TA-03

IoTを支えるビックデータソリューション

アマゾン データ サービス ジャパン株式会社
エコシステム ソリューション部
パートナーソリューションアーキテクト
榎並 利晃

自己紹介

名前

- 榎並 利晃（えなみ としあき）
- toshiake@amazon.co.jp

役割

- パートナーソリューションアーキテクト
- 主にエマージングパートナー様を担当

好きなAWSのサービス

- Amazon Kinesis
- Amazon DynamoDB

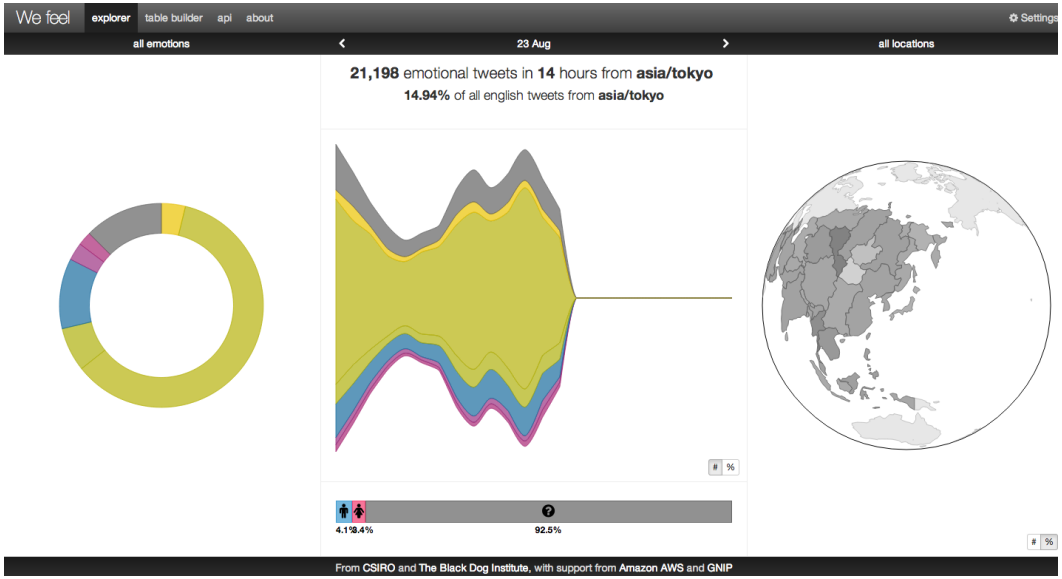
Agenda

- IoTがもたらす新たな価値
- データフロー
- Amazon Kinesis概要
- データ入力パターン
- データ処理パターン
- まとめ

IoTがもたらす新たな価値

IoT=「Internet of Things」

広がりつつあるIoTの世界



Amazonでの取り組み

MongoDB World 2014

「IoTが既にAmazonのビジネスを変えている」

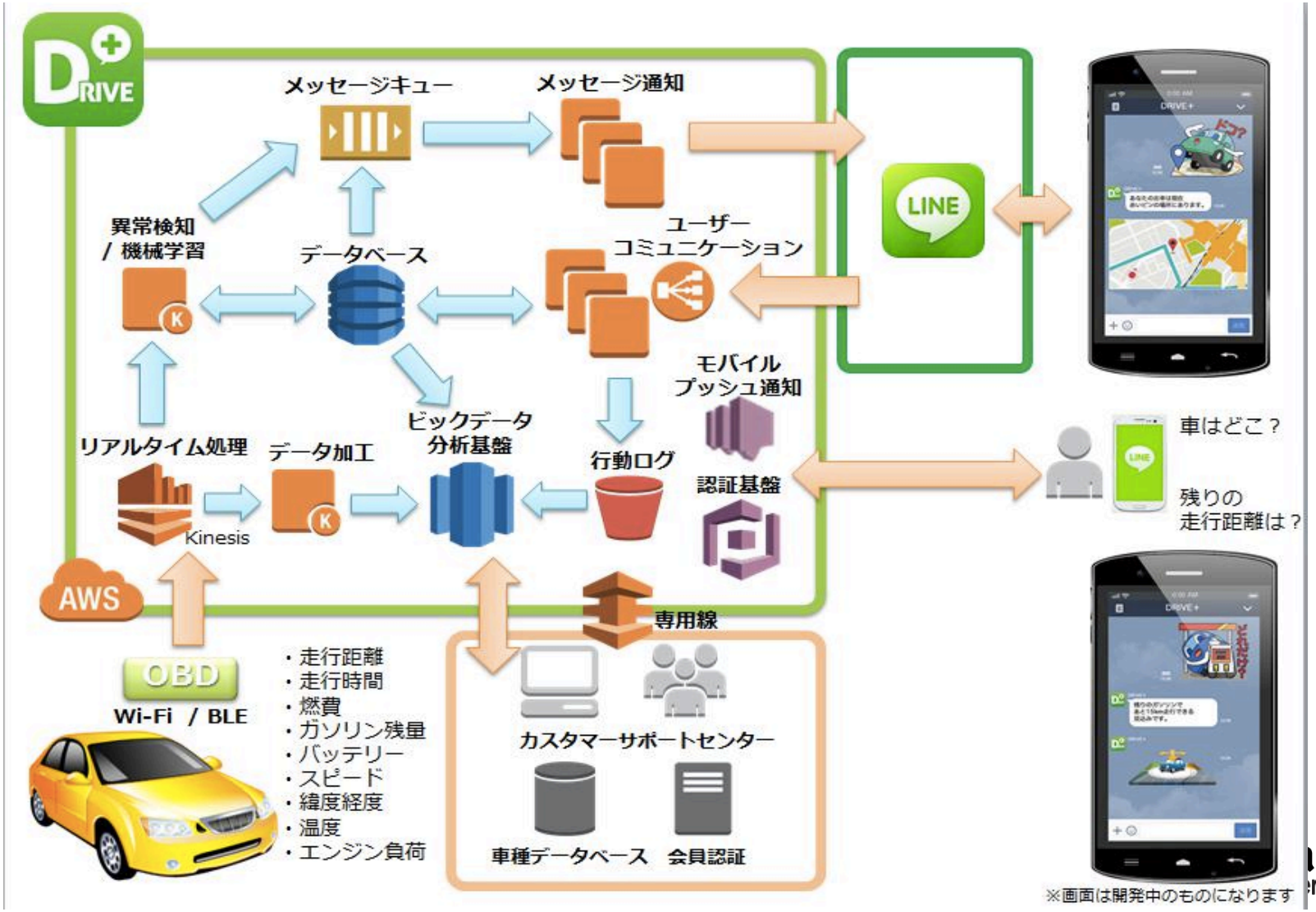
Amazon Drone

Amazon Dash

