片的记录处理程序完成的成功检查点的数目。 指标级别：明细 单位：计数
UpdateLease.Time	给定分片的每个检查点操作所花费的时间。 指标级别：明细 单位：毫秒
DataBytesProcessed	每个 ProcessTask 调用所处理的记录的总大小（以字节为单位）。 指标级别：摘要 单位：字节
RecordsProcessed	每个 ProcessTask 调用所处理的记录的数量。 指标级别：摘要 单位：计数
ExpiredIterator	的数量 ExpiredIteratorException 通话时收到GetRecords。 指标级别：摘要 单位：计数
MillisBehindLatest	当前迭代器晚于分片中最新记录 (tip) 的时间。此值为小于或等于响应中最新一条记录的时间和当前时间之间的时差。与比较最新一条响应记录中的时间戳相比，此指标可以更精确地反映分片相对于末端的距离。此值适用于最近一批记录，而不是每条记录中所有时间戳的平均值。 指标级别：摘要 单位：毫秒
RecordProcessor.processRecords	记录处理器的 processRecords 方法所花费的时间。 指标级别：摘要 单位：毫秒
成功	成功处理任务操作的数目。 指标级别：摘要 单位：计数

指标	描述
Time	处理任务操作所花费的时间。 指标级别：摘要 单位：毫秒

使用 Amazon CloudWatch 监控 Kinesis 生产者库

这些区域有：[Kinesis 创建者库](#)在 Amazon Kinesis Data Streams 中发布自定义的亚马逊 CloudWatch 代表您执行指标。您可以通过导航至[CloudWatch 控制台](#)然后选择自定义指标。有关自定义指标的更多信息，请参阅[发布自定义指标](#)中的亚马逊 CloudWatch 用户指南。

对于上传到的指标，存在象征性的少许费用。CloudWatch KPL；具体来说，亚马逊 CloudWatch 亚马逊自定义指标和 CloudWatch 将收取 API 请求费。有关更多信息，请参阅。[亚马逊 CloudWatch 定价](#)。本地指标收集不会产生 CloudWatch 费用。

主题

- [指标、维度和命名空间 \(p. 191\)](#)
- [指标级别和粒度 \(p. 191\)](#)
- [本地访问和亚马逊 CloudWatch 上传 \(p. 192\)](#)
- [指标的列表 \(p. 192\)](#)

指标、维度和命名空间

您可在启动 KPL 时指定应用程序名称，该名称随后在上传指标时将用作命名空间的一部分。这是可选的；如果未设置应用程序名称，则 KPL 将提供一个默认值。

您也可以配置 KPL，以便向指标添加任意其他维度。这在您希望获得更详细的数据时很有用 CloudWatch 指标。例如，您可将主机名作为维度添加，随后该维度将允许您标识队组中不均匀的负载分配。所有 KPL 配置设置是不可变的，因此您在初始化 KPL 实例后将无法更改其他维度。

指标级别和粒度

可通过两个选项控制上传到 CloudWatch 的指标的数目：

指标级别

这是对指标重要性的粗略估计。为每个指标分配了一个级别。在设置一个级别时，级别低于该级别的指标将不会发送到 CloudWatch。级别为 NONE、SUMMARY 和 DETAILED。默认设置为 DETAILED；即，所有指标。NONE 表示没有任何指标，因此实际上未为该级别分配任何指标。

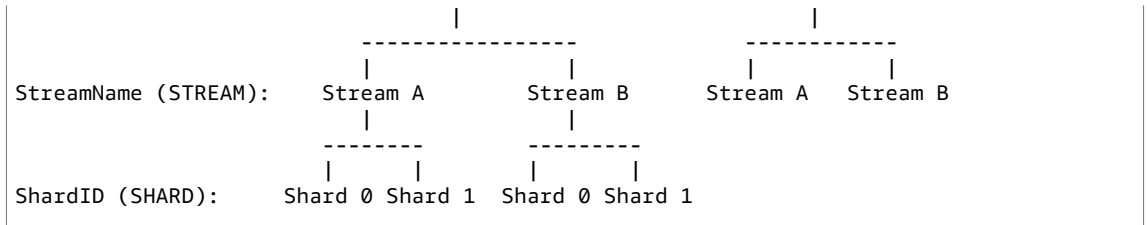
粒度

这可控制是否在其他粒度级别发出相同指标。级别为 GLOBAL、STREAM 和 SHARD。默认设置为 SHARD，其中包含粒度最高的指标。

选择 SHARD 后，发出将流名称和分片 ID 作为维度的指标。此外，还发出仅具有流名称维度的相同指标和不带流名称的指标。这意味着，对于特定指标，两个流每个分片都有两个分片将生成七个 CloudWatch 指标：一个针对每个分片的指标、一个针对整体的指标以及一个针对整体的指标；所有指标描述相同但处于不同级别粒度的统计数据。有关说明，请见下图。

层次结构中的不同粒度级别与系统中的所有指标构成了一个基于指标名称的树：

MetricName (GLOBAL):	Metric X	Metric Y
----------------------	----------	----------



并不会在分片级别提供所有指标；一些指标本来就属于流级别或全局级别。不会在分片级别生成这些指标，即使已启用分片级别指标（上图中的 Metric Y）。

在指定其他维度时，您需要提供 `tuple:<DimensionName, DimensionValue, Granularity>` 的值。粒度用于确定自定义维度在层次结构中的插入位置：GLOBAL 表示其他维度将插入到指标名称的后面；STREAM 表示其他维度将插入到流名称的后面；SHARD 表示其他维度将插入到分片 ID 的后面。如果为每个粒度级别提供多个其他维度，则这些维度将按给定顺序插入。

