Amazon Kinesis Data Streams

AWS Black Belt Online Seminar

Takayuki Enomoto

Solutions Architect
2023/04

AWS Black Belt Online Seminarとは

- 「サービス別」「ソリューション別」「業種別」などのテーマに分け、アマゾン ウェブ サービス ジャパン合同会社が提供するオンラインセミナーシリーズです
- AWSの技術担当者が、AWSの各サービスやソリューションについてテーマごとに動画を公開します
- 動画を一時停止・スキップすることで、興味がある分野・項目だけの聴講も可能、スキマ時間の学習にもお役立ていただけます
- 以下のURLより、過去のセミナー含めた資料などをダウンロードすることができます
  - https://aws.amazon.com/jp/aws-jp-introduction/aws-jp-webinar-service-cut/

内容についての注意点

- 本資料では 2023 年 04 月時点のサービス内容および価格について説明しています。最新の情報は AWS 公式ウェブサイト(https://aws.amazon.com/)よりご確認ください
- 資料作成には十分注意しておりますが、資料内の価格と AWS 公式ウェブサイト記載の価格に相違があった場合、AWS 公式ウェブサイトの価格を優先とさせていただきます
- 価格は税抜表記となっています。日本居住者のお客様には別途消費税をご請求させていただきます

自己紹介

名前：榎本 貴之 (Enomoto, Takayuki)

所属：アマゾンウェブサービスジャパン
データ事業本部
サービスソリューションアーキテクト本部
シニアソリューションアーキテクト

経歴：インフラエンジニア @システムインテグレーター
-> インフラエンジニア @ゲーム会社
-> Cloud Support Engineer @AWS
-> **Solutions Architect @AWS**

好きなAWSサービス: OpenSearch, AWS Support
Amazon QuickSight, Amazon Neptune,
Amazon EventBridge, AWS Config, Amazon CloudWatch,
Amazon Kinesis, Amazon MSK

本セミナーの対象者

リアルタイム分析の導入を検討されている方

Kinesis Data Streams におけるリアルタイム処理の実装方法を確認したい方

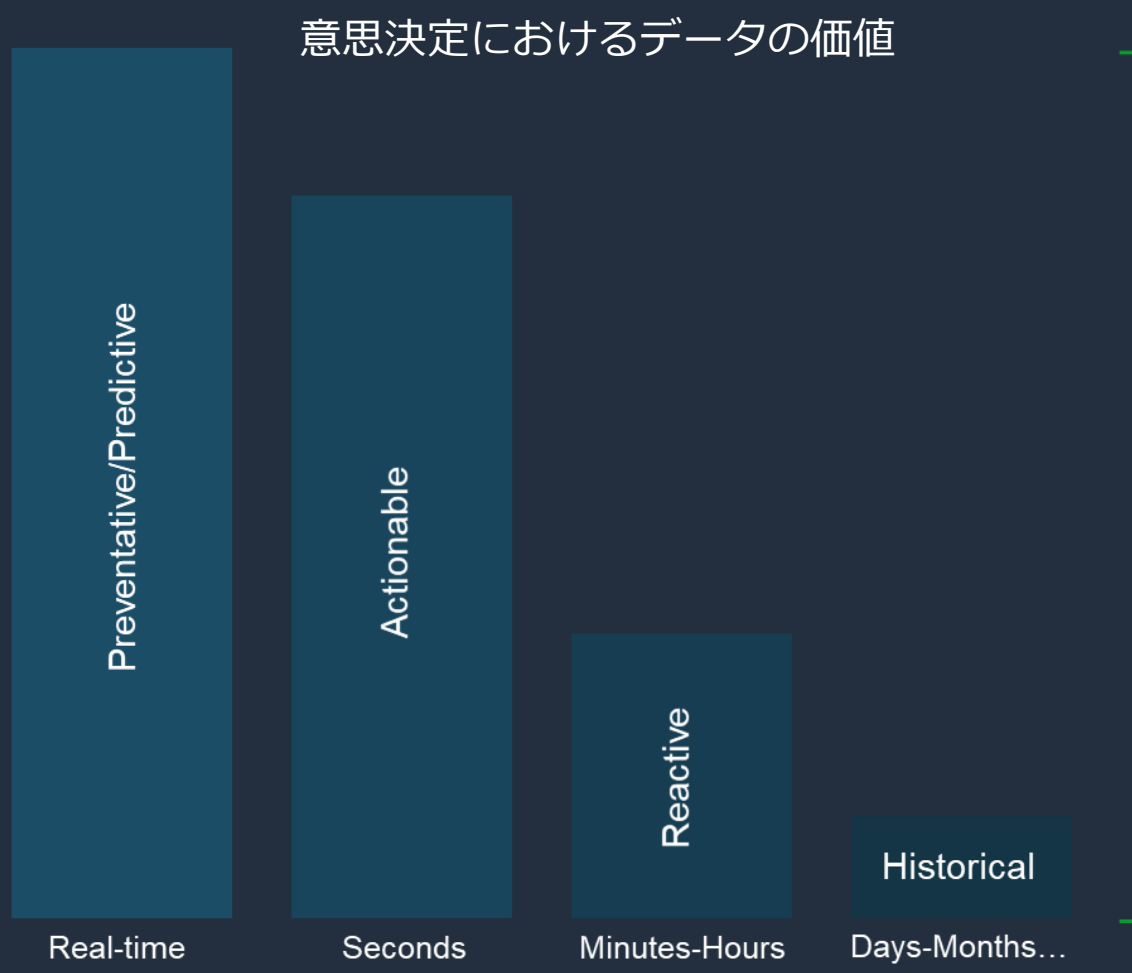
Kinesis Data Streams について改めて学びなおしたい方

アジェンダ

1. リアルタイム分析とストリーム処理
2. Amazon Kinesis Data Streams 概要
3. プロデューサーとコンシューマー
4. Kinesis Data Streams の運用
   (モード選択、拡張、セキュリティ、モニタリング、
   トラブルシュート)

リアルタイム分析とストリーム処理

データの価値は時間の経過とともに減少する



58% 顧客データの**リアルタイム分析**により、顧客維持率とロイヤルティが大幅に向上したと答えた回答者の割合

75% タイムリーでないデータがビジネスチャンスを妨げていると考えている調査対象企業の割合

https://www.forrester.com/report/Perishable+Insights+Stop+Wasting+Money+On+Unactionable+Analytics/-/E-RES135301#

https://hbr.org/resources/pdfs/comm/sas/Real.Time.Analytics.pdf

https://www.intersystems.com/news-events/news/news-item/survey-reveals-real-time-data-disconnect-leading-lost-business-opportunity/

リアルタイム分析のユースケース

異常検出、不正検出

リアルタイムな顧客体験のカスタマイズ

IoT 分析の強化

マーケティングキャンペーンの充実

リアルタイムなパーソナライゼーション

医療などの緊急性の高いサービスのサポート

従来のバッチ処理

- データストアに格納されたデータに対して一括で処理を行う
- 処理の性質上、データは予めデータストアに格納されていることが前提となる
- **データが発生してから処理可能になるまで時間がかかる**

データの生成

アプリケーションから
随時、定期的に
データが生成される

データの保存

アプリケーションが
生成したデータは、
データベースなどの
データストアに保存される

データの取得・分析

蓄積されたデータストアの
データを取得し、
分析を行う

処理・分析結果の格納

バッチ処理の結果は必要に応じて
再度データストアにロードされる

バッチ処理ではこうした要求にこたえることは難しい

milliseconds

マイクロサービス間の
メッセージング応答分析

Web ・モバイル
アプリケーションの通知

seconds

ログの取り込み

IoT デバイスのメンテナンス

Change Data Capture (CDC)

minutes

ストリーミングデータの
ETL 処理

データレイクや
データウェアハウスへの
データ配信

ストリーム処理によるリアルタイム分析

- さまざまなソースから生成される大量のデータをリアルタイムで収集し、少量ずつ処理する
- バッチ処理との大きな違いはデータの処理順序。データ格納は分析の後に行う

ソース

リアルタイムデータを高速に生成するデバイスやアプリケーション

ストリームデータ取り込み

数万のデータソースから生成されたデータが一つのストリームに書き込まれる

ストリームデータ変換

レコードを生成された順序で読み取り、リアルタイム分析・ETL 処理を実行

ストリームデータ配信

データウェアハウス

データストア

データレイク

分析ツール

ストリームデータからインサイトを獲得

Amazon Athena

Presto

Amazon QuickSight

ストリーム処理における課題

セットアップが困難

スケーリングが困難

高可用性の達成が困難

開発コストが高い

エラーが発生しやすく、管理が複雑

メンテナンスコストが高い

AWS が提供するデータストリーミングサービス

Amazon Kinesis Data Streams

ストリームデータの収集、保存

Amazon Kinesis Video Streams

ビデオストリームの収集、保存

Amazon Kinesis Data Firehose

ストリームデータを AWS または外部サービスへロード

Amazon Kinesis Data Analytics

Apache Flink によるストリームデータ分析

Amazon Managed Streaming for Apache Kafka

フルマネージド、高可用性、セキュアな Apache Kafka のマネージドサービス

AWS が提供するデータストリーミングサービス

Amazon Kinesis Data Streams

ストリームデータの収集、保存

Amazon Kinesis Video Streams

ビデオストリームの収集、保存

Amazon Kinesis Data Firehose

ストリームデータを AWS または外部サービスへロード

Amazon Kinesis Data Analytics

Apache Flink によるストリームデータ分析

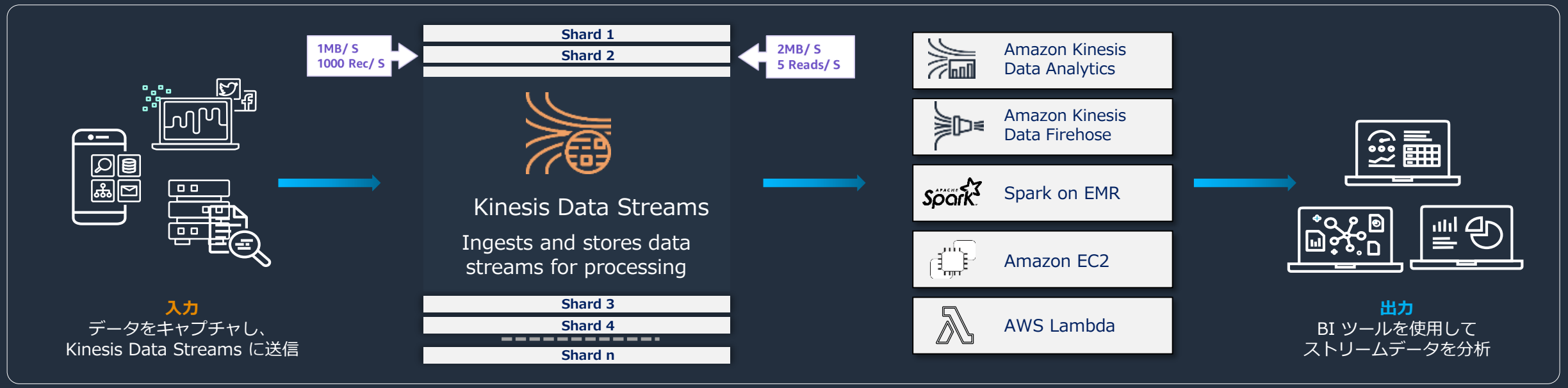
Amazon Managed Streaming for Apache Kafka

フルマネージド、高可用性、セキュアな Apache Kafka のマネージドサービス

Amazon Kinesis Data Streams 概要

Amazon Kinesis Data Streams

大量のストリームデータをリアルタイムで収集するための**サーバーレスストリーミングデータサービス**



- 高い管理性、コストパフォーマンス
- 柔軟かつ高いスケーラビリティ
- 可用性が高く暗号化に対応したストレージ
- 複数のリアルタイム分析アプリケーションとの連携
- 標準 24 時間、最長 1 年のデータ保持
- 単体の標準コンシューマーで平均 200ms のレイテンシ
- 拡張コンシューマーで平均 70ms のレイテンシ

Amazon Kinesis Data Streams の特徴

Easy-to-use

ストリームを作成するだけで、すぐにデータを取り込める

開発ライブラリ、エージェントは AWS より提供

高い拡張性

要件に応じて自由にスループットを調整可能

オートスケールをサポート

高可用性 高耐久性

3 つの AZ にデータを複製

AWS サービスとのシームレスな統合

各種AWS サービスによるネイティブサポート

運用の省力化

サーバーレスアーキテクチャー・フルマネージド

インフラ運用コスト削減

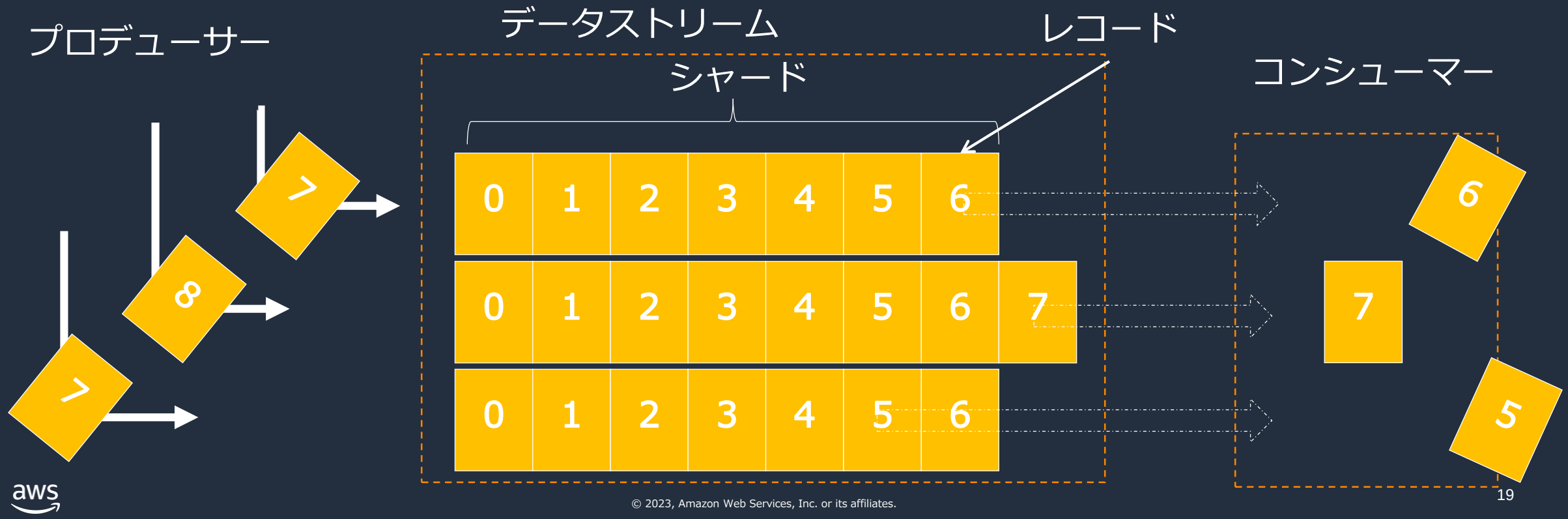
コスト パフォーマンス

ワークロードに応じて 2 種類の料金プランを選択可能

キャパシティまたはデータ量に応じた従量課金

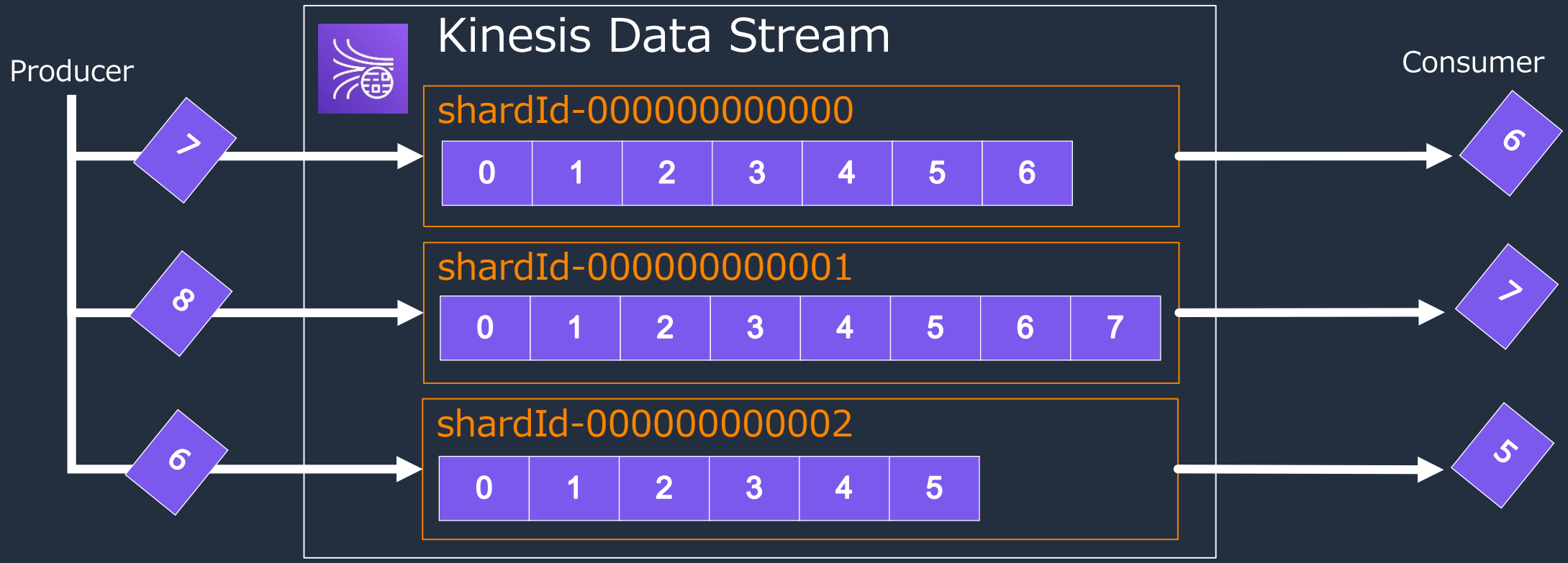
Kinesis Data Streams の利用の流れ

1. データストリームをセットアップする
2. プロデューサーをセットアップし、ソースから取得したデータレコードを送信する
3. コンシューマーアプリケーションを実装し、データレコードを取得して処理する



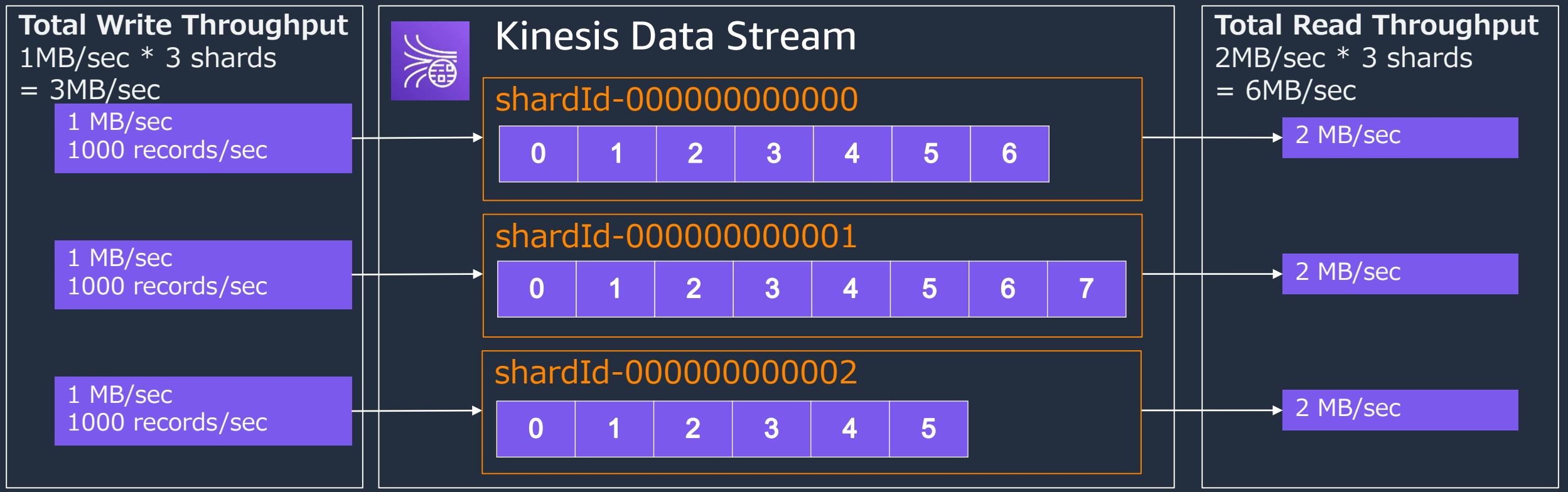
データストリーム

- **プロデューサー**から送信された**データレコード**を保持するデータストア
- 単一のデータストリームは複数の**シャード**から構成されている
- レコードは**コンシューマー**によって取得、処理される



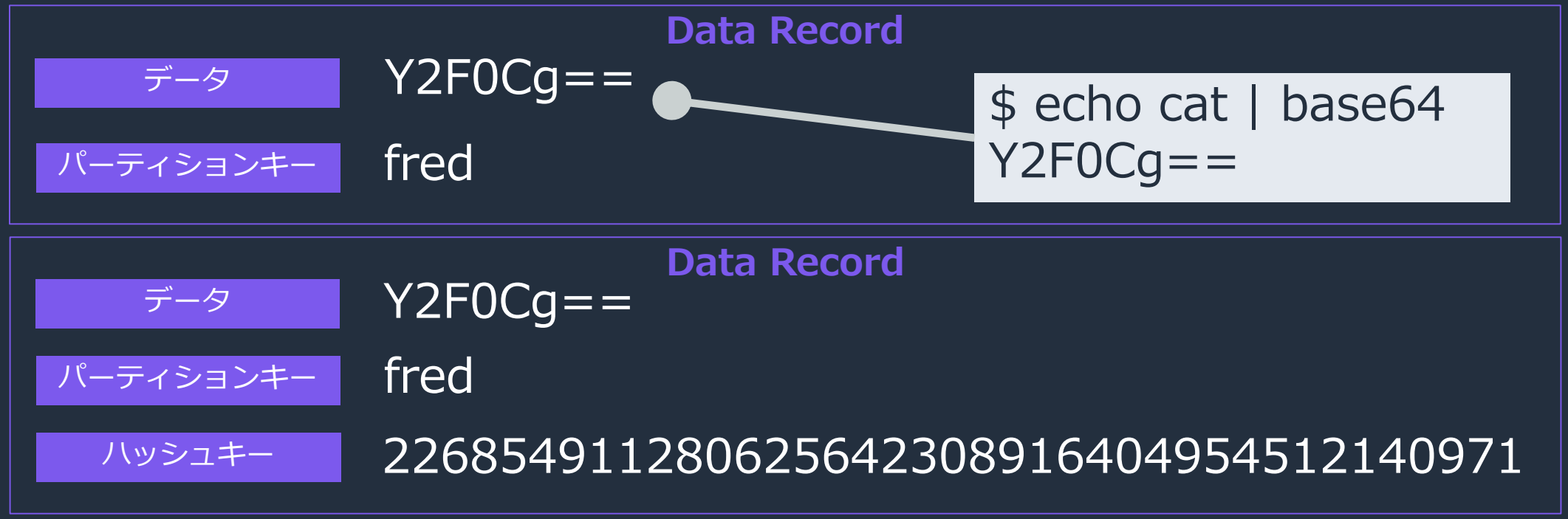
シャード

- ストリーム内のリソースの一単位。書き込み、読み取りのスループットがシャードごとに割り当てられており、シャードを増減させることでストリーム全体のキャパシティをコントロールできる
- データの書き込み、取り出しはシャードを指定して行う



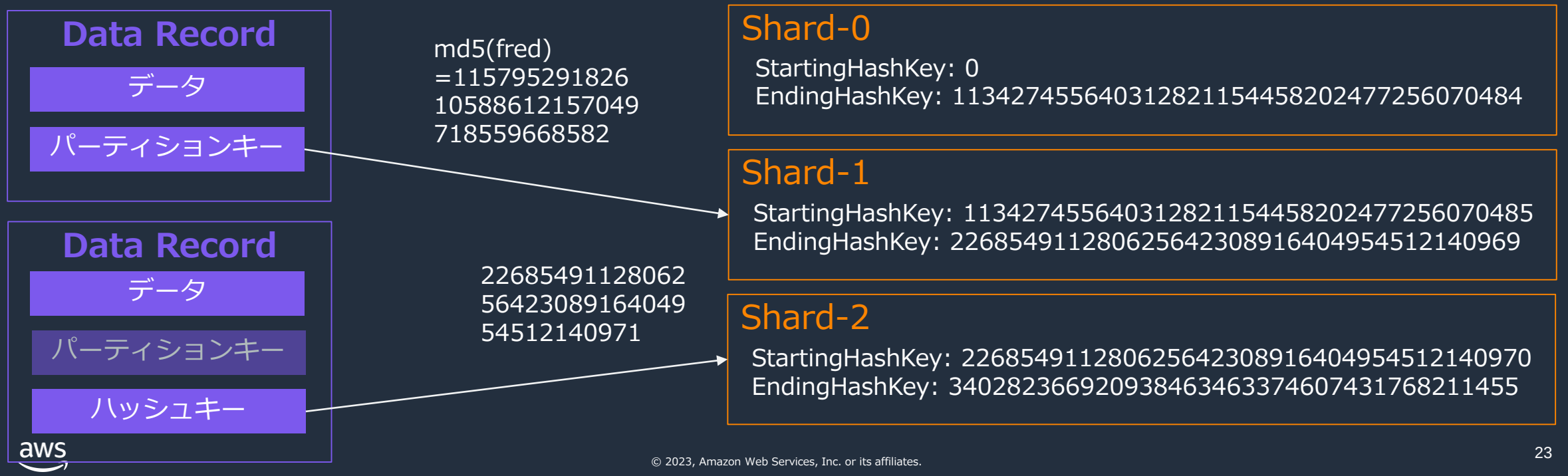
データレコード

- データストリームに送信されるメッセージの一単位
- Base 64 エンコードされたメッセージ本体(データ)、Unicode 文字列のパーティションキー、整数値のハッシュキー(オプション)で構成されている



データレコードの送信

- 各シャードにはハッシュキーレンジが割り当てられており、データレコードは対応するハッシュキーのシャードに送信される
- パーティションキーを指定した場合、ハッシュキーはパーティションキーを元にMD5 ハッシュ関数で生成される
- 明示的にハッシュキーを指定した場合、対応するシャードにそのままレコードが送信される



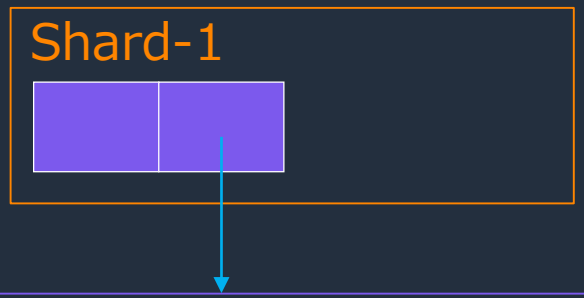
データレコードの制約

Data Record

データ	Base64 エンコードされた文字列。最大サイズは **1 MiB**
パーティションキー	Unicode 文字。最大 256 文字まで
ハッシュキー	整数。値の範囲は 0 から 2^128-1 まで

データレコードに付与されるメタデータ

- シャード内で一意のシーケンス番号が割り当てられる
- その他に、レコード送信時に指定されたパーティションキー、ストリームに到着したおよその時刻がメタデータとして付与される




```
{
  "SequenceNumber": "49622934722215519172145910627629375010673483513977634834",
  "ApproximateArrivalTimestamp": "2021-10-14T14:44:03.946000+09:00",
  "Data": "Y2F0Cg==",
  "PartitionKey": "fred"
}
```

データレコードの保持期間

- 保持期間はデフォルトで 24 時間。設定により 8760 時間(= 365 日)まで延長可能
  - 1 年ではなく 365 日であるという点に注意(閏年は考慮されない)
- 保持期間は動的に変更が可能
- 保持期間を変更すると、既存のレコードについても新しい保持期限が設定される
  - 保持期間を 24 時間から 48 時間に変更した場合、
    23 時間 55 分前にストリームに追加されたレコードは、さらに 24 時間後まで使用可能
- 保持期間を短縮した場合、保持期間外となったデータへのアクセスはできなくなる

標準期間 (24 h) 追加料金なし	標準期間(~168 h) 1 シャードにつき 0.026 USD/hour *	標準期間(~8760 h) データ保持: 1 GB につき 0.025 USD/month * データ取得: 1 GB につき 0.0273 USD *

* 東京リージョンにおける料金(2023/04 現在)

データレコードが取得可能になるまでの時間

通常は 1 秒未満

Kinesis Data Streams を使用する利点

Kinesis Data Streams は、さまざまなデータストリーミングの問題解決に使用できますが、一般的にデータのリアルタイム集計にも使用できます。集計データはその後でデータウェアハウスや MapReduce クラスターに読み込むことができます。

データは Kinesis Data Streams に取り込むことができるため、耐久性と伸縮性が確保されます。レコードがストリームに取り込まれてから取得されるまでの遅延 (put-to-getdelay) は通常、1 秒未満です。つまり、Kinesis Data Streams アプリケーションは、データが追加されると同時にストリームからデータを消費し始めることができます。Kinesis Data Streams は、マネージド型サービスであるため、データ取り込みパイプラインの作成と実行にかかわる運用負荷が軽くなります。MapReduce タイプのストリーミングアプリケーションを作成することができます。Kinesis Data Streams は伸縮性に優れており、ストリームをスケールアップまたはスケールダウンできるため、有効期限が切れる前にデータレコードがなくなることはありません。

https://docs.aws.amazon.com/ja_jp/streams/latest/dev/introduction.html#using-the-service

Kinesis Data Streams へのアクセス

- Kinesis Data Streams では、 Rest API を用いて
  リソースの管理やデータ操作を行う
- サービス共通、アカウント固有の Rest API エンドポイントが
  用意されている
- インターネット経由でアクセス可能なパブリックエンドポイント、
  VPC および VPC に接続されたプライベートネットワークから
  アクセス可能な VPC エンドポイントの 2 種類が提供されている

Kinesis Data Streams の API

- リソースを制御する control API と、レコード追加・取得に関する Data API が提供されている
- Consumer や Producer はアクティブなシャードを取得するために、ListShards など一部の control API も使用する

Control API

CreateStream
DeleteStream

SplitShard
MergeShards
UpdateShardCount
UpdateStreamMode
IncreaseStreamRetentionPeriod
DecreaseStreamRetentionPeriod

RegisterStreamConsumer
DeregisterStreamConsumer

ListStreams
DescribeLimits

StartStreamEncryption
StopStreamEncryption
EnableEnhancedMonitoring
DisableEnhancedMonitoring
AddTagsToStream
ListTagsForStream
RemoveTagsFromStream

DescribeStream
DescribeStreamSummary
ListShards
DescribeStreamConsumer
ListStreamConsumers

Data API

- GetRecords
- GetShardIterator
- PutRecord
- PutRecords
- SubscribeToShard

Kinesis Data Streams のエンドポイント

最新の SDK やライブラリでは、ストリーム指定時にストリーム名ではなくストリーム ARN を使用することで、アカウントエンドポイントへリクエストが発行される。API に応じて、control または data エンドポイントのいずれかが使用される。SDK や CLI を使用する場合、エンドポイントの判断は自動で行われる

- **<account-id>.control-kinesis.<region>.amazonaws.com**
- **<account-id>.data-kinesis.<region>.amazonaws.com**

ストリーム名を指定した場合は、従来通りサービス共通のエンドポイントが使われる

- **kinesis.<region>.amazonaws.com**

エンドポイントの確認

実際にアカウントエンドポイントが使用されているかは、デバッグログから確認可能

```
$ aws kinesis describe-stream-summary --stream-arn arn:aws:kinesis:ap-northeast-1:123456789012:stream/ExampleInputStream --debug
(...)
2023-01-12 12:48:25,178 - MainThread - urllib3.connectionpool - DEBUG - Starting new HTTPS connection (1): 123456789012.control-kinesis.ap-northeast-1.amazonaws.com:443
2023-01-12 12:48:25,325 - MainThread - urllib3.connectionpool - DEBUG - https://123456789012.control-kinesis.ap-northeast-1.amazonaws.com:443 "POST / HTTP/1.1" 200 396
(...)

$ aws kinesis get-shard-iterator --stream-arn arn:aws:kinesis:ap-northeast-1:123456789012:stream/ExampleInputStream --shard-id shardId-000000000006  --shard-iterator-type LATEST --debug
(...)
2023-01-12 12:52:18,026 - MainThread - urllib3.connectionpool - DEBUG - Starting new HTTPS connection (1): 123456789012.data-kinesis.ap-northeast-1.amazonaws.com:443
2023-01-12 12:52:18,144 - MainThread - urllib3.connectionpool - DEBUG - https://123456789012.data-kinesis.ap-northeast-1.amazonaws.com:443 "POST / HTTP/1.1" 200 268
(...)
```

プロデューサー

プロデューサー

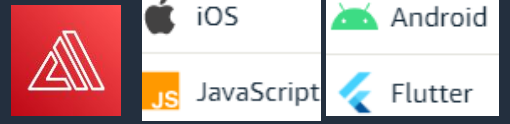
- Kinesis Data Streams にレコードを送信するコンポーネント
- 用途に応じて様々な形態で提供されるアプリケーションやフレームワークを使うことで、Kinesis Data Streams の **API** を意識することなくレコードを送信可能

エージェント, ライブラリ

AWS SDK

Kinesis Producer Library

AWS Amplify Libraries



Kinesis Agent

AWS サービス(一部)

AWS IoT

Amazon CloudWatch Logs

Amazon EventBridge

Amazon DynamoDB

Amazon API Gateway

サードパーティーツール

LOG4J

Flume

Fluentd

Fluent Bit

データレコード送信用の API

PutRecord API

- 単一の Kinesis データレコードを送信
- 各シャードにつき 1 秒あたり 1000 レコードまたは 1 MiB の書き込みをサポート
- SequenceNumberForOrdering オプションによるシャード内の厳密な順序保証をサポート

PutRecords API

- 複数の Kinesis データレコードをまとめて送信
- 最大で 500 データレコード、または 5 MiB まで一括送信可能。
  パーティションキーなども 5 MiB のペイロードサイズに含まれる
- PutRecords 自体の制限に加えて、シャードごとの制限が別途適用される
  (1 秒あたり 1000 レコードまたは 1 MiB の書き込み)

SequenceNumberForOrdering オプション

順序保証のためのオプション

- PutRecord API でのみ利用可能
- 前回の PutRecord 実行時に返却された SequenceNumber を用いることで、シーケンス番号の確実な増加を保証する
- あるパーティションに書き込みを行うクライアントが単一である際に、厳密な順序保証を行うケースで利用できる
- 同一パーティションに対して複数のクライアントが同時にデータを書き込むような場合、シーケンス番号の増加は保証されない
- 複数パーティションにまたがった順序保証はできない
- PutRecord を使うことになるため、スループットは PutRecords を使用する場合より低くなる。PutRecord ではスループット的に厳しい場合は、コンシューマーの方でユーザーレコード内の値を元にした順序保証を検討する

プロデューサーアプリケーション選択方針

① AWS サービスの Kinesis 連携機能を活用する

Amazon API Gateway

AWS IoT

Amazon EventBridge

Amazon Pinpoint

AWS WAF

その他多数

② アプリケーションログ集約を行う場合は、実行環境に適したエージェントを使用する

各種ログファイル

Kinesis Agent

Fluentd

コンテナアプリケーションが出力するログメッセージ

Fluent Bit

AWS サービスが出力するログメッセージ

Amazon CloudWatch Logs

③アプリケーションから直接メッセージを送信する場合は、ライブラリを活用する

モバイルアプリケーション, SPA

AWS Amplify Libraries

iOS Android JavaScript Flutter

サーバーサイドアプリケーション

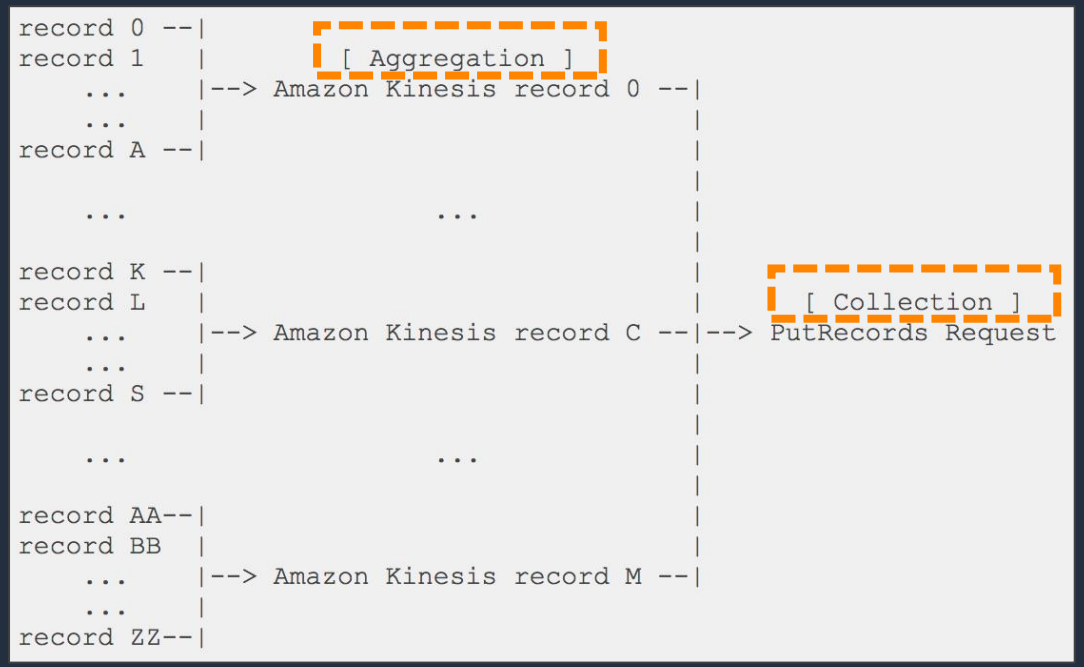
AWS SDK

Kinesis Producer Library

アプリケーションイベントの送信

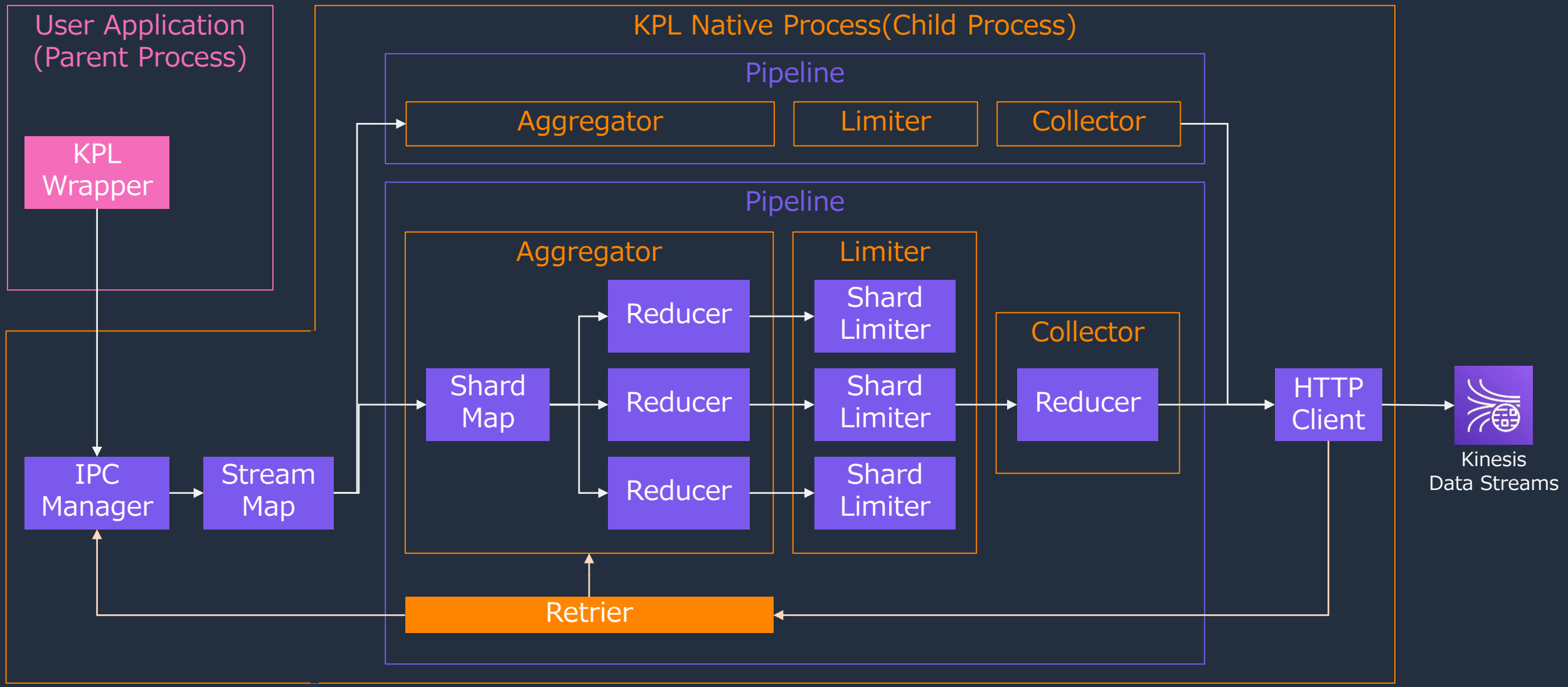
Amazon Kinesis Producer Library (KPL)