amazon web services

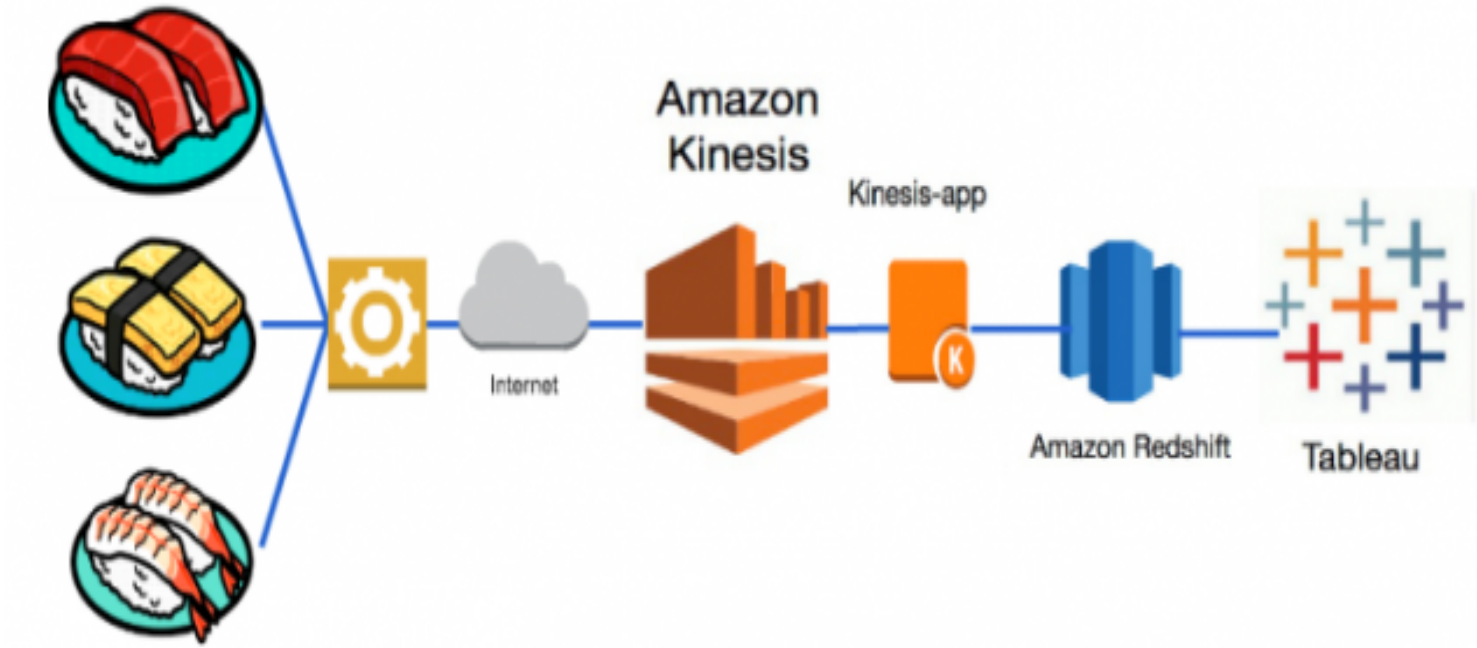
ガリバー様 Drive+

車のセンサー情報をクラウドに収集し、ドライバーに対して走行距離や位置情報などの情報を提供！



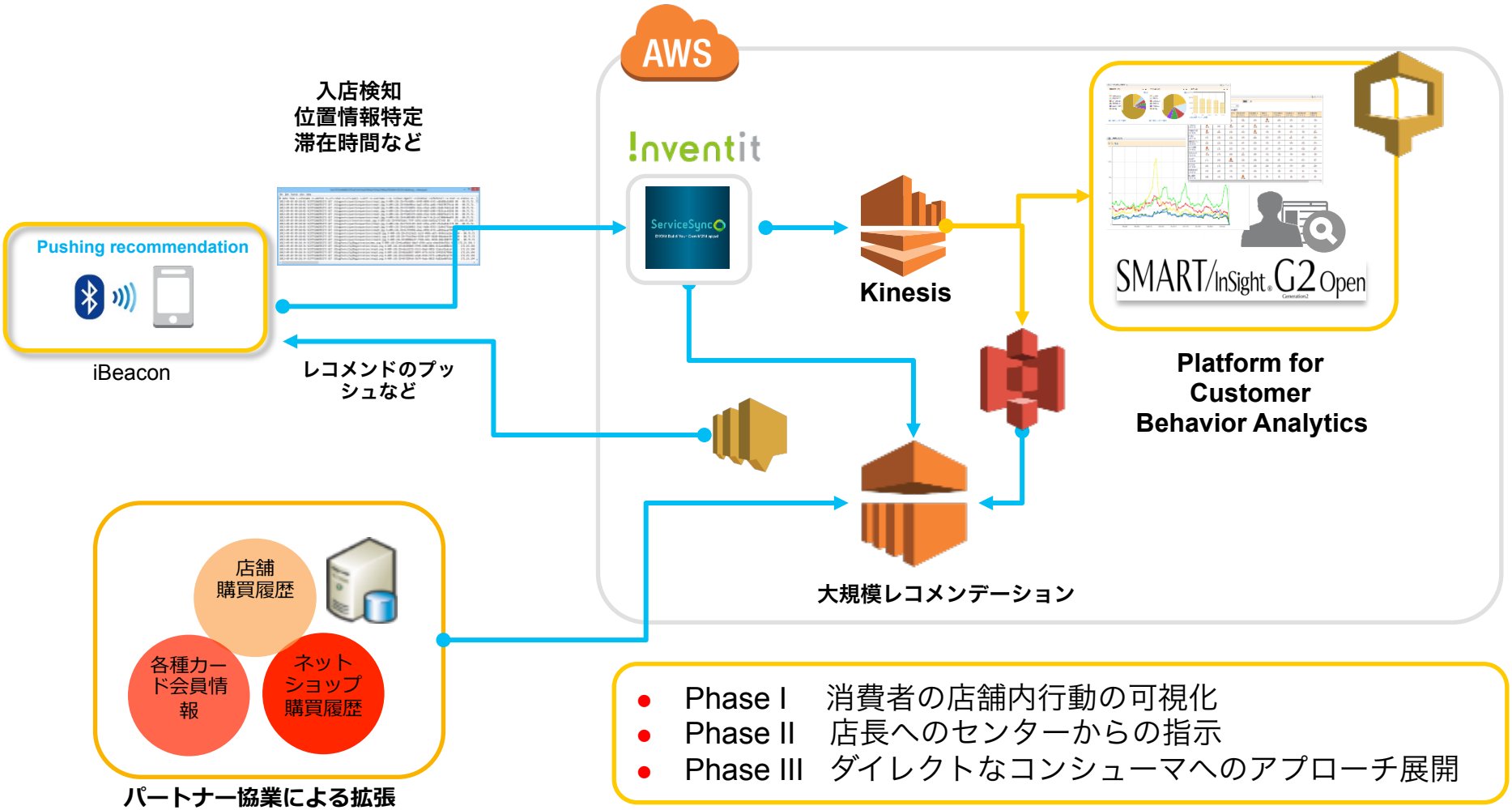
スシロー様

レーン上に流れる寿司情報をリアルタイムにクラウドに転送。来店状況やオペレーション状況をリアルタイム把握
リアルタイム収集しすてむを約1ヶ月でシステムを構築！



スマートインサイト様

センサー、モバイルデバイス、ログサーバーなど非常に膨大なデータを業務上意味のある情報にするためのSMART/InSight Cloud Real Time M2M Analyticsを開発！



- Phase I　消費者の店舗内行動の可視化
- Phase II　店長へのセンターからの指示
- Phase III　ダイレクトなコンシューマへのアプローチ展開

イノベーション:

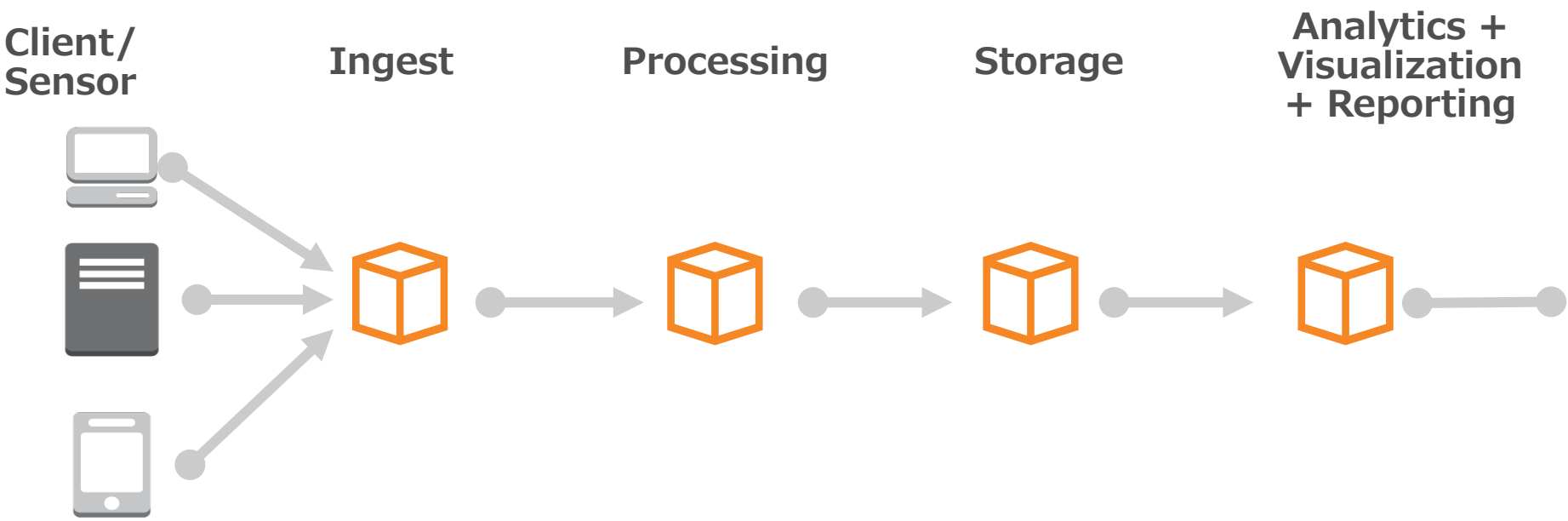
あらゆるものがインターネットにつながり、いままで見えなかった情報から新しいイノベーションを起こすことができる