本地访问和亚马逊 CloudWatch 上传

将实时本地提供当前 KPL 实例的指标；您可随时查询 KPL 以获得这些指标。KPL 本地计算每个指标的和、平均值、最大值和计数，就像在 CloudWatch 一样。

您可获取从程序启动到当前时间点或在过去 N 秒内（其中 N 为介于 1 和 60 之间的整数）使用滚动窗口累积的统计数据。

所有指标均可上传到 CloudWatch。这在跨多台主机聚合数据、监控和警报时特别有用。此功能在本地不可用。

如前所述，您可选择使用指标级别 和粒度 设置来上传的指标。未上传的指标在本地可用。

不支持单独上传数据点，因为如果流量过高，则每秒会生成数以百万计的上传。为此，KPL 会将指标本地聚合到 1 分钟存储桶中，并将统计数据对上传到。CloudWatch 每分钟一次，每个已启用指标。

指标的列表

指标	描述
UserRecordsReceived	KPL 内核为放置操作接收的逻辑用户记录的计数。在分片级别不可用。 指标级别：明细 单位：计数
UserRecordsPending	当前挂起的用户记录的数目的定期取样。如果记录当前已缓冲并等待发送，或已发送和正在发送到后端服务，则记录处于挂起状态。在分片级别不可用。 KPL 提供了专用方法来在全局级别检索此指标以便客户管理其放置速率。 指标级别：明细 单位：计数
UserRecordsPut	已成功放置的逻辑用户记录的计数。 KPL 不计算此指标的失败记录数。这允许用于提供成功率的平均值、用于提供总尝试数的计数以及用于提供失败计数的计数与总和之间的差值。

指标	描述
	指标级别：摘要 单位：计数
UserRecordsDataPut	逻辑用户记录成功放置的字节数。 指标级别：明细 单位：字节
KinesisRecordsPut	成功放置的 Kinesis Data Streams 记录的计数（每个 Kinesis Data Streams 记录均可包含多个用户记录）。 KPL 输出零作为失败记录数。这允许用于提供成功率的平均值、用于提供总尝试数的计数以及用于提供失败计数的计数与总和之间的差值。 指标级别：摘要 单位：计数
KinesisRecordsDataPut	Kinesis Data Streams 记录中的字节数。 指标级别：明细 单位：字节
ErrorsByCode	每种错误代码的计数。这引入了 ErrorCode 和 StreamName 等正常维度以及 ShardId 的其他维度。并非每个错误都可跟踪到分片。无法跟踪的错误仅在流或全局级别发出。此指标捕获有关限制、分片映射更改、内部故障、服务不可用、超时等的信息。 每条 Kinesis Data Streams 记录将计算一次计算的 Kinesis Data Streams。Kinesis Data Streams 记录中的多个用户记录不生成多个计数。 指标级别：摘要 单位：计数
AllErrors	这是由与 Errors by Code 相同的错误引发的，但并不区分类型。这在常规监控错误率而不需要手动汇总所有不同类型错误的计数时非常有用。 指标级别：摘要 单位：计数
RetriesPerRecord	每个用户记录执行的重试次数。为一次性尝试成功的记录发出零。 在用户记录完成时（成功时或不再能够重试时）发出数据。如果记录 time-to-live 是一个很大的值，此指标可能显著延迟。 指标级别：明细 单位：计数

指标	描述
BufferingTime	从用户记录到达 KPL 到达 KPL 和离开前往后端之间的时间。此信息将基于记录传回给用户，还可用作聚合的统计数据。 指标级别：摘要 单位：毫秒
Request Time	执行 PutRecordsRequests 所花费的时间。 指标级别：明细 单位：毫秒
User Records per Kinesis Record	已聚合到单个 Kinesis Data Streams 记录中的逻辑用户记录的数目。 指标级别：明细 单位：计数
Amazon Kinesis Records per PutRecordsRequest	已聚合到单个中的 Kinesis Data Streams 记录的数目PutRecordsRequest. 在分片级别不可用。 指标级别：明细 单位：计数
User Records per PutRecordsRequest	包含在 PutRecordsRequest 中的用户记录的总数。这大致相当于之前两个指标的产品。在分片级别不可用。 指标级别：明细 单位：计数

亚马逊 Kinesis 数据流中的安全

Amazon 十分重视云安全性。作为 Amazon 客户，您将会从专为满足大多数安全敏感型组织的要求而打造的数据中心和网络架构中受益。

安全性是 Amazon 和您的共同责任。[责任共担模式](#)将其描述为云的安全性和云中的安全性：

- 云的安全性 – Amazon 负责保护在 Amazon 云中运行 Amazon 服务的基础设施。Amazon 还向您提供可安全使用的服务。作为 [Amazon 合规性计划](#) 的一部分，我们的安全措施的有效性定期由第三方审计员进行测试和验证。要了解适用于 Kinesis Data Streams 的合规计划，请参阅[按合规计划划分的范围内的 Amazon 服务](#)。
- 云中的安全性 - 您的责任由您使用的 Amazon 服务决定。您还需要对其他因素负责，包括您的数据的敏感性、您组织的要求以及适用的法律法规。