- プロデューサーアプリケーション実装用のライブラリ。C++ で開発されており、Java アプリケーションに組み込むことでアプリケーションイベントをストリームに配信
- Amazon Kinesis Data Streams の特性に合った高い転送効率を実現する機能を提供
  - 複数件のデータを 1 データレコードに集約
  - 複数データレコードをバッファリングして送信
- 各種設定をプロパティベース調整可能
  - エラー時のリトライ挙動
  - タイムアウト時間の調整
  - データ送信時に利用するコネクション数
- メトリクスを Amazon CloudWatch に自動送信



https://github.com/awslabs/amazon-kinesis-producer

KPL > アーキテクチャー



KPL > 集約とバッファリング

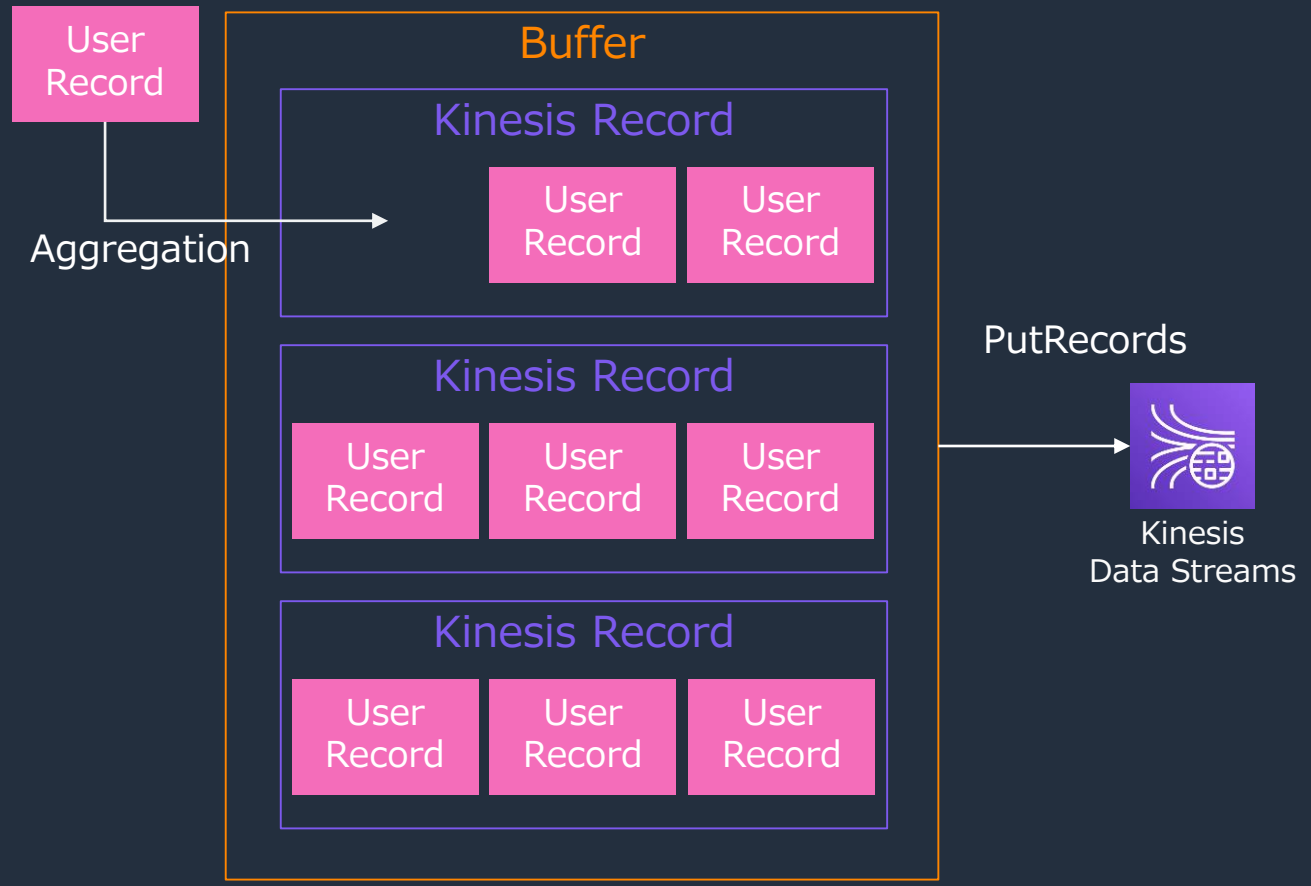
集約とバッファリングを活用することで、
転送効率のアップ、 **PUT ペイロード**
ユニット料金の削減が可能

集約(Aggregation)

複数のユーザーレコードを単一の
Kinesis レコードに結合

バッファリング(Buffering)

複数の Kinesis Data Streams
レコードを PutRecords API で
一括送信



KPL > 集約のチューニング

- Kinesis レコードに含められるユーザーレコードは、レコード数・サイズで調整可能
  - レコード数は 1 レコードから 4294967295 レコードまで指定可能。
    デフォルトは最大値の 4294967295。AggregationMaxCount パラメーターで調整する
  - サイズは 64 バイトから 1048576 バイト (1 MiB) まで指定可能。
    デフォルトは 51200 バイト (50 KiB)。AggregationMaxSize パラメーターで調整する
- AggregationMaxSize を 1 MiB に変更するなど過剰な集約設定を初手で行うことは非推奨。シャード間のスループット使用率の偏り、コンシューマーによるリトライ処理への影響がありえるので、デフォルト設定から初めて少しずつチューニングすること
- PutRecords API の実行頻度をコントロールしたい場合は、集約設定のチューニングよりバッファリングの設定調整をおすすめ

https://github.com/awslabs/amazon-kinesis-producer/blob/master/java/amazon-kinesis-producer-sample/default_config.properties
https://aws.amazon.com/jp/blogs/big-data/implementing-efficient-and-reliable-producers-with-the-amazon-kinesis-producer-library/

KPL > バッファリング設定のチューニング

- バッファリングされたデータレコードが PutRecords へ送信されるタイミングは以下の通り
  - バッファ内の Kinesis データレコードが 500 レコードに達した場合 (CollectionMaxCount パラメータで調整可能)
  - バッファ内の総データ量が 5 MiB に達した場合。バッファサイズはCollectionMaxSize パラメーターで調整可能
  - 前回 PutRecords を実行してから 100 ミリ秒が経過した場合。この間隔は RecordMaxBufferedTime パラメーターで調整可能
- CollectionMaxCount および CollectionMaxSize をデフォルトのままとし、RecordMaxBufferedTime を増やすことで PutRecords API の呼び出し頻度を下げることができる。バッファサイズが大きくなるためメモリ使用率が増加する点は注意

https://github.com/awslabs/amazon-kinesis-producer/blob/master/java/amazon-kinesis-producer-sample/default_config.properties
https://aws.amazon.com/jp/blogs/big-data/implementing-efficient-and-reliable-producers-with-the-amazon-kinesis-producer-library/

KPL > リトライ処理

- KPL では、Kinesis Data Stream へデータ挿入が失敗した場合にリトライを行う仕組みを持っている。
  - スロットリングが発生した際にリトライを行うかどうかは FailIfThrottled パラメーターで切り替えが可能。デフォルトは false = スロットリングはリトライの対象となる
- 対象データレコードのリトライは、TTL で指定した期間内いっぱい行われる
  - TTL は 100 ミリ秒から 9223372036854775807 ミリ秒まで指定可能で、デフォルトは 30000 ミリ秒
- TTL を変更することでリトライ期限を延ばすことができるが、メモリ不足による OOM のリスクは上がる
  - データストリームのリソース不足でスロットリングが長期化し、バッファに滞留するデータが増加した場合など

https://github.com/awslabs/amazon-kinesis-producer#back-pressure

KPL > 必要なメモリサイズについて

- ユーザーレコードのサイズ、1 秒あたりのユーザーレコード数、バッファ時間、TTL からバッファサイズをある程度推測することが可能
- バッファ領域はあくまで割り当てメモリの一部であるため、実際に必要なメモリサイズについては検証で確認すること

前提

- ユーザーレコードのサイズ: 50 KiB、1 秒あたりのユーザーレコード数: 1024
- バッファ時間: 1 秒、TTL: 1 時間(3600 秒)

想定バッファサイズ

- 通常ケースの場合: 50 KiB/record * 1024 records * 1 sec = 50 MiB
- ワーストケース: 50 KiB/record * 1024 records * 3600 sec ≒ 17.57 GiB

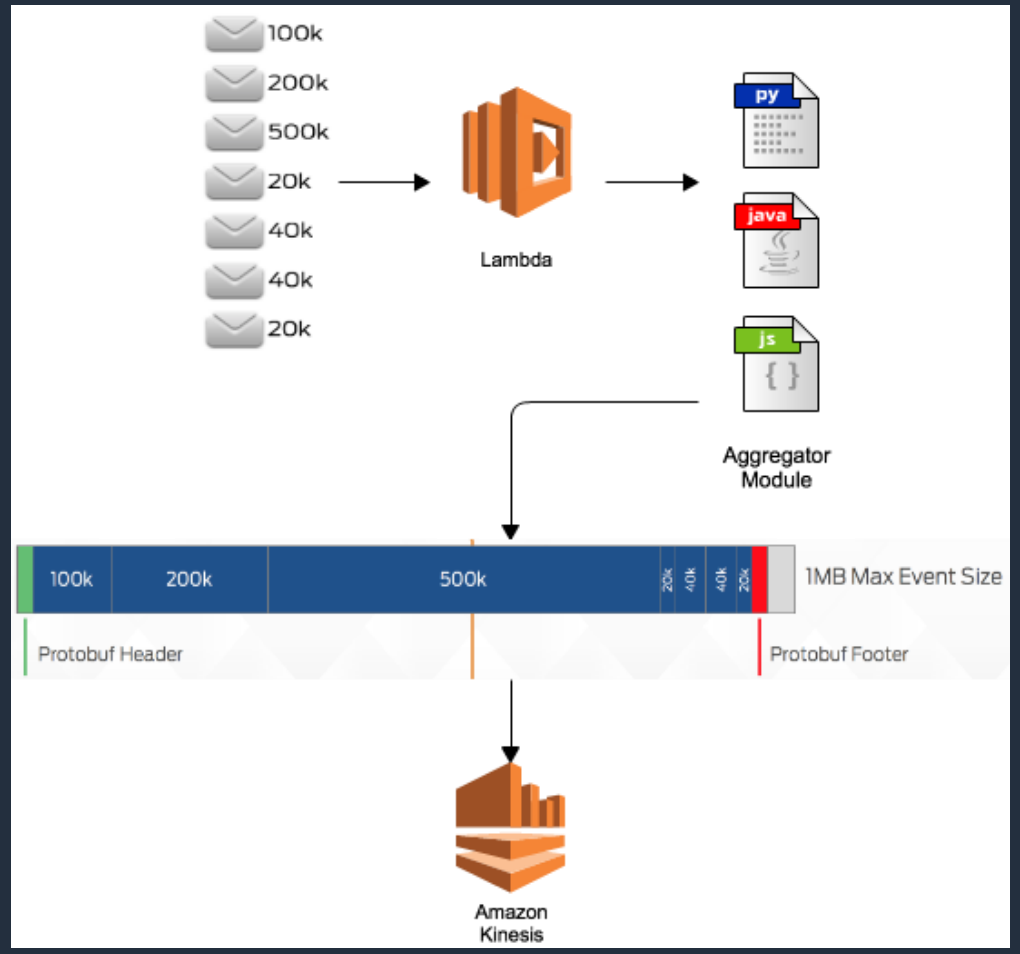
KPL > Tips: TTL によるレコード有効期限切れ時の対応

- 現状、FutureCallBack 内の onFailure にはユーザーレコードが渡されない。
- TTL によってユーザーレコードの失効を防ぐためには、自前で FutureCallBack 関数を拡張し、ユーザーレコードも CallBack に渡すような形での対処が必要

関連 Issue: https://github.com/awslabs/amazon-kinesis-producer/issues/76

Kinesis Record Aggregation & Deaggregation Modules for AWS Lambda

- メモリフットプリントが少ない Lambda 向けのライブラリ
- ユーザーレコードの集約、集約解除機能を提供
- Java、JavaScript、Python で利用可能



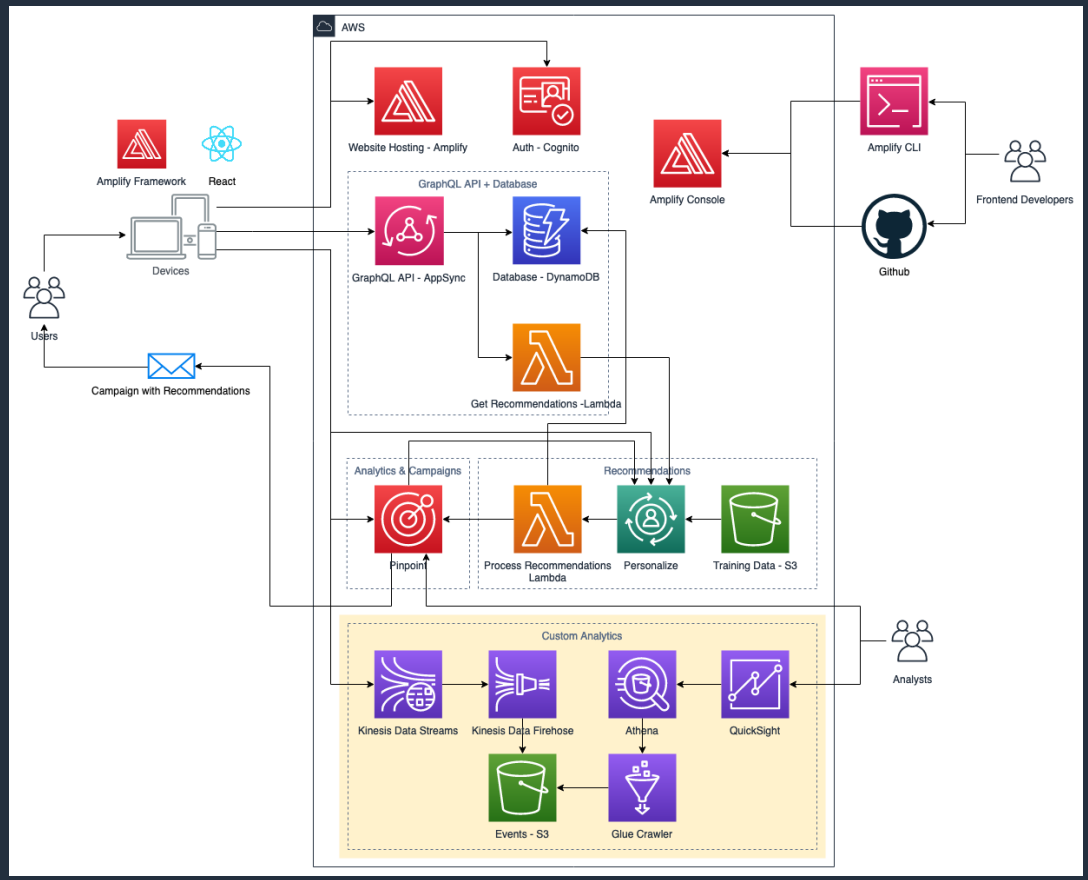
https://github.com/awslabs/kinesis-aggregation

AWS Amplify / AWS Mobile SDK

- Amplify の Analytics 機能では、
アプリケーションのイベントを
Kinesis Data Streams に送信可能

- 連携したデータを他の AWS サービスで
分析することで ユーザーの行動分析や
レコメンデーション実装を補助

- 利用可能な言語に制約がある点に注意。
Amplify 公式サイトで最新情報を確認すること

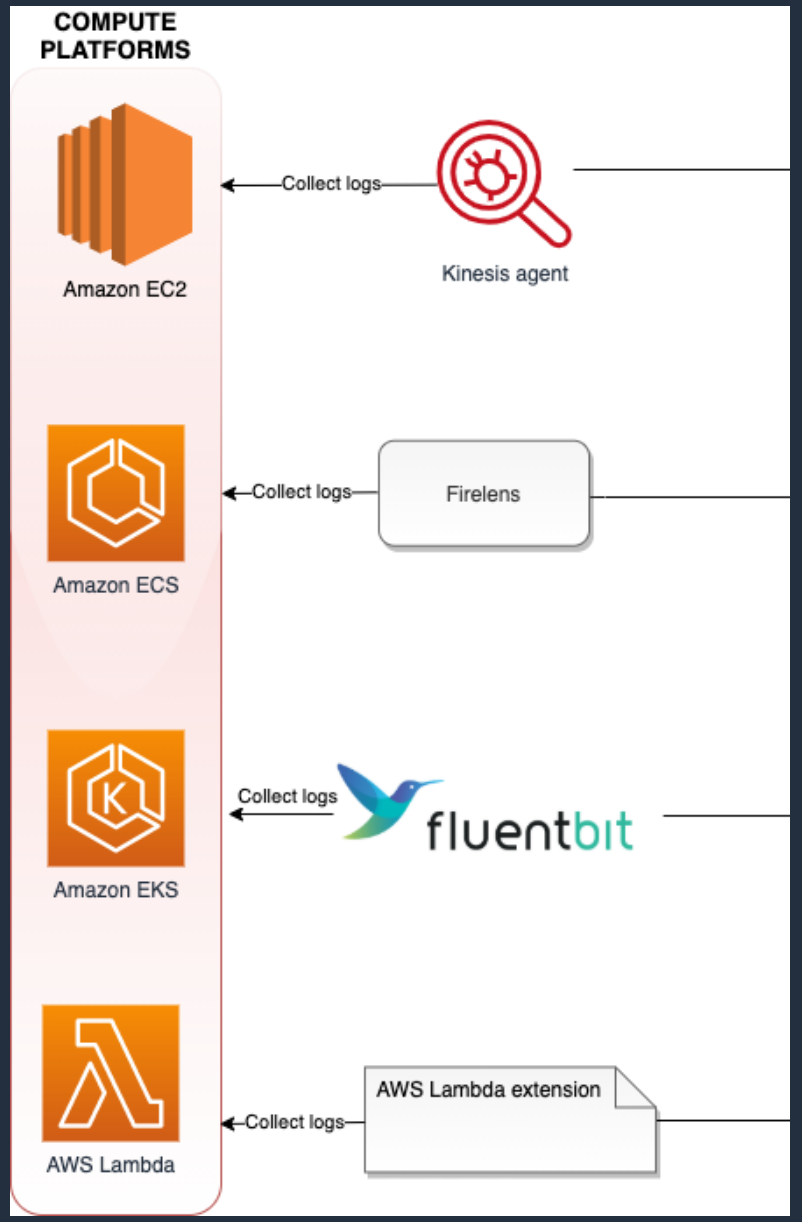


https://docs.amplify.aws/lib/analytics/getting-started/q/platform/js/

ログイベントの送信

ログ送信の概要

- アプリケーション実行環境によって最適なエージェントは異なる
- エージェントをインストール可能なEC2 等の環境では Kinesis Agent などのログエージェントを利用可能
- ECS や EKS などコンテナ上でアプリケーションを実行する場合は、FluentBit や Firelens を活用したサイドカーコンテナによるログ配信を利用可能
- AWS Lambda でアプリケーションを実行する場合は、CloudWatch Logs 経由の連携に加えて、Lambda Extensions によるログの直接送信も可能



https://aws.amazon.com/jp/blogs/big-data/unify-log-aggregation-and-analytics-across-compute-platforms/

Amazon Kinesis Agent for Linux

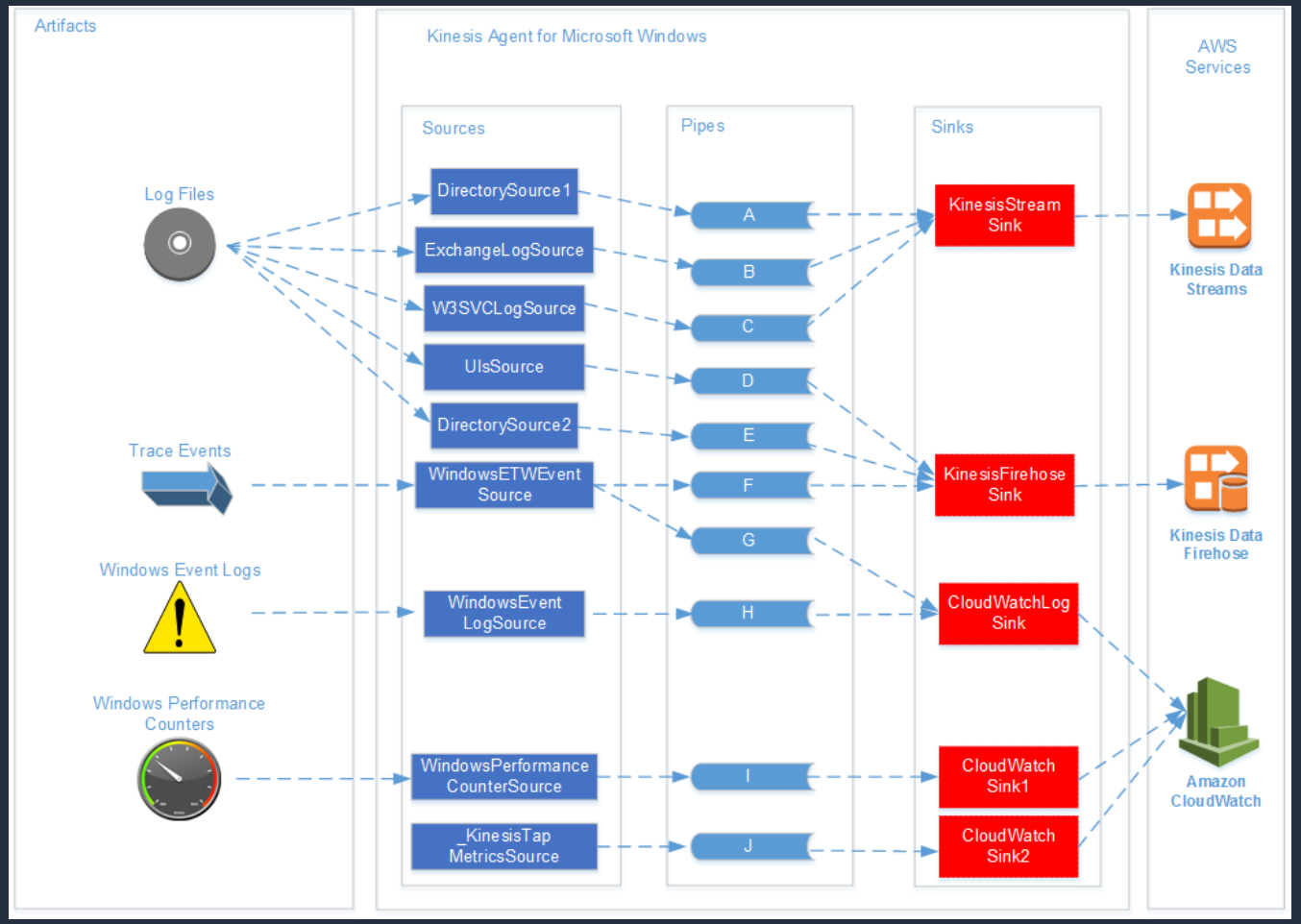
- スタンドアロン Java アプリケーションとして動作するログエージェント
- Amazon Kinesis Data Streams、Amazon Kinesis Data Firehose を宛先としてサポート
- パスで指定したファイルを検出し、差分を順次転送する。ファイルローテートにも対応
- 送信前のバッファリング、失敗時の再試行をサポート。またデータのフォーマット変換や、ログパースなどの前処理機能を提供
- エージェントのメトリクスデータを Amazon CloudWatch へ送信

```
# /etc/aws-kinesis/agent.json
{
    "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"
        }
    ]
}
```

https://github.com/awslabs/amazon-kinesis-agent

Amazon Kinesis Agent for Microsoft Windows

- Windows 向けの Kinesis Agent
- ログファイルに加えて Windows イベントログも収集可能
- データの簡易的な ETL をサポート
- Kinesis Data Streams、Kinesis Data Firehose を宛先としてサポート
- CloudWatch へのメトリクスの送信、CloudWatch Logs へのログ送信機能も提供



https://github.com/awslabs/kinesis-agent-windows

Fluent plugin for Amazon Kinesis v3

- Fluentd はオープンソースのログエージェント
- awslabs では、Fluentd から Kinesis にメッセージを直接送信するためのプラグインを提供している。3 つの output をサポートしている

Kinesis Data Streams output

1. kinesis_streams
2. kinesis_streams_aggregated

Kinesis Data Firehose output

3. kinesis_firehose

```
# Kinesis Data Streams 用の設定例
<match your_tag>
  @type kinesis_streams
  region us-east-1
  stream_name your_stream
  partition_key key
</match>
```

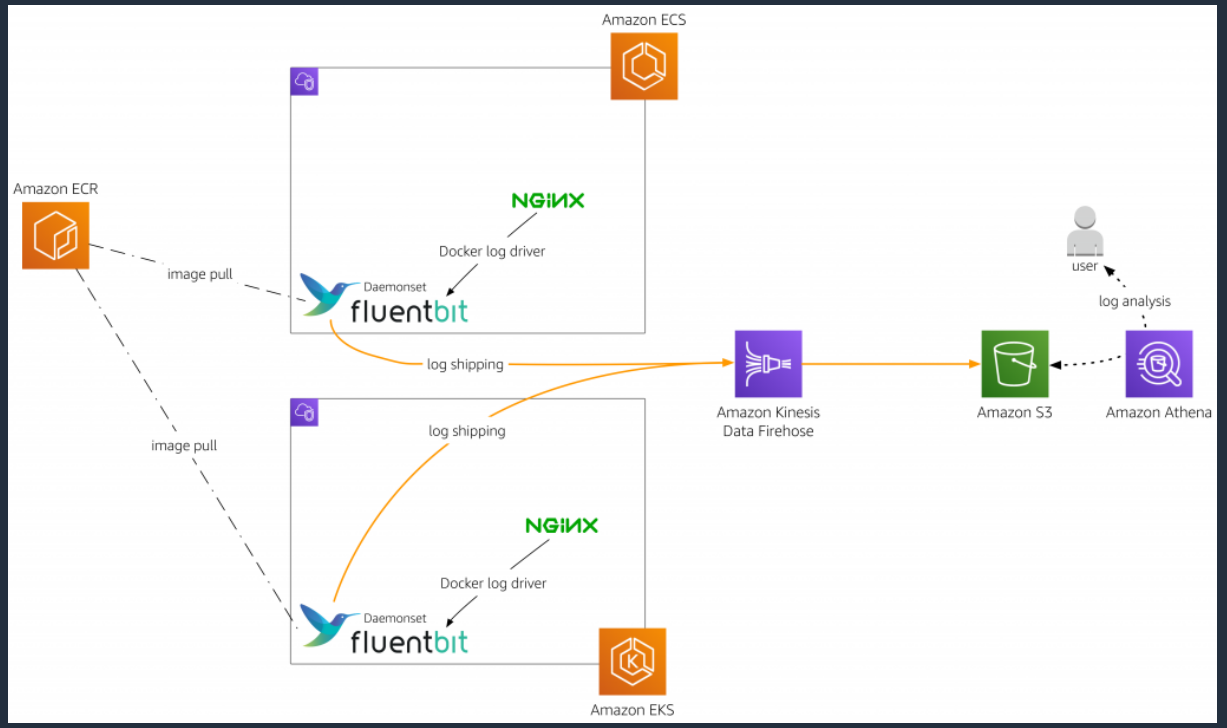
```
# Kinesis Data Firehose 用の設定例

<match your_tag>
    @type kinesis_firehose
    region us-east-1
    delivery_stream_name your_stream
</match>
```

https://github.com/awslabs/aws-fluent-plugin-kinesis

AWS for Fluent Bit

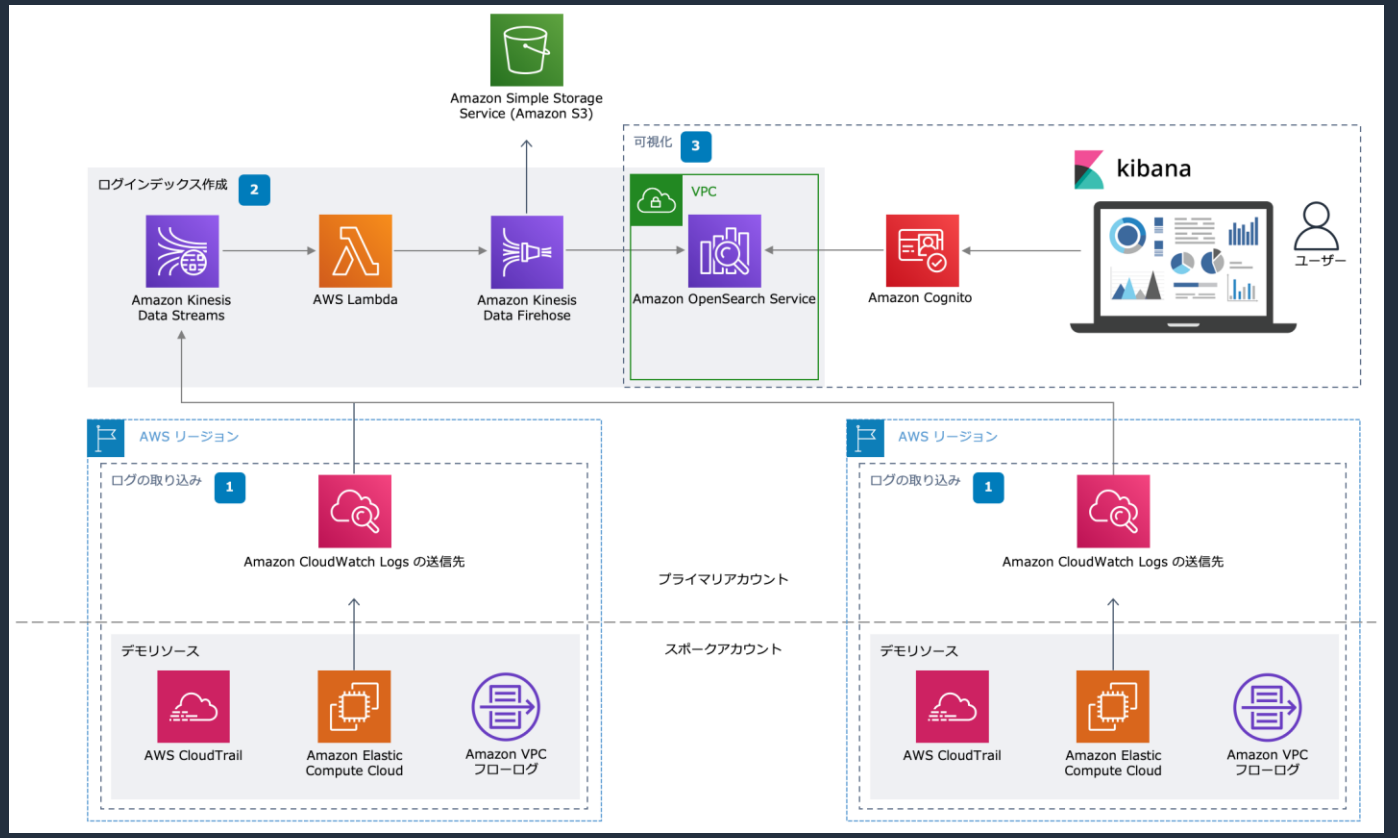
- OSS のログエージェント
- Fluentd より軽量に動作する特徴から、
  コンテナサービスのログ収集に活用される
- サーバーにログエージェントとして
  インストールして使用することも可能
- ECS、EKS に対応。Fargate 上でも実行可能
- Kinesis Data Streams、
  Kinesis Data Firehose に加えて、
  CloudWatch や OpenSearch Service も
  宛先としてサポートしている



https://github.com/aws/aws-for-fluent-bit

CloudWatch Logs

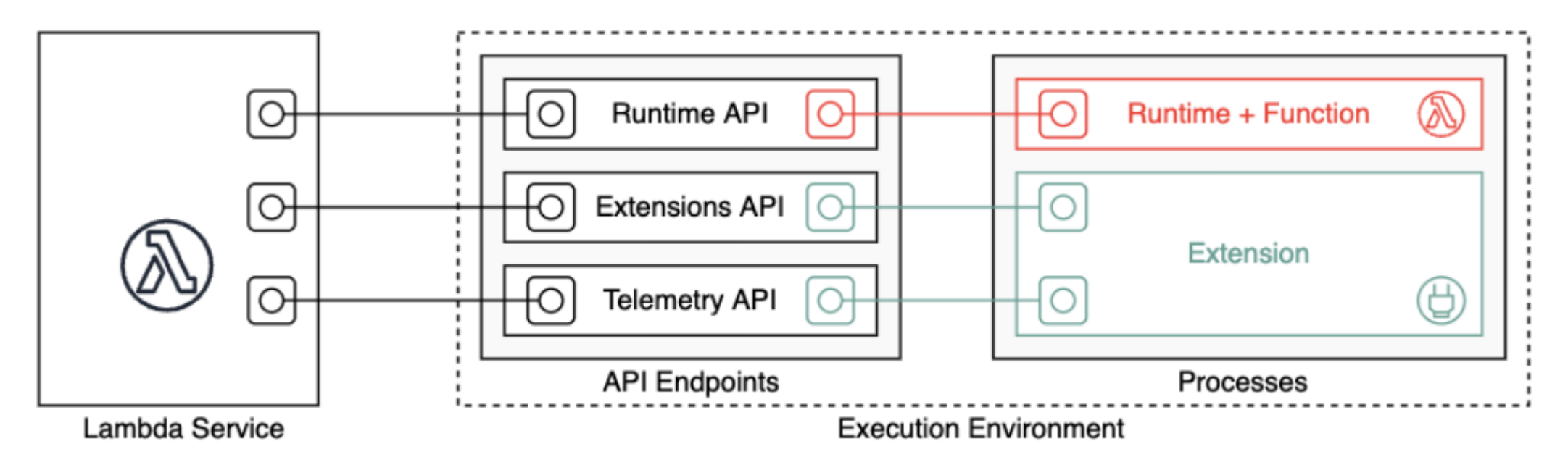
- AWS が提供する
  フルマネージドログ管理サービス
- 多数の AWS サービスが
  CloudWatch Logs を
  サポートしている
- ユーザーは簡単な設定を行うだけで、
  AWS Lambda の アプリケーション
  ログや Amazon VPC Flow Logs
  といったサービスログを
  CloudWatch Logs へ記録可能
- サブスクリプションフィルターと
  呼ばれる機能を使うことで、
  Kinesis Data Streams 等の
  サービスに対して、収集したログを
  ニアリアルタイムで配信可能



https://aws.amazon.com/jp/solutions/implementations/centralized-logging/

Lambda Extension

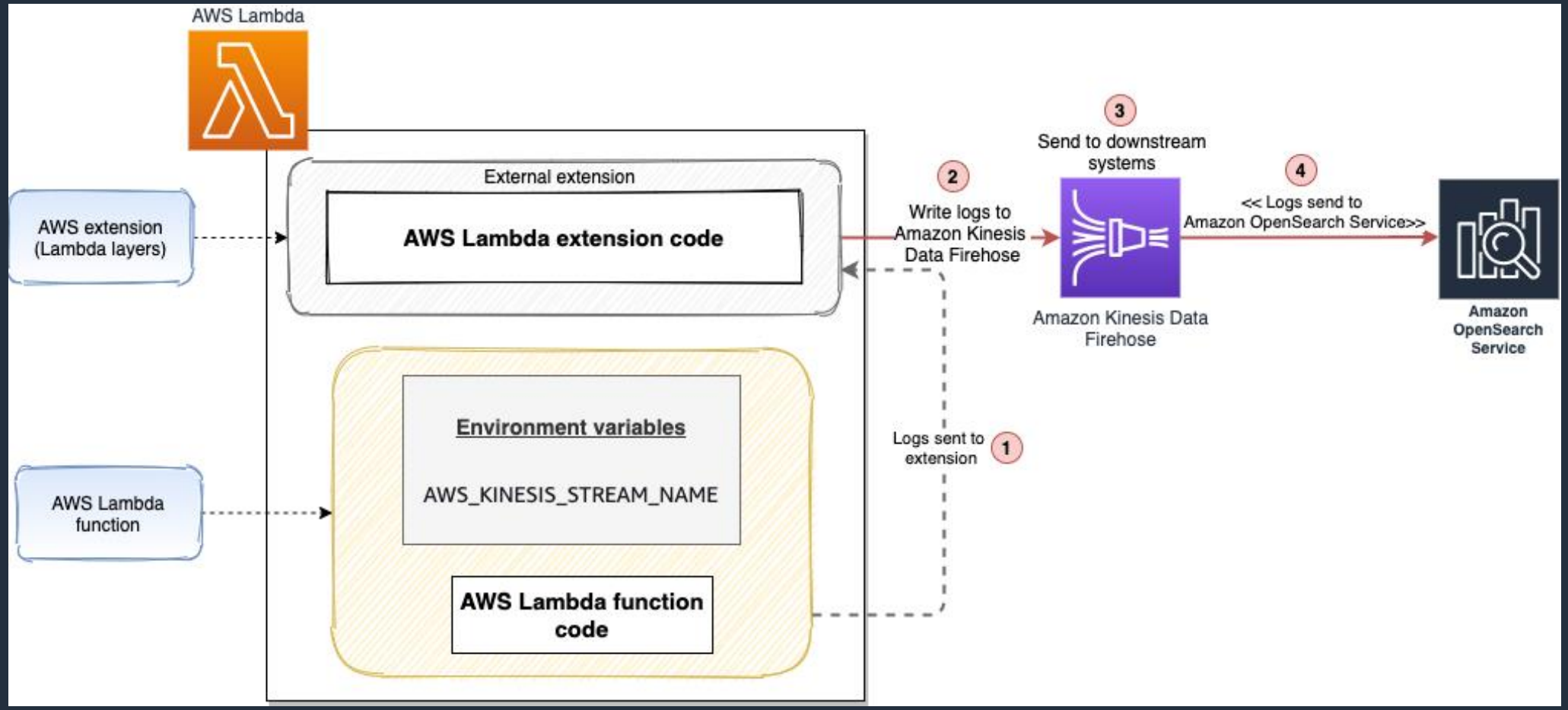
- AWS Lambda の拡張機能
- Extensions から Telemetry API (旧 Logs API) を介してログやテレメトリデータを取得し、外部サービスに連携可能



https://aws.amazon.com/jp/blogs/compute/introducing-the-aws-lambda-telemetry-api/

Lambda Extension から Kinesis へのログ送信例

Kinesis 経由でログを連携することが目的の場合、Extensions を使用することで
CloudWatch Logs を介さずに直接 Kinesis へデータを送信可能。コスト削減が期待できる。