どのようにデータを集め、
どのようにデータを処理するのか。

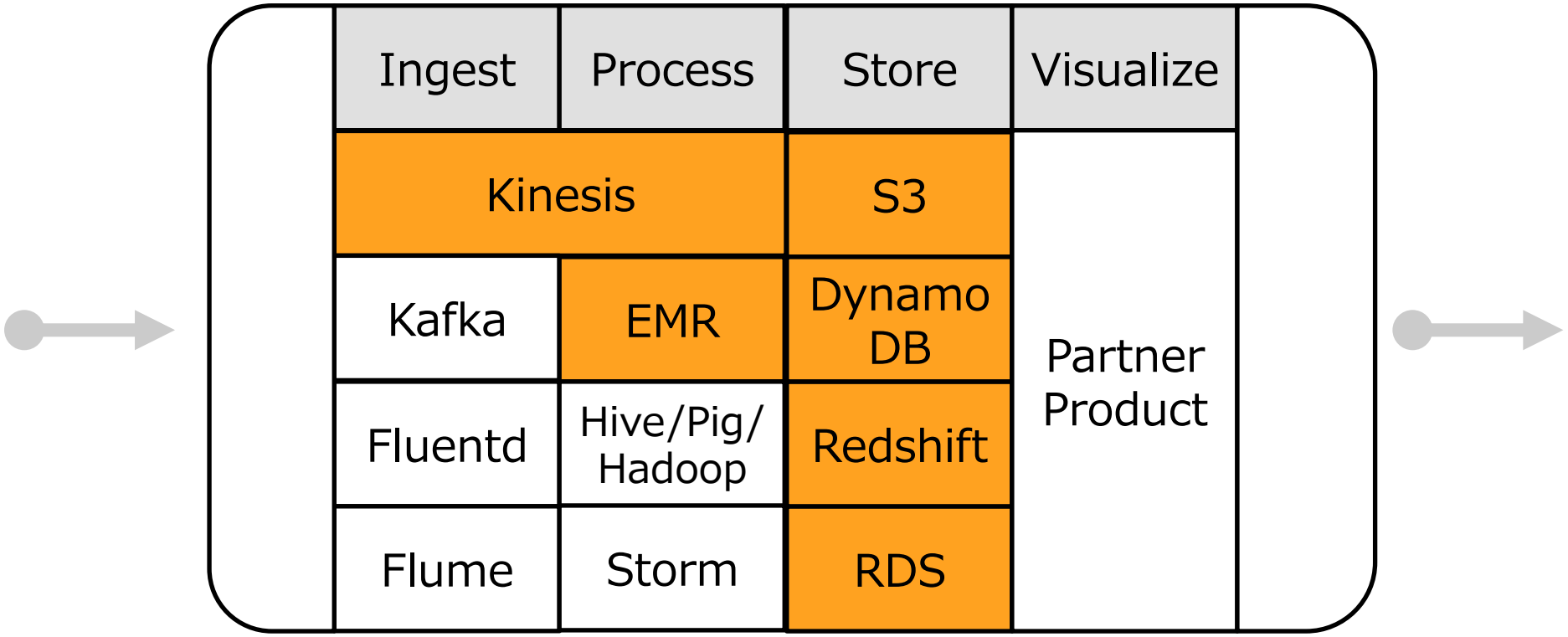
amazon web services

データフロー

典型的なデータフロー



データフローからみた
ソリューションマッピング

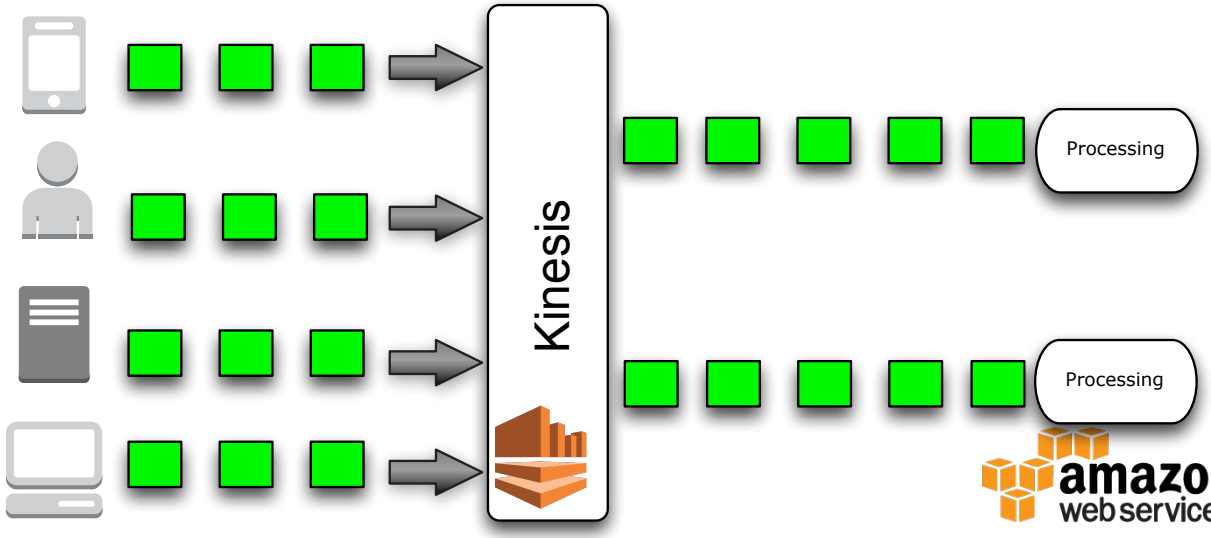


（＊）色付きがAWSサービス

Ingest Layerの重要性

- 構造の異なるデータソースに対する高速処理
  - 耐障害性とスケールに対する考慮
  - 高い信頼性の維持
  - 順序性

- ランダムにくるデータをまとめて、シーケンスストリームの形に変換
  - シーケンスデータによる容易な処理の実現
  - 容易なスケール
  - 永続化データ



Ripplation様により発表
AWS Summit Tokyo 2014

- 秒間10万のデータ収集（Ingest Layer）の仕組みを検討した結果、膨大なリソース（人的、金銭的）が必要であることが判明！

（Ripplation様発表資料より抜粋）

結果、Amazon Kinesisを採用することに！

Amazon Kinesis概要