本文档可帮助您了解在使用 Kinesis Data Streams 时如何应用分担责任模型。以下主题向您展示如何配置 Kinesis 数据流以实现您的安全和合规性目标。您还将学习如何使用其他 Amazon 服务来帮助您监控和保护 Kinesis Data Streams 资源。

主题

- [亚马逊 Kinesis 数据流中的数据保护 \(p. 195\)](#)
- [使用 IAM 控制对亚马逊 Kinesis 数据流资源的访问权限 \(p. 201\)](#)
- [亚马逊 Kinesis 数据流的合规性验证 \(p. 205\)](#)
- [亚马逊 Kinesis 数据流中的弹性 \(p. 205\)](#)
- [Kinesis 数据流中的基础设施安全 \(p. 206\)](#)
- [Kinesis 数据流的安全最佳实践 \(p. 206\)](#)

亚马逊 Kinesis 数据流中的数据保护

使用 Amazon Key Management Service (Amazon KMS) 密钥进行服务器端加密，使您可以在 Amazon Kinesis Data Streams 中对静态数据进行加密，从而轻松满足严格的数据管理要求。

Note

如果在通过命令行界面或 API 访问 Amazon 时需要经过 FIPS 140-2 验证的加密模块，请使用 FIPS 端点。有关可用的 FIPS 端点的更多信息，请参阅 [《美国联邦信息处理标准 \(FIPS\) 第 140-2 版》](#)。

主题

- [什么是 Kinesis 数据流的服务器端加密？ \(p. 196\)](#)
- [费用、区域和性能注意事项 \(p. 196\)](#)
- [如何开始使用服务器端加密？ \(p. 197\)](#)
- [创建和使用用户生成的 KMS 主密钥 \(p. 197\)](#)
- [使用用户生成的 KMS 主密钥的权限 \(p. 197\)](#)
- [验证 KMS 密钥权限和排查问题 \(p. 199\)](#)
- [将亚马逊 Kinesis 数据流与接口 VPC 终端节点一起使用 \(p. 199\)](#)

什么是 Kinesis 数据流的服务器端加密？

服务器端加密是 Amazon Kinesis Data Streams 中的一项功能，它使用您指定的 Amazon KMS 客户主密钥 (CMK) 在数据处于静止状态之前自动对其进行加密。数据在写入 Kinesis 流存储层之前进行加密，在从存储中检索数据后进行解密。因此，您的数据将在 Kinesis 数据流服务中进行静态加密。这样，您就可以满足严格的监管要求并增强您数据的安全性。

使用服务器端加密，您的 Kinesis 直播制作者和使用者无需管理主密钥或加密操作。您的数据在进入和离开 Kinesis Data Streams 服务时会自动加密，因此您的静态数据会被加密。Amazon KMS 提供服务器端加密功能使用的所有主密钥。Amazon KMS 便于使用由 Amazon 用户指定的 CMK 或导入到服务中的主密钥管理的 Kinesis Amazon KMS CMK。Amazon KMS

Note

服务器端加密仅在启用加密后加密传入数据。启用服务器端加密后，未加密流中预先存在的数据不会加密。

费用、区域和性能注意事项

在您应用服务器端加密时，会向您收取 Amazon KMS API 使用费和密钥费。与自定义 KMS 主密钥不同，(Default) aws/kinesis 客户主密钥 (CMK) 是免费提供的。但是，您仍然必须支付 Amazon Kinesis Data Streams 代表您产生的 API 使用费用。

API 使用费适用于所有 CMK，包括自定义的 CMK。Kinesis 数据流在轮换数据密钥时 Amazon KMS 大约每五分钟调用一次。在 30 天内，由 Kinesis 数据流发起的 Amazon KMS API 调用的总费用应低于几美元。此费用随着数据创建者和使用者上使用的用户凭证数的增加而提高，因为每个用户凭证都需要对 Amazon KMS 进行唯一的 API 调用。当您使用 IAM 角色进行身份验证时，每次代入角色调用都会生成唯一的用户证书。为了节约 KMS 成本，您可能想要缓存代入角色调用返回的用户凭证。

下面按资源介绍各项费用：

键

- 由 Amazon (别名 =aws/kinesis) 管理的 Kinesis 的 CMK 是免费的。
- 用户生成的 KMS 密钥需要收取 KMS 密钥费用。有关更多信息，请参阅[Amazon 密钥管理服务定价](#)。

API 使用费适用于所有 CMK，包括自定义的 CMK。Kinesis 数据流在轮换数据密钥时大约每 5 分钟调用一次 KMS。在 30 天内，由 Kinesis 数据流发起的 KMS API 调用的总费用应低于几美元。请注意，此费用会随着您在数据创建者和使用者上使用的用户凭证数量而变化，因为每个用户凭证都需要对 Amazon KMS 进行唯一的 API 调用。当您使用 IAM 角色进行身份验证时，每个角色都 assume-role-call 将生成唯一的用户证书，您可能需要缓存返回的用户证书 assume-role-call 以节省 KMS 成本。

KMS API 使用量

对于每个加密流，当从 TIP 读取并在读取器和写入器之间使用单个 IAM 账户/用户访问密钥时，Kinesis 服务大约每 5 分钟调用该 Amazon KMS 服务 12 次。不读取 TIP 可能会导致更高的 Amazon KMS 服务呼叫次数。生成新数据加密密钥的 API 请求需要收取 Amazon KMS 使用费。有关更多信息，请参阅[Amazon 密钥管理服务定价：使用情况](#)。