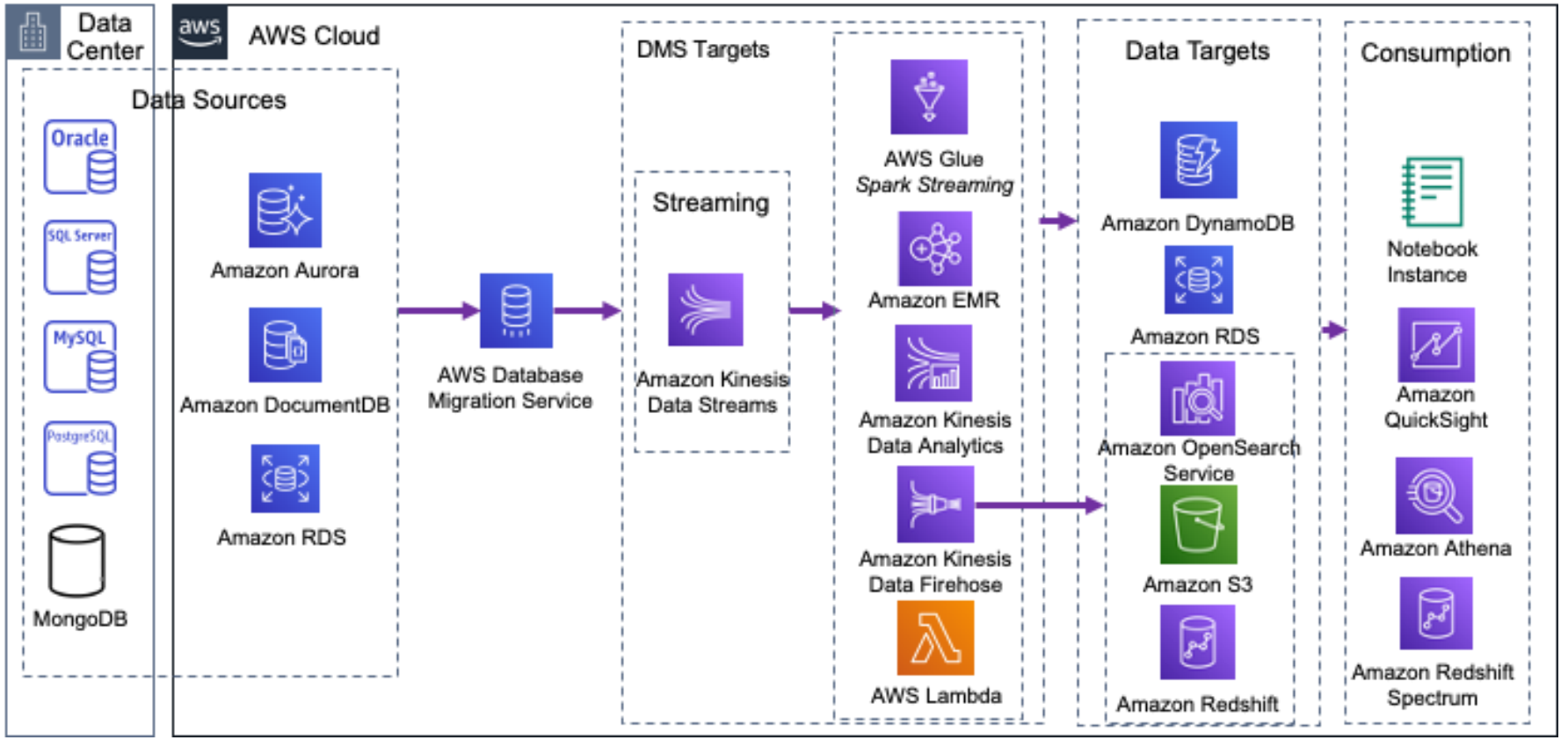


https://github.com/aws-samples/aws-lambda-extensions/tree/main/kinesis-stream-logs-extension-demo

データベース変更情報のキャプチャと配信

AWS DMS + Kinesis による変更情報の取得・処理

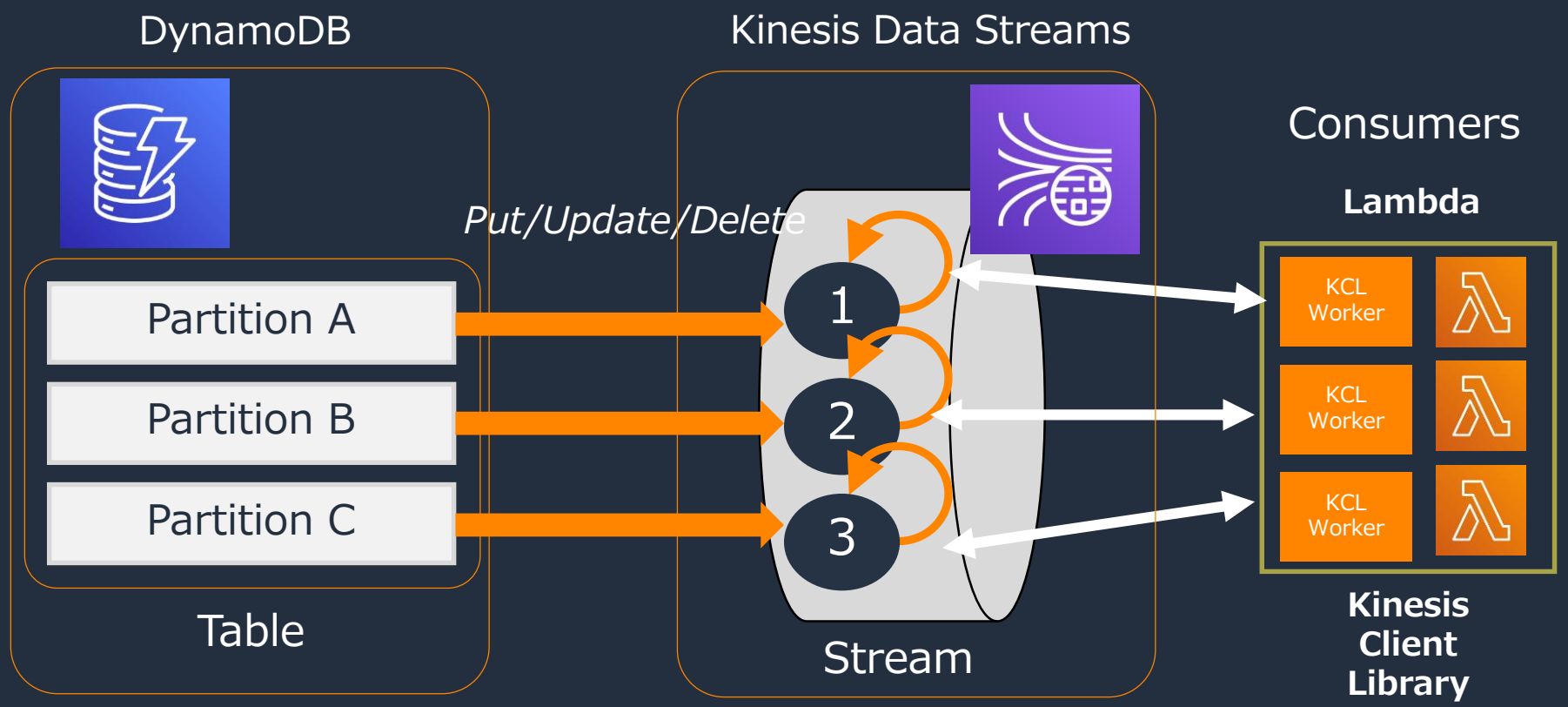
Kinesis Data Streams に DMS から変更データを配信することで、
複数データストアへのリアルタイムなデータ配信、リアルタイム処理を行うことができる



https://aws.amazon.com/jp/blogs/big-data/stream-change-data-to-amazon-kinesis-data-streams-with-aws-dms/

DynamoDB + Kinesis Data Streams

- テーブル変更情報を Kinesis Data Streams に送信
- 要求に応じた DynamoDB Streams との使い分けが可能



DynamoDB Streams との使い分け

- 24時間以上のデータ保持, AWS サービスとのより柔軟な連携、複雑な処理が必要な場合は Kinesis Data Streams を検討
- Exactly Once やデータの並び順などが重要な場合は DynamoDB Streams を検討

プロパティ	Kinesis Data Streams	DynamoDB Streams
データ保持期間	24時間 から 1 年	24 時間
コンシューマーの数	最大 5 (標準コンシューマー)、最大20(拡張ファンアウト)	最大 2 つまで
レコードの順序	Consumer の方で, レコードタイムスタンプから順序を識別	ストリーム内のレコードは, 実際に発生した変更と同じ順序で出力
レコード重複の有無	重複は起こりうる (At Least Once)	重複は発生しない (Exactly Once)
拡張性	別途でスケールする必要あり	テーブルに追従して自動的にスケール
Consumer	- Kinesis Data Firehose - Kinesis Data Analytics(SQL, Flink) - Kinesis Client Library - AWS Glue Streaming ETL Jobs - AWS Lambda	- DynamoDB Streams Kinesis Adapter - Kinesis Data Analytics for Flink - AWS Lambda
料金(東京リージョン)	- DDB Streams 変更データキャプチャ (0.1142 USD/ 1,000,000 CDC) - 上記に加え, Kinesis Data Streams の標準料金	- DymamoDB Streams 読み込みリクエスト 0.0228 USD/100,000 リクエスト (最初の 2,500,000 リクエストは無料) - Lambda によって行われる読み込みリクエストについては料金がかからない
監査	- CloudTrail のデータイベントは未サポート	- CloudTrail のデータイベント対応 (GetRecords, GetShardIterator)

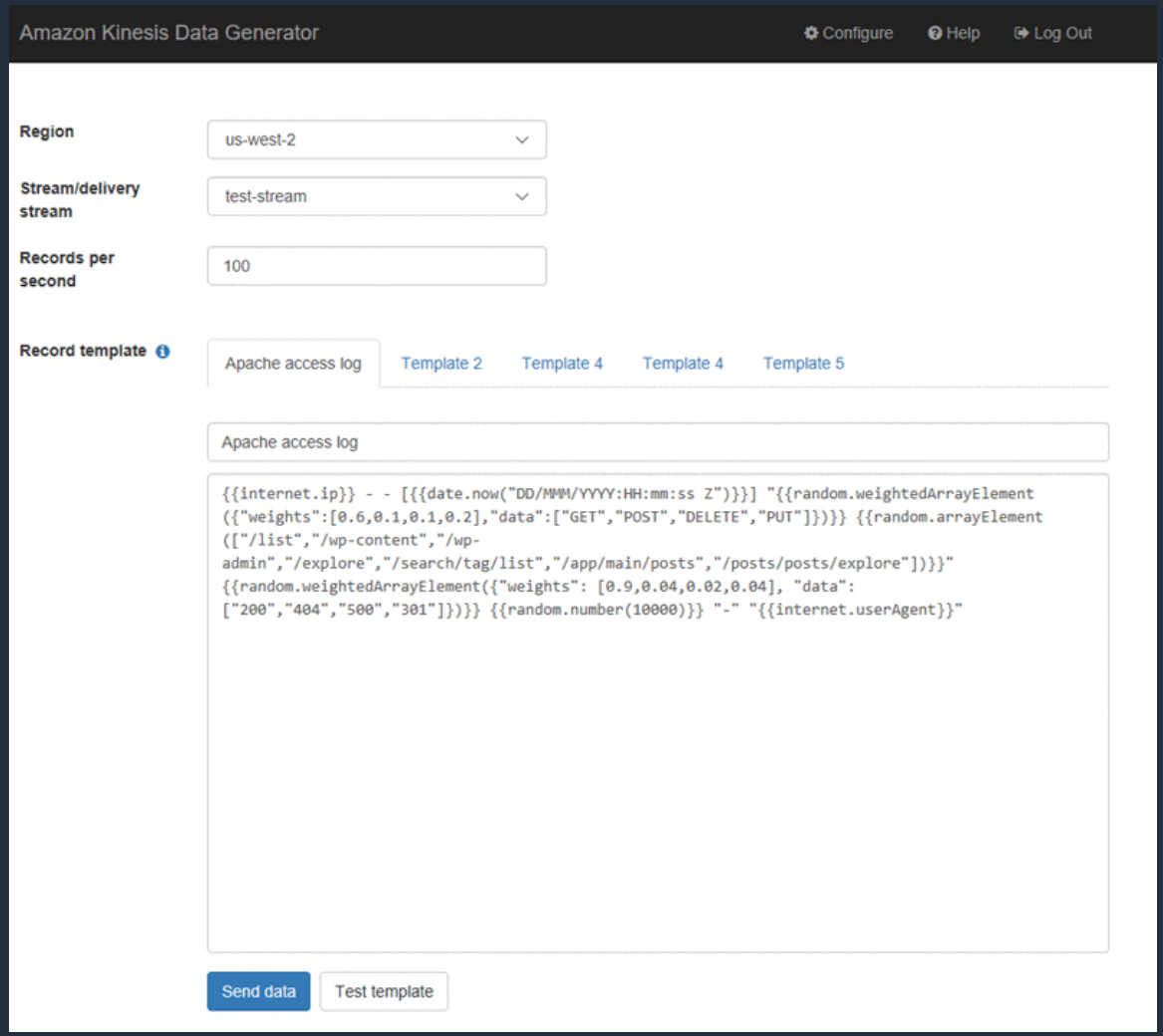
テストデータの送信

Amazon Kinesis Data Generator (KDG)

- HTML と JavaScript で実装された
  ダミーデータ生成・送信ツール

- Amazon Kinesis Data Streams または
  Amazon Kinesis Data Firehose に
  テストデータを簡単に送信できる

- CloudFormation テンプレートから素早く
  環境を構築し利用可能

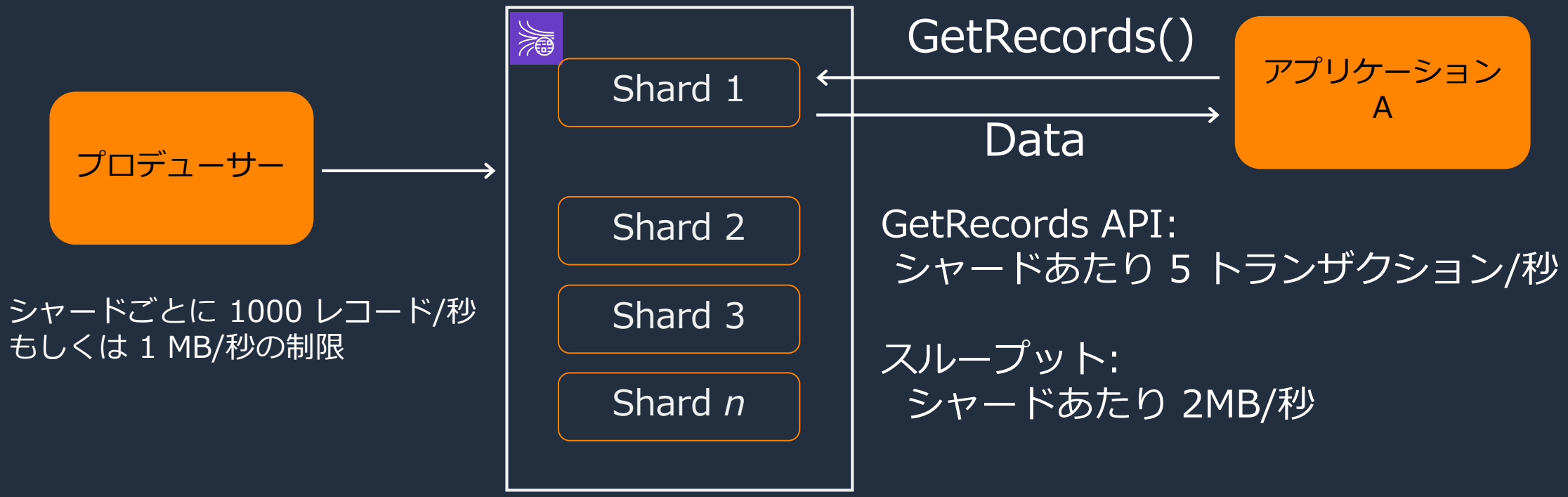


https://github.com/awslabs/amazon-kinesis-data-generator

コンシューマー

標準コンシューマー

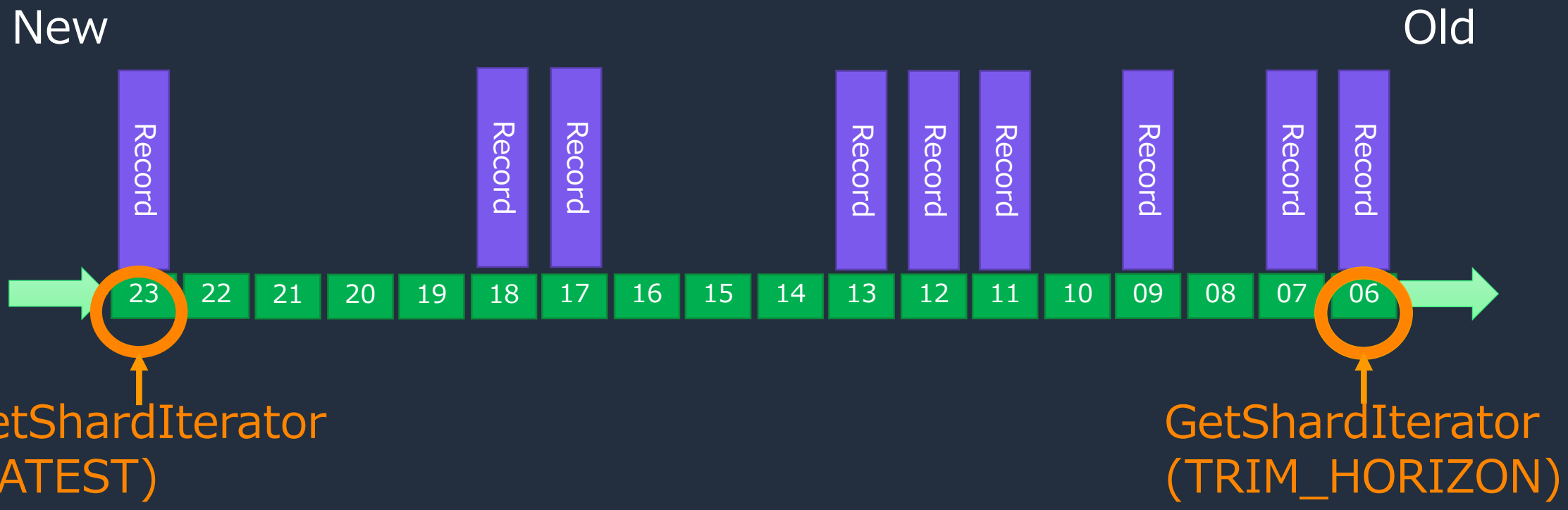
- GetRecords API によりレコードを取得。1 秒ごとに 5 トランザクションまで発行可能
- コンシューマーアプリケーションが 1 つだけの場合、
  データレコードは 1 秒 / 5 = 200 ミリ秒ごとに取得可能



標準コンシューマーによるデータ取得

取得開始位置を特定してから, レコードを取得していく

1. シャード内のレコード取得開始位置(ShardIterator)を取得
2. 取得したシャード内の位置(ShardIterator)を指定し、レコードを取得



標準コンシューマーによるデータ取得

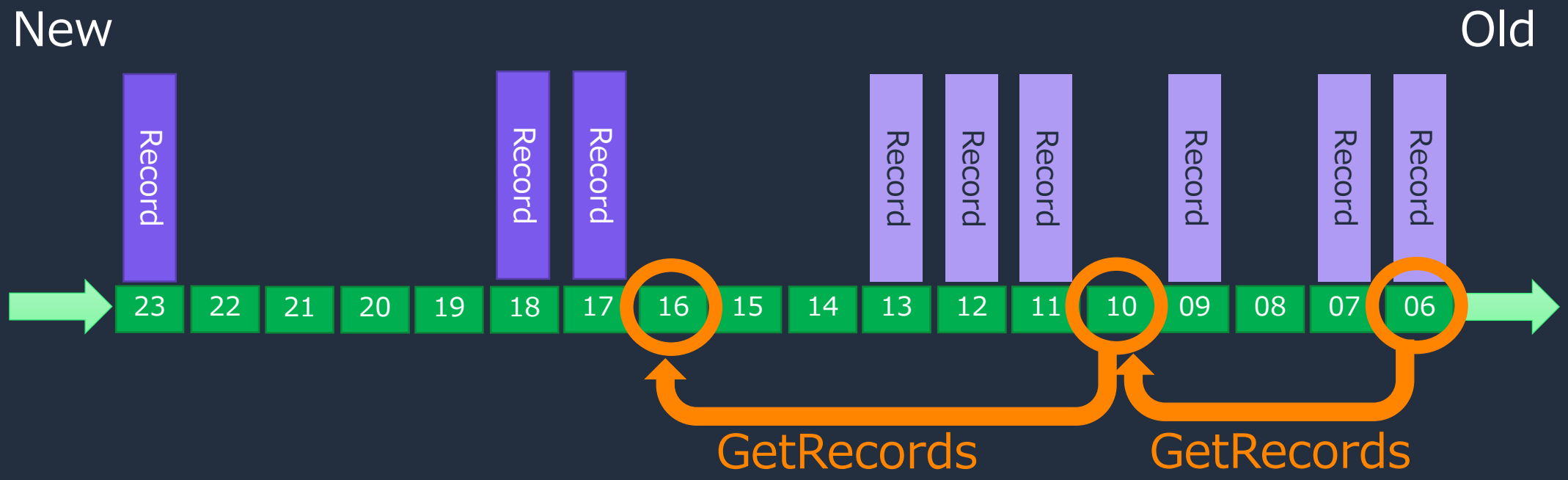
取得開始位置を特定してから, レコードを取得していく

1. シャード内のレコード取得開始位置(ShardIterator)を取得
2. 取得したシャード内の位置(ShardIterator)を指定し、レコードを取得

標準コンシューマーによるデータ取得

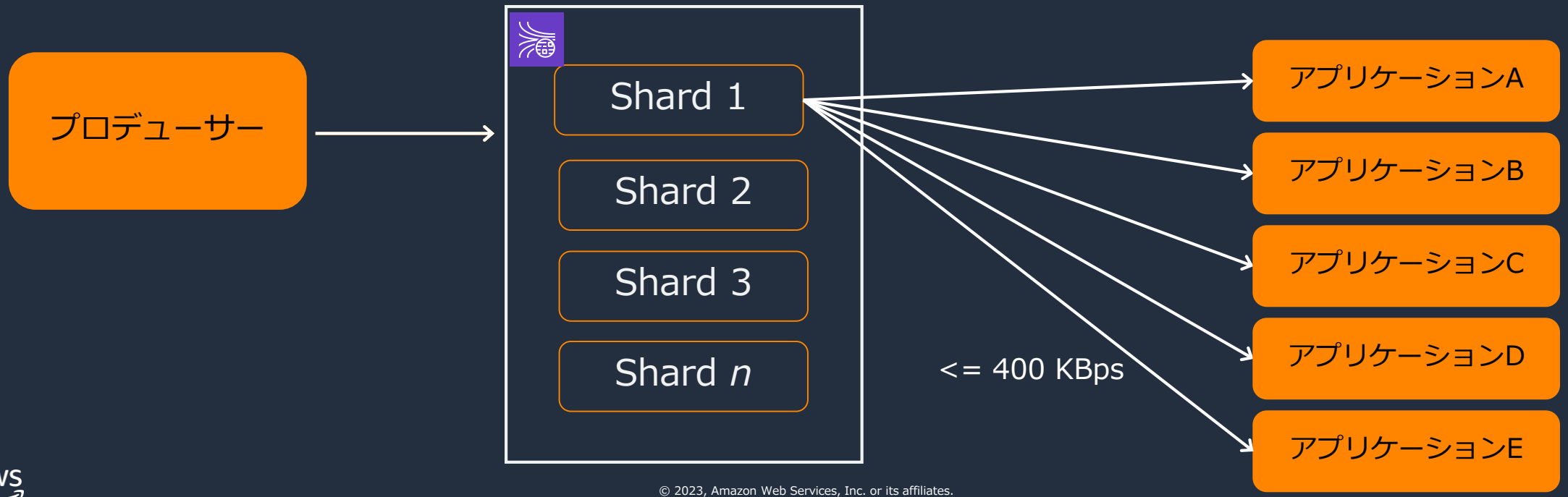
取得開始位置を特定してから, レコードを取得していく

1. シャード内のレコード取得開始位置(ShardIterator)を取得
2. 取得したシャード内の位置(ShardIterator)を指定し、レコードを取得



標準コンシューマーの課題

- コンシューマーアプリケーションの増加がレイテンシの増大を招く
- コンシューマーアプリケーションが 5 つ存在する場合、各アプリケーションがシャードからデータを取得可能な頻度は、5 回/秒を 5 等分した 1 回/秒となる。
秒間スループットも2 MiB/s を 5 等分した 400 KiB/s 以下となる
- 遅延解消のためには、シャード数を増やし 1 シャードあたりのデータ流量を下げるなどの対応が必要となる

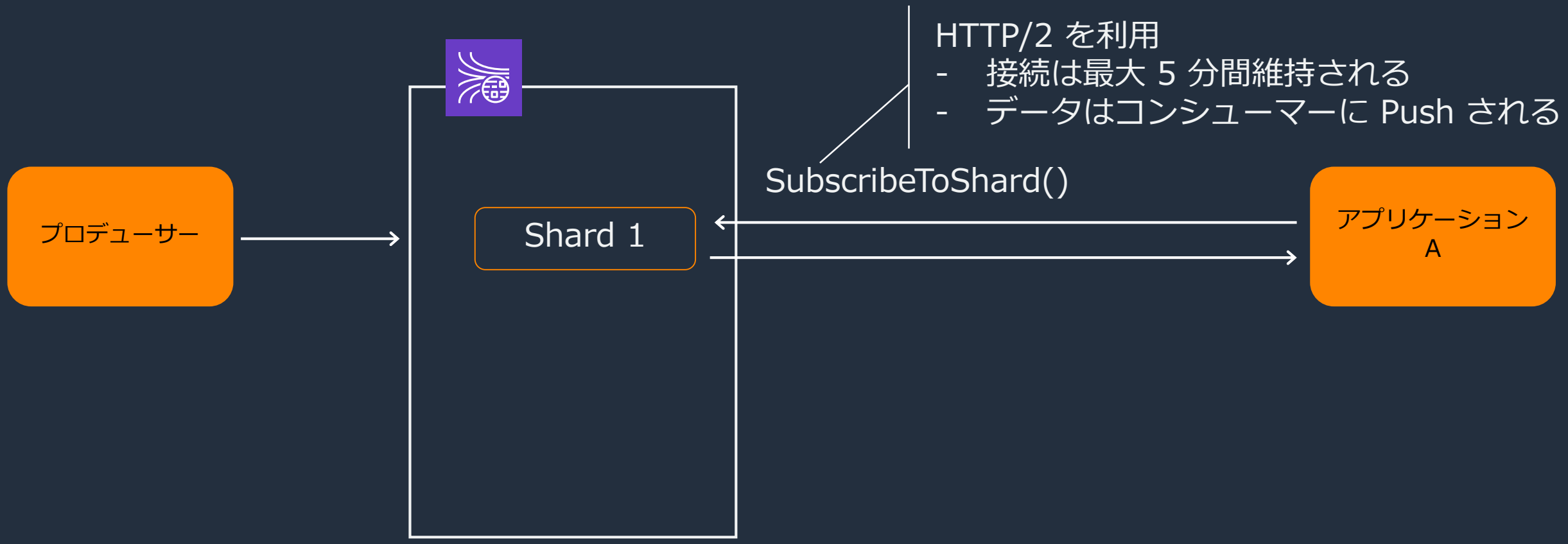


GetRecords API の制限

- 複数のシャードにまたがってデータを取得することはできない
  - API 呼び出し時に ShardIterator = シャードと取得開始位置を指定する必要あり
- シャードにつき 1 秒間に 5 回までの呼び出し制限あり
- 1 回の API 呼び出しにつき 10000 レコードまたは 10 MB までのデータを取得可能
- 2 MB を超えるデータを 1 回の API 呼び出しで取得した場合、以降の API 呼び出しがしばらくスロットルされる
  - シャードに割り当てられている読み取り帯域は 2 MB/s
  - 仮に 10 MB のデータを 1 回の API 呼び出しで取得した場合、その後 5 秒間は対象シャードに対する GetRecords がスロットルされうる

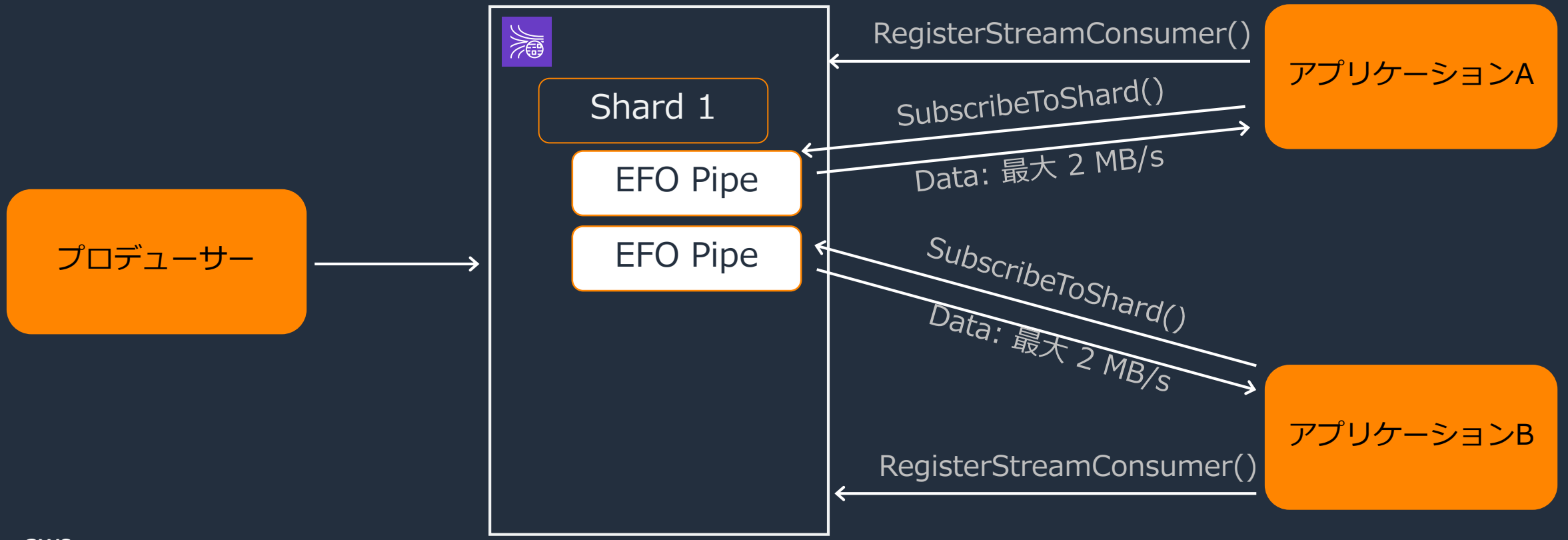
拡張ファンアウトコンシューマー

コンシューマーアプリケーションは、シャードに対して Subscribe を行い、
データを継続的に受け取る



拡張ファンアウトコンシューマーの特徴

- コンシューマーごとに専用の帯域が割り当てられるため、処理の競合による スロットルのリスクがない
- HTTP/2 プロトコルによる低レイテンシなデータ配信



コンシューマーの使い分け

標準コンシューマーを使うケース

- コンシューマーアプリケーションが 2 つに収まる場合。
  - シャードごとの帯域上限は出力が 2MB/s、入力が 1MB/s と、出力帯域は入力帯域の倍
  - コンシューマーアプリケーションが 2 つまでであれば、帯域不足は起こりにくい
  - API の呼び出し回数は 2 分割される点には注意が必要
- コストを最小化したい場合。拡張ファンアウトは追加コストが発生するため

拡張ファンアウトを検討するケース

- 3 つ以上のコンシューマーを使用する場合
- 低レイテンシの要件に対応したい場合。標準コンシューマーの一般的なレイテンシは 200ms、拡張ファンアウトは 70ms

コンシューマーアプリケーション

マネージドサービスやフレームワークを使用することで、低レイヤーの API を意識せずに
コンシューマーアプリケーションの開発、運用が可能

Kinesis

Amazon
Kinesis Data
Analytics

Flink

beam

SQL

Kinesis Client Library
+
Connector Library

Kinesis Data Firehose

AWS Services

AWS Lambda

Amazon EMR

AWS Glue

3rd party

Spark

Apache Spark

APACHE STORM

DATADOG

mongoDB.

Anodot

databricks

Qubole

VOLTDB

MEMSQL

splunk>

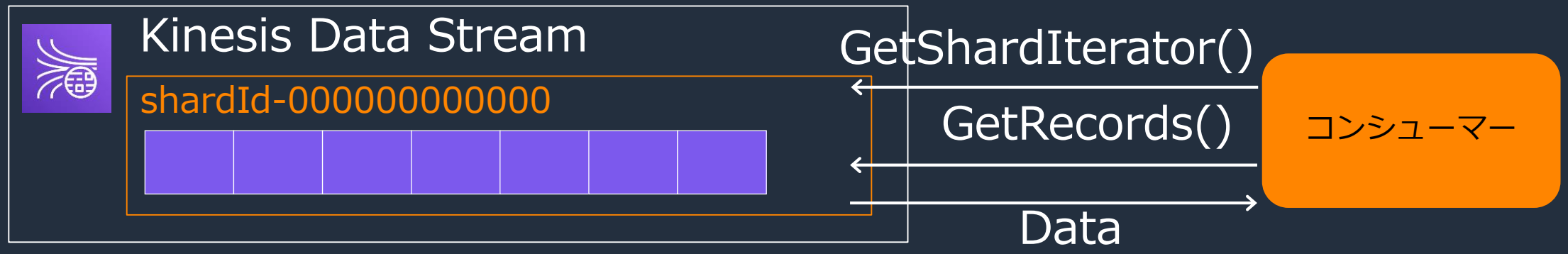
コンシューマーライブラリによるデータの取得

コンシューマーアプリケーション実装上の課題

実装コスト

コンシューマーアプリケーションの実装は煩雑。
自前で実装する場合、以下の処理を組み込む必要が有る。

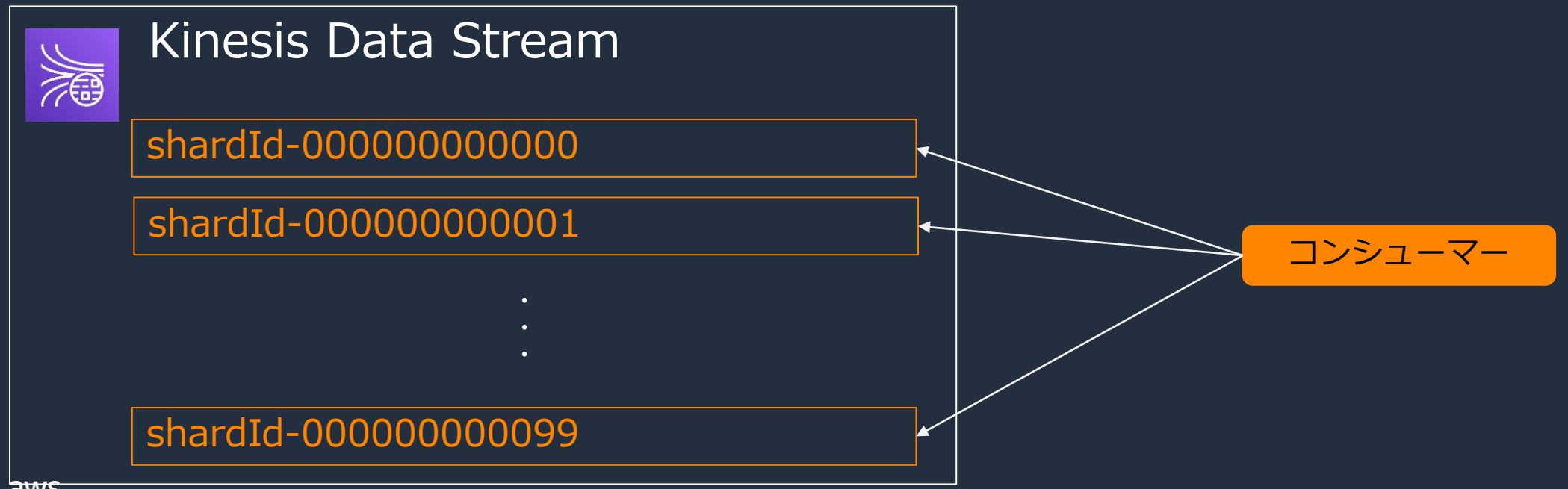
- GetShardIterator API で開始地点を取得
- GetRecords API でレコードを取得
- GetRecords API の戻り値に含まれる Iterator を取得し、再度 GetRecords API を実行 (以降繰り返し)