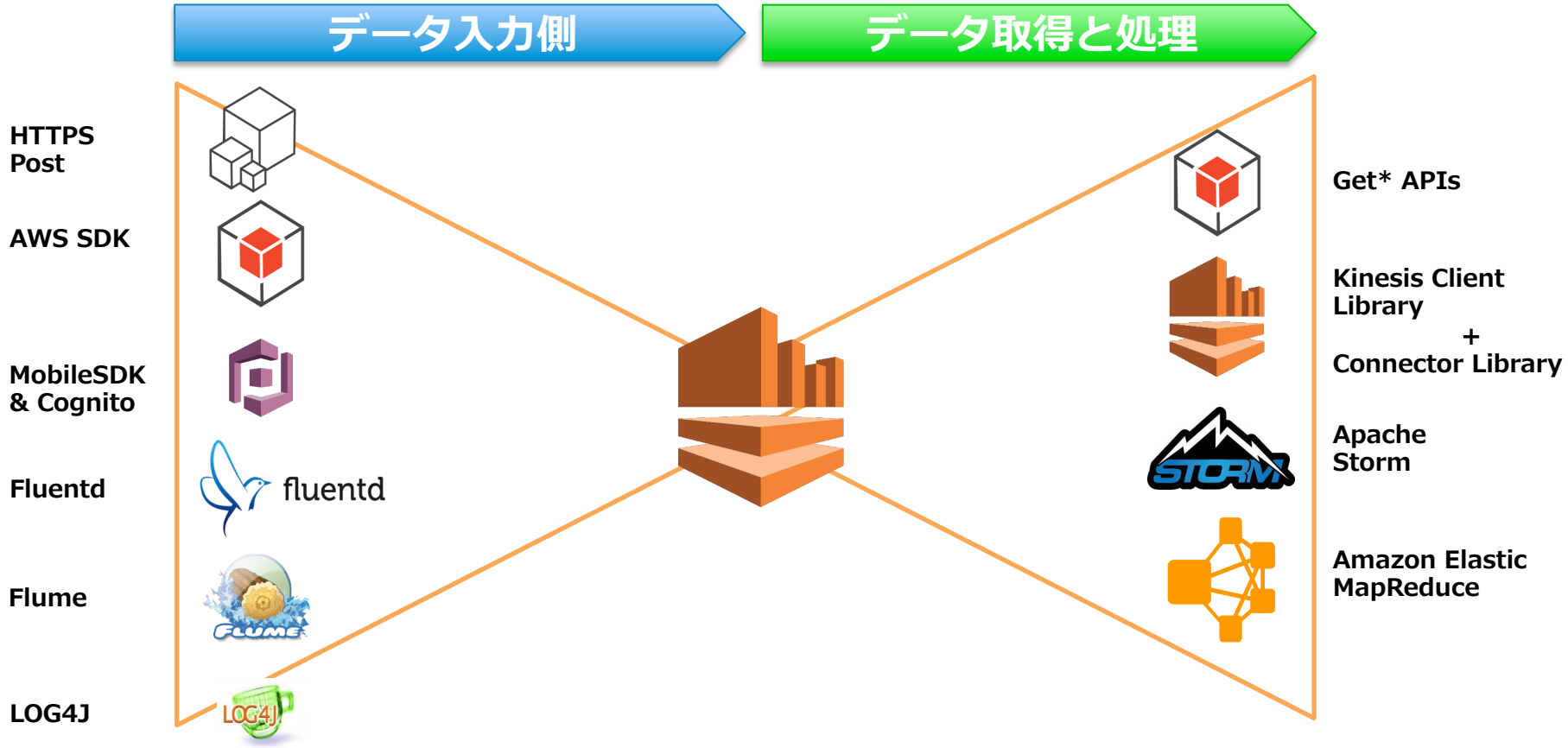
amazon web services

Amazon Kinesisとは？

- ハイボリュームな連続したデータをリアルタイムで処理可能なフルマネージドサービス

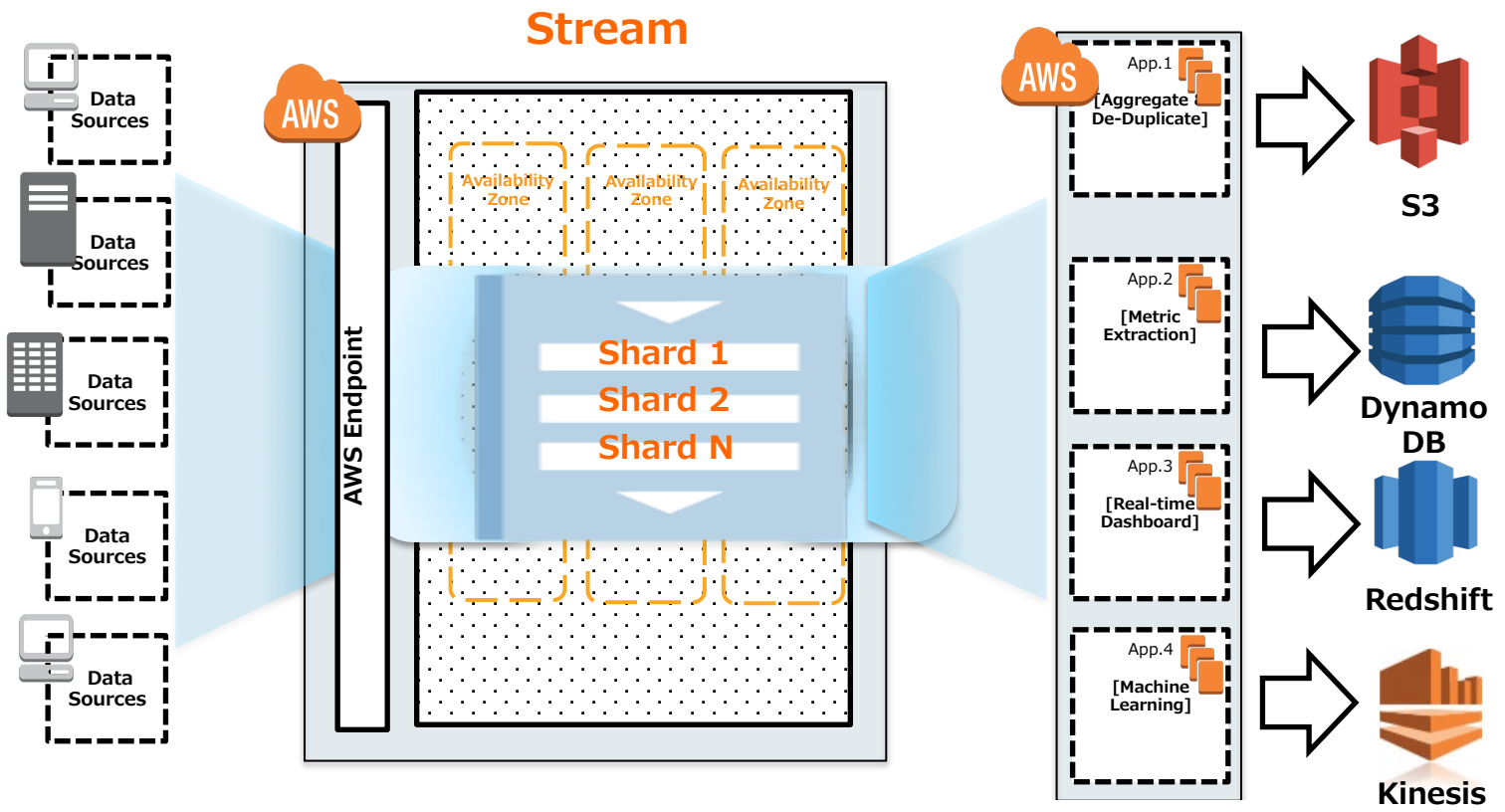
- Kinesisは、数十万のデータソースからの1時間辺り数テラバイトのデータを処理することができ、かつ、格納されたデータは、複数のAZに格納する信頼性と耐久性をもつサービス

Amazon Kinesis 概要



amazon web services

Amazon Kinesis 構成内容



- 用途単位で**Stream**を作成し、Streamは、1つ以上の**Shard**で構成される
- Shardは、データ入力側 1MB/sec, 1000 TPS、データ処理側 2 MB/sec, 5TPSのキャパシティを持つ
- 入力するデータを**Data Record**と呼び、入力されたData Recordは、24 時間かつ複数のAZに保管される
- Shardの増加減によってスケールの制御が可能