按区域的服务器端加密的可用性

目前，Kinesis 数据流的服务器端加密适用于支持 Kinesis 数据流的所有区域，包括 Amazon GovCloud (美国西部) 和中国区域。有关 Kinesis 数据流支持区域的更多信息，请参阅<https://docs.aws.amazon.com/general/latest/gr/ak.html>。

性能注意事项

由于应用加密存在服务开销，应用服务器端加密的开销会将 PutRecord、PutRecords 和 GetRecords 的典型延迟增加不到 100 微秒。

如何开始使用服务器端加密？

开始使用服务器端加密的最简单方法是使用 Amazon Web Services Management Console 和 Amazon Kinesis KMS 服务密钥。aws/kinesis

以下过程演示如何为 Kinesis 流启用服务器端加密。

为 Kinesis 数据流启用服务器端加密

1. 登录 Amazon Web Services Management Console 并打开 [亚马逊 Kinesis 数据流控制台](#)。
2. 在中创建或选择 Kinesis 直播。Amazon Web Services Management Console
3. 选择 Details (详细信息) 选项卡。
4. 在 Server-side encryption (服务器端加密) 中，选择 edit (编辑)。
5. 除非您希望使用用户生成的 KMS 主密钥，否则应确保选中 (Default) aws/kinesis ((默认) aws/kinesis) KMS 主密钥。这是 Kinesis 服务生成的 KMS 主密钥。选择启用，然后选择保存。

Note

默认 Kinesis 服务主密钥是免费的，但是 Kinesis 对该 Amazon KMS 服务进行的 API 调用受 KMS 使用成本的约束。

6. 流结束挂起状态。一旦该流恢复到活动状态并启用加密，对于所有写入该流的传入数据，都将使用您选择的 KMS 主密钥进行加密。
7. 要禁用服务器端加密，请在的 Server-side encryption 中选择 Disabled Amazon Web Services Management Console，然后选择 Save。

创建和使用用户生成的 KMS 主密钥

本节介绍如何创建和使用您自己的 KMS 主密钥，而不是使用 Amazon Kinesis 管理的主密钥。

创建用户生成的 KMS 主密钥

有关创建自己的主密钥的说明，请参阅《Amazon Key Management Service 开发者指南》中的 [创建密钥](#)。为账户创建密钥后，Kinesis Data Streams 服务会在 KMS 主密钥列表中返回这些密钥。

使用用户生成的 KMS 主密钥

将正确的权限应用于您的消费者、生产者和管理员后，您可以在自己的 Amazon 账户或其他 Amazon 账户中使用自定义 KMS 主密钥。在中的 KMS Master Key Amazon Web Services Management Console 列表内显示您账户中的所有 KMS 主密钥。

要使用位于另一个账户中的自定义 KMS 主密钥，您需要有使用这些密钥的权限。您还必须在 Amazon Web Services Management Console 的 ARN 输入框中指定 KMS 主密钥的 ARN。

使用用户生成的 KMS 主密钥的权限

在您可以通过用户生成的 KMS 主密钥进行服务器端加密之前，您必须配置 Amazon KMS 密钥策略，以允许加密流以及加密和解密流记录。有关 Amazon KMS 权限的示例和更多信息，请参阅 [Amazon KMS API 权限：操作和资源参考](#)。

Note

使用默认服务密钥进行加密不需要申请自定义 IAM 权限。

在使用用户生成的 KMS 主密钥之前，请确保您的 Kinesis 流创建者和使用者（IAM 委托人）是 KMS 主密钥策略中的用户。否则，与流相关的读写操作会失败，这可能最终导致数据丢失、处理延迟或应用程序挂起。您可以使用 IAM 策略管理 KMS 密钥的权限。有关更多信息，请参阅在 [AmazonKMS 中使用 IAM 策略](#)。

创建者权限示例

您的 Kinesis 直播制作人必须 `kms:GenerateDataKey` 获得许可。

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

使用者权限示例

您的 Kinesis 直播用户必须 `kms:Decrypt` 获得许可。

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

亚马逊 Kinesis 数据分析并 Amazon Lambda 使用角色来消费 Kinesis 直播。确保将 `kms:Decrypt` 权限添加到这些使用者使用的角色。

流管理员权限

Kinesis 直播管理员必须获得授权才能致电 `kms:List*` 和 `kms:DescribeKey*`

验证 KMS 密钥权限和排查问题

在 Kinesis 直播上启用加密后，我们建议您使用以下亚马逊 CloudWatch 指标监控 `putRecord`、`putRecords`、和 `getRecords` 呼叫的成功情况：

- `PutRecord.Success`
- `PutRecords.Success`
- `GetRecords.Success`

有关更多信息，请参阅 [监控 Amazon Kinesis Data Streams \(p. 168\)](#)

将亚马逊 Kinesis 数据流与接口 VPC 终端节点一起使用

您可以使用接口 VPC 终端节点来防止您的 Amazon VPC 和 Kinesis 数据流之间的流量离开亚马逊网络。接口 VPC 终端节点不需要 Internet 网关、NAT 设备、VPN 连接或 Amazon Direct Connect 连接。接口 VPC 终端节点由 Amazon PrivateLink 一项 Amazon 技术提供支持，该技术支持使用弹性网络接口与您的 Amazon VPC 中的私有 IP 进行 Amazon 服务之间的私有通信。有关更多信息，请参阅 [Amazon 虚拟私有云](#) 和 [接口 VPC 终端节点 \(Amazon PrivateLink\)](#)。