コンシューマーアプリケーション実装上の課題

スケーラビリティ・性能限界

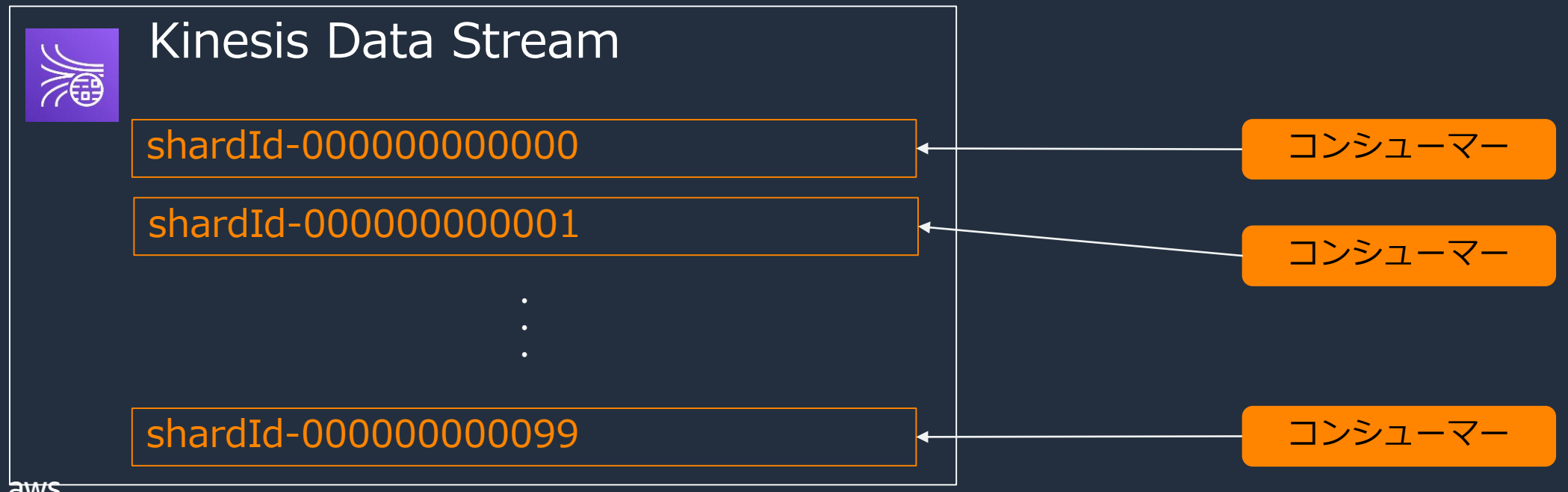
多数のシャード上のデータを、単一コンシューマーで
取得・処理することは困難



コンシューマーアプリケーション実装上の課題

コストパフォーマンス

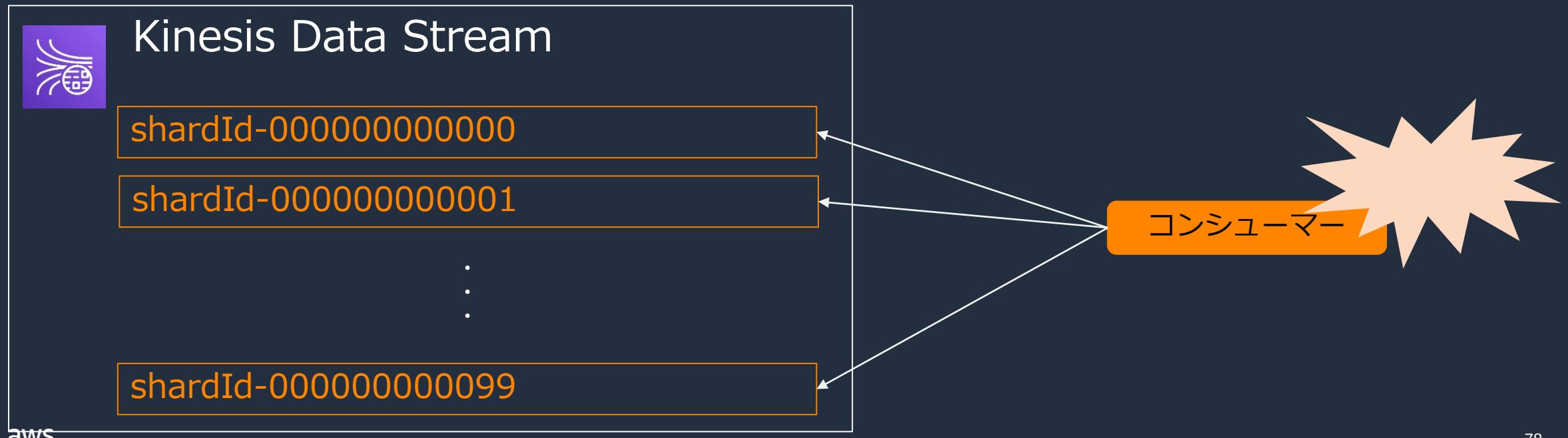
シャードごとにインスタンスを割り当てると、
リソースを過剰に割り当てることに



コンシューマーアプリケーション実装上の課題

可用性

ステート情報を外部に保存していないと、
ノードやアプリケーション障害発生時に
最初からデータ処理をすることになる

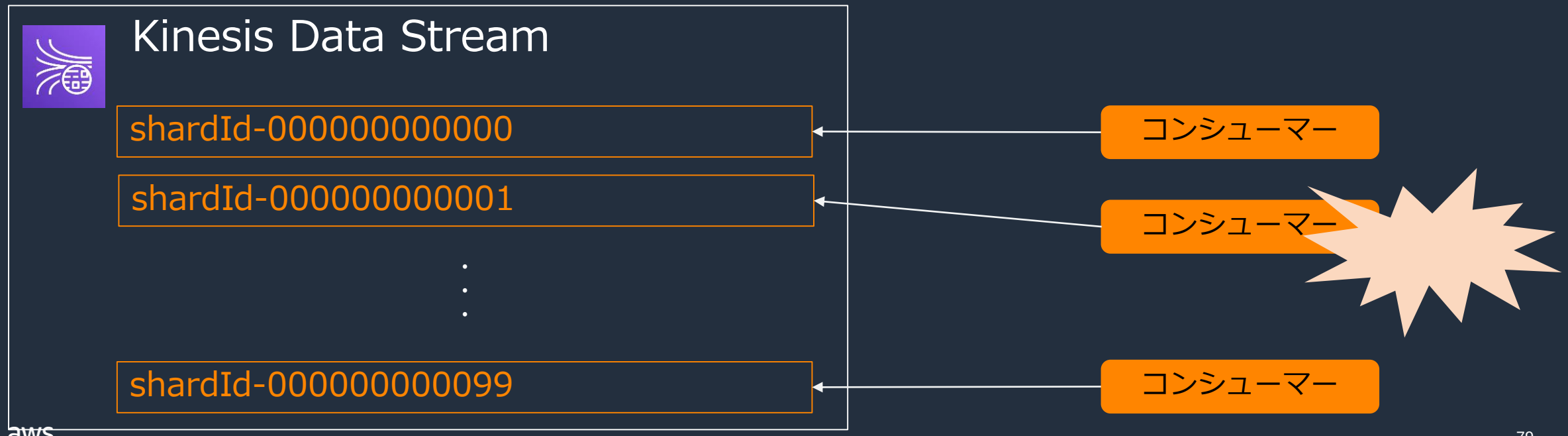


コンシューマーアプリケーション実装上の課題

エラーハンドリングの実装

複数コンシューマー構成だと、特定のコンシューマーで

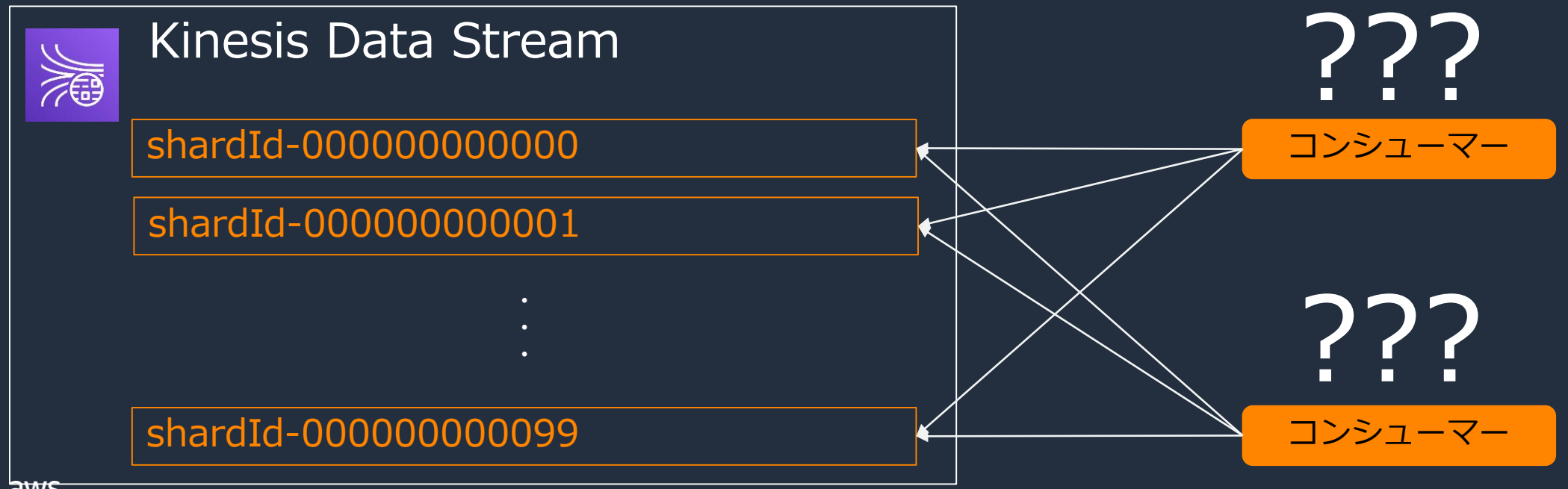
障害が発生した場合の処理のテイクオーバーが必要



コンシューマーアプリケーション実装上の課題

コンシューマー間のバランシング

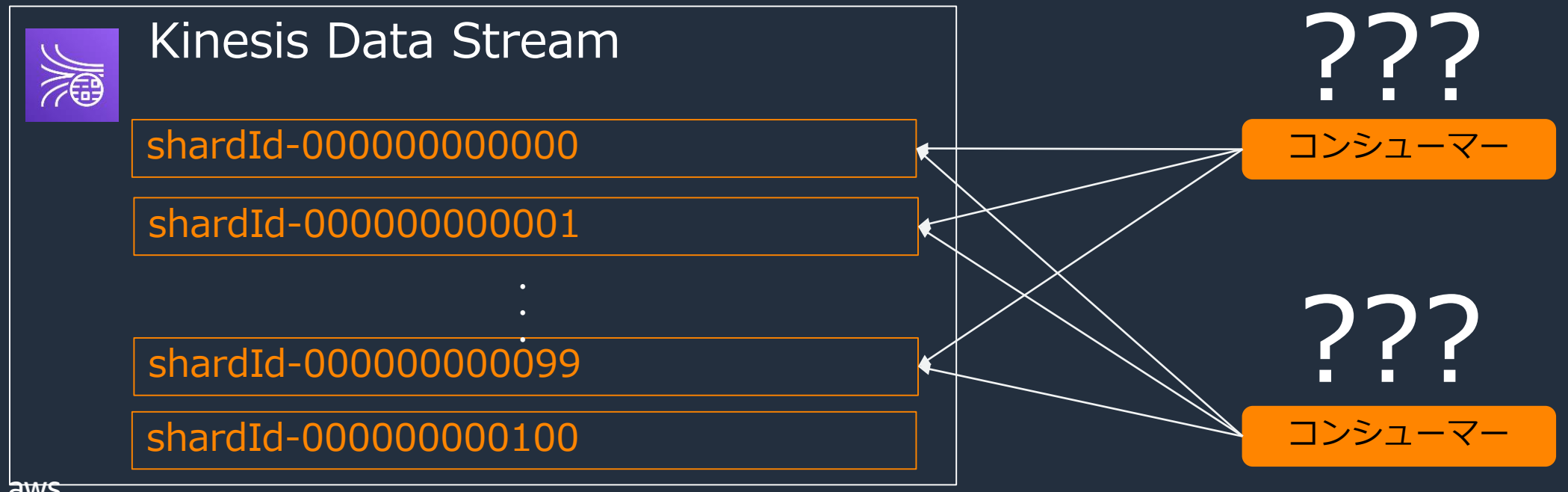
シャードとインスタンスを N:M 関係にすると、
どのインスタンスがどのシャードを担当するべきかの調整が必要



コンシューマーアプリケーション実装上の課題

シャード変動の追跡

シャードの増減が発生をトレースする必要あり



Amazon Kinesis Client Library (KCL)

Kinesis クライアントライブラリとは

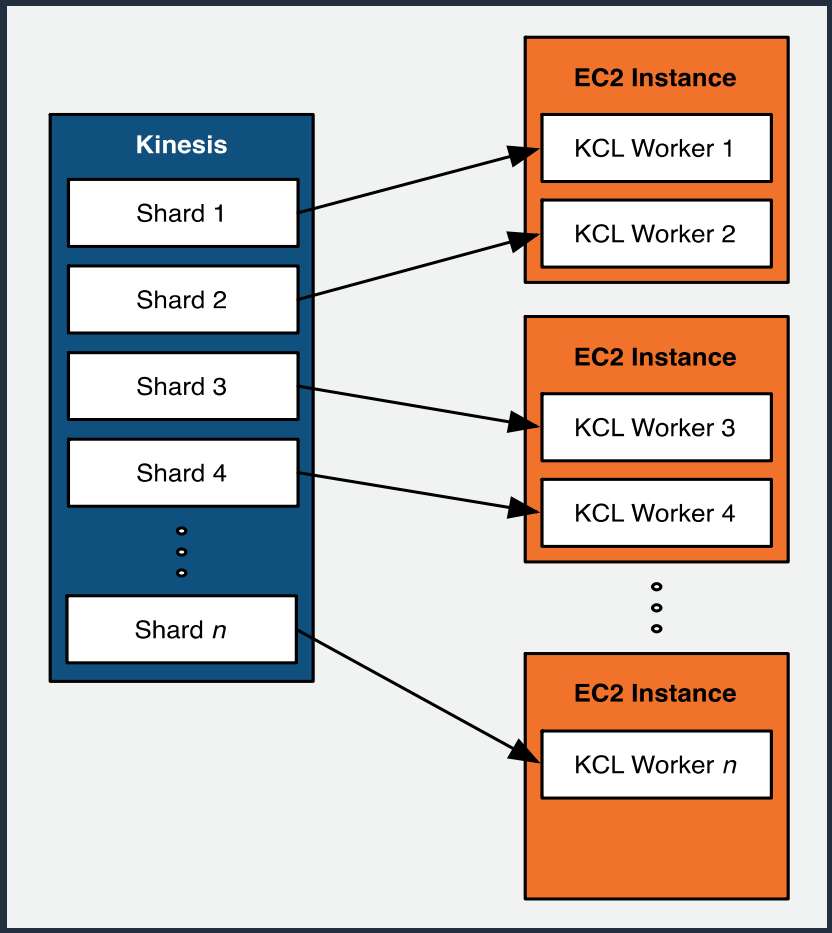
KCL は、分散コンピューティングに関連する複雑なタスクの多くを処理することで、Kinesis データストリームからデータを消費および処理するのに役立ちます。これには、複数のコンシューマーアプリケーションインスタンス間での負荷分散、コンシューマーアプリケーションインスタンスの障害に対する応答、処理済みのレコードのチェックポイント作成、リシャーディングへの対応が挙げられます。KCL はこれらのサブタスクをすべて処理するため、カスタムレコード処理ロジックの作成に集中できます。

KCLの責務

利用者の責務

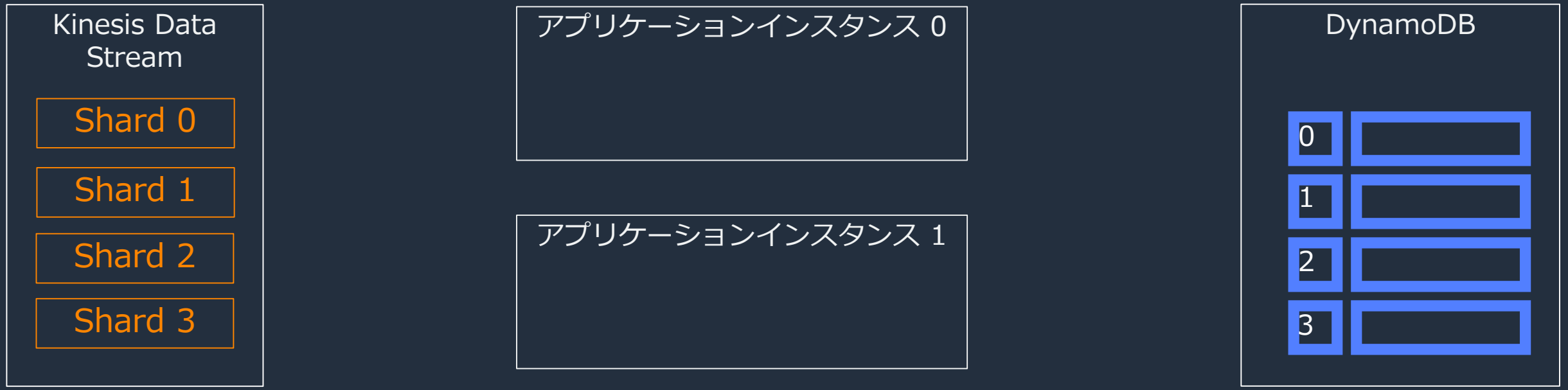
Amazon Kinesis Client Library (KCL)

- Kinesis Data Streams 用に Java で開発された Consumer ライブラリ
- Ruby、Python、Node.js、 .NET から利用可能な MultiLangDaemon も提供されている
- アプリケーションの処理状況を DynamoDB に保存することで、中断個所からの再開やノード間での処理引継ぎを可能に
- Kinesis Producer Library で集約されたレコードの集約解除をサポート



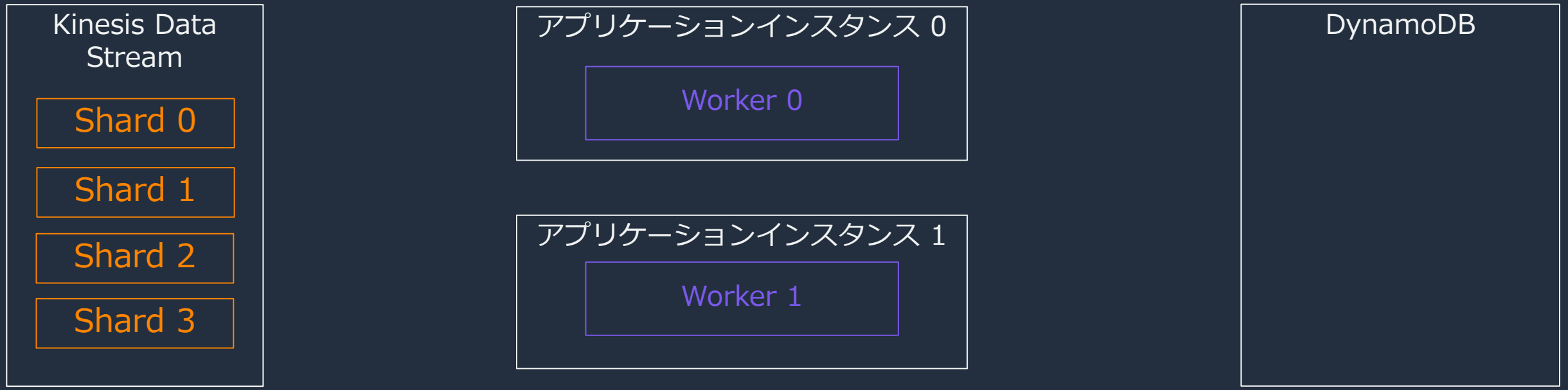
https://github.com/search?q=org%3Aawslabs+kinesis-client

KCL ライフサイクル > 起動時



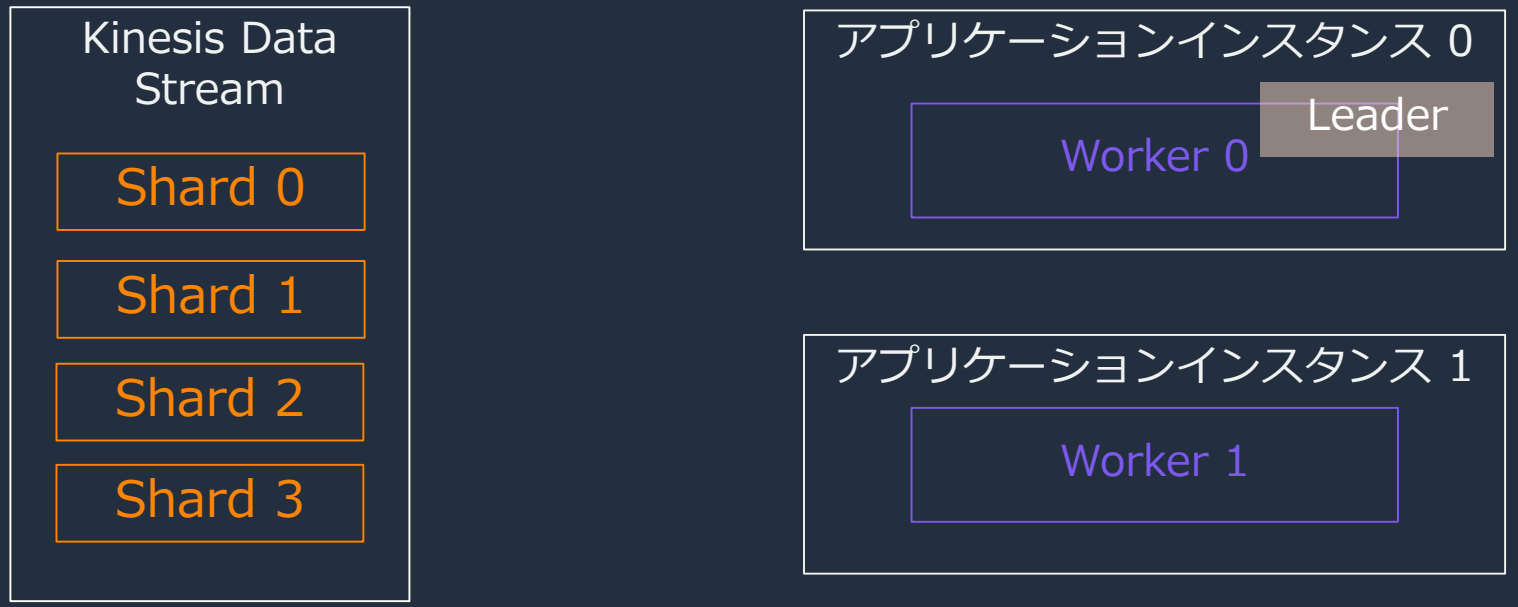
KCL ライフサイクル > 起動時

- ワーカーの起動



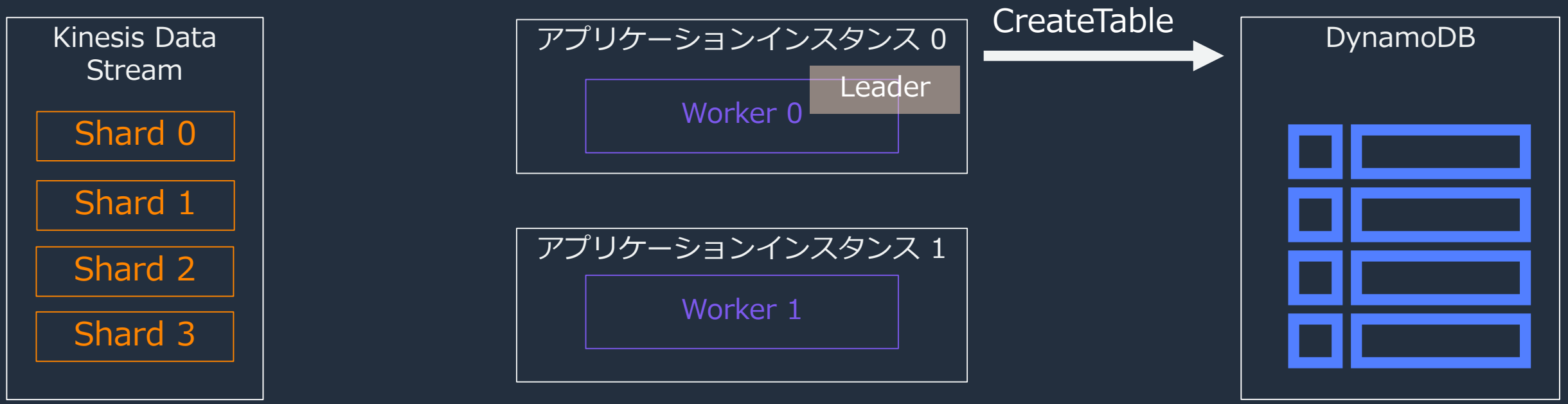
KCL ライフサイクル > 起動時

- ワーカーの起動
- リーダーの選出



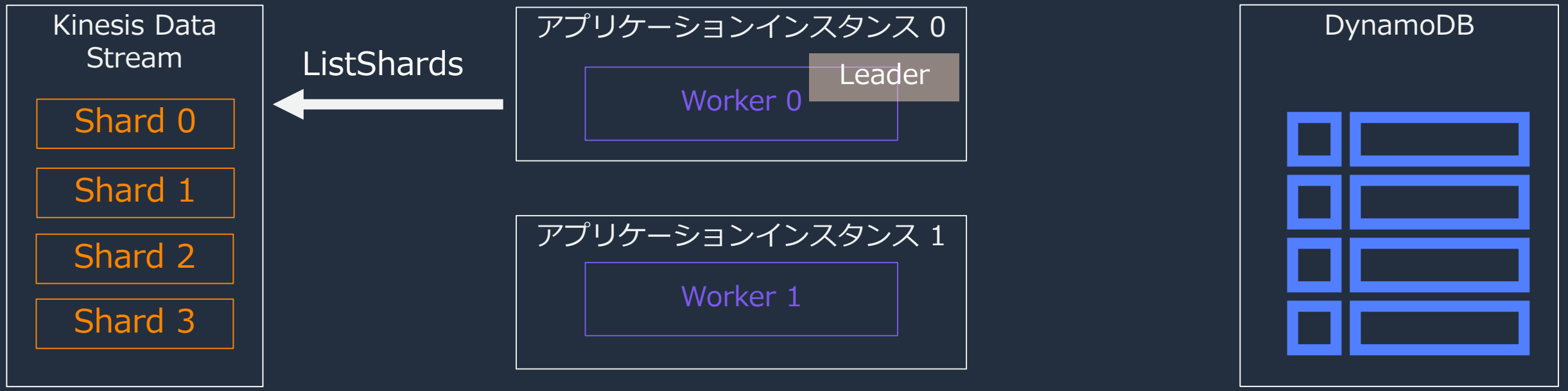
KCL ライフサイクル > 起動時

- ワーカーの起動
- リーダーの選出
- DynamoDB テーブルの作成 (リーダーのみ)



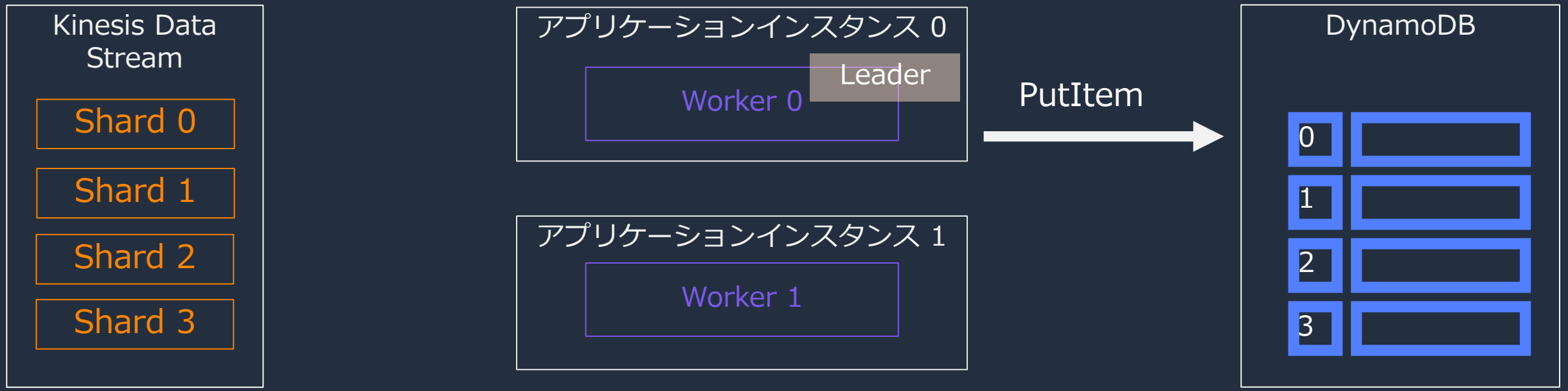
KCL ライフサイクル > 起動時

- ワーカーの起動
- リーダーの選出
- DynamoDB テーブルの作成 (リーダーのみ)
- シャードのリストアップ



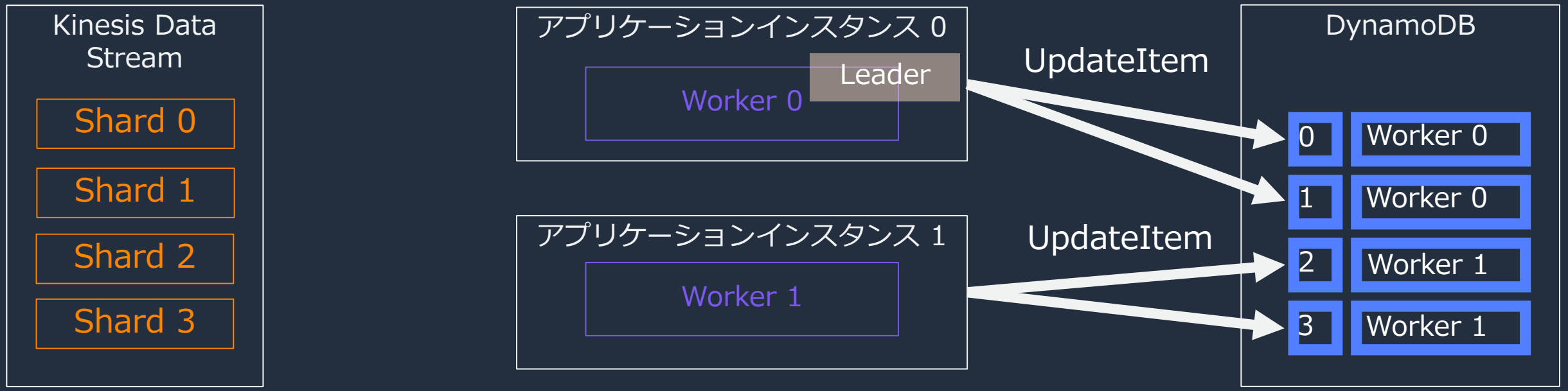
KCL ライフサイクル > 起動時

- ワーカーの起動
- リーダーの選出
- DynamoDB テーブルの作成 (リーダーのみ)
- シャードのリストアップ、リース情報(ワーカーとシャードのマッピング)の登録 (リーダーのみ)



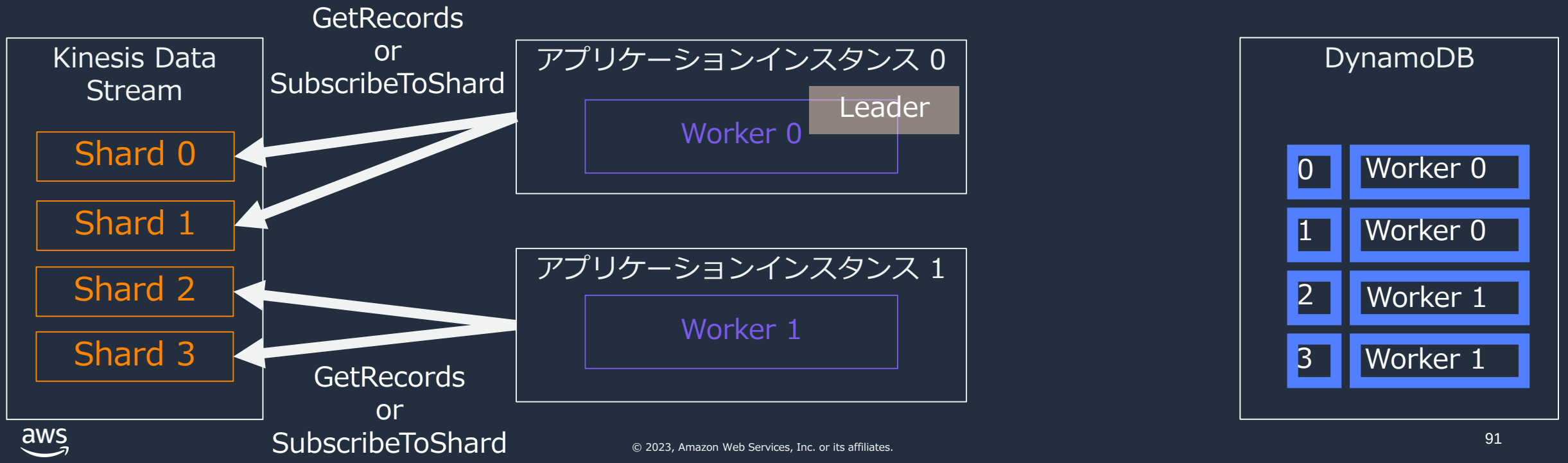
KCL ライフサイクル > 起動時

- リースの取得、リーステーブルの更新



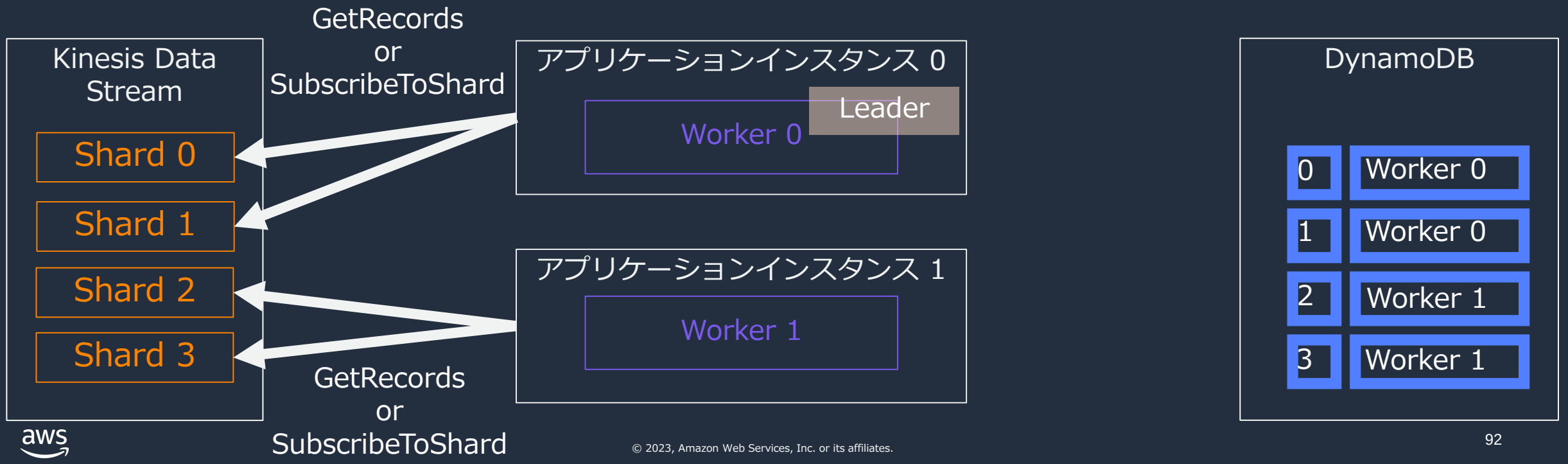
KCL ライフサイクル > 起動時

- リースの取得、リーステーブルの更新
- データレコード取得、処理開始



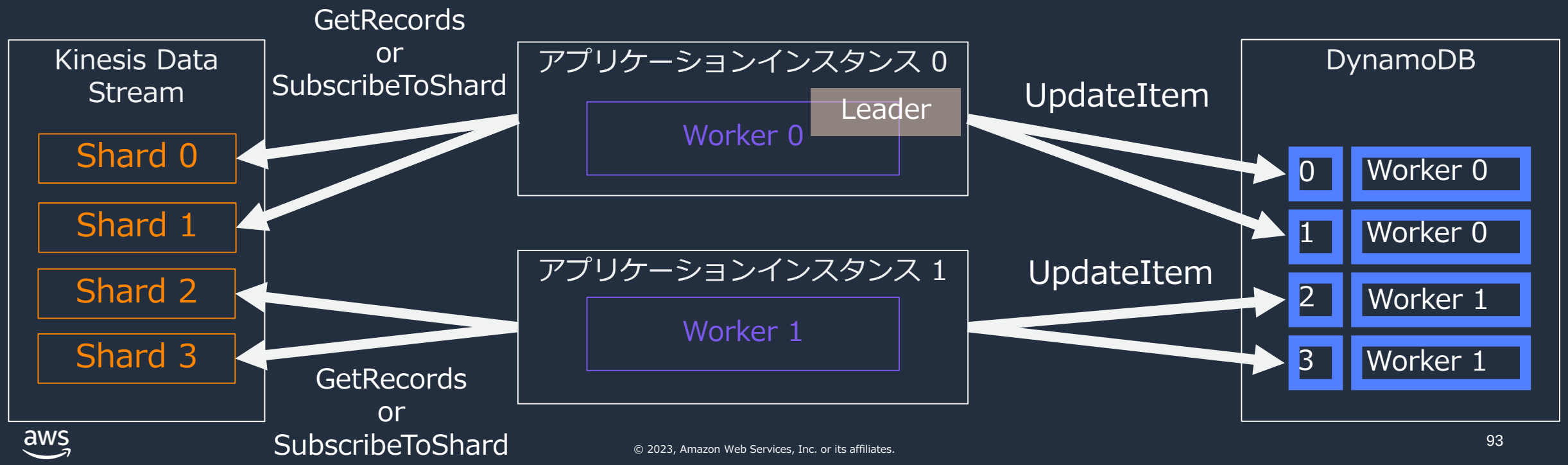
KCL ライフサイクル > 起動後

- データレコードの取得と処理は継続



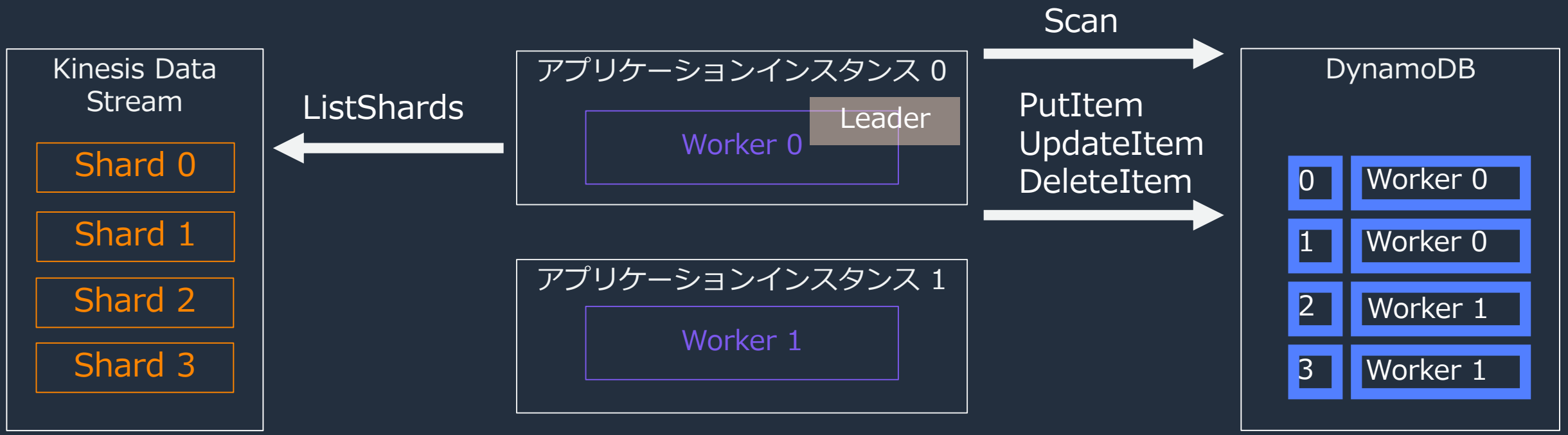
KCL ライフサイクル > 起動後

- データレコードの取得と処理は継続
- 定期的に DynamoDB テーブル上のチェックポイント情報(シャード内のレコードをどこまで取得しているか)を更新。更新処理はアプリケーション内で明示的に呼び出す必要あり
- リースのチェック・更新も定期的に実行。リース管理はユーザーロジック外で実行される