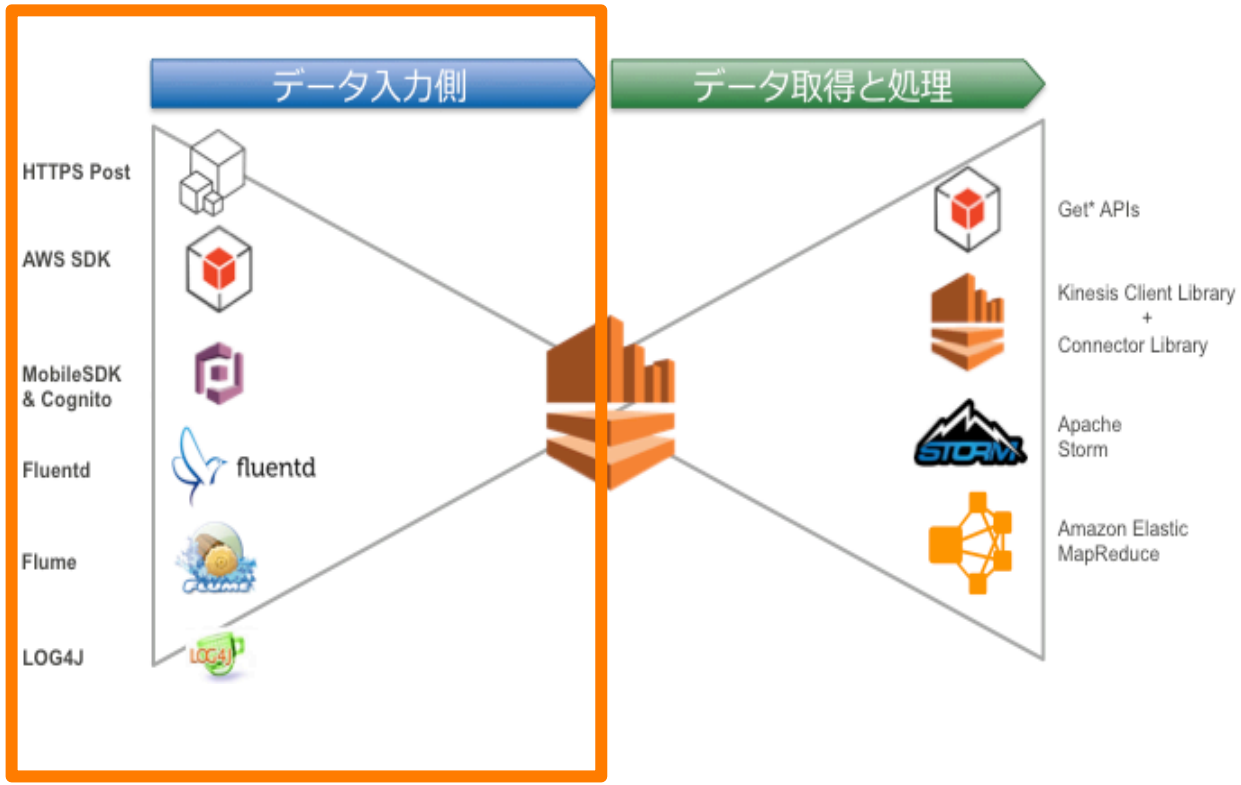
amazon web services

コスト
従量課金 & 初期費用不要

課金項目	単価
シャード利用料	$0.0195/shard/時間
Putトランザクション	$0.043/100万Put

- シャード１つで、一ヶ月約$14
- Getトランザクションは無料
- インバウンドのデータ転送料は無料
- アプリケーションが走るEC2は通常の料金がかかります

データ入力デザインパターン



amazon web services

データ入力方法

仕様

- PutRecord API でデータ入力が可能
  - http://docs.aws.amazon.com/kinesis/latest/APIReference/API_PutRecord.html
- AWS CLI、AWS SDK for Java, Javascript, Python, Ruby, PHP, .Net が利用可能

AWS CLI を利用した例：

```
$ aws kinesis put-record \
--stream-name StreamName --data 'foo' --partition-key $RANDOM
```

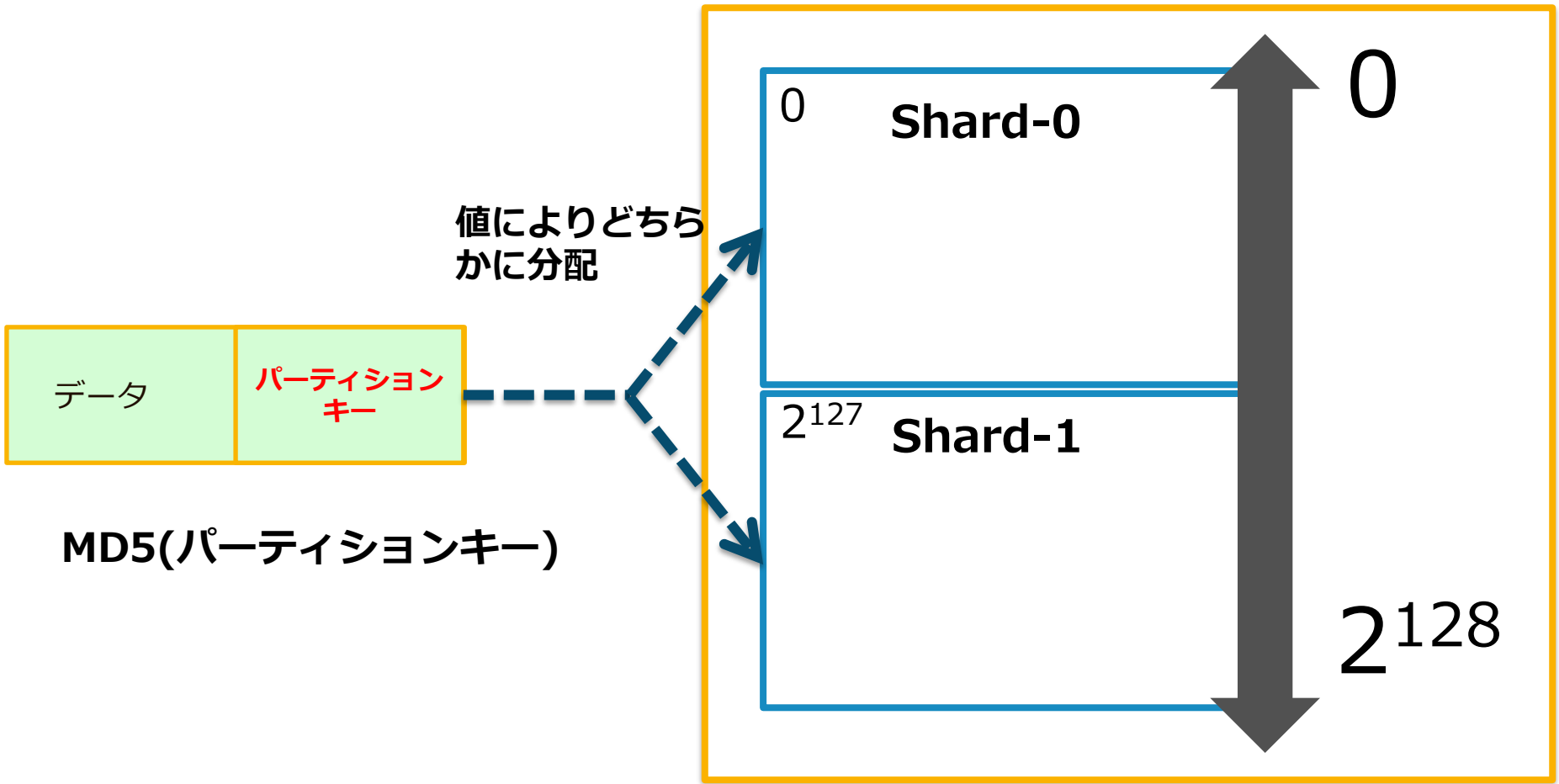
レスポンス

```
{
    "ShardId": "shardId-000000000013",
    "SequenceNumber":
"49541296383533603670305612607160966548683674396982771921"
}
```

データ入力及び分配イメージ

仕様

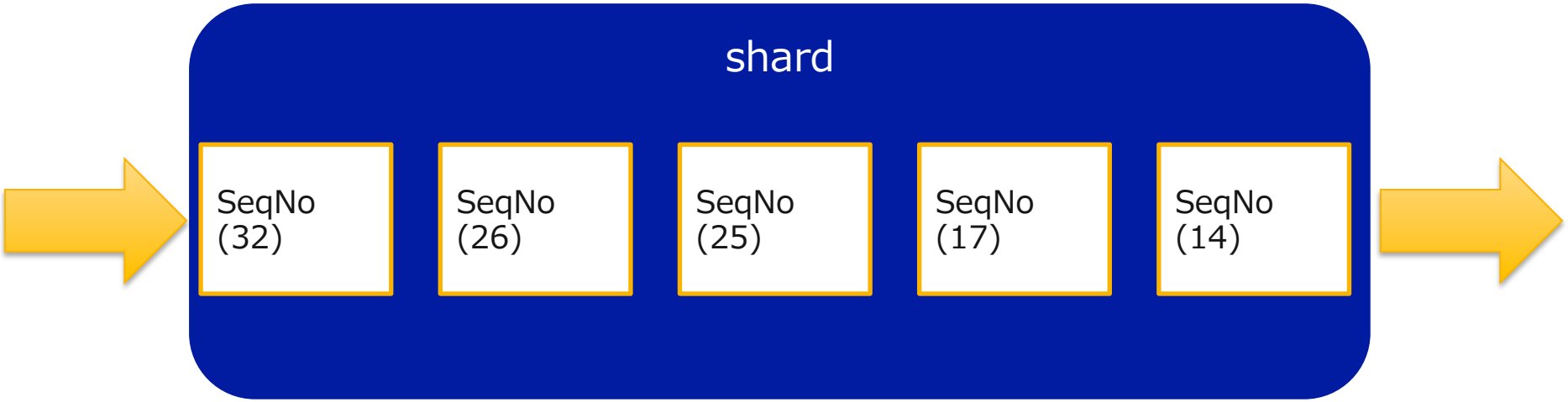
- DataRecordに設定されたパーティションキーを基にShardに分配
- Shardは担当するレンジを持ち、パーティションキーをMD5でハッシュ化した値によって該当のShardに分配される