主题

- [为 Kinesis 数据流使用接口 VPC 终端节点 \(p. 199\)](#)
- [控制 Kinesis 数据流对 VPCE 端点的访问权限 \(p. 199\)](#)
- [Kinesis 数据流的 VPC 终端节点策略的可用性 \(p. 200\)](#)

为 Kinesis 数据流使用接口 VPC 终端节点

要开始使用，您不需要更改流、创建者或用户的设置。只需创建接口 VPC 终端节点，您就可以开始流入和流向 Amazon VPC 资源的 Kinesis 数据流流量了 VPC 接口终端节点。有关更多信息，请参阅 [创建接口终端节点](#)。

Kinesis Producer Library (KPL) 和 Kinesis 消费者库 (KCL) 使用公共终端节点或私有接口 VPC 终端节点（以正在使用的为准）调用亚马逊 CloudWatch 和亚马逊 DynamoDB 等 Amazon 服务。例如，如果您的 KCL 应用程序在启用了 VPC 终端节点的 DynamoDB 接口的 VPC 中运行，则 DynamoDB 和您的 KCL 应用程序之间的调用将流经接口 VPC 终端节点。

控制 Kinesis 数据流对 VPCE 端点的访问权限

借助 VPC 终端节点策略，您可以控制访问，其方式是：将策略附加到 VPC 终端节点或使用附加到 IAM 用户、组或角色的策略中的额外字段，从而限制只能通过特定 VPC 终端节点进行访问。这些策略可用于限制对特定流的访问、当与 IAM 策略共同使用时对特定 VPC 终端节点的访问，以及只能通过特定 VPC 终端节点对 Kinesis 数据流进行授权访问。

以下是用于访问 Kinesis 数据流的示例终端节点策略。

- VPC 策略示例：只读访问权限-此示例策略可以附加到 VPC 终端节点。（有关更多信息，请参阅 [控制对 Amazon VPC 资源的访问](#)）。它限制仅能通过其附加的 VPC 终端节点列出或描述 Kinesis 数据流。

```
{
  "Statement": [
    {
      "Sid": "ReadOnly",
      "Principal": "*",
      "Action": [
        "kinesis:List*",
        "kinesis:Describe*"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

- VPC 策略示例：限制对特定 Kinesis 数据流的访问 – 此示例策略可以附加到 VPC 终端节点。它限制通过其附加的 VPC 终端节点访问特定的数据流。

```
{
  "Statement": [
    {
      "Sid": "AccessToSpecificDataStream",
      "Principal": "*",
      "Action": "kinesis:*",
      "Effect": "Allow",
      "Resource": "arn:aws:kinesis:us-east-1:123456789012:stream/MyStream"
    }
  ]
}
```

- IAM 策略示例：限制只能通过特定的 VPC 终端节点访问特定流 – 此示例策略可以附加到 IAM 用户、角色或组。它限制只能通过特定的 VPC 终端节点访问特定的 Kinesis 数据流。

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

Kinesis 数据流的 VPC 终端节点策略的可用性

以下区域支持带策略的 Kinesis Data Streams 接口 VPC 终端节点：

- 欧洲地区（巴黎）
- 欧洲地区（爱尔兰）
- 美国东部（弗吉尼亚州北部）
- 欧洲地区（斯德哥尔摩）

- 美国东部 (俄亥俄州)
- 欧洲地区 (法兰克福)
- 南美洲 (圣保罗)
- 欧洲地区 (伦敦)
- 亚太地区 (东京)
- 美国西部 (北加利福尼亚)
- 亚太地区 (新加坡)
- 亚太地区 (悉尼)
- 中国 (北京)
- China (Ningxia)
- 亚太地区 (香港)
- 中东 (巴林)
- 中东 (阿联酋)
- 欧洲地区 (米兰)
- 非洲 (开普敦)
- 亚太地区 (孟买)
- 亚太地区 (首尔)
- 加拿大 (中部)
- 美国西部 (俄勒冈) 除了 usw2-az4
- AmazonGovCloud (美国东部)
- AmazonGovCloud (美国西部)
- 亚太地区 (大阪)
- 欧洲 (苏黎世)
- 亚太地区 (海得拉巴)

使用 IAM 控制对亚马逊 Kinesis 数据流资源的访问权限

Amazon Identity and Access Management(IAM) 使您能够执行以下操作：

- 在您的 Amazon 账户下创建用户和组
- 为您的 Amazon 账户下的每个用户分配唯一的安全凭证
- 控制每个用户使用 Amazon 资源执行任务的权限
- 允许另一 Amazon 账户的用户共享您的 Amazon 资源
- 创建 Amazon 账户角色并定义可以担任这些角色的用户或服务
- 借助企业的现有身份，授予使用 Amazon 资源执行任务的权限

通过将 IAM 与 Kinesis Data Streams 结合使用，您可以控制组织中的用户是否可以使用特定的 Kinesis Data Streams API 操作执行任务，以及他们是否可以使用特定 Amazon 资源。

如果您正在使用 Kinesis 客户端库 (KCL) 开发应用程序，则您的策略必须包含亚马逊 DynamoDB 和亚马逊的权限 CloudWatch；KCL 使用 DynamoDB 跟踪应用程序的状态信息，并 CloudWatch 代表您向其发送 KCL 指标。CloudWatch 有关 KCL 的更多信息，请参阅 [开发 KCL 1.x 使用者 \(p. 117\)](#)。

有关 IAM 的更多信息，请参阅以下文档：

- [Amazon Identity and Access Management \(IAM\)](#)
- [入门](#)
- [IAM 用户指南](#)

有关 IAM 和亚马逊 DynamoDB 的更多信息，请参阅亚马逊 DynamoDB 开发[者指南中的使用 IAM 控制对亚马逊 DynamoDB 资源的访问权限](#)。

有关 IAM 和 Amazon 的更多信息 CloudWatch，请参阅亚马逊[用户指南中的控制 CloudWatch 用户对您 Amazon 账户的访问权限](#)。

目录

- [策略语法 \(p. 202\)](#)
- [针对 Kinesis 数据流的操作 \(p. 202\)](#)
- [Kinesis 数据流的亚马逊资源名称 \(ARN\) \(p. 203\)](#)
- [Kinesis 数据流的策略示例 \(p. 203\)](#)

策略语法

IAM policy 是包含一个或多个语句的 JSON 文档。每个语句的结构如下：