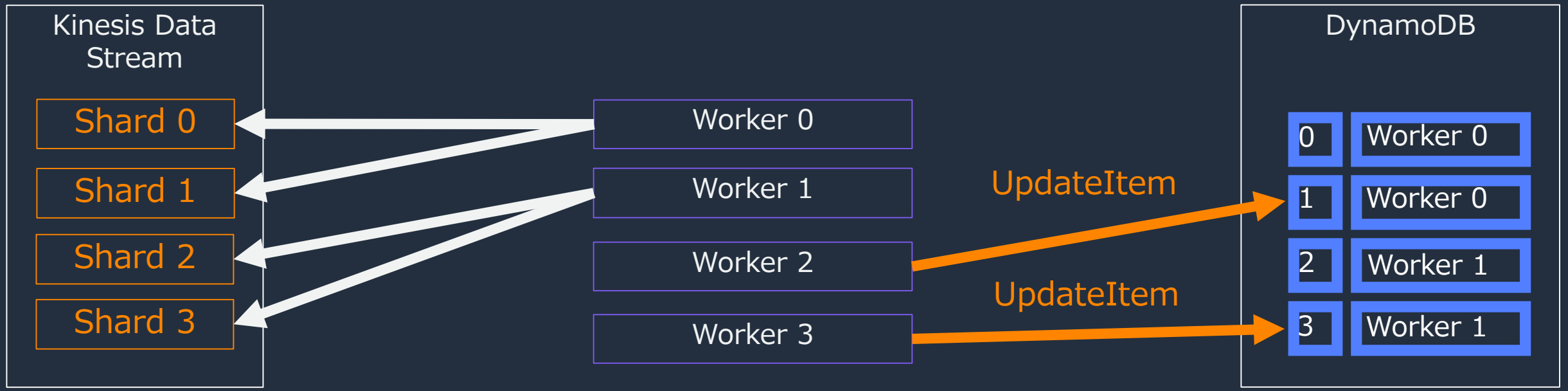
KCL ライフサイクル > 起動後

各ワーカーによる更新処理に失敗している、ワーカーの入れ替えが発生した等の理由による不整合を解消するために、リーダーは定期的なリース情報の一括更新を行う



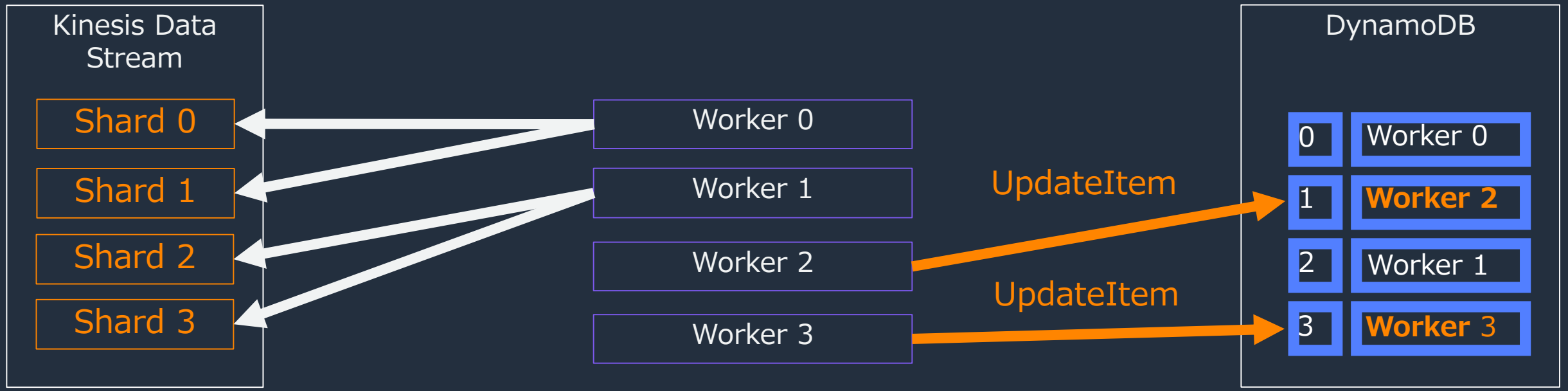
KCL ライフサイクル > ワーカーの追加

- 新規に追加されたワーカーがリースを取得



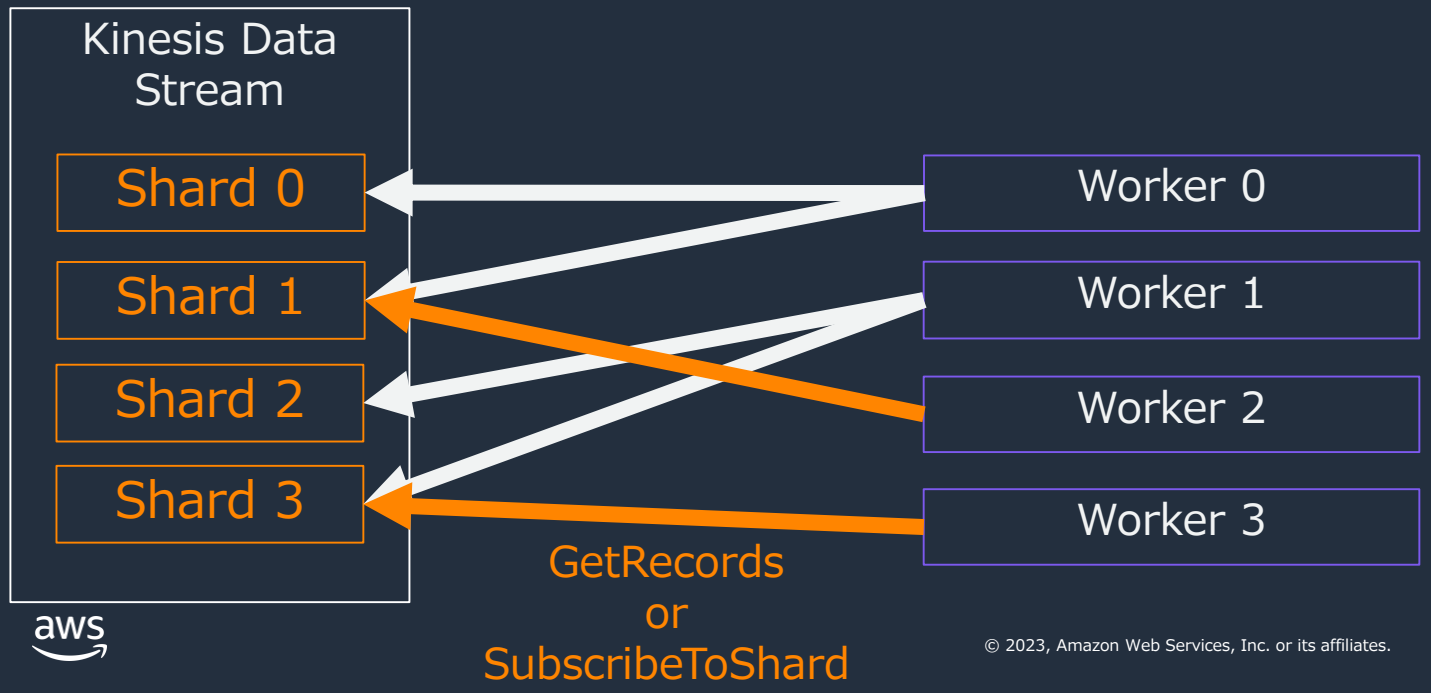
KCL ライフサイクル > ワーカーの追加

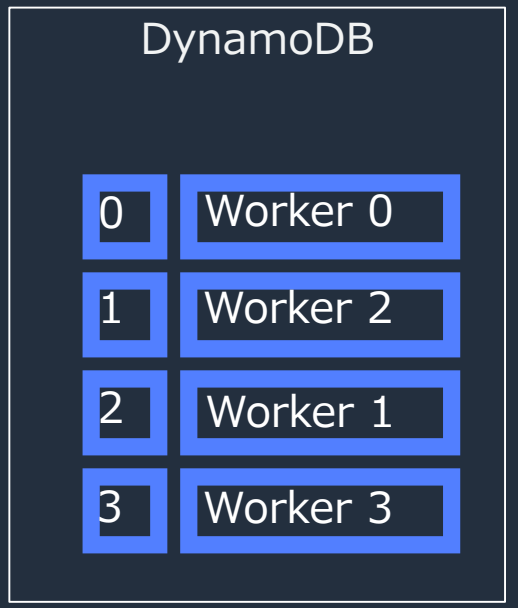
- 新規に追加されたワーカーがリースを取得



KCL ライフサイクル > ワーカーの追加

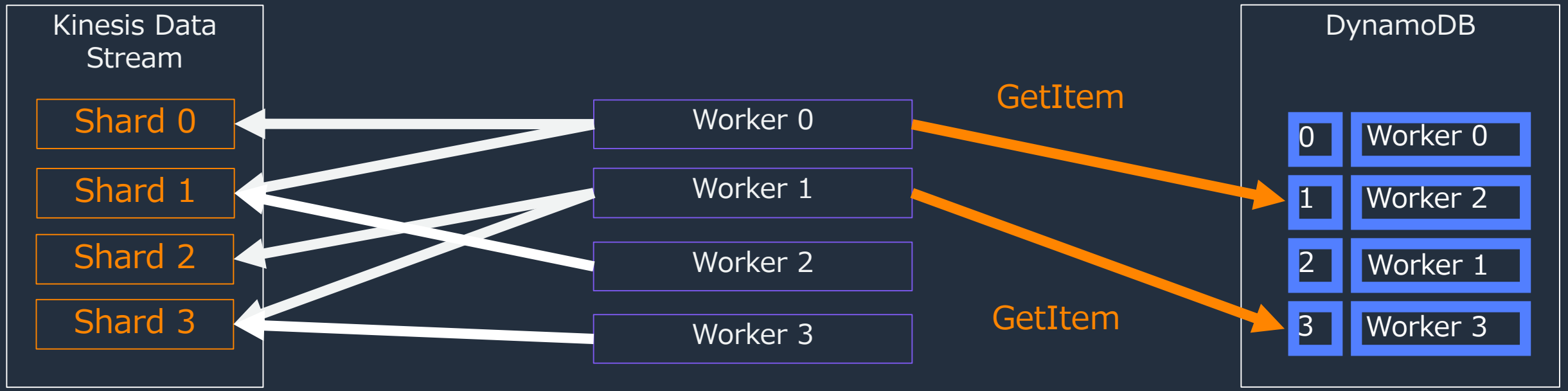
- 新規に追加されたワーカーがリースを取得
- データレコード取得、処理開始





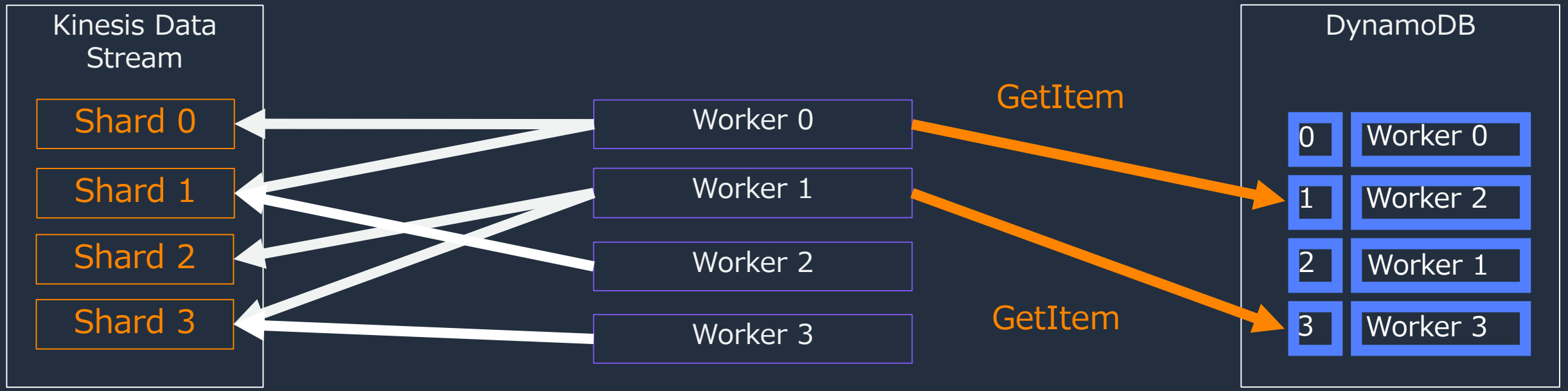
KCL ライフサイクル > 既存ワーカーの処理停止

- 既存ワーカーは、リース更新時にリースが他のワーカーによって取得された(奪われた)ことを検出する



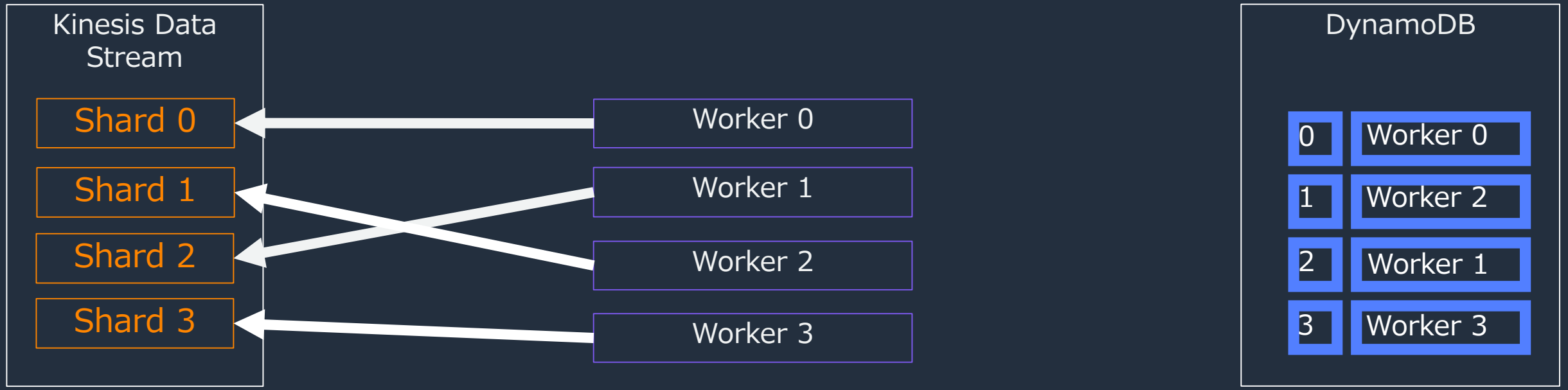
KCL ライフサイクル > 既存ワーカーの処理停止

- 既存ワーカーは、リース更新時にリースが他のワーカーによって取得された(奪われた)ことを検出する



KCL ライフサイクル > 既存ワーカーの処理停止

- 既存ワーカーは、リース更新時にリースが他のワーカーによって取得された(奪われた)ことを検出する
- 既存ワーカーは、リースが奪われたシャードからのレコード取得処理を停止する



KCL のチューニングポイント > リース

リース関係の設定を調整することでワーカーインスタンスの負荷をコントロールすることが可能

maxLeasesForWorker

- ワーカーあたりのリース数 = 処理可能なシャード数
- 1 に設定すると、シャード分割時に子シャードのリースを取れずスタックしてしまうため、2 以上を推奨

maxLeasesToStealAtOneTime

- リース取得処理の並列度

KCL のチューニングポイント > DDB キャパシティ

KCL によって作成される DynamoDB テーブルのデフォルト設定は、KCL のバージョンによって異なる

デフォルト値

- KCL v2.3.6 まで: プロビジョンドモード。WCU/RCU はそれぞれ 10
- KCL v2.3.7 以降: オンデマンドモード

要求キャパシティは、シャード数、アプリケーションワークロード、チェックポイント更新頻度によって変わるため、不足する場合はキャパシティ追加やモード変更を行う

KCL チューニングポイント > データ取得間隔

レコードの取得間隔はデフォルトで 1 秒。以下のパラメーター変更で短縮可能だが、
GetRecords API の TPS (5 TPS / sec) Limit に注意すること

- shardConsumerDispatchPollIntervalMillis (KCL v2)
- idleTimeBetweenReadsInMillis (KCL v1)

本設定によるポーリング間隔は、GetRecords により取得されたレコードの件数がゼロ件である場合も同様に変更される。
idleTimeBetweenReadsInMillis を 500ms に設定した場合の挙動は以下の通り

- レコード件数 1 件以上: レコード処理を行ってから 500ms 待機し次のレコードを取得
- レコード件数 0 件: 500ms 待機し次の GetRecords を実行

KCL > よくある疑問

Q. チェックポイント間隔は？

アプリケーション実装次第。取得したレコードを処理する RecordProcessor から、任意のタイミングでチェックポイントを更新する

Q. CLOSED シャードのリースはいつ解放されるのか？

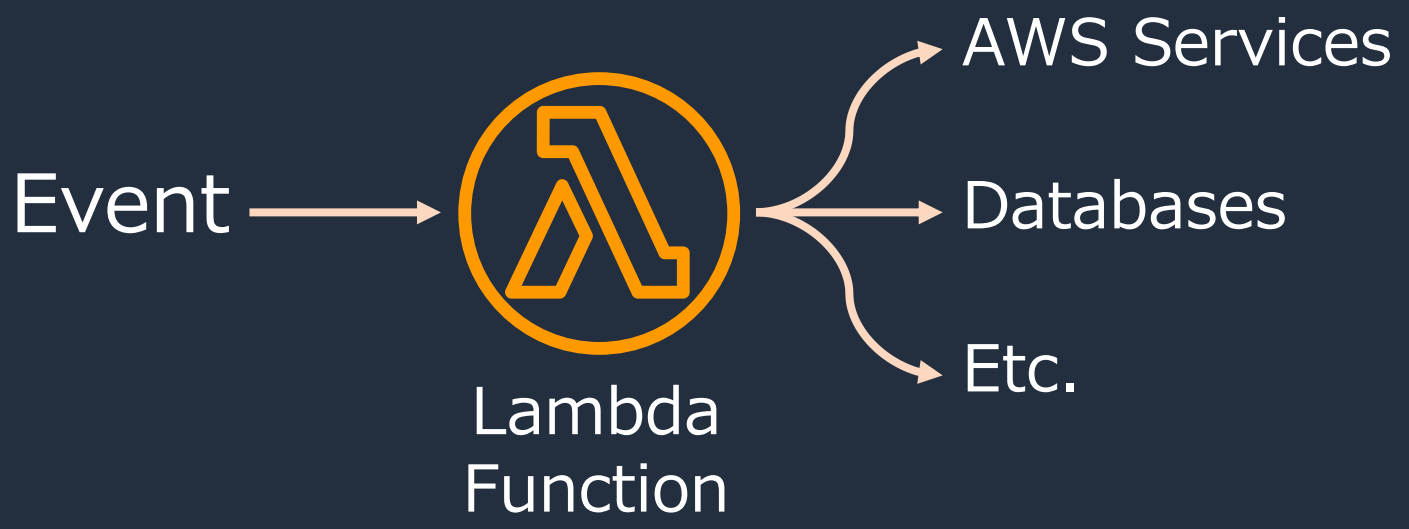
対象シャードが CLOSED であり、かつ最後のデータまで処理が行われてから 5 分が経過すると解放される

https://github.com/awslabs/amazon-kinesis-client/blob/master/amazon-kinesis-client-multilang/src/main/java/software/amazon/kinesis/coordinator/KinesisClientLibConfiguration.java#L984-L1003

その他のコンシューマー

AWS Lambda

- サーバーレスのコード実行サービス。イベント呼び出しをサポート
- 様々なランタイムに加えて、カスタムランタイムやコンテナをサポート
- コード実行時間(ミリ秒単位)と割り当てメモリに応じた課金

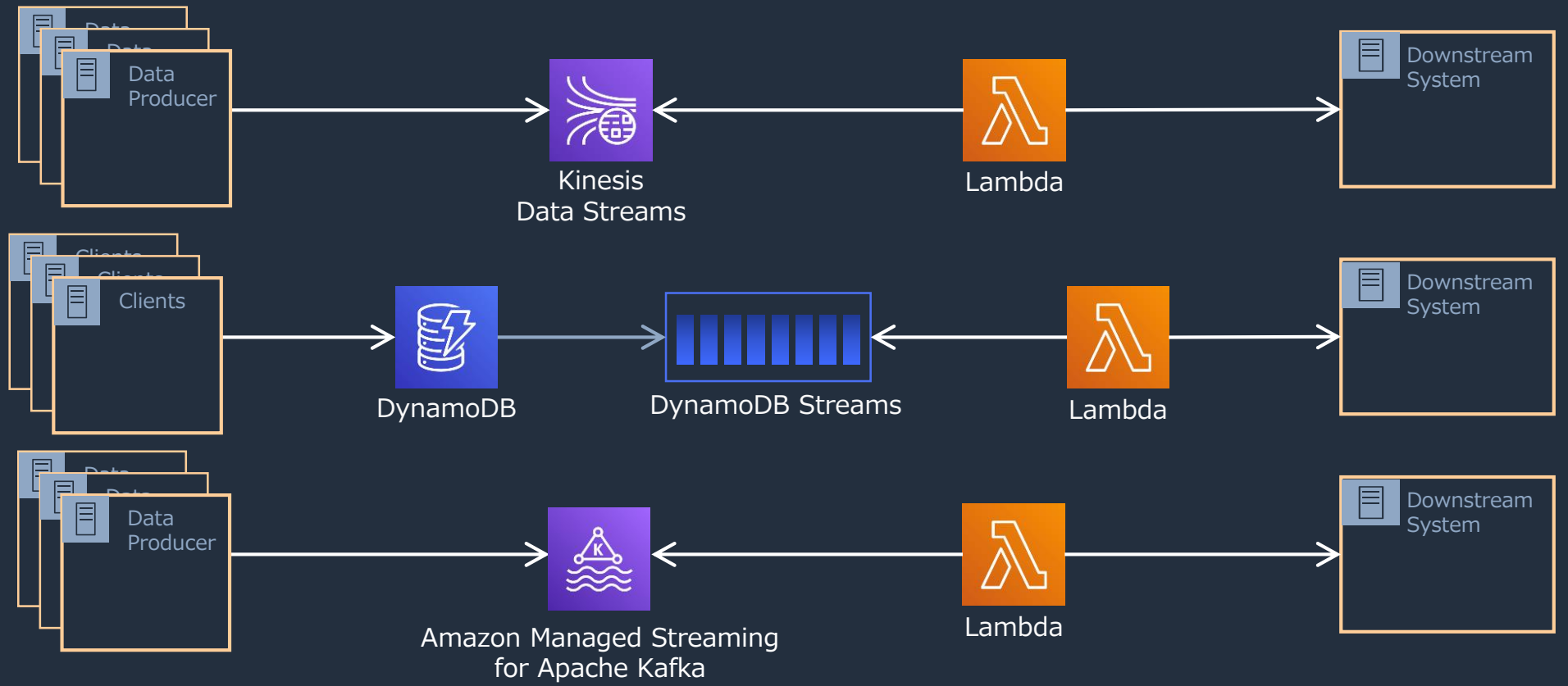


Python, Javascript
Java, Golang, C#
BYOL, Container images

https://aws.amazon.com/jp/lambda/

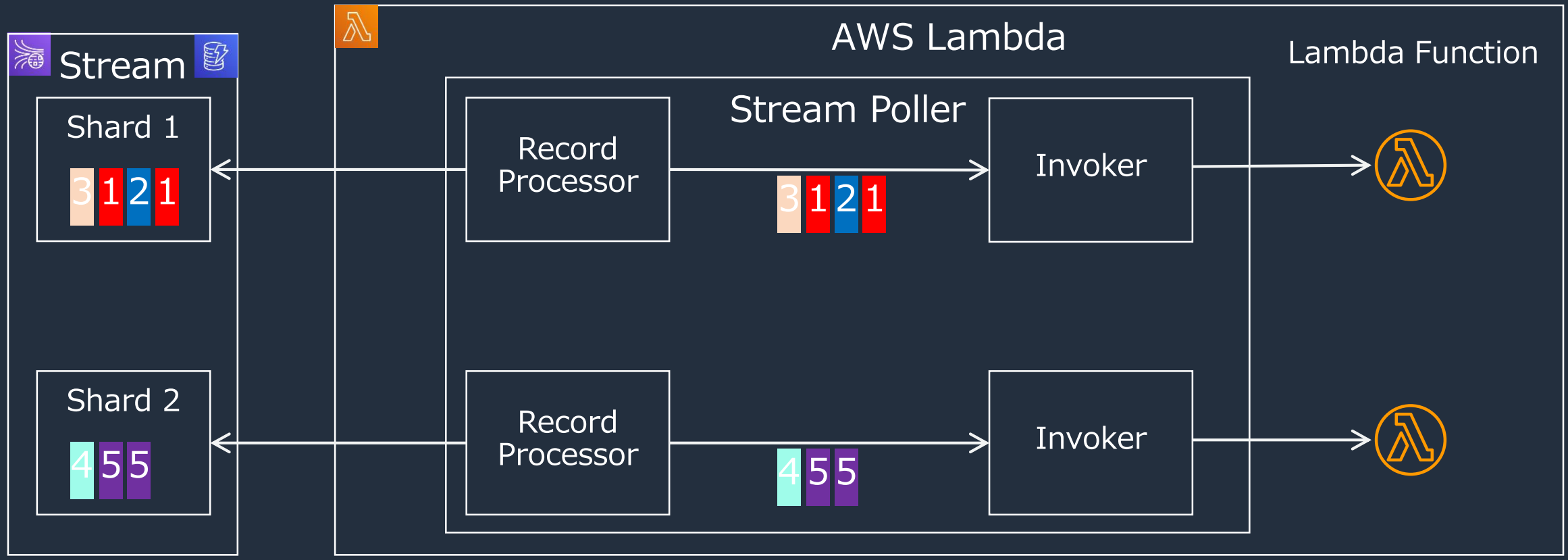
Lambda Consumer

- Kinesis Data Streams、DynamoDB Streams、Kafka のストリームデータを取得可能
- リトライ回数、リトライ期間、バッチサイズ、バッチ期間の調整、デッドレターキューなど、様々な Lambdaの機能を活用できる



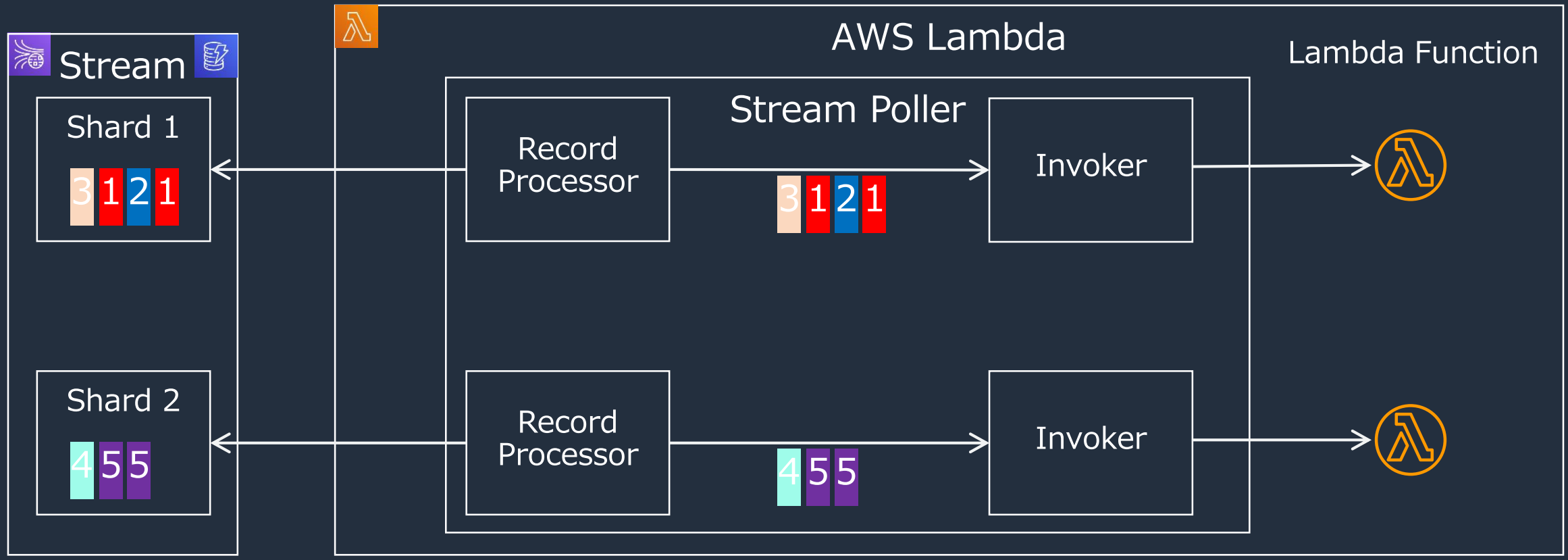
Lambda Consumer > アーキテクチャ

- Lambda サービスがストリームに対して定期的にポーリングを実施し、レコードを取得する
- 複数レコードが Lambda Function 呼び出し時のイベントとしてまとめて渡される



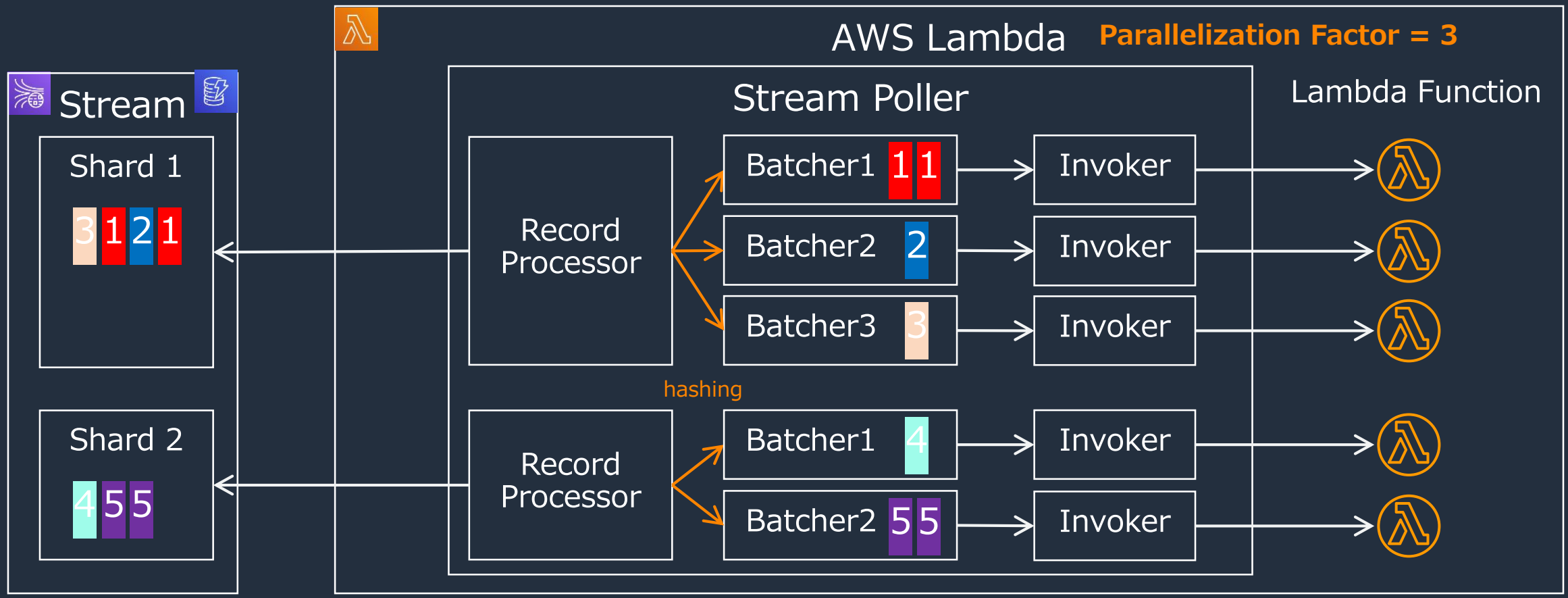
Lambda Consumer > スケーラビリティ

デフォルトでは、ストリームのシャード数に応じて Lambda Function の並列実行数が制限される



Lambda Consumer > スケーラビリティ

- Parallelization Factor により、並列度ををデフォルトの 1 倍から最大 10 倍まで調整可能
- パーティションキーに応じて内部振り分けが行われるため、キーのカーディナリティが重要

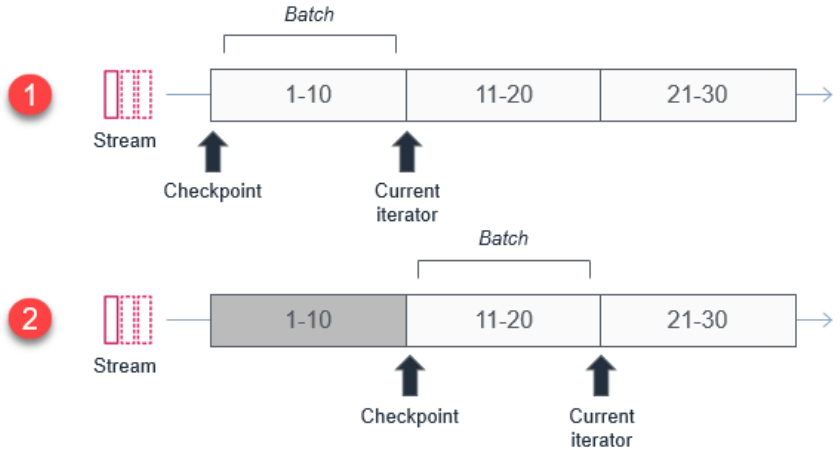


Lambda Consumer > リトライ機構

- レコードバッチの処理でエラーが発生した場合、関数に設定したリトライ回数、リトライ期限、もしくはデータストリームの保持期限に達するまでリトライが行われる
- リトライ回数・期限超過時に、デッドレターキューにイベントを移動することも可能

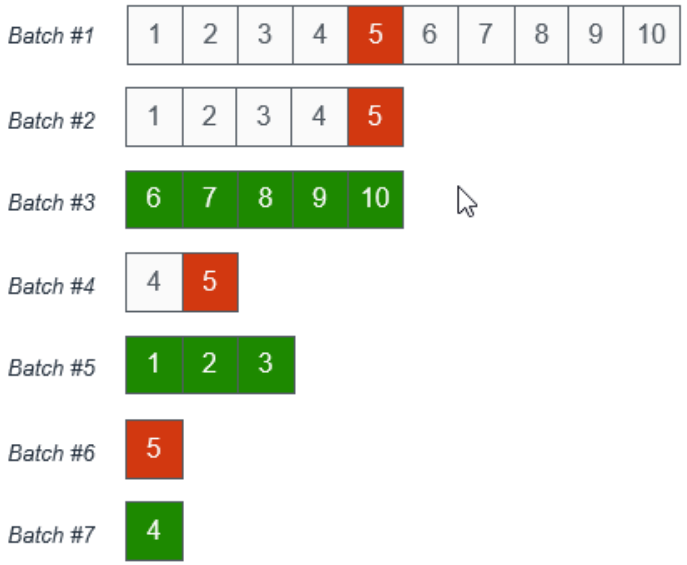
Default settings

エラーが発生したバッチを再処理



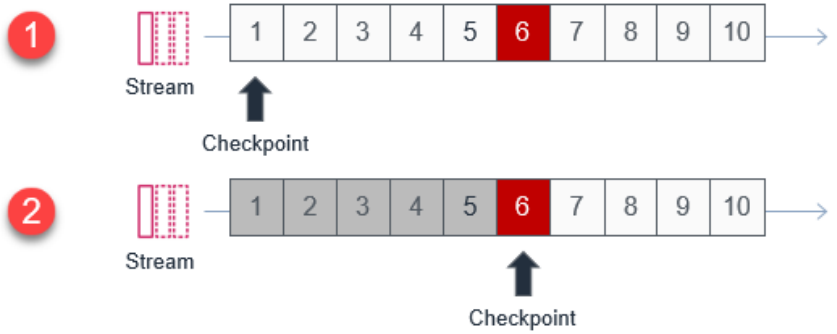
Bisect batch

エラーが発生したバッチを分割して再処理



Custom Checkpoint

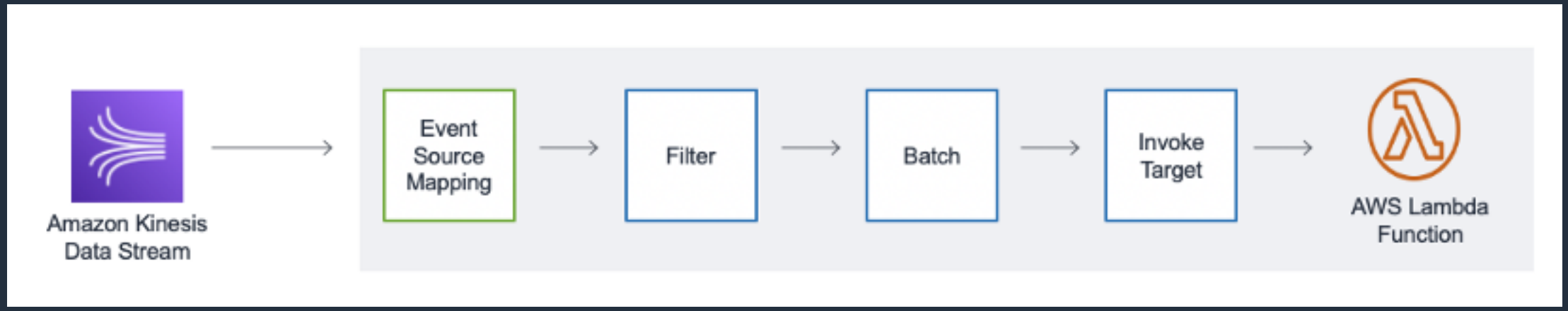
開始地点を指定して再処理



Lambda Consumer > イベントフィルタリング

- 条件に合致したデータレコードだけを Lambda Function に渡す
- Lambda のコスト削減に有効
- EventBridge の Rule と同じ記法を採用しており，数値比較、文字列比較などによるフィルタが可能

```
{
  "data": {
    "tire_pressure": [{
      "numeric": ["<", 32]
    }]
  }
}
```

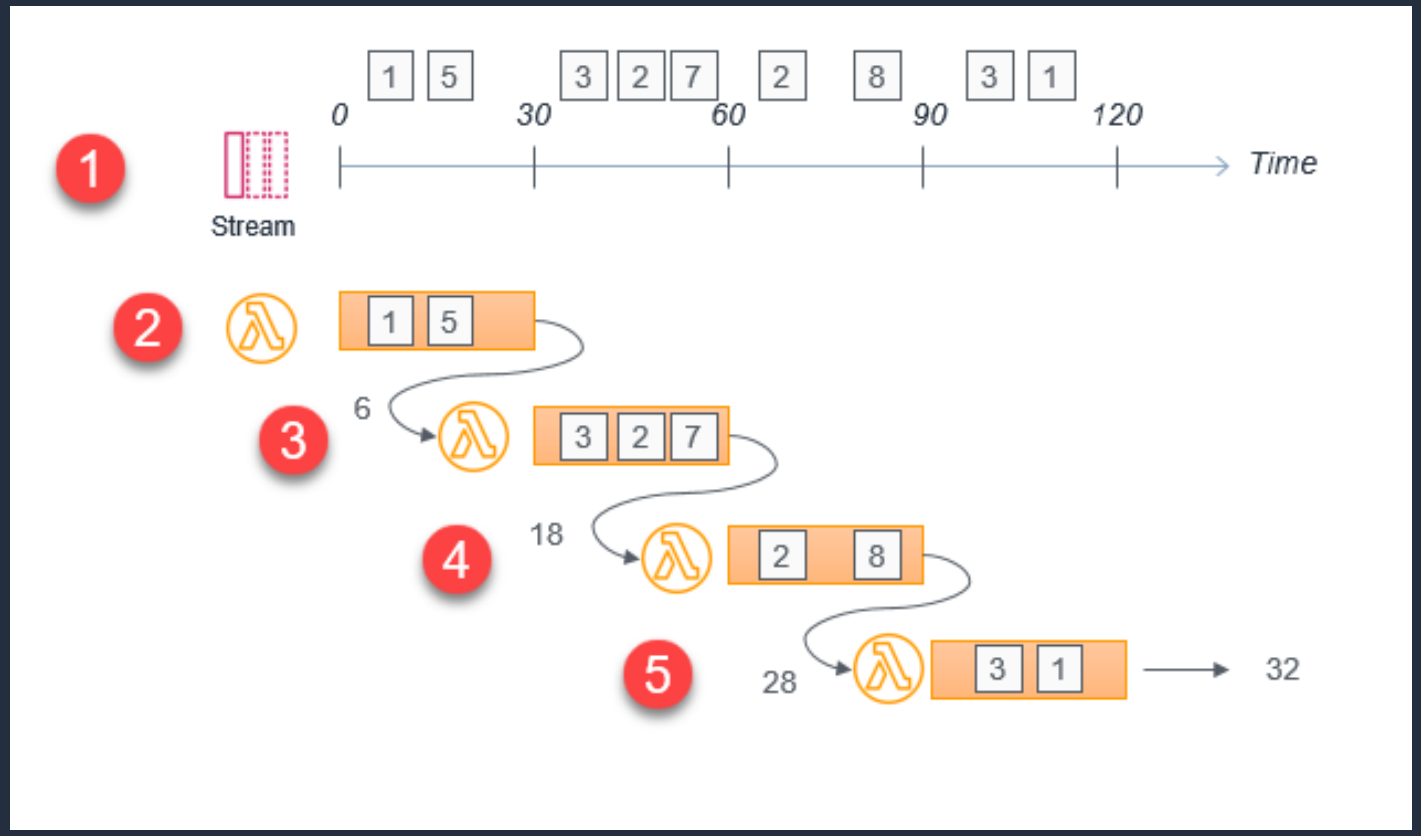


https://aws.amazon.com/jp/blogs/compute/filtering-event-sources-for-aws-lambda-functions/
https://docs.aws.amazon.com/lambda/latest/dg/invocation-eventfiltering.html#filtering-syntax

Lambda Consumer > ウィンドウ処理

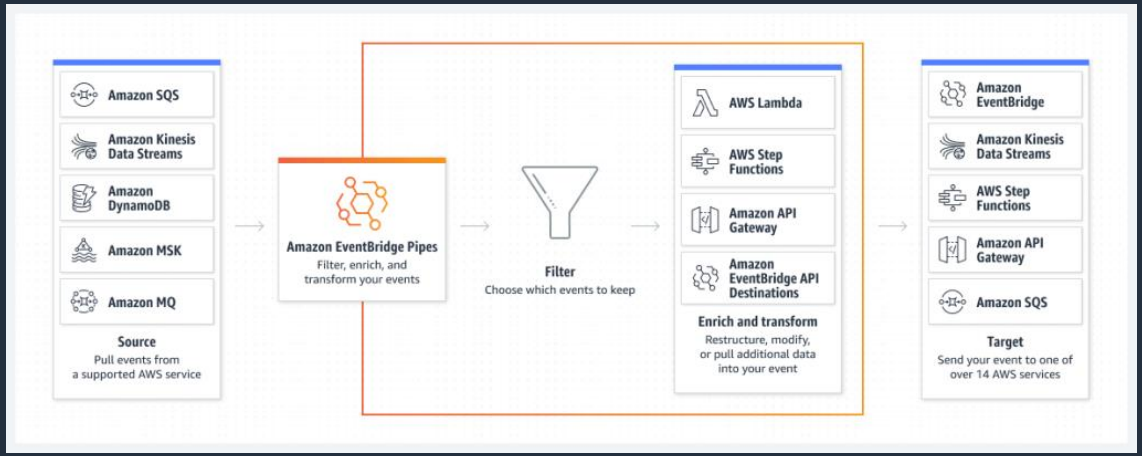
- 指定した秒数ごとにレコードバッチを区切って関数を実行する

- 関数に渡されるレコードバッチはシャード単位で区切られているため、シャードにまたがった集計処理は実行できない



Amazon EventBridge Pipes によるレコードの配信

- イベントバス不要。ソースとターゲットを
  シームレスに接続
- イベントのフィルタリング、変換、
  Amazon API Gateway、AWS Step Functions、
  カスタム API と連携したエンリッチも可能
- 複数のイベントをバッファリングし、
  マイクロバッチとして一括処理することも可能
- ソース側でイベントの順序性が保証されている
  場合、順序を保ったままターゲットに配信
- Amazon SQS -> Amazon Kinesis など、
  異なるサービス間の連携を
  ノーコード/ローコードで実現