シーケンス番号

仕様

- KinesisがStream内でユニークなシーケンス番号を付与
- データもシーケンス番号も不変
- シーケンス番号でデータが何回でも取得できる（24時間以内）
- 何度取得してもシーケンス番号の順番はかわらない

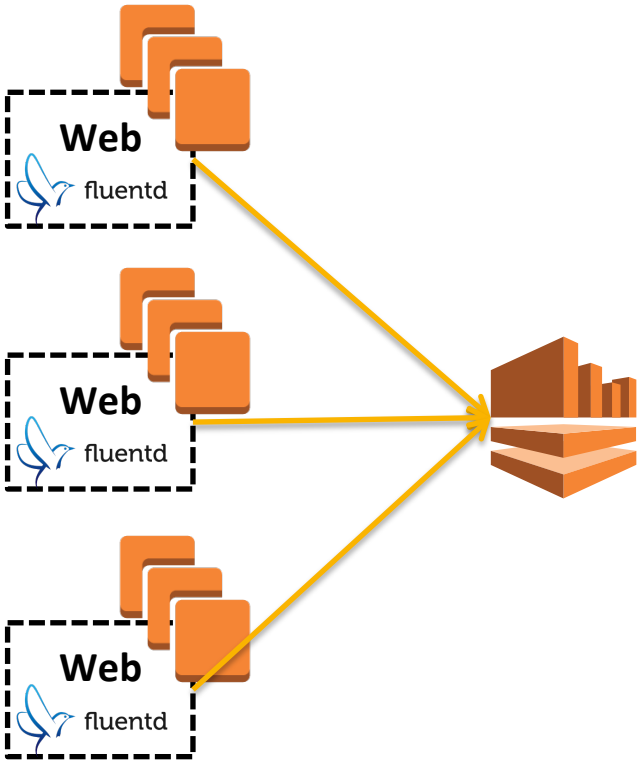


データ入力パターン分類

パターン	ユースケース
データ入力パターン１	サーバに蓄積されたログを入力データとしたい場合 – リアルタイムダッシュボード – オンラインリコメンデーション
データ入力パターン２	センサーが収集したデータを入力データとしたくかつ軽量なプロトコルを利用したい場合 – センサ異常検知 – O2O
データ入力パターン３	モバイルアプリが生み出すデータを直接入力データとしたいパターン – モバイルアプリ利用状況把握（ダッシュボード） – モバイルアプリに対するサービス機能提供

データ入力パターン１

- Fluentd Pluginを利用し、データ入力するパターン
- Webサーバ、アプリケーションサーバなどにあるログデータの入力に最適
- GithubからPluginを取得することが可能
  https://github.com/awslabs/aws-fluent-plugin-kinesis

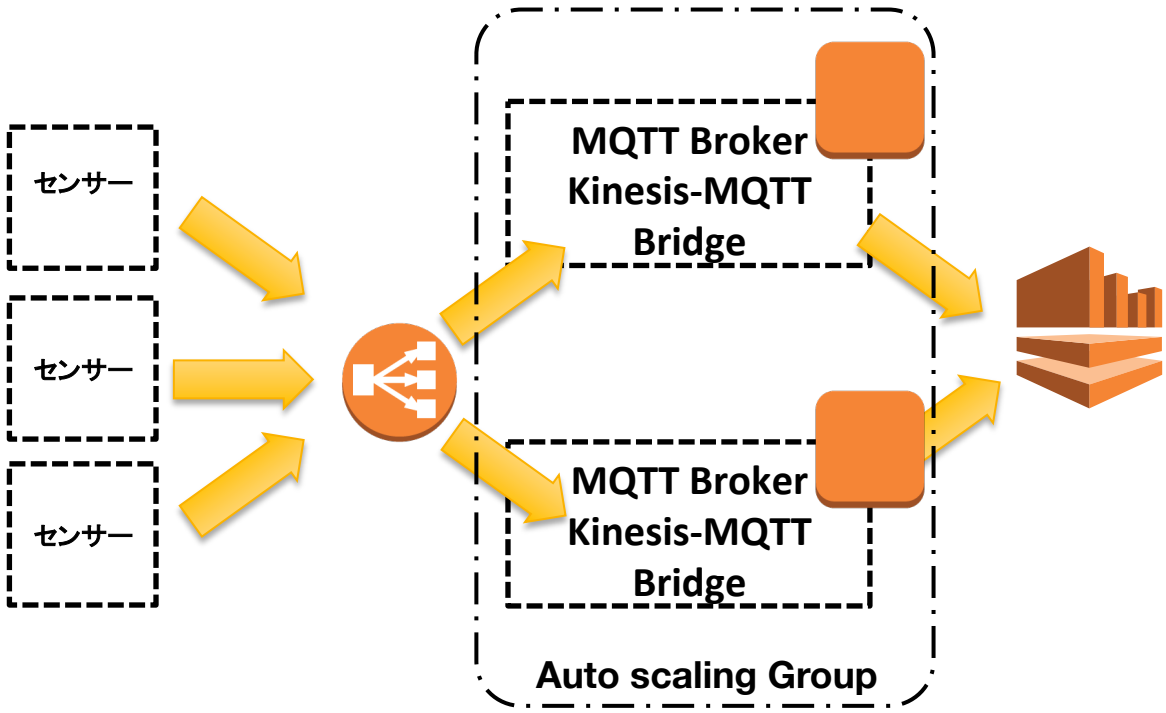


設定ファイル例

```
<match your_tag>
type kinesis

stream_name YOUR_STREAM_NAME

aws_key_id YOUR_AWS_ACCESS_KEY
aws_sec_key YOUR_SECRET_KEY

region us-east-1

partition_key name
partition_key_expr 'some_prefix-' + record
</match>
```

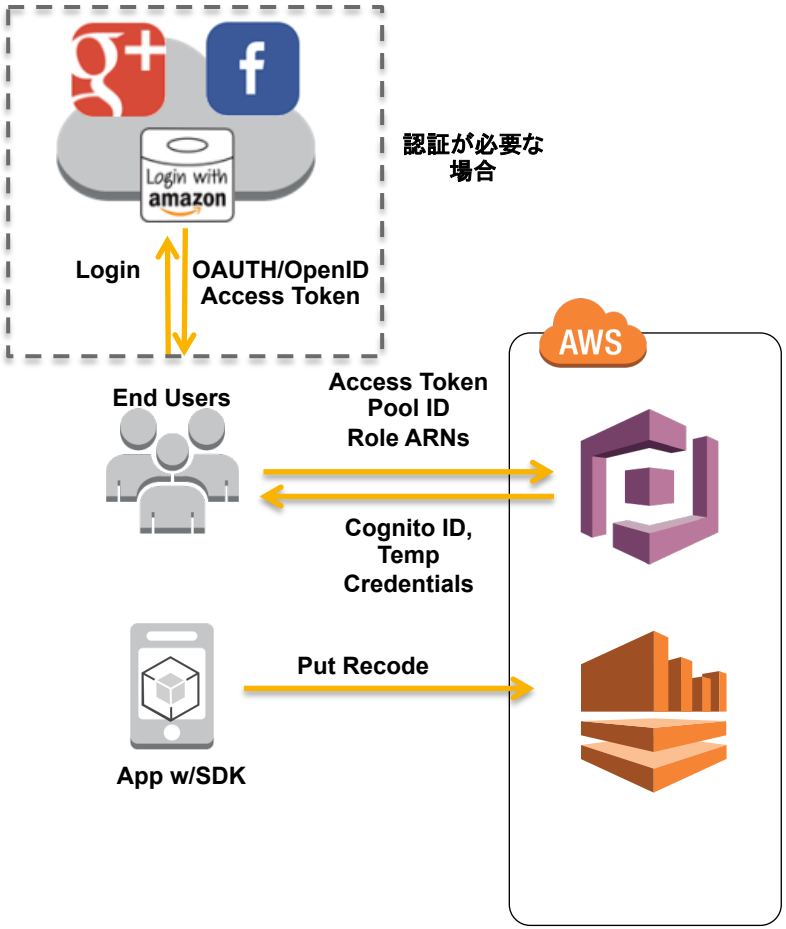
データ入力パターン２

- MQTTアダプタ利用パターン
- センサーデバイスなどライトウェイトなプロトコル（MQTT)を利用したい場合に最適
- MQTT BrokerとMQTT-Kinesis Bridgeを用いてメッセージをKinesisに入力することが可能
- GithubからMQTT-Kinesis Bridgeサンプルソースが取得可能
  https://github.com/awslabs/mqtt-kinesis-bridge