```
{
  "Statement": [{
    "Effect": "effect",
    "Action": "action",
    "Resource": "arn",
    "Condition": {
      "condition": {
        "key": "value"
      }
    }
  }]
}
```

组成语句的各个元素如下：

- Effect：此 effect 可以是 Allow 或 Deny。在默认情况下，IAM 用户没有使用资源和 API 操作的许可，因此，所有请求均会被拒绝。显式允许将覆盖默认规则。显式拒绝将覆盖任何允许。
- Action：action 是对其授予或拒绝权限的特定 API 操作。
- Resource：受操作影响的资源。要在语句中指定资源，您需要使用其 Amazon 资源名称 (ARN)。
- Condition：条件是可选的。它们可以用于控制策略生效的时间。

在创建和管理 IAM 策略时，您可能需要使用 [IAM 策略生成器](#) 和 [IAM 策略模拟器](#)。

针对 Kinesis 数据流的操作

在 IAM policy 语句中，您可以从支持 IAM 的任何服务中指定任何 API 操作。对于 Kinesis 数据流，请使用以下前缀作为 API 操作的名称：kinesis:。例如：kinesis:CreateStream、kinesis:ListStreams 和 kinesis:DescribeStreamSummary。

要在单个语句中指定多项操作，请使用逗号将它们隔开，如下所示：

```
"Action": ["kinesis:action1", "kinesis:action2"]
```

您也可以使用通配符指定多项操作。例如，您可以指定名称以单词“Get”开头的所有操作，如下所示：

```
"Action": "kinesis:Get*"
```

要指定所有 Kinesis 数据流操作，请按如下方式使用 * 通配符：

```
"Action": "kinesis:*"
```

有关 Kinesis Data Streams API 操作的完整列表，请参阅[亚马逊 Kinesis API 参考](#)。

Kinesis 数据流的亚马逊资源名称 (ARN)

每个 IAM policy 语句适用于您使用资源的 ARN 指定的资源。

对于 Kinesis 数据流，使用以下 ARN 资源格式：

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

例如：

```
"Resource": arn:aws:kinesis:*:111122223333:stream/my-stream
```

Kinesis 数据流的策略示例

以下示例策略演示了如何控制用户对 Kinesis 数据流的访问权限。

Example 1：允许用户从流中获取数据

此策略允许用户或组对特定流执行 DescribeStreamSummary、GetShardIterator 和 GetRecords 操作，对任何流执行 ListStreams 操作。此策略可应用于应该能够从特定流获取数据的用户。

```
{
  "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：允许用户将数据添加到账户中的流

此策略允许用户或组对账户的任一使用 PutRecord 操作。此策略可应用于应该能够向账户中的所有流添加数据记录的用户。

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "Action": [  
        "kinesis:PutRecord"  
      ],  
      "Resource": [  
        "arn:aws:kinesis:us-east-1:111122223333:stream/*"  
      ]  
    }  
  ]  
}
```

Example 3：允许对特定流执行任何 Kinesis 数据流操作

此策略允许用户或群组对指定流使用任何 Kinesis 数据流操作。此策略可应用于应该对特定流有管理控制权限的用户。

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "Action": "kinesis:*",  
      "Resource": [  
        "arn:aws:kinesis:us-east-1:111122223333:stream/stream1"  
      ]  
    }  
  ]  
}
```