- **ソースとして利用可能なサービス**
  Amazon SQS, **Amazon Kinesis**, Amazon DynamoDB, Amazon MSK, セルフマネージド型 Apache Kafka, Amazon MQ
- **ターゲットとして利用可能なサービス**
  Amazon Kinesis Data Streams, Amazon Kinesis Data Firehose など、 イベントバスで利用できるターゲットと同様

Kinesis Data Firehose によるデータ配信

Extract
Sources (20+)

Streaming as a source

- **Amazon Kinesis Data Streams**

Direct PUT

- AWS SDK
- Customer Application Producers (e.g., mobile app)
- 20+ AWS services (e.g., AWS IoT, CloudWatch, AWS WAF, SNS, **VPC Flow Logs**)
- Agents (e.g., Kinesis Agent, Fluentd, FluentBit, Apache Nifi)

New

Transform

- Lambda Custom Transformation
- Built-in Transformation

- Parquet, ORC
- S3 Dynamic Partitioning
- Record Delimiters
- Compression
- Buffer and batch

Load
Outputs (15+)

AWS destination

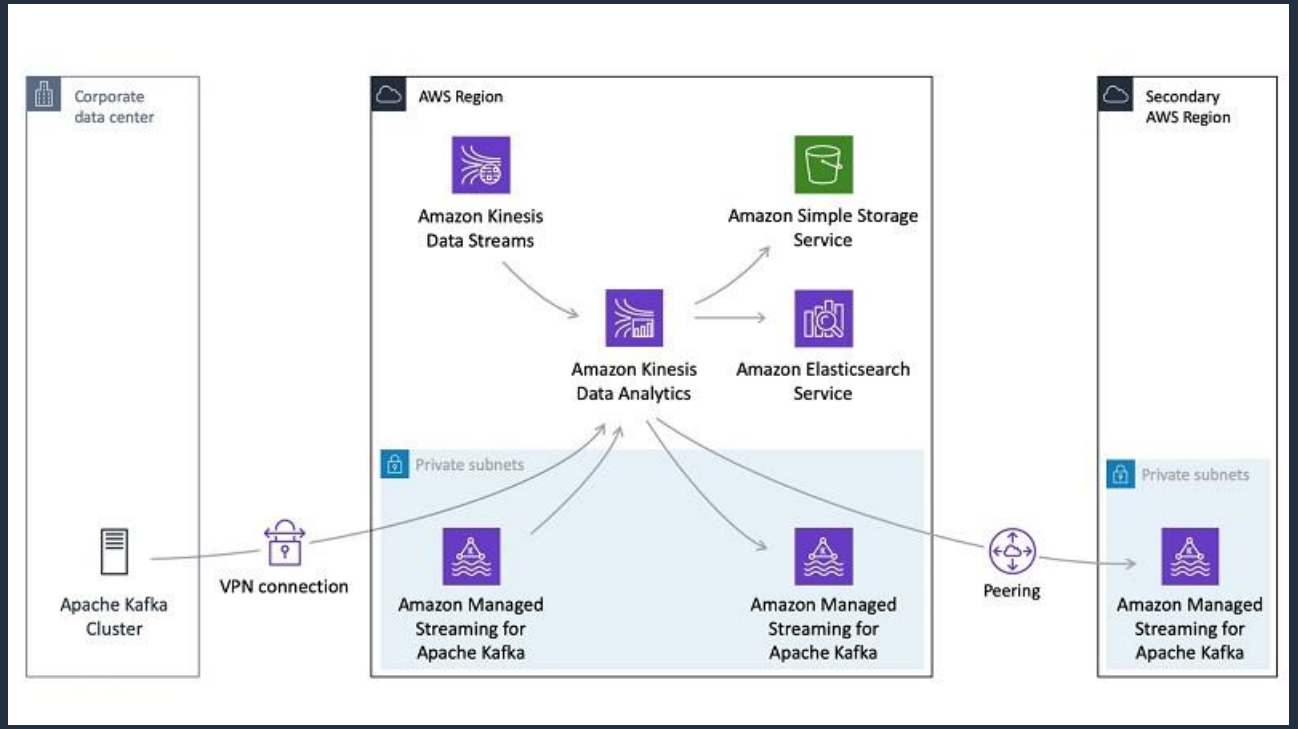
- Amazon S3
- Amazon Redshift
- Amazon OpenSearch and **Amazon OpenSearch Serverless**
- Amazon API Gateway

Third-party integration destination

- Splunk
- HTTP endpoint (9+ partners such as **Logz.io**, Datadog, etc…)

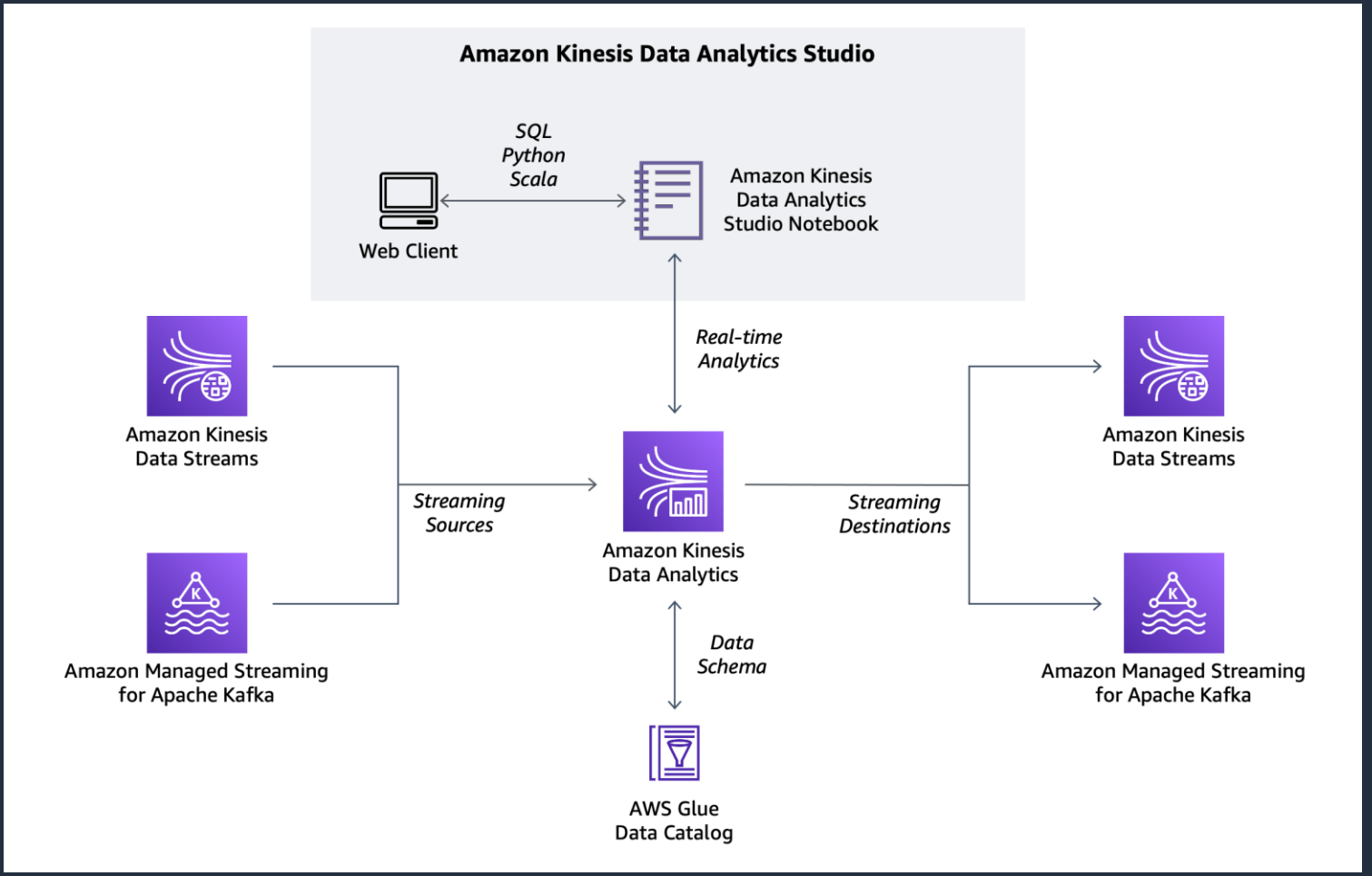
Amazon Kinesis Data Analytics for Flink との連携

- Kinesis Data Analytics for Flink は、Flink アプリケーションを実行するフルマネージドサービス
- Flink は Kafka をはじめとして多数のサービス、データストアに対応しており、柔軟なデータ処理、データ連携を実現可能
- Java, SQL, Python による記述をサポート



Amazon Kinesis Data Analytics Studio を利用した開発

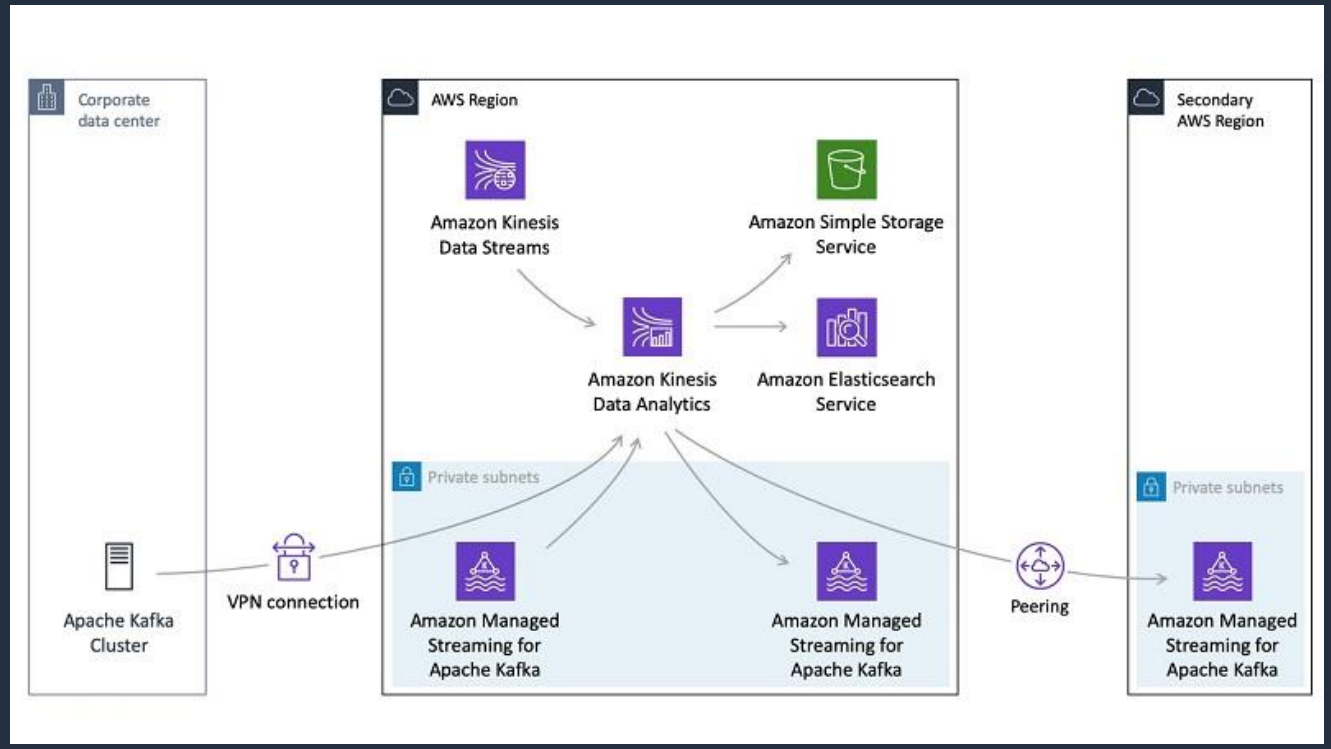
- マネージドな Zeppelin ノートブックを使用して、アドホックなトピックのデータ分析が可能
- 作成した分析ロジックを Flink アプリケーションとして保存、本番へデプロイすることも可能
- Java, Scala, Python, SQL など複数言語に対応



https://aws.amazon.com/jp/blogs/news/introducing-amazon-kinesis-data-analytics-studio-quickly-interact-with-streaming-data-using-sql-python-or-scala/

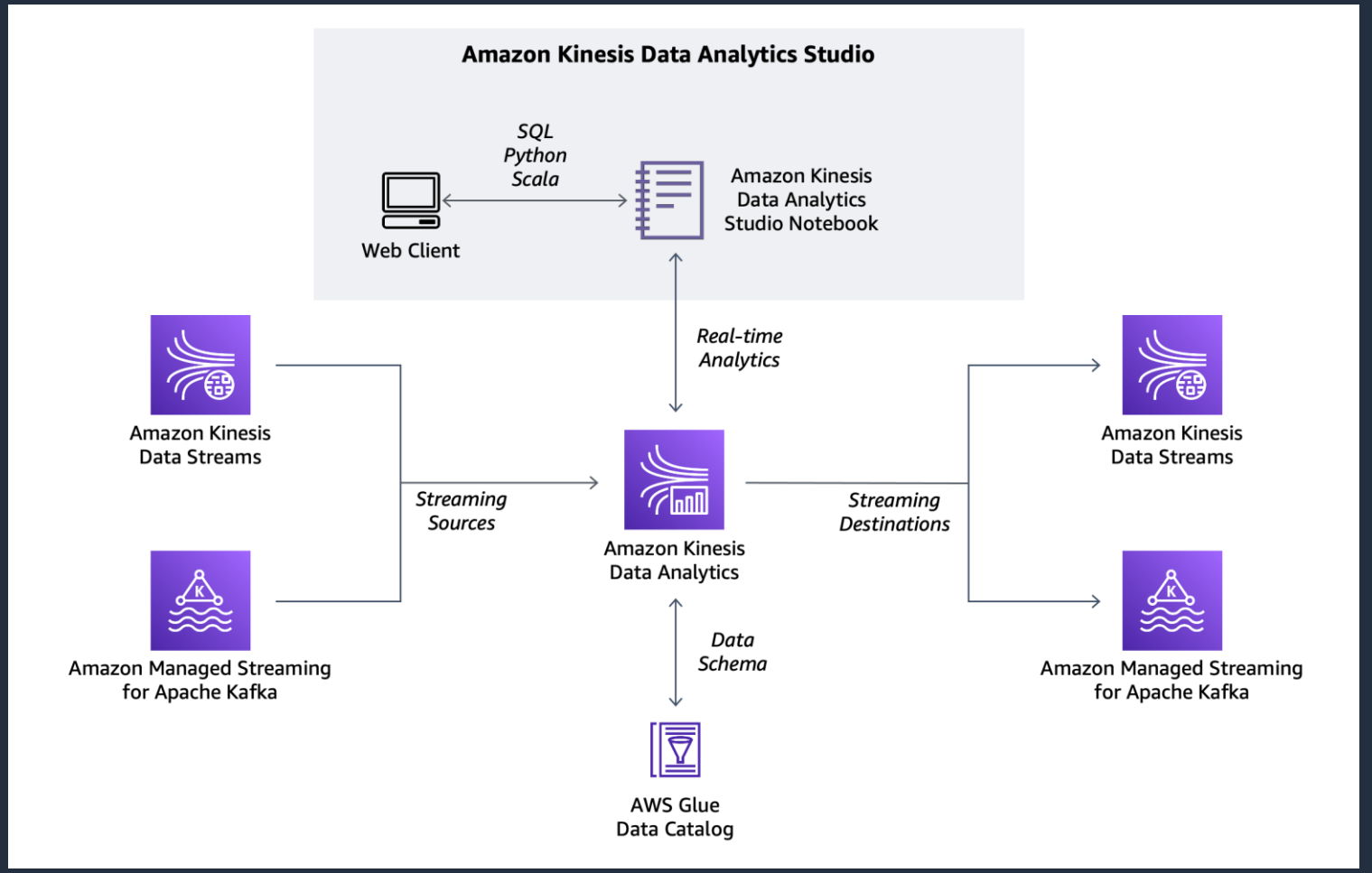
Amazon Kinesis Data Analytics for Flink との連携

- Kinesis Data Analytics for Flink は、Flink アプリケーションを実行するフルマネージドサービス
- Flink は Kafka をはじめとして多数のサービス、データストアに対応しており、柔軟なデータ処理、データ連携を実現可能
- Java, SQL, Python による記述をサポート



Amazon Kinesis Data Analytics Studio を利用した開発

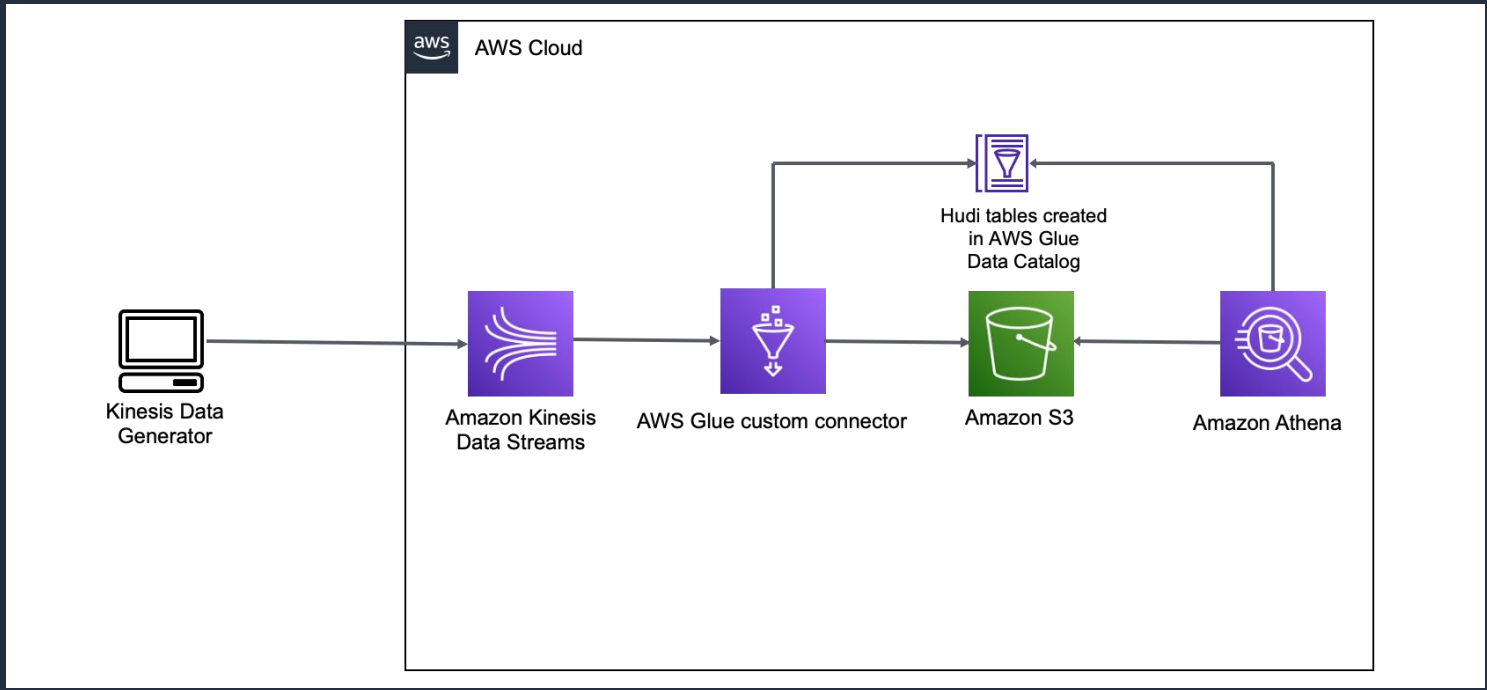
- マネージドな Zeppelin ノートブックを使用して、アドホックなトピックのデータ分析が可能
- 作成した分析ロジックを Flink アプリケーションとして保存、本番へデプロイすることも可能
- Java, Scala, Python, SQL など複数言語に対応



https://aws.amazon.com/jp/blogs/news/introducing-amazon-kinesis-data-analytics-studio-quickly-interact-with-streaming-data-using-sql-python-or-scala/

AWS Glue Streaming ETL ジョブによるデータ処理

- AWS Glue ではストリームデータを継続的に処理するための Spark Streaming ジョブタイプを使用可能
- Glue がネイティブサポートしている Apache Hudi などのデータレイクストレージフレームワークと連携させることで、データレイクに対するニアリアルタイムな増分データ処理も実装可能



https://docs.aws.amazon.com/ja_jp/glue/latest/dg/add-job-streaming.html
https://aws.amazon.com/jp/blogs/big-data/build-a-serverless-pipeline-to-analyze-streaming-data-using-aws-glue-apache-hudi-and-amazon-s3/

Consumer の選定基準

既存アプリケーションに組み込む場合

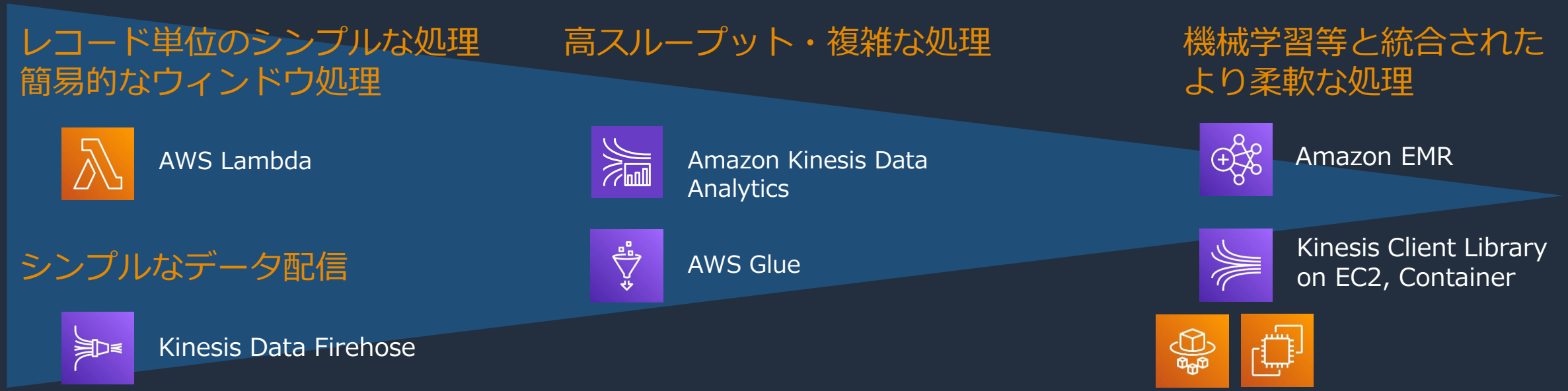
- Kinesis Client Library の利用を検討

新規に Consumer アプリケーションを作成する場合

- 用途(データ配信とデータ処理のどちらがメインか)
- 連携サービス(Amazon Kinesis のみか、それ以外との連携もあるか)
- 処理の複雑性(単純なデータ配信か、それとも複雑な集計が求められるか)
- 想定スループット(求められる処理性能はどの程度か)
- レイテンシ要件(どの程度の遅延までなら許容可能か)
- 開発生産性(使い慣れた言語、フレームワークを使用したいか、
  ノーコード、ローコードで実装したいか)
- メンテナンス性(運用の省力化を重視するか)

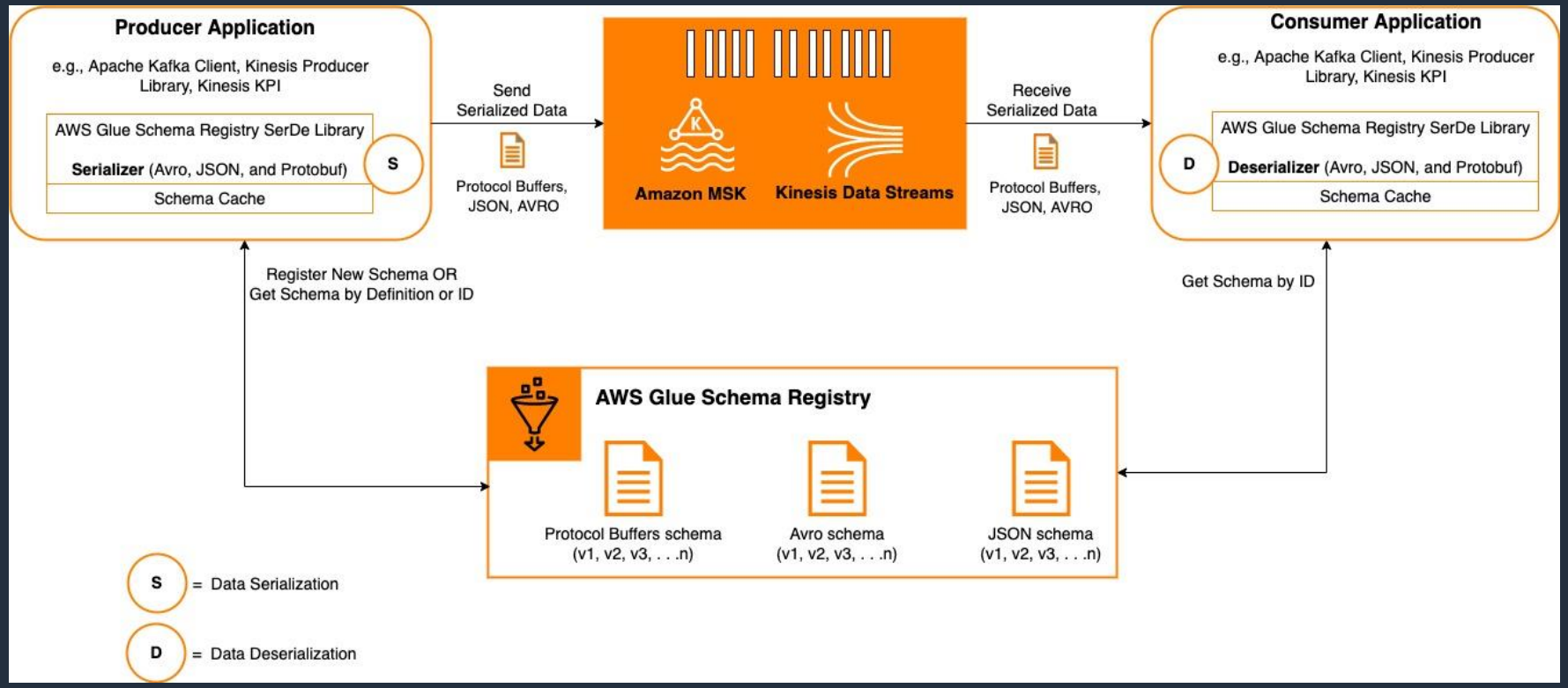
Consumer 選定のポイント

- マネージドサービスの中から、スループット要件、処理要件、利用可能な言語、コスト要件に応じてサービスを絞る
- 処理の複雑さに合わせて、管理負荷が低いものを選択する
  - EC2 やコンテナ、EMR を選択すると Savings Plan 等の恩恵に預かれるが、インフラの運用コストの負担がある。この辺りはトレードオフとなる



Glue Schema Registry によるスキーマ管理

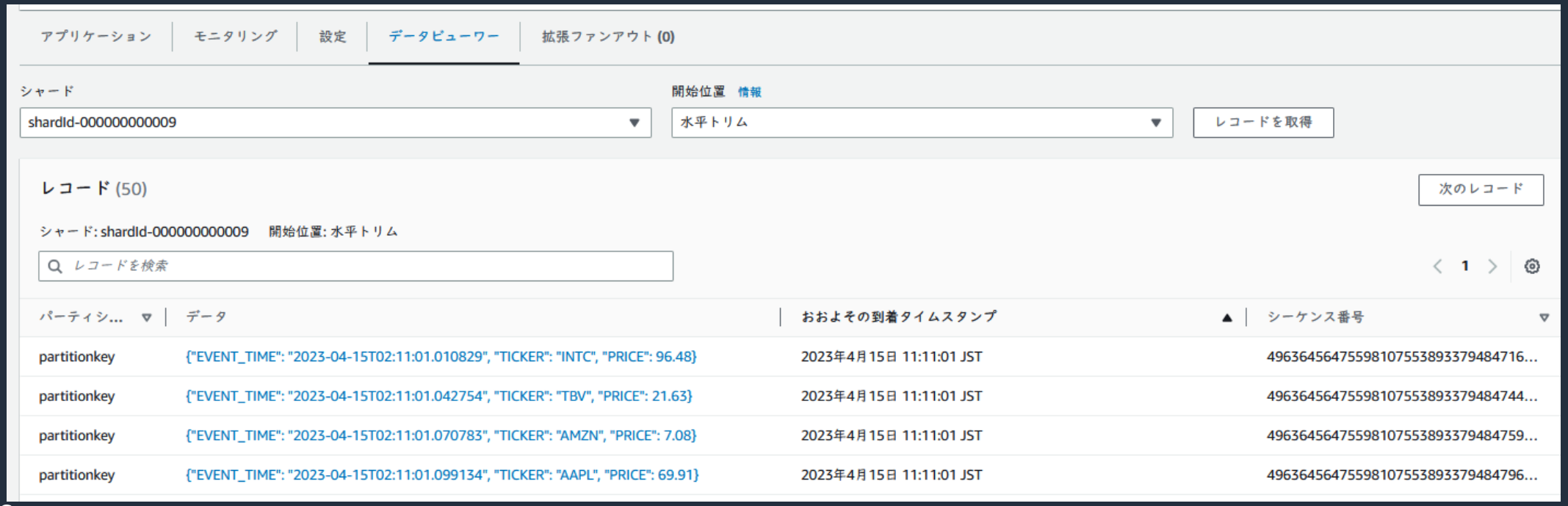
- ストリーミング処理におけるスキーマをバージョン管理することで、変更時の変化にも対応
- ライブラリを使用することで透過的なシリアライズ、デシリアライズを行うことが可能
- Protobuf, Avro, JSON フォーマットがサポートされている



https://aws.amazon.com/jp/blogs/big-data/introducing-protocol-buffers-protobuf-schema-support-in-amazon-glue-schema-registry/
https://github.com/awslabs/aws-glue-schema-registry

Data Viewer によるレコード閲覧

- Kinesis Data Streams マネジメントコンソールよりレコードの閲覧が可能。開発、デバッグ、トラブルシュートに有用
- GetRecords API を使用しているため、既存の標準コンシューマーアプリケーションとコールレートを共有することになる。使用時のスロットリングに注意



キャパシティモード

オンデマンドモードとプロビジョンドモード

- 要件に応じた 2 つのモードが提供されている
- オートスケールの有無、料金モデル、ストリームあたりのシャード数上限が異なる
- モードはオンラインで切り替え可能だが、24 時間に 2 回までの切り替え制限あり

https://docs.aws.amazon.com/ja_jp/streams/latest/dev/how-do-i-size-a-stream.html

モードの選定基準

当初はオンデマンドで運用を開始し、トラフィックが安定したら
プロビジョンドモードに切り替えるといった方法も有る

オンデマンドモードの利用を検討するケース

- イニシャルのスループット見積もりを行わずに、すぐに利用を開始したい
- 予測不可能、不安定なワークロードのため、オートスケールを活用したい

プロビジョンドモードの利用を検討するケース

- 予測可能で安定したワークロードであり、手動スケールで問題なく運用できる
- 1000 シャード以上の大規模なストリームを運用したい

どちらがコスト最適かは試算、もしくは実際の請求をベースに判断

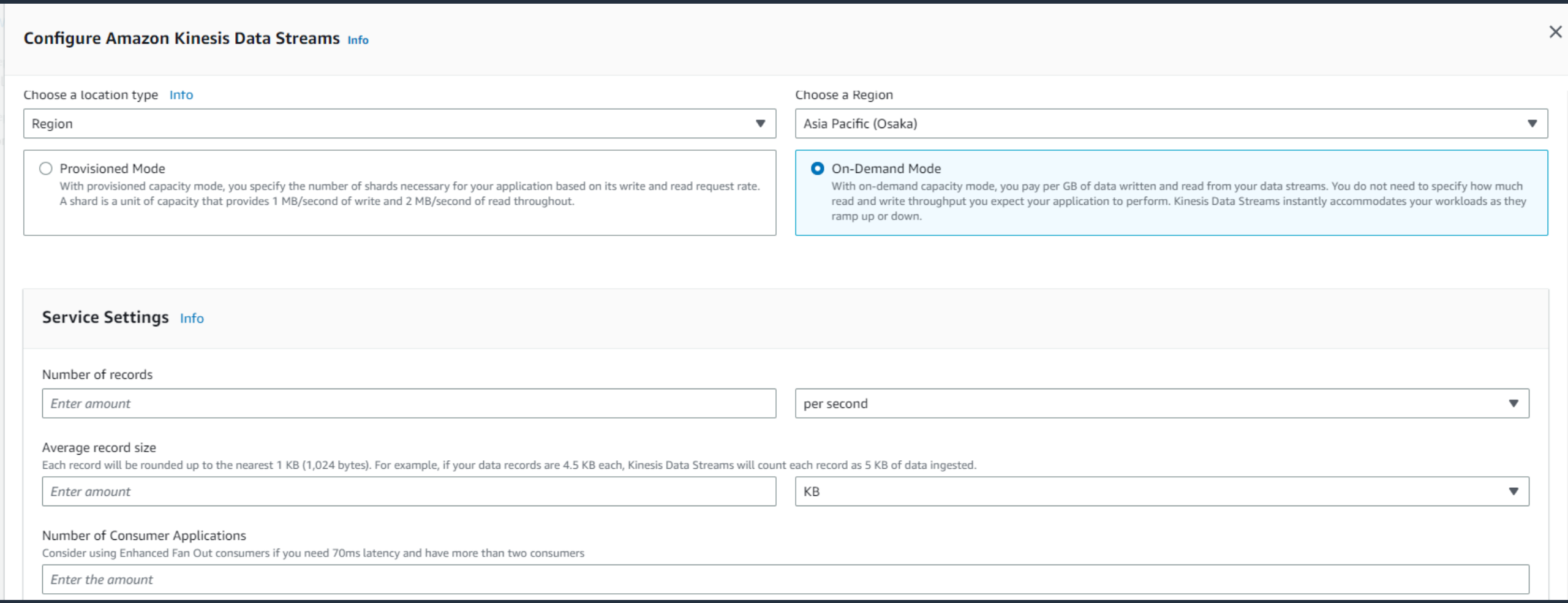
料金比較(東京リージョン)

	オンデマンド	プロビジョンド
リソースに対する課金	$ 0.052/hour per stream	$ 0.0195/hour per shard
書き込みに対する課金	$ 0.104/GB (レコードサイズは 1KB 単位で切り上げ)	$ 0.0215/1M payload unit (1 payload unit = 25 KB. レコードサイズは 25 KB 単位で切り上げ)
読み取りに対する課金	$ 0.052/GB (サイズ切り上げ無し)	課金なし
データ長期保存	無料 (24 時間まで) $ 0.12/GB per month (7 日間まで) $ 0.025/GB per month(7 日目以降)	無料 (24 時間まで) $ 0.026/hour per shard (7 日間まで) $ 0.025/GB per month (7 日目以降)
長期保存データの取り出し	追加の課金なし (上の”読み取りに対する課金”は発生)	$ 0.0273/GB
拡張ファンアウトリソースに対する課金	課金なし	$ 0.0195/hour per shard
拡張ファンアウトによるデータの取り出し	$ 0.065/GB (サイズ切り上げ無し)	$ 0.0169/GB (サイズ切り上げ無し)

https://aws.amazon.com/jp/kinesis/data-streams/pricing/

Pricing Calculator による試算

オンデマンド、プロビジョンド 2 つのモードでの試算をサポート



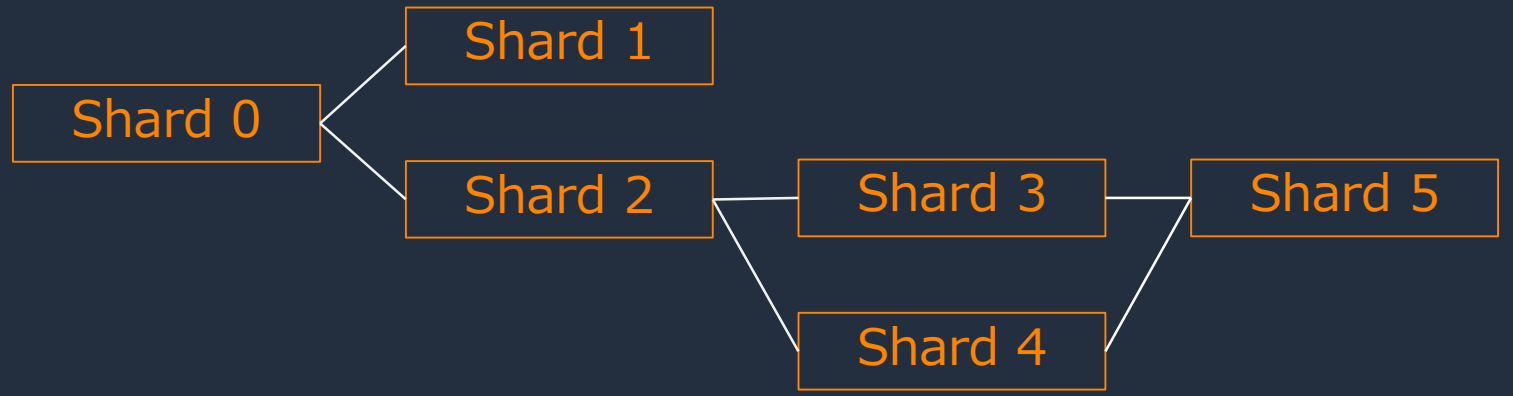
https://calculator.aws/#/addService/KinesisDataStreams

拡張性

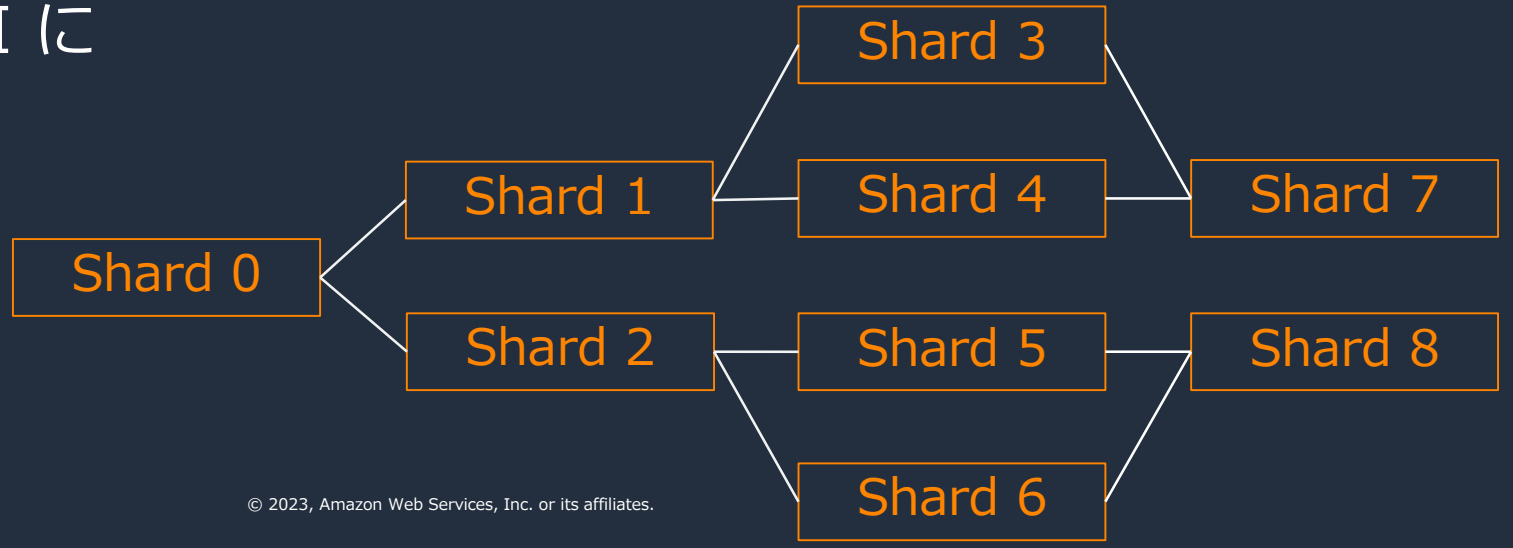
シャード数の変更

シャードの増減はオンラインで実行可能。 2 つの方式をサポート

Split/Merge API による特定シャードの分割、統合



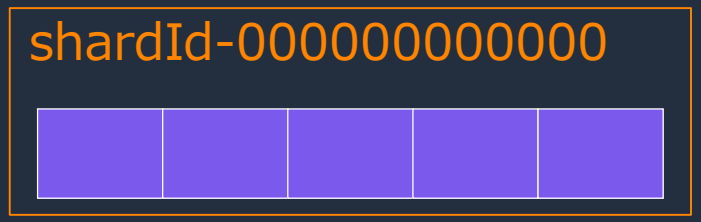
UpdateShardCount API による、ストリーム全体のシャード数の変更