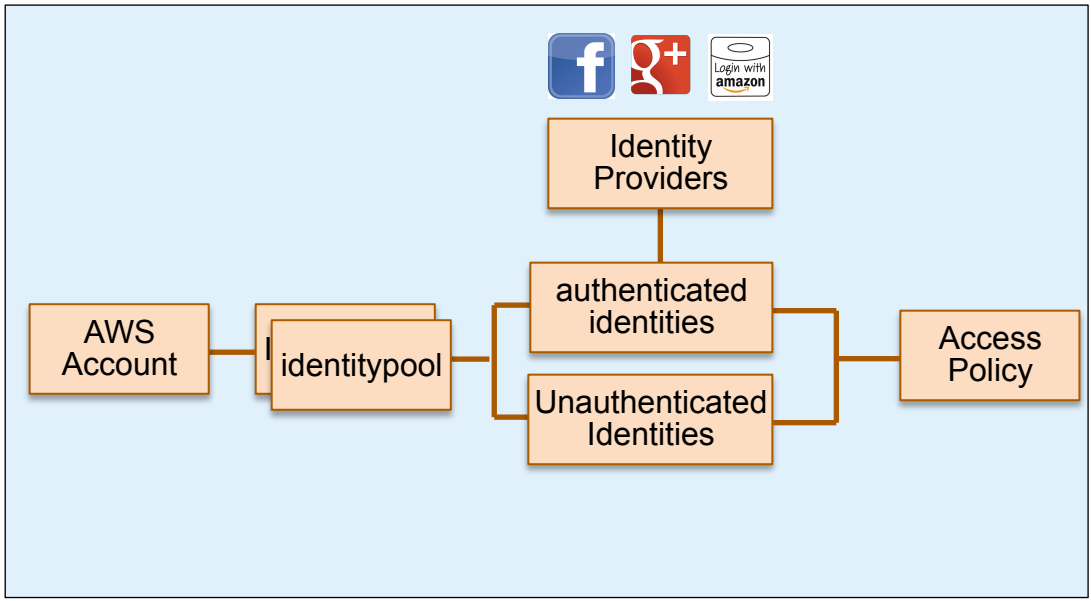


データ入力パターン3

- モバイルアプリから直接入力パターン
- CognitoとMobileSDKを用いて容易にKinesisにデータ入力が可能
- 認証または、非認証でKinesisへのアクセストークンをテンポラリに取得しデータ入力が可能

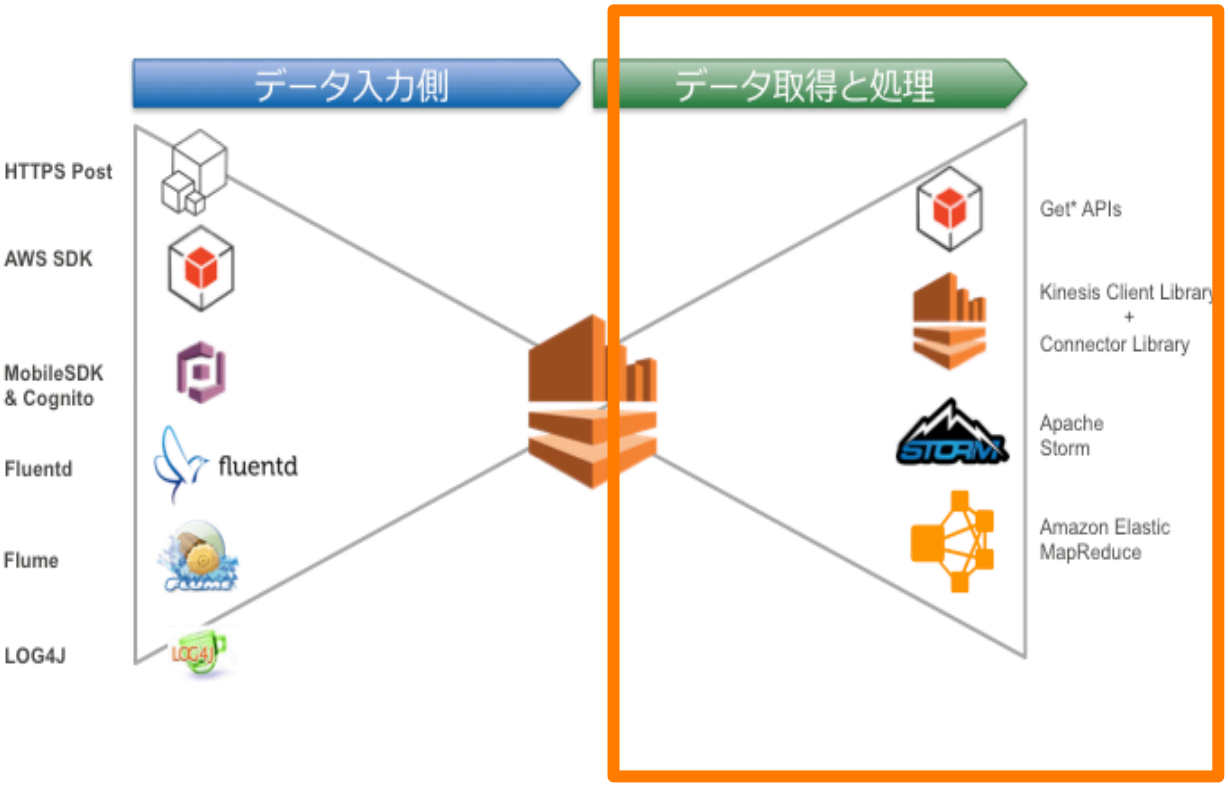


Amazon Cognito - IDブローカー



amazon web services

データ処理デザインパターン



amazon web services

データ取得方法

仕様

- GetShardIterator APIでShard内のポジションを取得し、GetRecords APIでデータ入力が可能
  - http://docs.aws.amazon.com/kinesis/latest/APIReference/API_GetShardIterator.html
  - http://docs.aws.amazon.com/kinesis/latest/APIReference/API_GetRecords.html
- AWS CLI、AWS SDK for Java, Javascript, Python, Ruby, PHP, .Net が利用可能

```
$ aws kinesis get-shard-iterator --stream-name StreamName \
--shard-id shardId-000000000013 --shard-iterator-type AT_SEQUENCE_NUMBER \
--starting-sequence-number
49541296383533603670305612607                                    21
{
    "ShardIterator": ”FakeIterator"
}
$ aws kinesis get-records  --shard-iterator FakeIterator --limit 1
{
    "Records": [
        {
            "PartitionKey": "16772",
            "Data": "Zm9v",
            "SequenceNumber":
"49541296383533603670305612607160966548683674396982771921"
        }
    ],
    "NextShardIterator": “YetAnotherFakeIterator"
}
```