Example 4：允许对任何流进行任何 Kinesis 数据流操作

此策略允许用户或群组对账户中的任何数据流使用任何 Kinesis 数据流操作。由于此策略会授予对您的所有流的完全访问权限，您应该将其限制为仅对管理员可用。

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "Action": "kinesis:*",  
      "Resource": [  
        "arn:aws:kinesis:*:111122223333:stream/*"  
      ]  
    }  
  ]  
}
```

亚马逊 Kinesis 数据流的合规性验证

作为多个合规计划的一部分，第三方审计师会评估 Amazon Kinesis Data Streams 的安全性、合规性和其他方面。其中包括 SOC、PCI、FedRAMP、HIPAA 及其它。

有关特定合规计划范围内的 Amazon 服务的列表，请参阅[合规性计划范围内的 Amazon 服务](#)。有关一般信息，请参阅[Amazon 合规性计划](#)。

您可以使用 Amazon Artifact 下载第三方审计报告。有关更多信息，请参阅[下载 Amazon Artifact 中的报告](#)。

使用 Kinesis Data Streams 时的合规责任由数据的敏感性、贵公司的合规目标以及适用的法律和法规决定。如果您对 Kinesis Data Streams 的使用必须符合 HIPAA、PCI 或 FedRAMP 等标准，请提供资源来帮助您：Amazon

- [《安全性与合规性快速入门指南》](#) – 这些部署指南讨论了架构注意事项，并提供了在 Amazon 上部署基于安全性和合规性的基准环境的步骤。
- [设计符合 HIPAA 安全性和合规性要求的架构白皮书](#) – 此白皮书介绍公司如何使用 Amazon 创建符合 HIPAA 标准的应用程序。
- [Amazon 合规资源](#) – 这组工作簿和指南可能适用于您的行业和地点
- [Amazon Config](#) – 此 Amazon 服务评估您的资源配置在多大程度上符合内部惯例、行业准则和法规。
- [Amazon Security Hub](#) – 此 Amazon 服务提供了 Amazon 中安全状态的全面视图，可帮助您检查是否符合安全行业标准和最佳实践。

亚马逊 Kinesis 数据流中的弹性

Amazon 全球基础设施围绕 Amazon 区域和可用区构建。Amazon 区域提供多个在物理上独立且隔离的可用区，这些可用区通过延迟低、吞吐量高且冗余性高的网络连接在一起。利用可用区，您可以设计和操作在可用区之间无中断地自动实现故障转移的应用程序和数据库。与传统的单个或多个数据中心基础设施相比，可用区具有更高的可用性、容错性和可扩展性。

有关 Amazon 区域和可用区的更多信息，请参阅[Amazon 全球基础设施](#)。

除了 Amazon 全球基础架构外，Kinesis Data Streams 还提供多项功能，以帮助支持您的数据恢复和备份需求。

亚马逊 Kinesis 数据流中的灾难恢复

当您使用 Amazon Kinesis Data Streams 应用程序处理流中的数据时，故障可能发生在以下级别：

- 记录处理器可能失败
- 工作程序可能失败，或者已实例化工作程序的应用程序的实例可能失败
- 托管该应用程序的一个或多个实例的 EC2 实例可能失败

记录处理器失败

工作器使用 Java [ExecutorService](#) 任务调用记录处理器方法。如果任务失败，工作程序将保留对记录处理器之前在处理的分片的控制。工作程序启动一项新的记录处理器任务来处理该分片。有关更多信息，请参阅[读取限制 \(p. 167\)](#)：

工作程序或应用程序失败

如果工作人员（或 Amazon Kinesis Data Streams 应用程序的实例）出现故障，则应检测并处理这种情况。例如，如果 `Worker.run` 方法引发了一项异常，则您应捕获并处理它。

如果应用程序本身失败，您应检测此应用程序并重新启动它。该应用程序启动时，它会实例化一个新工作程序，这会反过来实例化自动获得要处理的分片的新记录处理器。这些分配可能是记录处理器在失败前正在处理的相同分片或这些处理器新接触的分片。

在工作程序或应用程序失败、未检测到失败且应用程序有其他实例正在其他 EC2 实例上运行的情况下，这些实例上的工作程序会处理该失败。它们将创建额外的记录处理器来处理不再由失败的工作程序处理的分片。这些其他 EC2 实例上的负载也会相应地增加。

此处描述的场景假设尽管工作线程或应用程序出现故障，但托管 EC2 实例仍在运行，因此未由 Auto Scaling 组重启。

亚马逊 EC2 实例故障

我们建议您在 Auto Scaling 组中为您的应用程序运行 EC2 实例。这样，如果其中一个 EC2 实例出现故障，Auto Scaling 组会自动启动一个新实例来替换它。您应将实例配置为在启动时启动您的 Amazon Kinesis 数据流应用程序。

Kinesis 数据流中的基础设施安全

作为一项托管服务，Amazon Kinesis Data Streams 受 Amazon 全球网络安全程序的保护，如[亚马逊网络服务：安全流程概述白皮书](#)所述。

您可以使用 Amazon 已发布的 API 调用通过网络访问 Kinesis 数据流。客户端必须支持传输层安全性 (TLS) 1.2 或更高版本。客户端还必须支持具有完全向前保密 (PFS) 的密码套件，例如 Ephemeral Diffie-Hellman (DHE) 或 Elliptic Curve Ephemeral Diffie-Hellman (ECDHE)。大多数现代系统（如 Java 7 及更高版本）都支持这些模式。

此外，必须使用访问密钥 ID 和与 IAM 委托人关联的秘密访问密钥来对请求进行签名。或者，您可以使用[Amazon Security Token Service](#) (Amazon STS) 生成临时安全凭证来对请求进行签名。

Kinesis 数据流的安全最佳实践

Amazon Kinesis Data Streams 提供了许多安全功能，供您在制定和实施自己的安全策略时考虑。以下最佳实践是一般指导原则，并不代表完整安全解决方案。由于这些最佳实践可能不适合您的环境或不满足您的环境要求，因此将其视为有用的考虑因素而不是惯例。