Split API によるシャードの分割

- Split API を実行すると、

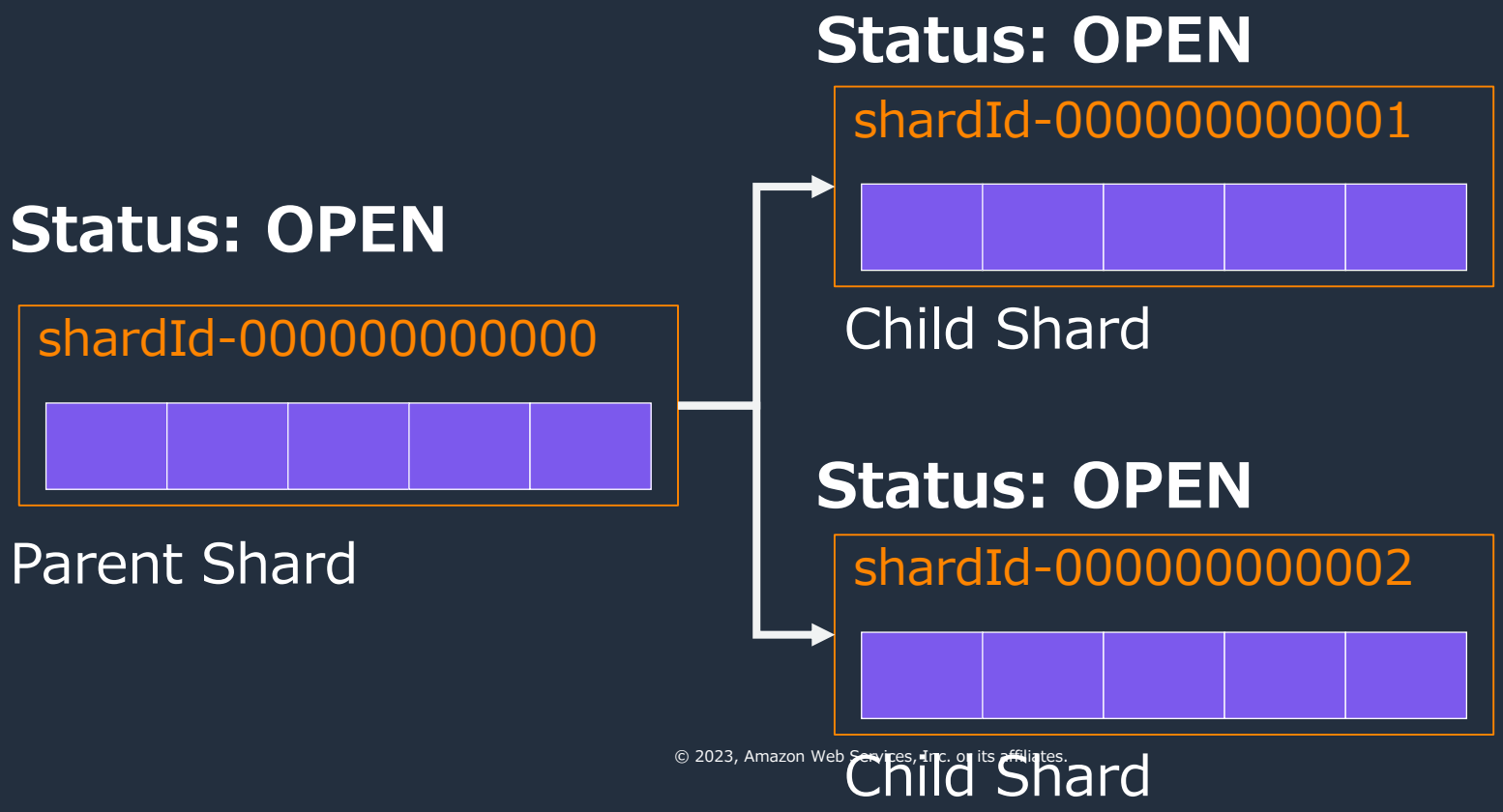
Status: OPEN



Parent Shard

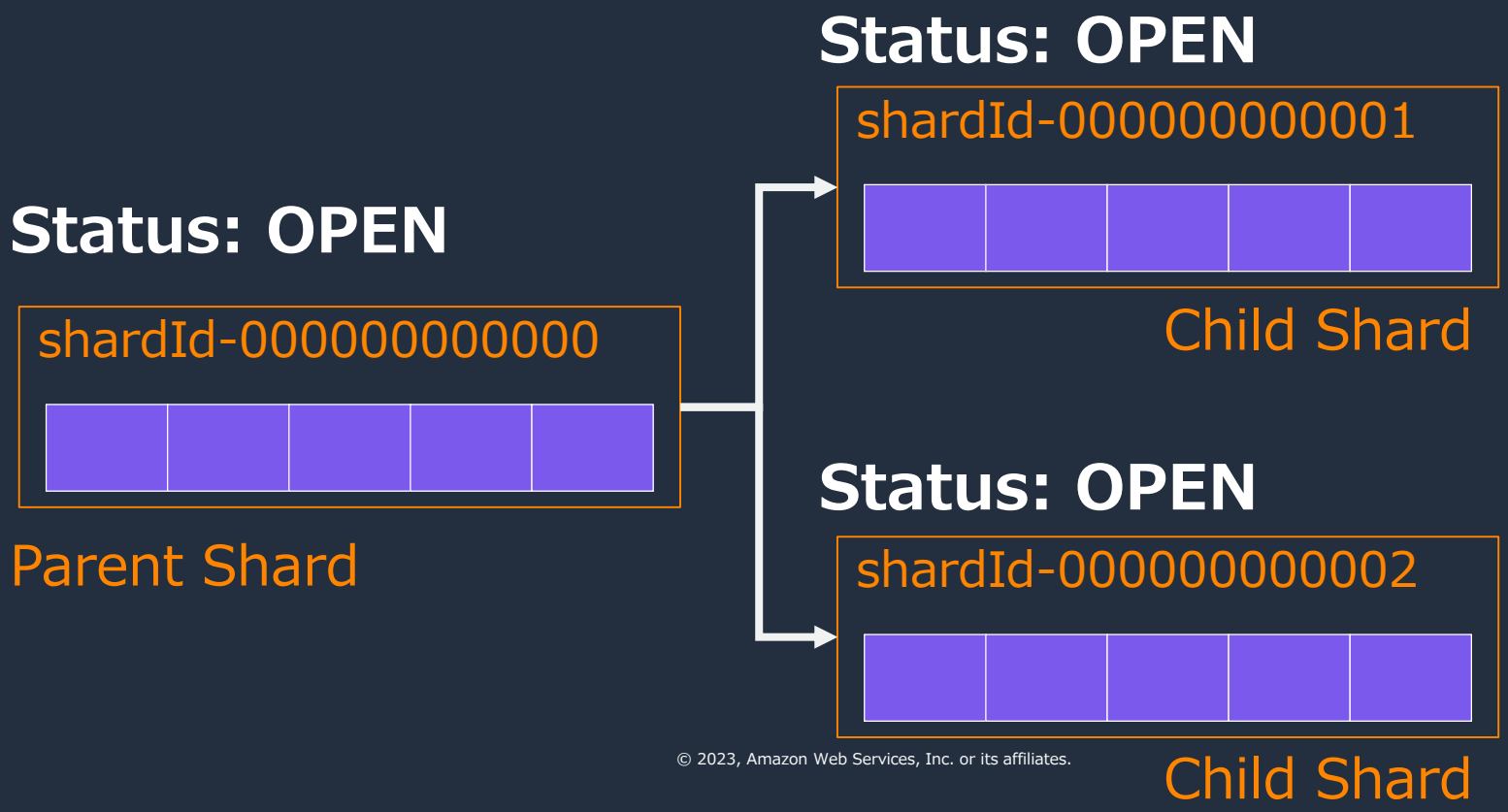
Split API によるシャードの分割

- Split API を実行すると、特定のシャードが 2 つのシャードに分割される



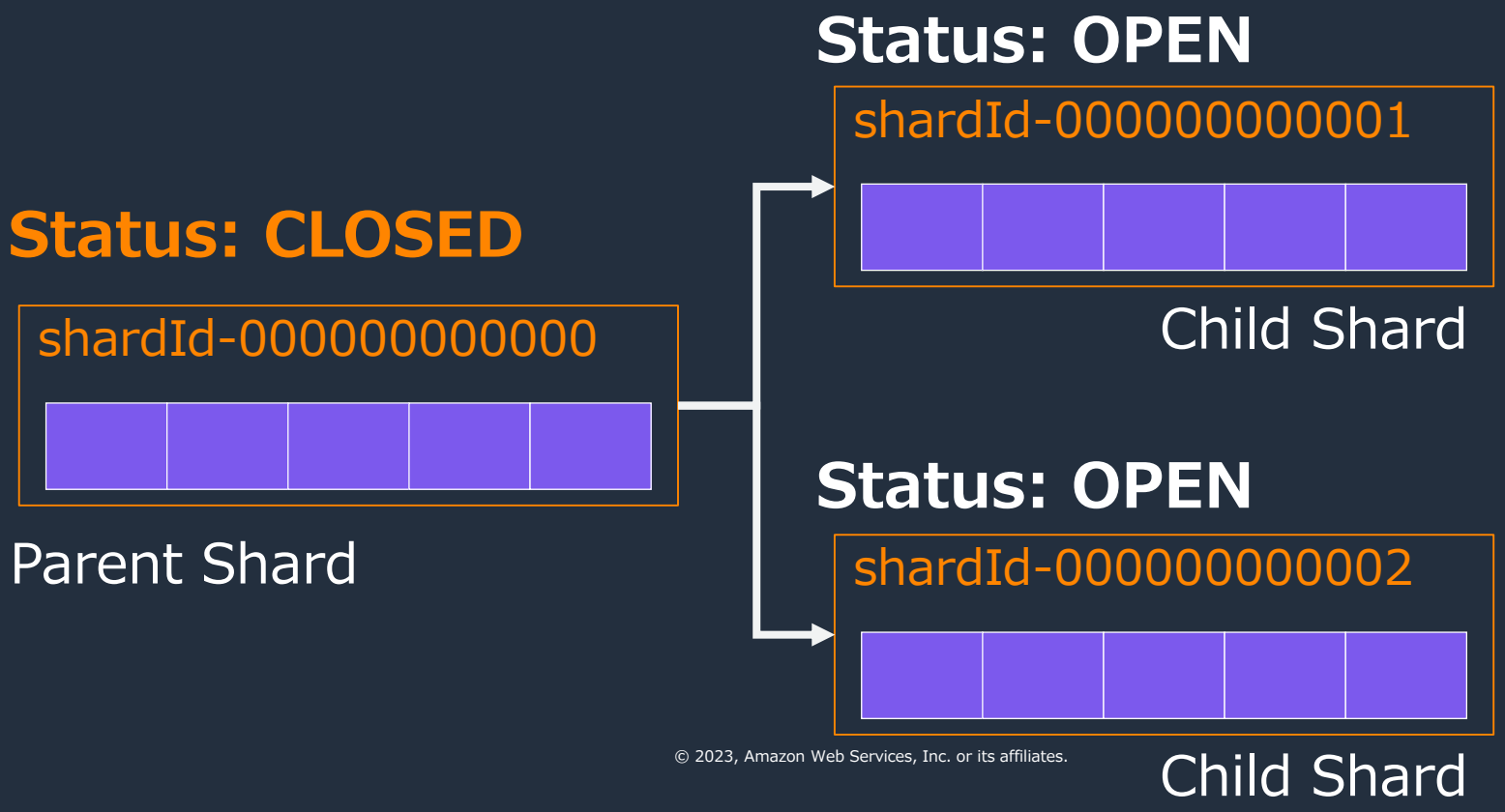
Split API によるシャードの分割

- Split API を実行すると、特定のシャードが 2 つのシャードに分割される
- 分割されたシャードには親子関係が生じる



Split API によるシャードの分割

- Split API を実行すると、特定のシャードが 2 つのシャードに分割される
- 分割されたシャードには親子関係が生じる
- 分割が完了すると親シャードのステータスは CLOSED に変化する。
  CLOSED になるとデータレコードの送信はできないが、取得は引き続き可能



ハッシュキーレンジの分割

Split API によりシャードが分割されると、ハッシュキーレンジも分割される

```
$ aws kinesis split-shard --stream-name sample --
shard-to-split shardId-000000000000 --new-starting-
hash-key
170141183460469231731687303715884105727
```

shardId-000000000000

StartingHashKey: 0
EndingHashKey:
340282366920938463463374607431768211455

shardId-000000000001

StartingHashKey: 0
EndingHashKey:
170141183460469231731687303715884105726

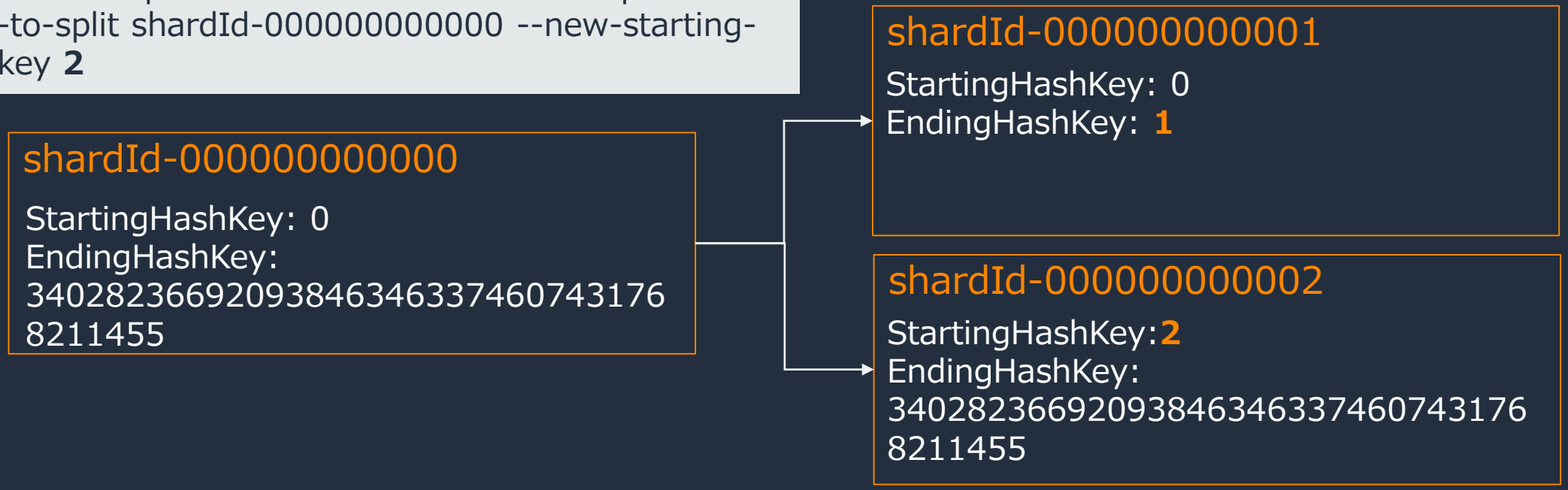
shardId-000000000002

StartingHashKey:
170141183460469231731687303715884105727
EndingHashKey:
340282366920938463463374607431768211455

ハッシュキーレンジの分割

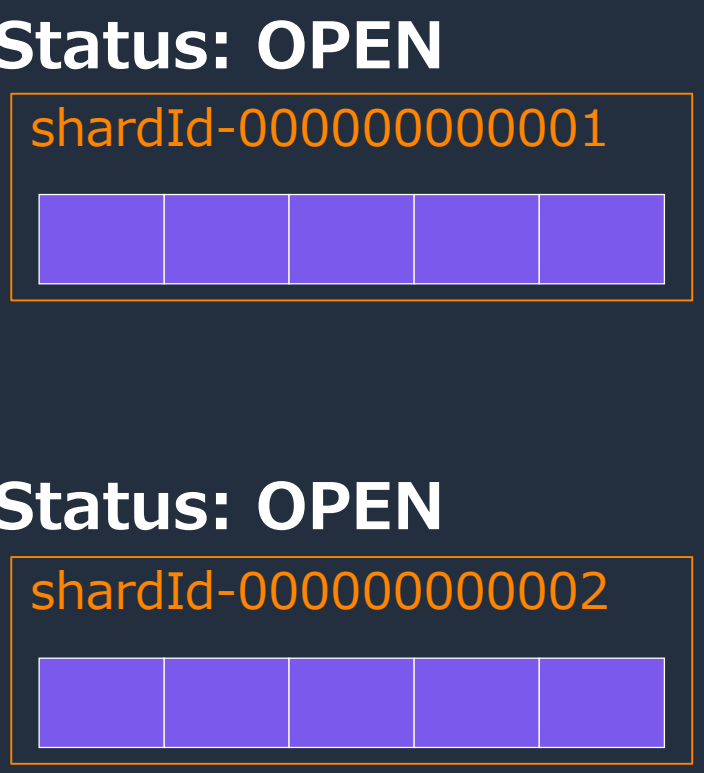
- 極端なレンジでの分割も可能だが、シャードごとのデータ流量のバランスが崩れるため、非推奨
- パーティションキー設計上特定のシャードにデータが偏ってしまった場合の回避策としては機能するが、パーティションキーの再検討は行った方がよい

```
$ aws kinesis split-shard --stream-name sample --
shard-to-split shardId-000000000000 --new-starting-
hash-key 2
```



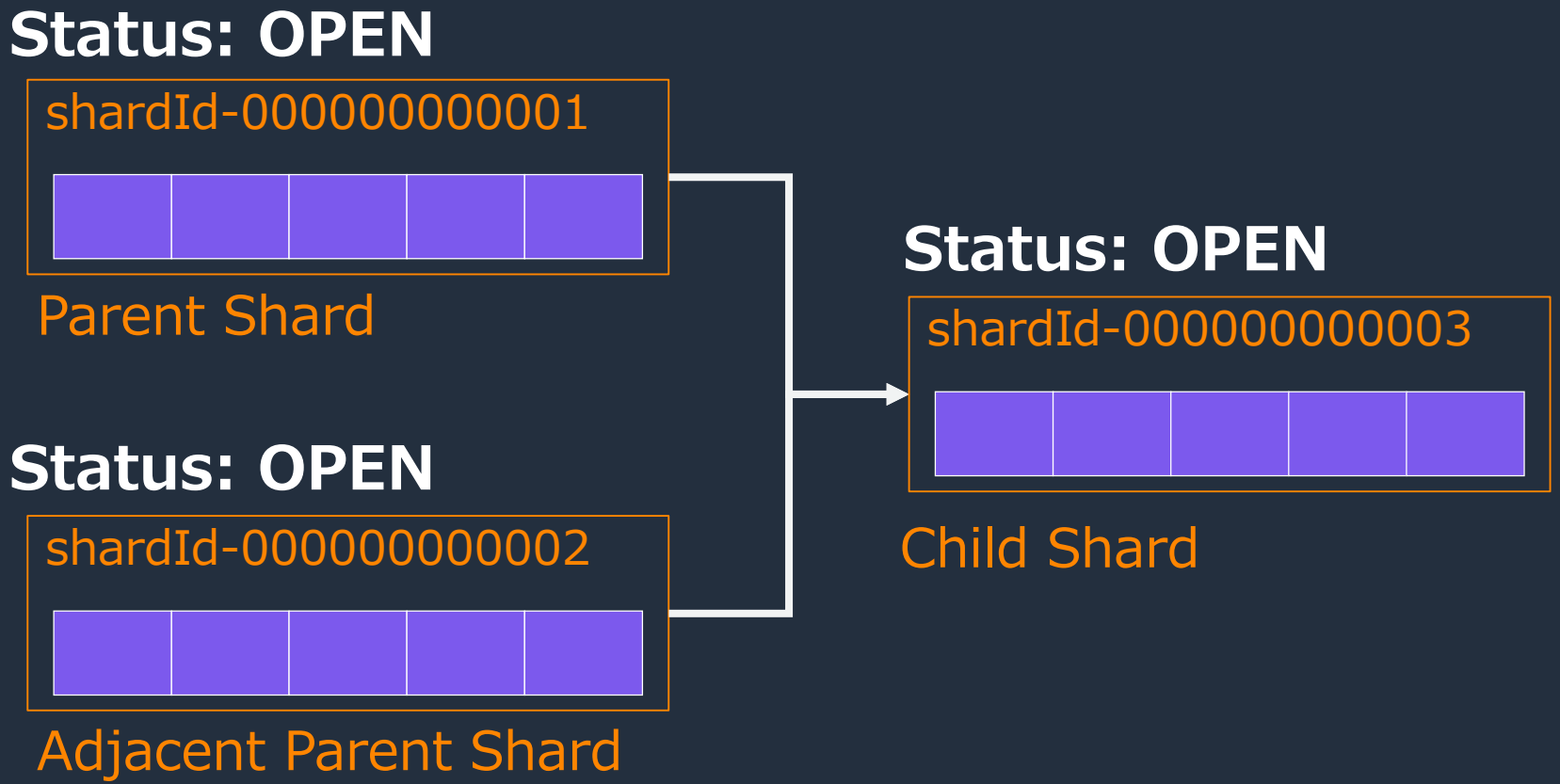
Merge API によるシャードの統合

- Merge API を実行すると、



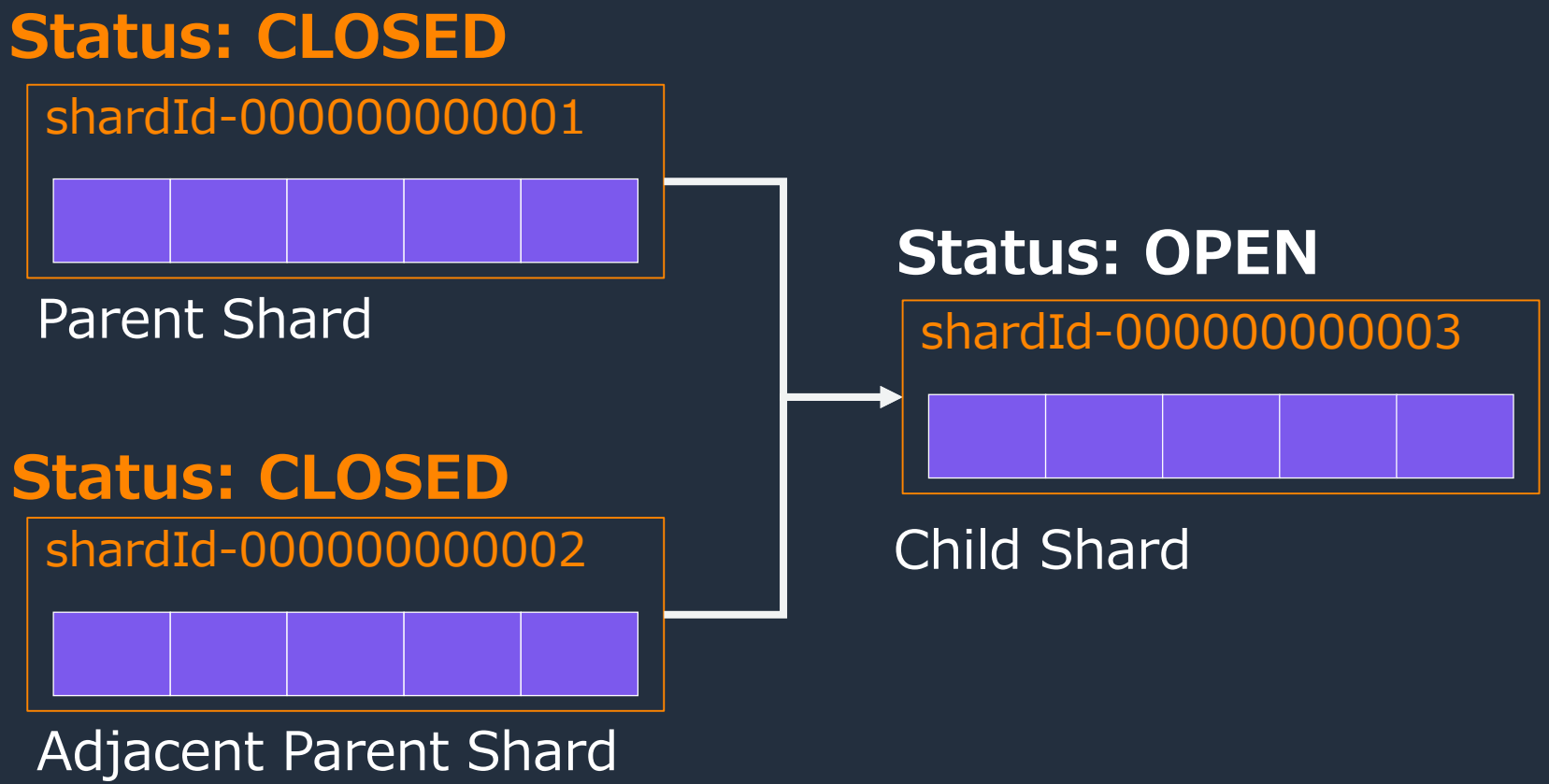
Merge API によるシャードの統合

- Merge API を実行すると、指定した 2 つのシャードが 1 つのシャードに統合される
- シャードの親子関係や親シャードが CLOSED に遷移するなどの動作は Split API と同様



Merge API によるシャードの統合

- Merge API を実行すると、指定した 2 つのシャードが 1 つのシャードに統合される
- シャードの親子関係や親シャードが CLOSED に遷移するなどの動作は Split API と同様



Merge API の実行条件

ハッシュキーレンジが隣接するシャード同士のみ統合可能

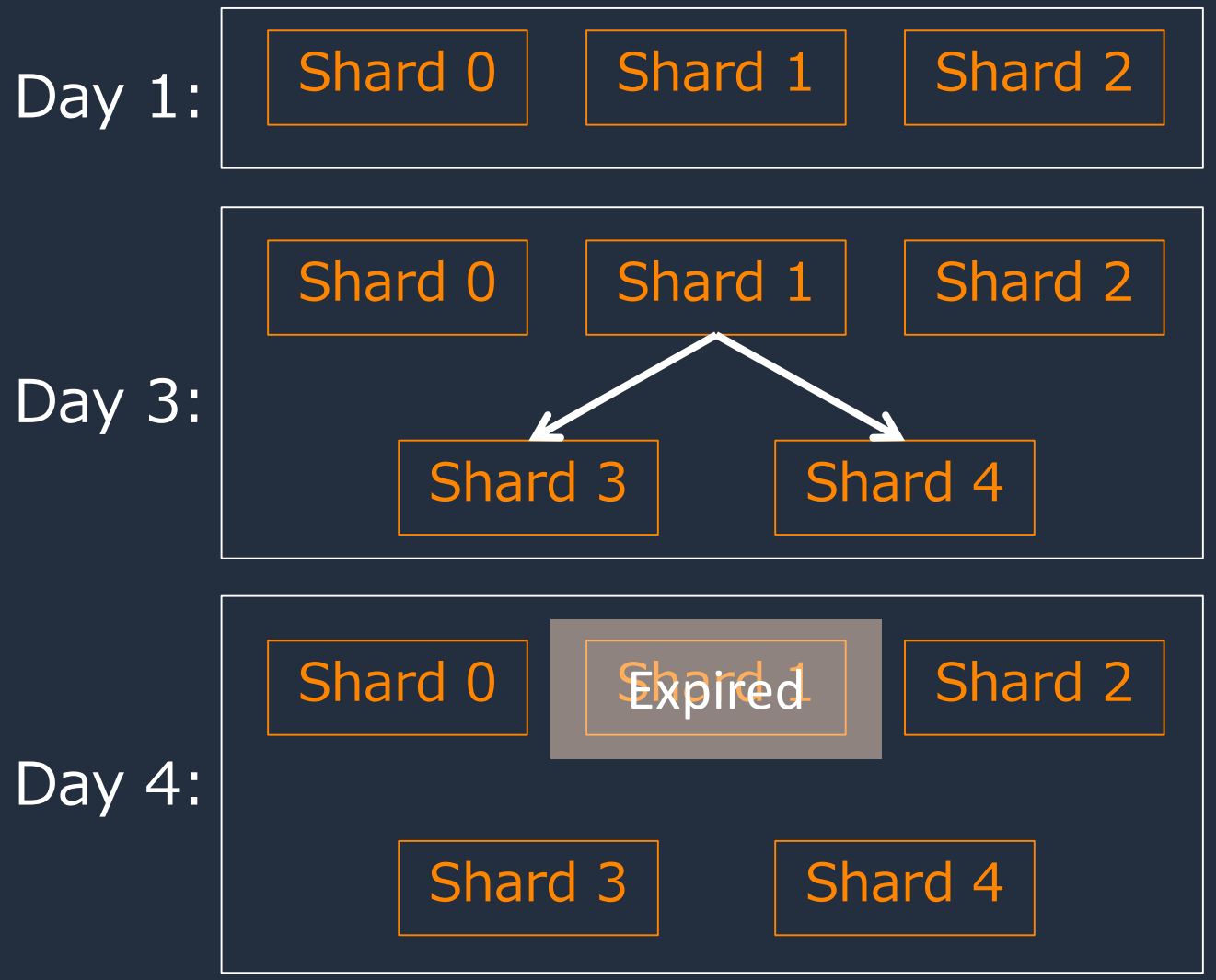
Merge API によるシャードの統合

隣接していないシャード同士の統合は不可能

親シャードの削除タイミング

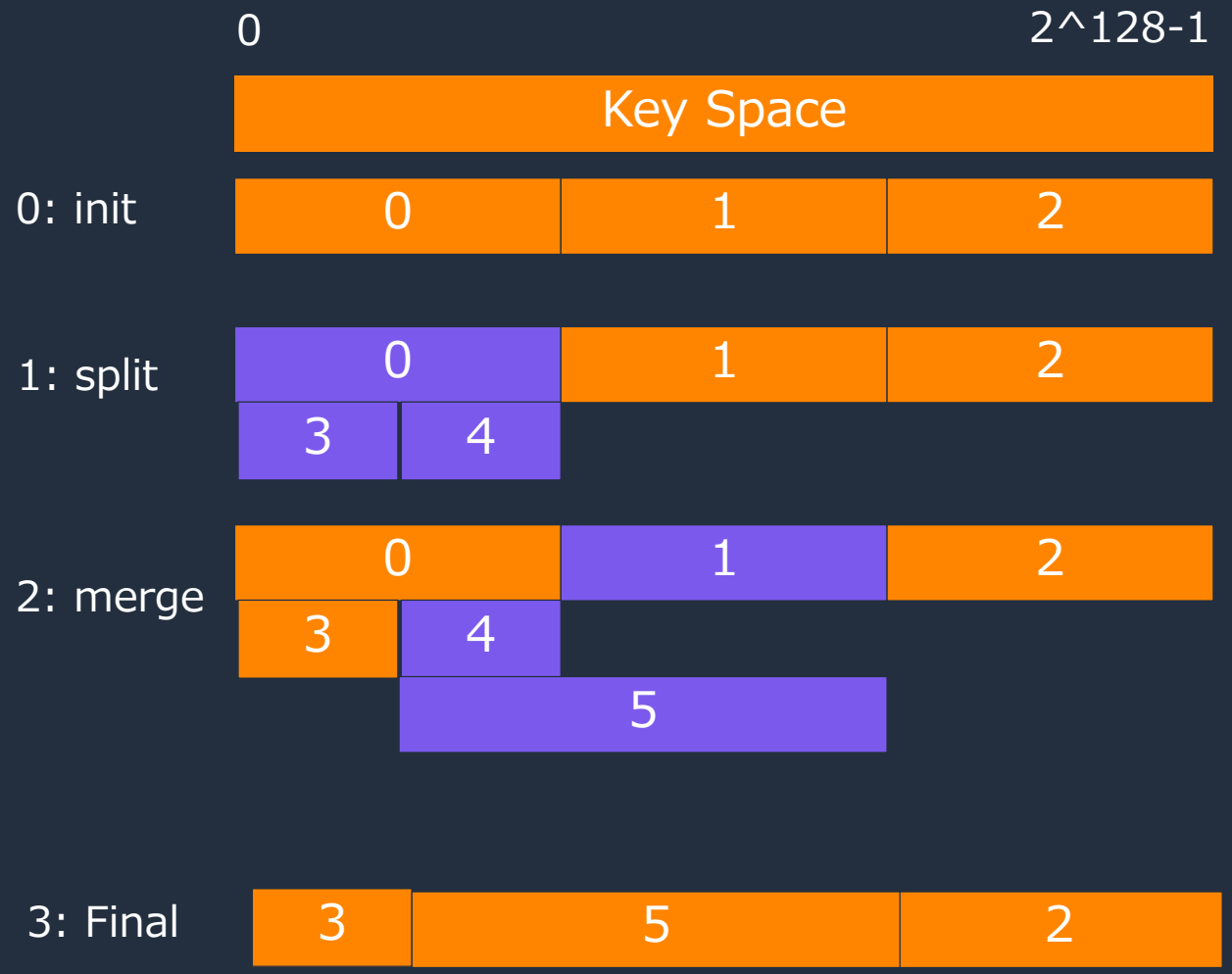
データ保持期限: 24 h

- 親シャード内のデータレコードが全て保持期間を超過すると、親シャードのステータスは EXPIRED となる
- EXPIRED シャードに対するデータ送信、取得は不可能となる。ListShards の結果からも削除される
- KCL や Lambda などのライブラリ、サービスを使っている場合は、EXPIRED シャードを除外する仕組みが備わっているため、アプリケーション実装時にこれらを考慮する必要はない



Split/Merge API の課題

- Merge 実行時に隣接しているシャードを確認する必要がある
- Split 実行時のハッシュキーレンジ計算が煩雑
- 適切にハッシュキーレンジを指定しないで Split/Merge を繰り返すと、シャード間のハッシュキーレンジの長さが不一致となるリスクがある

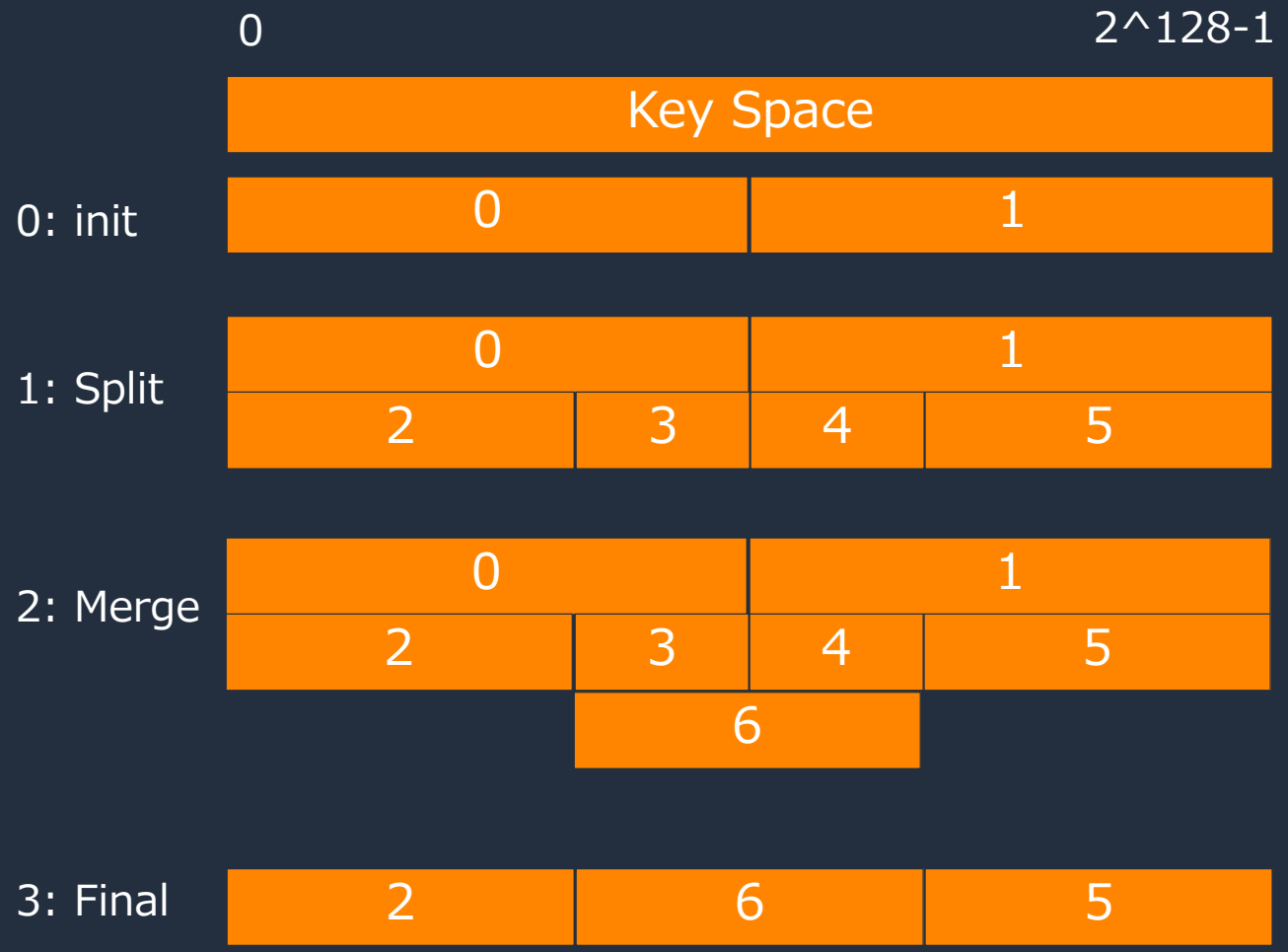


UpdateShardCount API によるシャード数の変更

- 指定したシャード数を
ターゲットとして、
ストリーム内のシャードを
自動でリバランス

- 現在のシャード数と
ターゲットのシャード数を
元に、Split と Merge を
繰り返し実行

- シャードごとのキーレンジは
均等になる



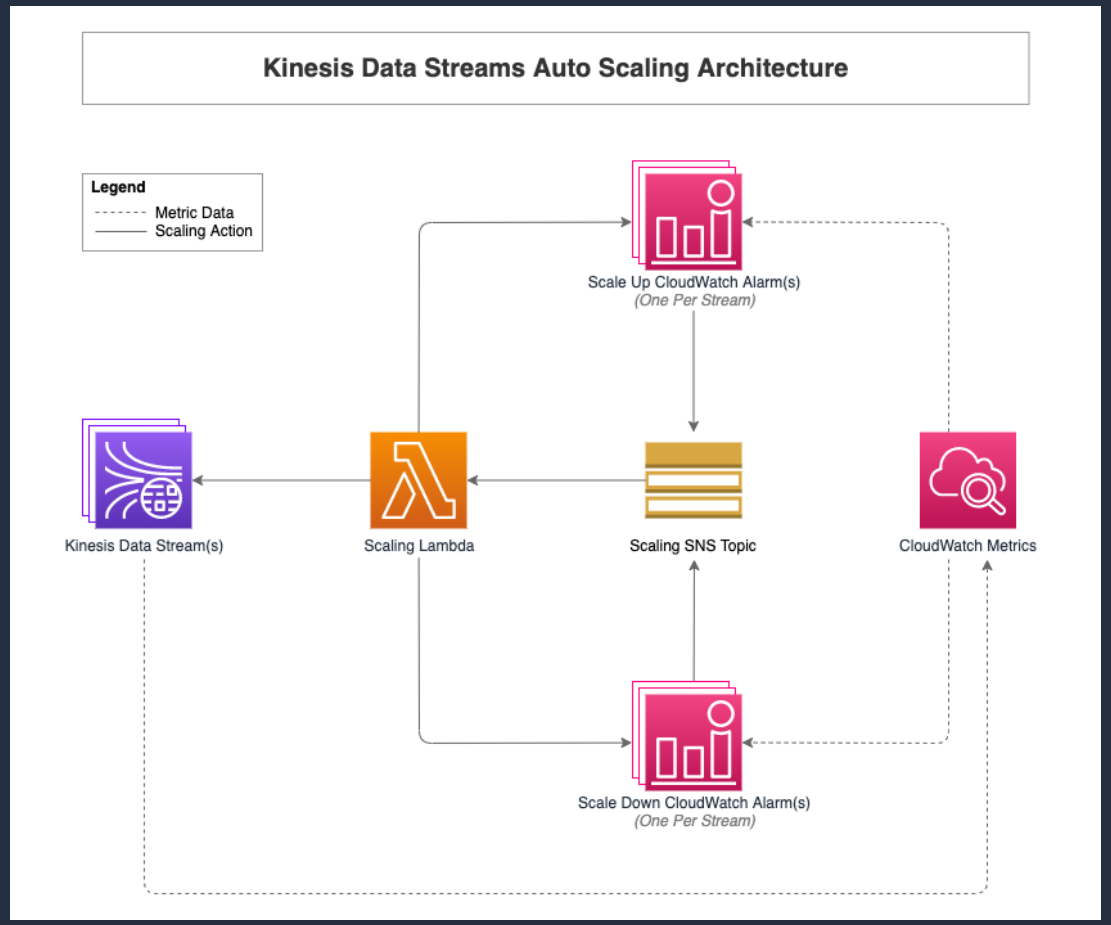
UpdateShardCount API の制限

- 24 時間ごとに 10 回までの実行回数制限あり
- スケールの上限は現在のシャード数の 2 倍、もしくは 10,000 シャードまで
  - 現在のシャード数が 2 シャードの場合、引き上げ可能なシャード数の最大値は 4
  - 現在のシャード数が 7,000 シャードの場合、引き上げ可能なシャード数の最大値は 10,000
- スケールの下限は現在のシャード数の半分まで。例えば、現在のシャード数が 4 シャードの場合は、引き下げ可能なシャード数の最低値は 2 となる
- アカウント、リージョンごとに設定されたシャード数の制限を超えてのスケールは不可能。
  - Service Quotas またはサポートケースから上限緩和申請を行うこと

Kinesis Auto Scaling

- Amazon CloudWatch のアラームから Amazon SNS を経由して AWS Lambda を呼び出すことで自動的なスケールを実装

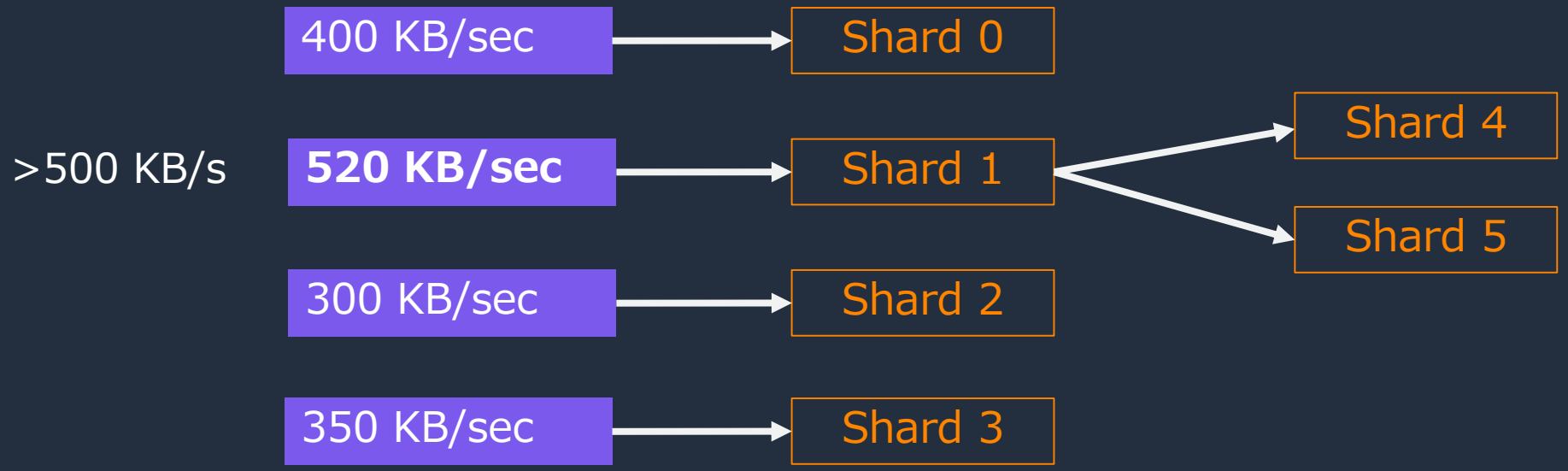
- プロビジョンドモードで
オートスケールを行う際に有用



https://aws.amazon.com/jp/blogs/big-data/auto-scaling-amazon-kinesis-data-streams-using-amazon-cloudwatch-and-aws-lambda/
https://github.com/aws-samples/kinesis-auto-scaling

オンデマンドモードにおけるオートスケーリング

- 新規に作成されたストリームの初期シャード数は 4となる。
  初期スループットは書き込み 4 MB/s、読み取り 8 MB/s となる
- 各シャードごとに書き込みのスループットが 50% (500 KB/s or 500 records/s) を超えるとシャードの分割が実行される
- デフォルトで最大で 200 シャードまで、上限緩和により 1000 シャードまでスケールする



オンデマンドモードにおけるオートスケーリング > 補足

- Provisioned mode と同様パーティションキー設計を適切に行う必要が有る。
  シャードの分割時に指定される Hash Key は Hash Range の中央値。
  これはシャードは均等に分割されることを意味する。データが特定の
  Hash Key に偏って書き込まれるケースだと性能上問題が出る場合も

- 一般的なシャード数であればスケールは 15 分以内に完了する

- リトライ処理はプロビジョンドモードと同様必要。
  過去 30 日間のトラフィック使用量の 2 倍のトラフィックをさばけるように
  スケールするため、過去の 2倍以上のトラフィックが突発的に発生した場合は、
  スケールが完了するまでにスロットリングエラーが発生する可能性がある

オンデマンドモードにおけるオートスケーリング > 補足

- 読み取りトラフィックに応じたスケールは行われない。
  3 つ以上のコンシューマーアプリケーションが存在する場合は
  拡張ファンアウトの利用を検討すること

- ユーザー側で任意のスループットを指定することはできない。
  任意のスループットで利用したい場合はプロビジョンドモードを利用すること

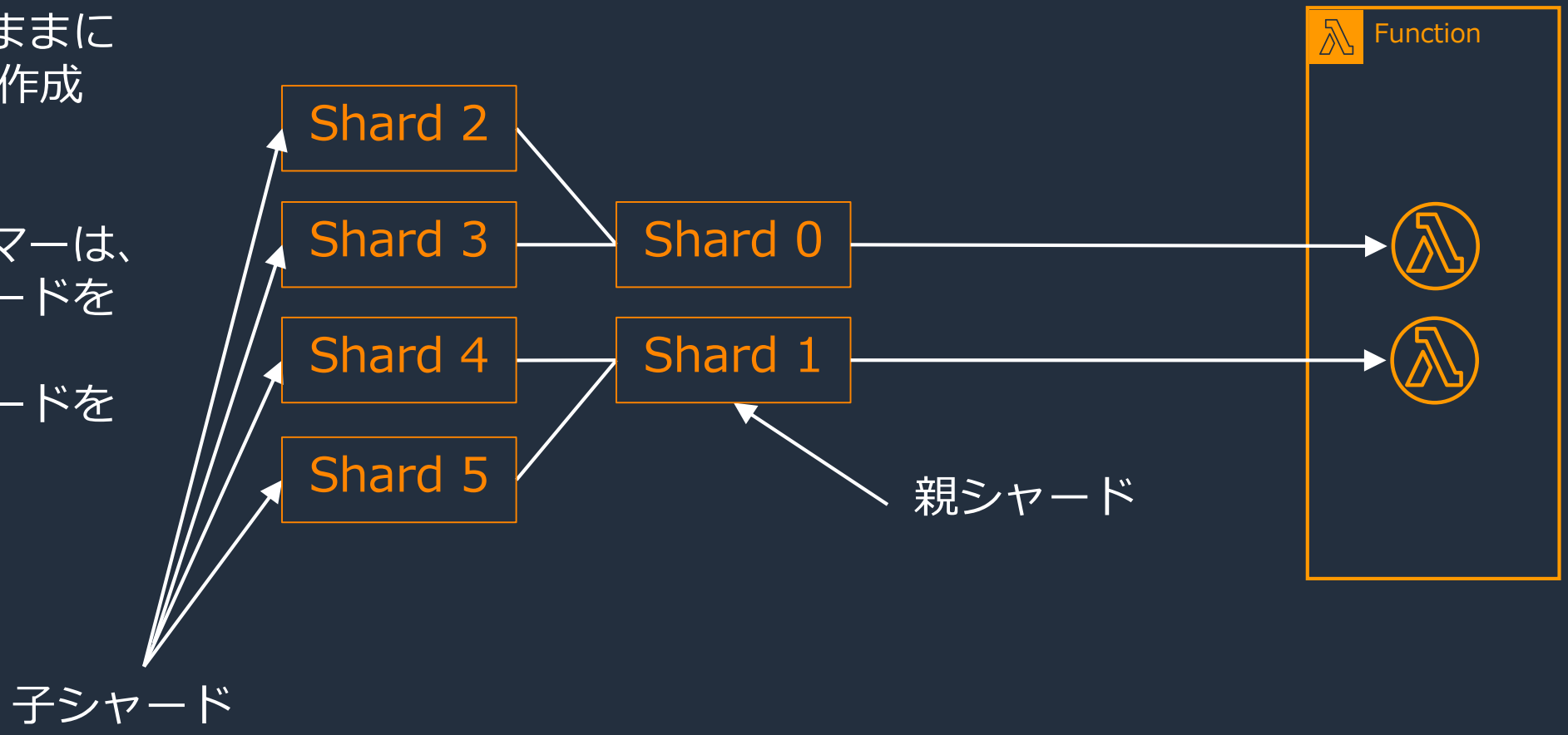
- プロビジョンドモードとオンデマンドモードの切り替えは 24 時間に 2 回まで

- プロビジョンドモードからオンデマンドモードに切り替えた場合、
  切り替え前のシャード数と書き込みスループットに応じて、即時に
  スケーリングが行われる。オンデマンドモードからプロビジョンドモードに
  切り替えた場合は、切り替え前のシャード数が維持される

シャード数変更時の考慮事項

シャード数を変更しても, アプリケーションのスループットは即座に変化しない

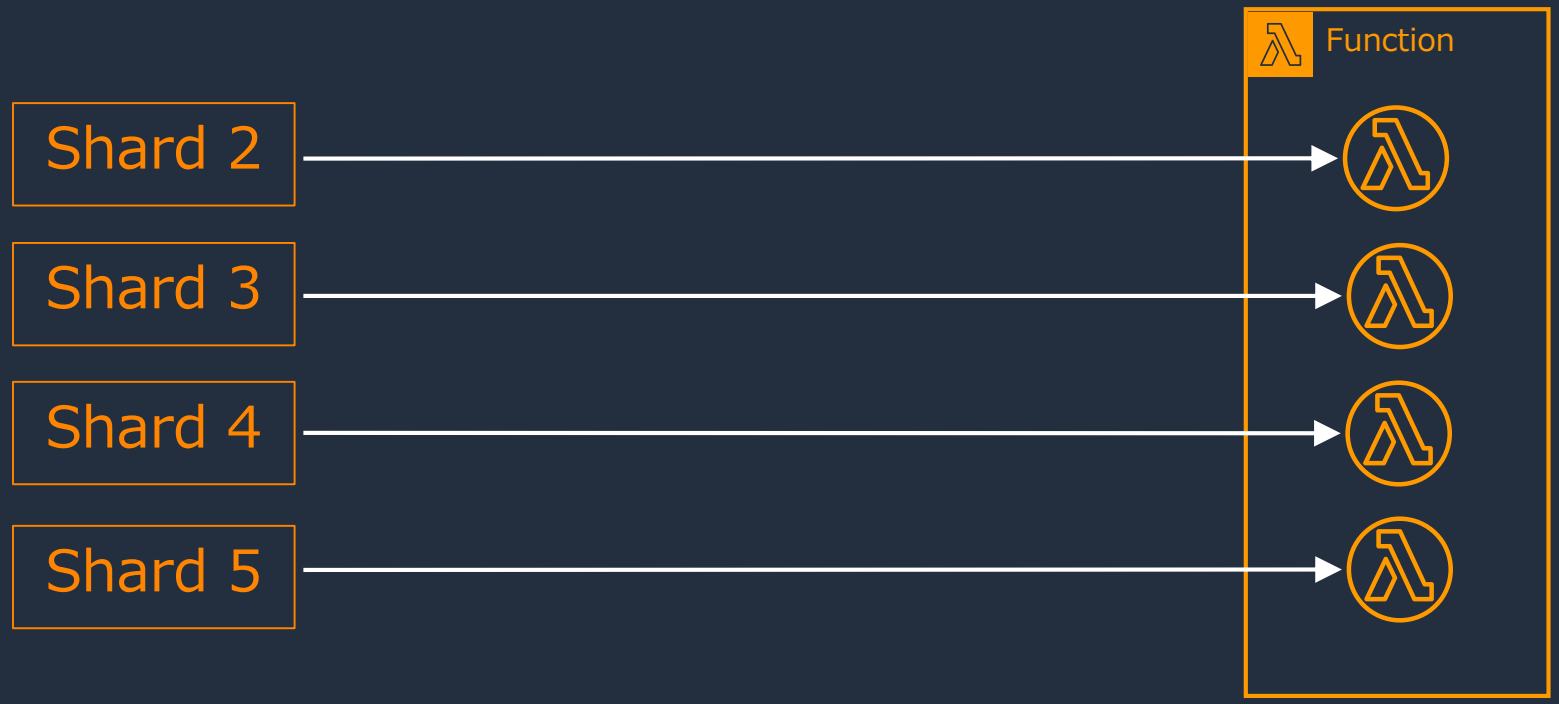
- シャードを分割すると、親シャードはそのままに子シャードが 2 つ作成される
- 通常のコンシューマーは、親シャードのレコードを全て処理するまで、子シャードのレコードを取得しない



シャード数変更時の考慮事項

シャード数を変更しても, アプリケーションのスループットは即座に変化しない

- 親シャード内のレコードを全て取得し終えるか、親シャードが EXPIRED となると、クライアントは子シャードからレコードを取得し始める
- KCL や Lambda などで実装されたコンシューマーは、親シャードの処理完了後子シャードの数に合わせてスケールする形となる



セキュリティ

サーバーサイド暗号化

- AWS KMS と連携したサーバーサイド暗号化の機能を提供している
- 暗号化を有効化した際の追加レイテンシは 100 マイクロ秒以下
- 有効化した場合は、プロデューサー、コンシューマーがストリームに対してレコードの送信・取得を行うために、IAM ポリシーで CMK へのアクセスを許可する必要が有る

プロデューサーのサンプルポリシー

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

https://docs.aws.amazon.com/streams/latest/dev/permissions-user-key-KMS.html

暗号化の有効化、無効化

- オンラインで有効化、無効化、キーの変更が可能
- 数秒～数分程度で設定変更は反映されるが、ストリーム内の全てのレコードに対する暗号化、もしくは暗号化の解除を行うために、追加で最大 5 秒程度時間を要す場合がある
- PUT したレコードに対する SSE (Server Side Encryption) が有効化されていることは、PutRecord, PutRecords, GetRecords の実行結果から判断できる
- 暗号化にかかる設定変更は 24 時間につき 25 回までの制限あり

```
$ aws kinesis put-record --stream-arn arn:aws:kinesis:ap-northeast-
1:123456789012:stream/ExampleInputStream --data Y2F0Cg== --partition-key 1
{
  "ShardId": "shardId-000000000009",
  "SequenceNumber": "49636456475598107553893232849861586131003170243548807314",
  "EncryptionType": "KMS"
}
```


https://docs.aws.amazon.com/kinesis/latest/APIReference/API_StartStreamEncryption.html

アクセス制御

利用可能なアクセス制御方式(Control API, Data API)

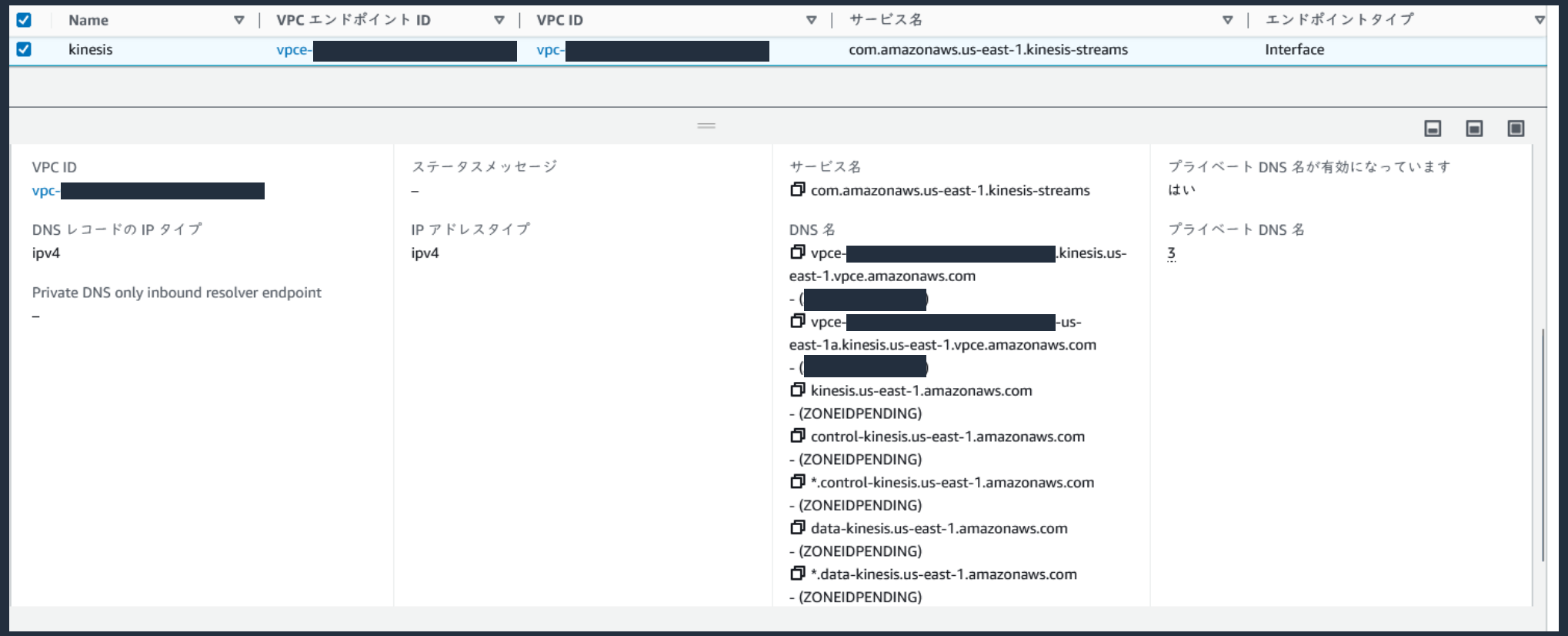
- IAM ポリシー
- VPC エンドポイントポリシー (VPC エンドポイント使用時のみ)

異なるアカウントからのアクセスについて

- Amazon Kinesis Data Streams はリソースベースのアクセスポリシーをサポートしていないため、異なる AWS アカウントの認証情報を使用したクロスアカウントアクセスは不可能
- 異なる AWS アカウントのリソースからアクセスする場合は、AssumeRole でアクセス先の AWS アカウントの権限を引き受けるなどの対応が必要

VPC エンドポイント

- 特定 VPC 、または Direct Connect 等を経由して VPC に接続されたプライベート NWから Kinesis Data Streams に安全にアクセスするためのエンドポイントを作成可能
- control と data それぞれのエンドポイントに対応。エンドポイントポリシーもサポート



https://docs.aws.amazon.com/ja_jp/streams/latest/dev/vpc.html

CloudTrail による API コールの記録

- CloudTrail による監査に対応
- ストリーム作成などの
  コントロール API が記録対象
- データ API は記録対象外


```
{
    "eventVersion": "1.01",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::012345678910:user/Alice",
        "accountId": "012345678910",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice"
    },
    "eventTime": "2014-04-19T00:16:31Z",
    "eventSource": "kinesis.amazonaws.com",
    "eventName": "CreateStream",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "127.0.0.1",
    "userAgent": "aws-sdk-java/unknown-version Linux/x.xx",
    "requestParameters": {
        "shardCount": 1,
        "streamName": "GoodStream"
    },
    "responseElements": null,
    "requestID": "db6c59f8-c757-11e3-bc3b-57923b443c1c",
    "eventID": "b7acfcd0-6ca9-4ee1-a3d7-c4e8d420d99b"
}
```


https://docs.aws.amazon.com/ja_jp/streams/latest/dev/logging-using-cloudtrail.html

モニタリング
・パフォーマンストラブルシュート

ダッシュボードのデータ反映が遅れている！

何を調べますか？ どこから調べますか？

Source **Producer** **Data Stream** **Consumer** **Destination**

User Application

Kinesis Producer Library

Amazon Kinesis Data Streams

AWS Lambda

Amazon OpenSearch Service

調査の第一歩

データストリームを起点に調査する

User Application

Kinesis Producer Library

Amazon Kinesis Data Streams

AWS Lambda

Amazon OpenSearch Service

Kinesis Data Streams と CloudWatch Metrics

Kinesis Data Streams は以下 2 レベルのメトリクスを提供

- ストリームレベルのメトリクス (標準)
- シャードレベルのメトリクス (要追加設定、追加料金)

ストリーム自体のエラー発生状況に加えて、コンシューマーの処理遅延などもモニタリングが可能。

標準コンシューマー、拡張ファンアウトで見るべきメトリクスが異なる

https://docs.aws.amazon.com/ja_jp/streams/latest/dev/monitoring-with-cloudwatch.html

データストリームの見るべきポイント

メトリクスを見るだけでなく、全てのシャードが同様のトレンドを示しているか、特定シャードに偏って異常が見られないかを確認することが重要

メトリクス名	説明	メトリクスの見方
ReadProvisionedThroughputExceeded	単位時間あたりの GetRecords リクエストがシャードの制限(2MB/s, 5 TPS)を超過した回数	本メトリクスの値がストリーム全体で恒常的に 1 以上を記録している場合は, キャパシティの増強または拡張ファンアウトの導入が必要. 特定のシャードで発生している場合は, 特定シャードに書き込みが偏っている可能性がある
WriteProvisionedThroughputExceeded	単位時間あたりの PutRecord, PutRecords リクエストがシャードの制限(1MB/s, 1000 records/s)を超過した回数	本メトリクスの値がストリーム全体で恒常的に 1 以上を記録している場合は, キャパシティの増強が必要. 特定のシャードで発生している場合は, 特定シャードに書き込みが偏っている可能性がある
GetRecords.Success PutRecord.Success PutRecords.Success	ストリームに対する各 API の実行回数と成功回数を記録したもの. 平均(Average)の統計から API の成功率が確認できる	成功率が急激に低下している場合はサービスで問題が発生している可能性がある

https://docs.aws.amazon.com/streams/latest/dev/monitoring-with-cloudwatch.html#kinesis-metrics

データストリームの問題が疑われるケースと対処

WriteProvisionedThroughputExceeded > 1 なら、書き込みがスロットルされている

単純にキャパシティ不足に起因していることが多いが、書き込みが特定シャードに偏っている可能性もある。その場合はクライアントの実装見直しを推奨

ストリーム全体で発生している場合

- 帯域使用量起因でスロットルされている場合は、シャード追加によるストリーム全体のスループットの増強を検討
- Records per Sec 起因でスロットルされている場合もシャード追加が有効だが、帯域使用量が低い場合はクライアント側でレコードを集約することでも対処可能

一部のシャードでのみ発生している場合

- パーティションキー設計を見直し、データが適切に分散されるようにする

データストリームの問題が疑われるケースと対処

ReadProvisionedThroughputExceeded ＞ 1 なら、読み取り処理がスロットルされている

読み取り処理のスロットリングは、3 つ以上のコンシューマーが同一のシャードから
データを取得している場合に起こりやすい

ストリーム全体で発生している場合

- シャード追加によるストリーム全体のスループットの増強
- 拡張ファンアウトを検討

一部のシャードでのみ発生している場合

- プロデューサーが書き込みに使用するパーティションキーの設計を見直し、
  データが適切に分散されるようにする

Consumer の遅延が疑われる場合は？

Consumer で問題が発生しているのか、ターゲットのデータストアで問題が発生しているかを確認する

ターゲットの調査(OpenSearch Service)

リクエストがスロットルされていないか、エラーにより書き込みに失敗していないか確認する

メトリクス名	説明	メトリクスの見方
2xx, 3xx, 4xx, 5xx	各ステータスコードの発生回数	5xx が発生している場合は何らかのサーバーエラーの発生が疑われる。4xx が発生している場合はリクエストの問題、ペイロードサイズの超過、キュー溢れが疑われる
IndexingLatency IndexingRate	ドキュメントの Indexing 処理所要時間 単位時間あたりの Indexing リクエストの統計	IndexingRate が変わらないか、減少しているにも関わらず IndexingLatency が増加傾向にある場合、リソース使用状況を確認し、スケールなどで対処する
ThreadpoolWriteRejected CoordinatingWriteRejected PrimaryWriteRejected ReplicaWriteRejected	書き込みリクエストが拒否された回数	これらのメトリクスが増加している場合、キューあふれ、Indexing Request 用のヒープ領域不足が想定される。恒常的に発生している場合はドメインをスケールさせるか、クライアントのリクエスト方法が妥当か確認したうえで見直しを行う

https://docs.aws.amazon.com/opensearch-service/latest/developerguide/managedomains-cloudwatchmetrics.html
https://docs.aws.amazon.com/opensearch-service/latest/developerguide/handling-errors.html

Consumer の調査(AWS Lambda の場合)

関数内部のエラーであるか、データ起因のエラーであるか、単純に処理に時間がかかっているか、関数呼び出しの同時実行数の上限に達してスロットルされているか、などを切り分ける

メトリクス名	説明	メトリクスの見方
Invocations	Function の実行回数	Function が正しく呼び出されているかをまずは確認する Duration に変化がないものの Invocations が増加している場合は、データ量が増加している可能性がある。オーバーヘッド削減のために、レコードバッチのサイズを増やすことも検討する
IteratorAge	ストリームがレコードを受信後、Poller から Function にイベントを送信するまでの時間	Kinesis Data Streams 側の GetRecords.IteratorAgeMilliseconds(Stream-Level) ないし IteratorAgeMilliseconds(Shard-Level) から遅延が判断できる場合、本メトリクスを合わせて確認することで、どのコンシューマーが遅延しているか判断できる
Duration	Function の実行時間	Duration が増加している場合は Lambda Function 側の問題が疑われる。遅延の原因はロジックの問題、リソース不足、処理データ量の増加、Function 内から呼び出している別サービスの遅延など様々である。CloudWatch Logs に出力されているログの確認や、AWS X-Ray などのトレース分析を活用し、調査を行う
Errors	エラーの発生回数	ロジックの問題、不正データの問題などが疑われる。ログから調査を行い対処する。 不正データ起因で発生している問題については、対象のデータをデッドレターキューに退避して後続処理を実行するなどの対処を検討する
Throttles	Function の呼び出しがスロットルされた回数	関数に対する予約された同時実行数が少ないか、アカウント全体の同時実行数が不足している。 関数の設定変更もしくは上限緩和申請で対処する

https://docs.aws.amazon.com/ja_jp/lambda/latest/dg/monitoring-metrics.html
https://docs.aws.amazon.com/lambda/latest/dg/lambda-troubleshooting.html

データストリームの調査(Kinesis Data Streams)

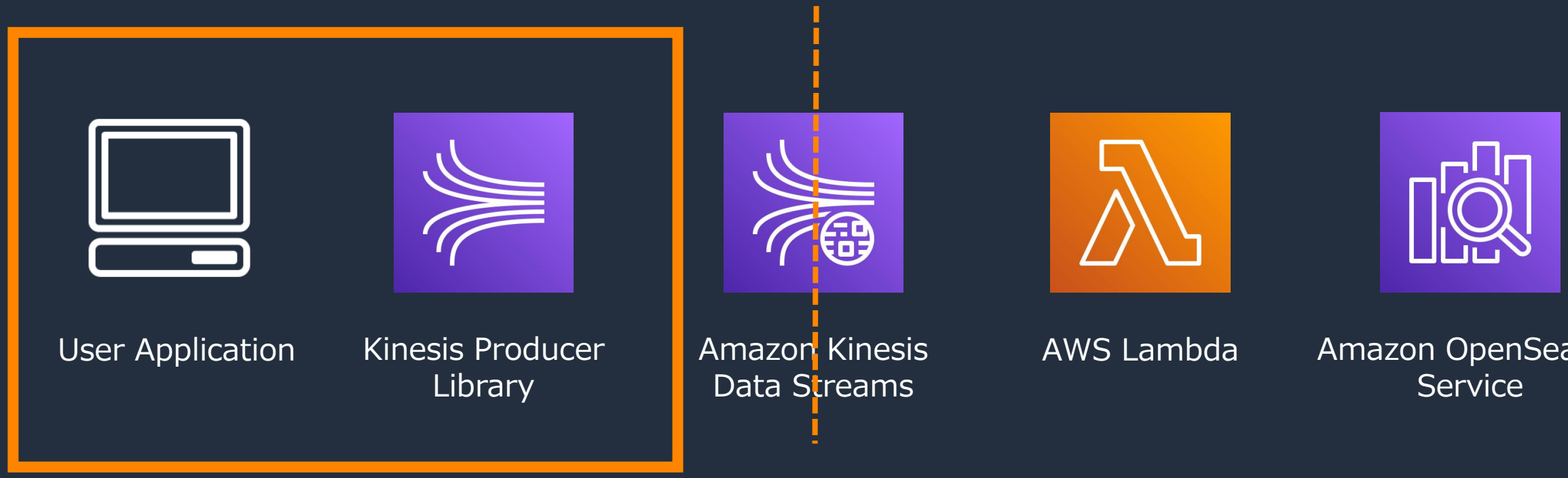
問題の発生個所だけでなく、ストリーム全体で同様の傾向にあるか、
特定シャードで偏った傾向がみられるかを合わせて切り分けるとよい

メトリクス名	説明	メトリクスの見方
GetRecords.IteratorAgeMilliseconds (Stream-Level) IteratorAgeMilliseconds (Shard-Level)	現在の時刻と Consumer による GetRecords 呼び出しの最後のレコードがストリーム(シャード)に書き込まれた時刻との差	当メトリクスの値が 0 に近いほど、コンシューマーの処理遅延が少ない。 値が継続的に増加している場合は、コンシューマーの処理遅延の可能性がある。 一時的なスパイクは、再起動や新しいコンシューマーが追加されるなどの理由で起こりえるため、すぐに解消するようであれば問題は無いと考えられる

https://docs.aws.amazon.com/streams/latest/dev/monitoring-with-cloudwatch.html#kinesis-metrics

プロデューサーの遅延が疑われる場合は？

プロデューサーが生成したイベントを、遅延・エラーなく
データストリームに送信できているか調査する



プロデューサーの調査(Kinesis Producer Library)

- イベントがプロデューサーに送り込まれてから、実際にデータストリームにデータが送信されるまでのタイムラグ、処理エラー、送信エラーの有無を確認
- ネットワークや内部処理の問題でストリームにデータが到達していない場合、データストリームのメトリクスからは問題を検出できない。
  このようなケースでプロデューサーのメトリクス確認は重要

メトリクス名	説明	メトリクスの見方
BufferingTime	ユーザーレコードが KPL に到着してからバックエンドに送信されるまでの時間	値が増加している場合, 処理に時間がかかっている可能性が疑われる。 ErrorsByCode が増加していればエラーの原因を、ErrorsByCode が増加していなければリソース不足の有無を調査する
ErrorsByCode	各種類のエラーコードの数	エラー内容毎に異なる ErrorCode ディメンションでメトリクスが出力されるため、ディメンションから原因を判断する。合わせて、シャード ID ごとのエラー件数偏りの有無をチェックすることで、ストリーム全体の問題か特定シャードの問題かを特定する
UserRecordsReceived UserRecordsPending UserRecordsPut	KPL が受信したレコード数、 KPL 内で保留状態にあるレコード数 ストリームに PUT されたレコード数	UserRecordsPending が増加しており、UserRecordsReceived が UserRecordsPut を継続的に上回っている場合は、KPL 内で処理が滞留している。 ErrorsByCode から処理エラーの増加が疑われる場合はエラーと実装の確認を、 スペック不足であればスケールを検討する

https://docs.aws.amazon.com/ja_jp/streams/latest/dev/monitoring-with-kpl.html

その他の監視ポイント

タイムラグ

- アプリケーションによるイベントを生成してから、宛先にデータが取り込まれるまでのタイムラグは SLA 内か
- タイムラグの継続的な増加はみられるか

スループット

- スループットの急激な増加、あるいは低下は発生しているか
- トラフィックの継続的な増加はみられるか

キャパシティ、リソース使用率

- リソース使用率が上限に達していないか
- 割り当てリソースに対する使用率が極端に低くないか

モニタリングに用いる主なメトリクス

タイムラグ

- IteratorAgeMilliseconds

スループット,キャパシティ, リソース使用率

- GetRecords.Latency, GetRecords.Bytes, GetRecords.Records
- OutcomingBytes, OutcomingRecords
- PutRecords.Latency, PutRecords.Bytes, PutRecords.Records
- IncomingBytes, IncomingRecords

スロットリング

- ReadProvisionedThroughputExceeded
- WriteProvisionedThroughputExceeded

パフォーマンス問題の切り分け まとめ

- ストリーム処理のパフォーマンス問題は
  中央(データストリーム) を起点に切り分ける

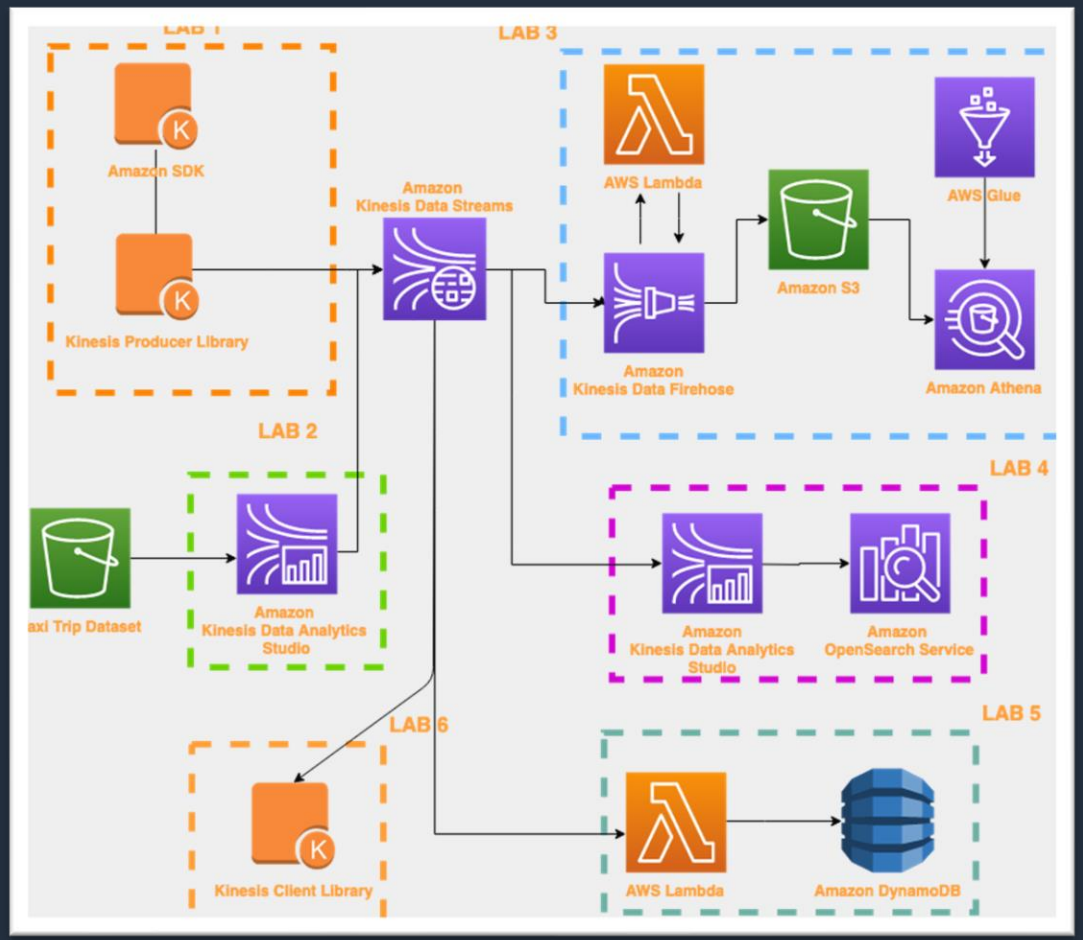
- AWS サービスが提供する Amazon CloudWatch メトリクスは
  有用な情報が数多く含まれている

- エラーによって処理が滞留しているのか、スペック不足
  (スループット不足)によって処理が滞留しているのかは
  メトリクスから確認可能

ワークショップの紹介

Real Time Streaming with Amazon Kinesis

Kinesis Data Streams を含む
リアルタイム処理でよく使われる
サービスの組み合わせを体験できる
ハンズオン形式のワークショップ



https://catalog.us-east-1.prod.workshops.aws/workshops/2300137e-f2ac-4eb9-a4ac-3d25026b235f/en-US

まとめ

まとめ

- Amazon Kinesis Data Streams はフルマネージドな
サーバーレスストリーミングデータ処理サービス。
すぐにストリームを作成でき、AWS により提供されている
クライアントやライブラリを活用することで、素早く
ストリームデータの配信・取得を開始することができる
- 様々な AWS サービスと統合されているため、サービスログの
ニアリアルタイム取得、処理も可能
- キャパシティ・スループットに着目して
モニタリングとスケールを行うことが運用上のポイント
- レコード集約を行うことでコストパフォーマンスを最大化できる

その他リソース

- サービス概要: https://aws.amazon.com/jp/kinesis/data-streams/
- よくある質問: https://aws.amazon.com/jp/kinesis/data-streams/faqs/
- 料金: https://aws.amazon.com/jp/kinesis/data-streams/pricing/
- ドキュメント: https://docs.aws.amazon.com/streams/latest/dev/introduction.html
- 制限事項: https://docs.aws.amazon.com/streams/latest/dev/service-sizes-and-limits.html
- トラブルシューティング
  - https://docs.aws.amazon.com/streams/latest/dev/troubleshooting-producers.html
  - https://docs.aws.amazon.com/streams/latest/dev/troubleshooting-consumers.html
- ナレッジセンター: https://repost.aws/tags/TASWsn6-d_TEOCnpTjcgKxzQ/amazon-kinesis-data-streams

本資料に関するお問い合わせ・ご感想

技術的な内容に関しましては、有料のAWSサポート窓口へ
お問い合わせください

https://aws.amazon.com/jp/premiumsupport/

料金面でのお問い合わせに関しましては、カスタマーサポート窓口へ
お問い合わせください（マネジメントコンソールへのログインが必要です）

https://console.aws.amazon.com/support/home#/case/create?issueType=customer-service

具体的な案件に対する構成相談は、後述する個別相談会をご活用ください

ご感想はTwitterへ！ハッシュタグは以下をご利用ください
#awsblackbelt

その他コンテンツのご紹介

ウェビナーなど、AWSのイベントスケジュールをご参照いただけます

https://aws.amazon.com/jp/events/

ハンズオンコンテンツ

https://aws.amazon.com/jp/aws-jp-introduction/aws-jp-webinar-hands-on/

AWS 個別相談会

AWSのソリューションアーキテクトと直接会話いただけます

https://pages.awscloud.com/JAPAN-event-SP-Weekly-Sales-Consulting-Seminar-2021-reg-event.html

Thank you!