实施最低权限访问

授予权限时，由您决定谁将获得哪些 Kinesis Data Streams 资源的权限。您可以对这些资源启用希望允许的特定操作。因此，您应仅授予执行任务所需的权限。实施最低权限访问对于减小安全风险以及可能由错误或恶意图造成的影响至关重要。

使用 IAM 角色

生产者和客户端应用程序必须具有有效凭据才能访问 Kinesis 数据流。您不应将 Amazon 证书直接存储在客户端应用程序或 Amazon S3 存储桶中。这些是不会自动轮换的长期凭证，如果它们受到损害，可能会对业务产生重大影响。

相反，您应该使用 IAM 角色来管理生产者和客户端应用程序的临时证书，以访问 Kinesis 数据流。在使用角色时，您不必使用长期凭证（如用户名和密码或访问密钥）来访问其他资源。

有关更多信息，请参阅 IAM 用户指南中的以下主题：

- [IAM 角色](#)
- [针对角色的常见情形：用户、应用程序和服务](#)

实施从属资源中的服务器端加密

可以在 Kinesis 数据流中对静态数据和传输中的数据进行加密。有关更多信息，请参阅[亚马逊 Kinesis 数据流中的数据保护 \(p. 195\)](#)：

使用 CloudTrail 监控 API 调用

Kinesis Data Streams 与一项服务集成在一起 Amazon CloudTrail，该服务在 Kinesis Data Streams 中提供用户、角色或 Amazon 服务所采取的操作的记录。

使用收集的信息 CloudTrail，您可以确定向 Kinesis Data Streams 发出的请求、发出请求的 IP 地址、谁发出了请求、何时发出以及其他详细信息。

有关更多信息，请参阅 [the section called “记录 Amazon Kinesis Data Streams API 调用 Amazon CloudTrail” \(p. 178\)](#)。

文档历史记录

下表介绍了 Amazon Kinesis Data Streams 文档的重要更改。

更改	描述	更改日期
添加了对按需和预配置数据流容量模式的支持。	增加了 选择数据流容量模式 (p. 59) 。	2021 年 11 月 29 日
新增了服务器端加密的相关内容	增加了 亚马逊 Kinesis 数据流中的数据保护 (p. 195) 。	2017 年 7 月 7 日
新增内容，用于增强 CloudWatch 指标。	已更新 监控 Amazon Kinesis Data Streams (p. 168) 。	2016 年 4 月 19 日
新增有关增强的 Kinesis 代理的内容。	已更新 使用 Amazon Kinesis Data Streams s Kinesis Streams (p. 90) 。	2016 年 4 月 11 日
新增有关使用 Kinesis 代理的内容。	增加了 使用 Amazon Kinesis Data Streams s Kinesis Streams (p. 90) 。	2015 年 10 月 2 日
更新 0.10.0 版的 KPL 内容。	增加了 使用 Amazon Kinesis 生产者库开发生产者 (p. 76) 。	2015 年 7 月 15 日
更新可配置指标的 KCL 指标主题。	增加了 使用 Amazon CloudWatch 监控 Kinesis 客户端库 (p. 182) 。	2015 年 7 月 9 日
重新组织了内容。	为提供更简洁的树视图和更有逻辑的分组，内容主题经过了大幅地重新组织。	2015 年 7 月 01 日
“新 KPL 开发人员的指南”主题。	增加了 使用 Amazon Kinesis 生产者库开发生产者 (p. 76) 。	2015 年 6 月 02 日
“新 KCL 指标”主题。	增加了 使用 Amazon CloudWatch 监控 Kinesis 客户端库 (p. 182) 。	2015 年 5 月 19 日
对 KCL .NET 的支持	增加了 在 .NET 中开发 Kinesis Client Library 使用者 (p. 124) 。	2015 年 5 月 1 日
对 KCL Node.js 的支持	增加了 在 Node.js 中开发 Kinesis Client Library 使用者 (p. 121) 。	2015 年 3 月 26 日
对 KCL Ruby 的支持	添加了指向 KCL Ruby 库的链接。	2015 年 1 月 12 日
新 API PutRecords	添加了有关新增信息 PutRecords API 到 the section called “使用添加多个记录PutRecords” (p. 87) 。	2014 年 12 月 15 日
对标签的支持	增加了 在 Amazon Kinesis Data Streams 中为流添加标签 (p. 72) 。	2014 年 9 月 11 日
New CloudWatch 公制	向 GetRecords.IteratorAgeMilliseconds 添加了指标 Amazon Kinesis Data Streams 维度和指标 (p. 168) 。	2014 年 9 月 3 日
新的监控章节	添加了 监控 Amazon Kinesis Data Streams (p. 168) 和 使用 Amazon CloudWatch 监控 Amazon Kinesis Data Streams (p. 168) 。	2014 年 7 月 30 日

更改	描述	更改日期
默认分片限制	更新了 配额和限制 (p. 6) ：默认分片限制从 5 个提高到 10 个。	2014 年 2 月 25 日
默认分片限制	更新了 配额和限制 (p. 6) ：默认分片限制从 2 个提高到 5 个。	2014 年 1 月 28 日
API 版本更新	更新版本 2013-12-02 的 Kinesis Data Streams API。	2013 年 12 月 12 日
首次发布	首次发布 Amazon Kinesis 开发人员指南。	2013 年 11 月 14 日

Amazon词汇表

有关最新 Amazon 术语，请参阅《Amazon Web Services 一般参考》中的 [Amazon 术语表](#)。

本文属于机器翻译版本。若本译文内容与英语原文存在差异，则一律以英文原文为准。