get-shard-iterator

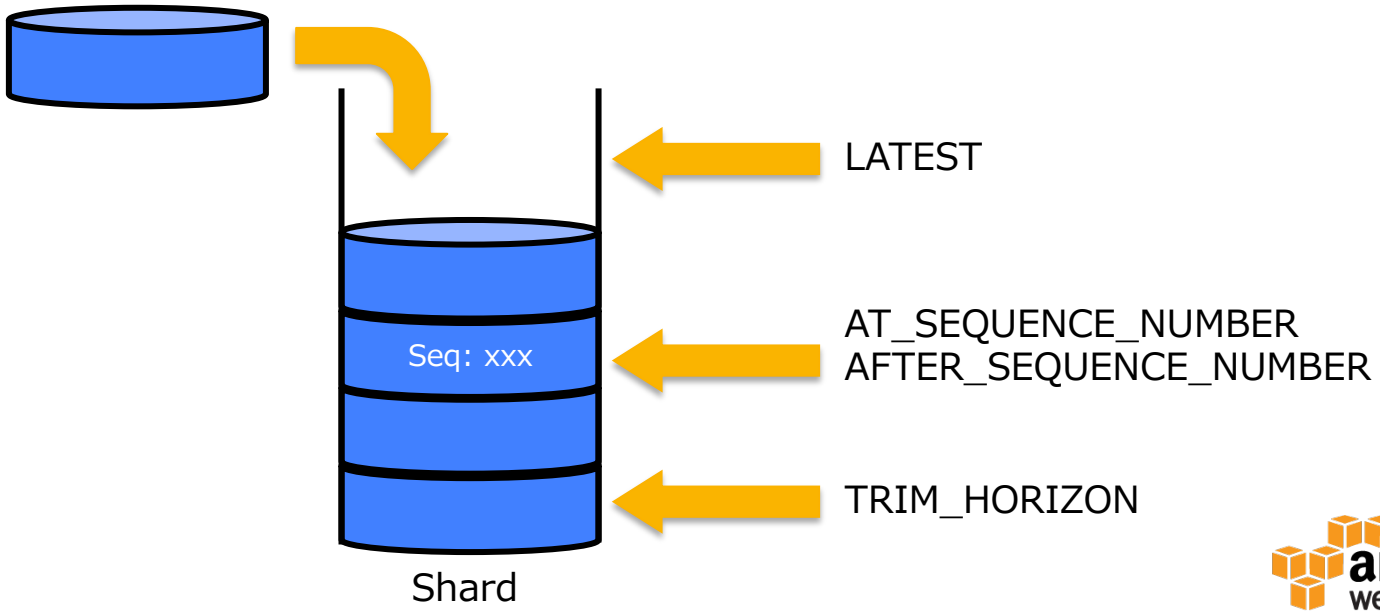
get-records

GetShardIteratorでのデータ取得指定方法

仕様

- GetShardIterator APIでは、ShardIteratorTypeを指定してポジションを取得する。
- ShardIteratorTypeは以下の通り
  - AT_SEQUENCE_NUMBER (指定のシーケンス番号からデータ取得)
  - AFTER_SEQUENCE_NUMBER (指定のシーケンス番号以降からデータ取得)
  - TRIM_HORIZON (Shardにある最も古いデータからデータ取得)
  - LATEST (最新のデータからデータ取得)

GetShardIteratorの動作イメージ



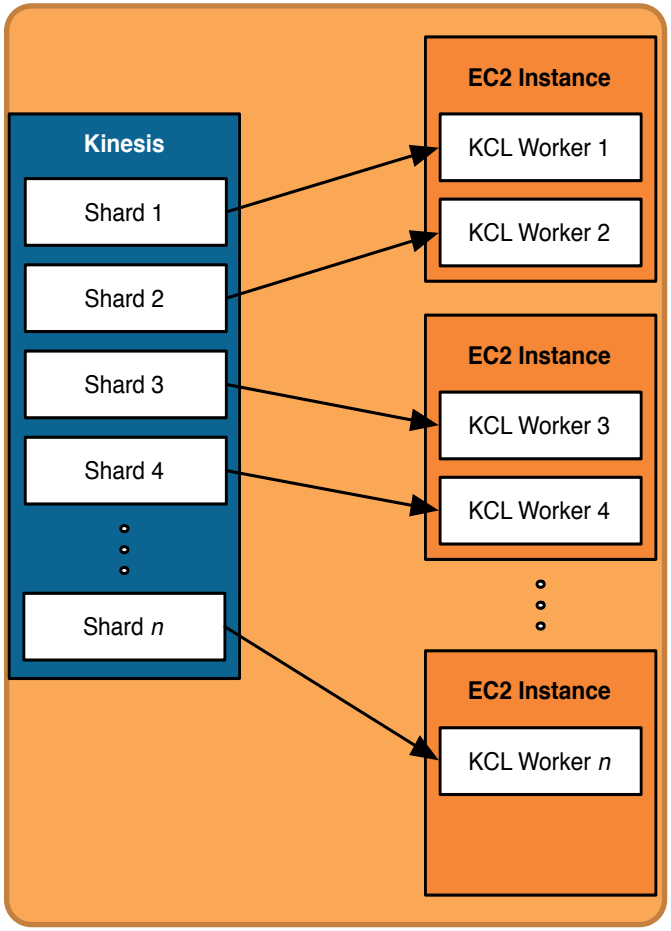
amazon web services

Kinesis Client Library (KCL)
Client library for fault-tolerant, at least-once, Continuous Processing

仕様

- Shardと同じ数のWorker
- Workerを均等にロードバランシング
- 障害感知と新しいWorkerの立ち上げ
- シャードの数に応じてworkerが動作する
- AutoScalingでエラスティック
- チェックポインティングとAt least once 処理

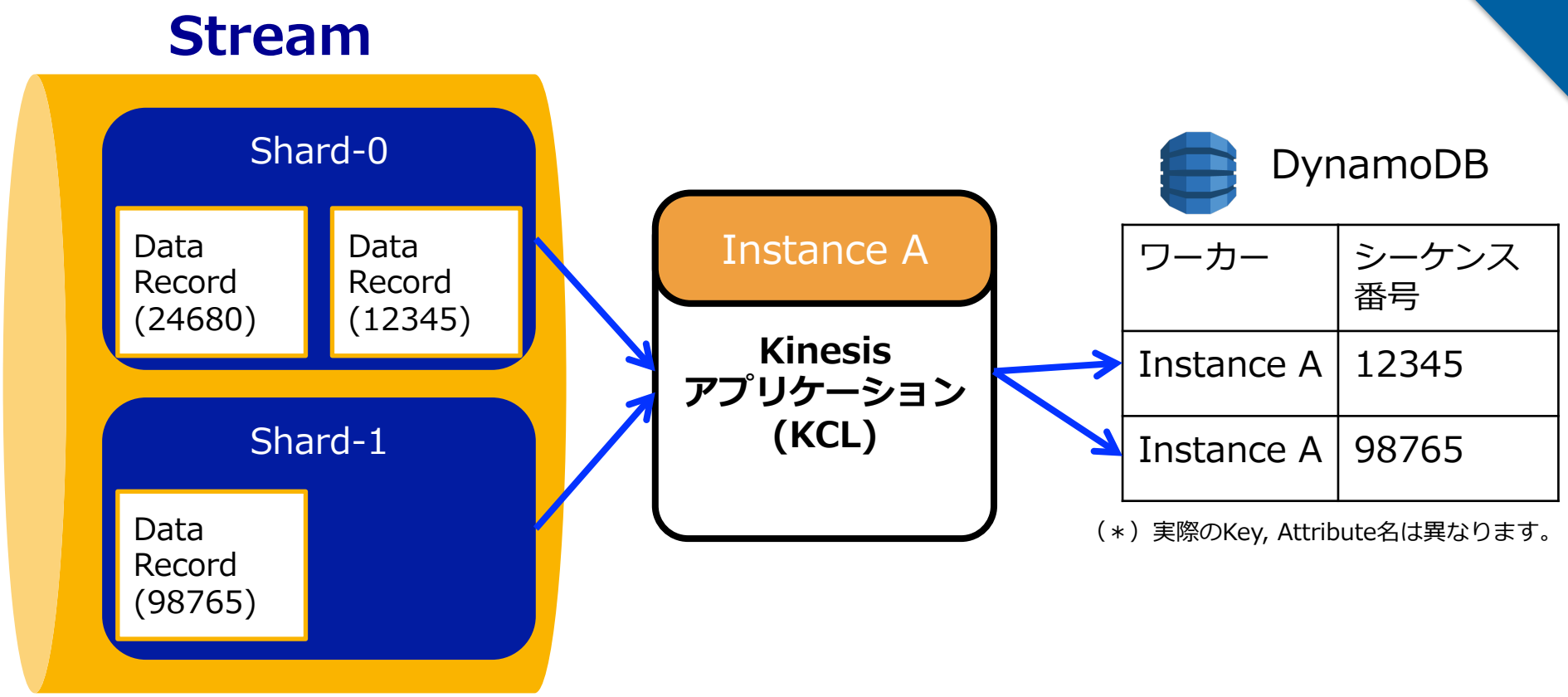
これらの煩雑な処理を意識することなく
ビジネスロジックに集中することができる。



amazon web services

Kinesis Client Libraryの動き

仕様

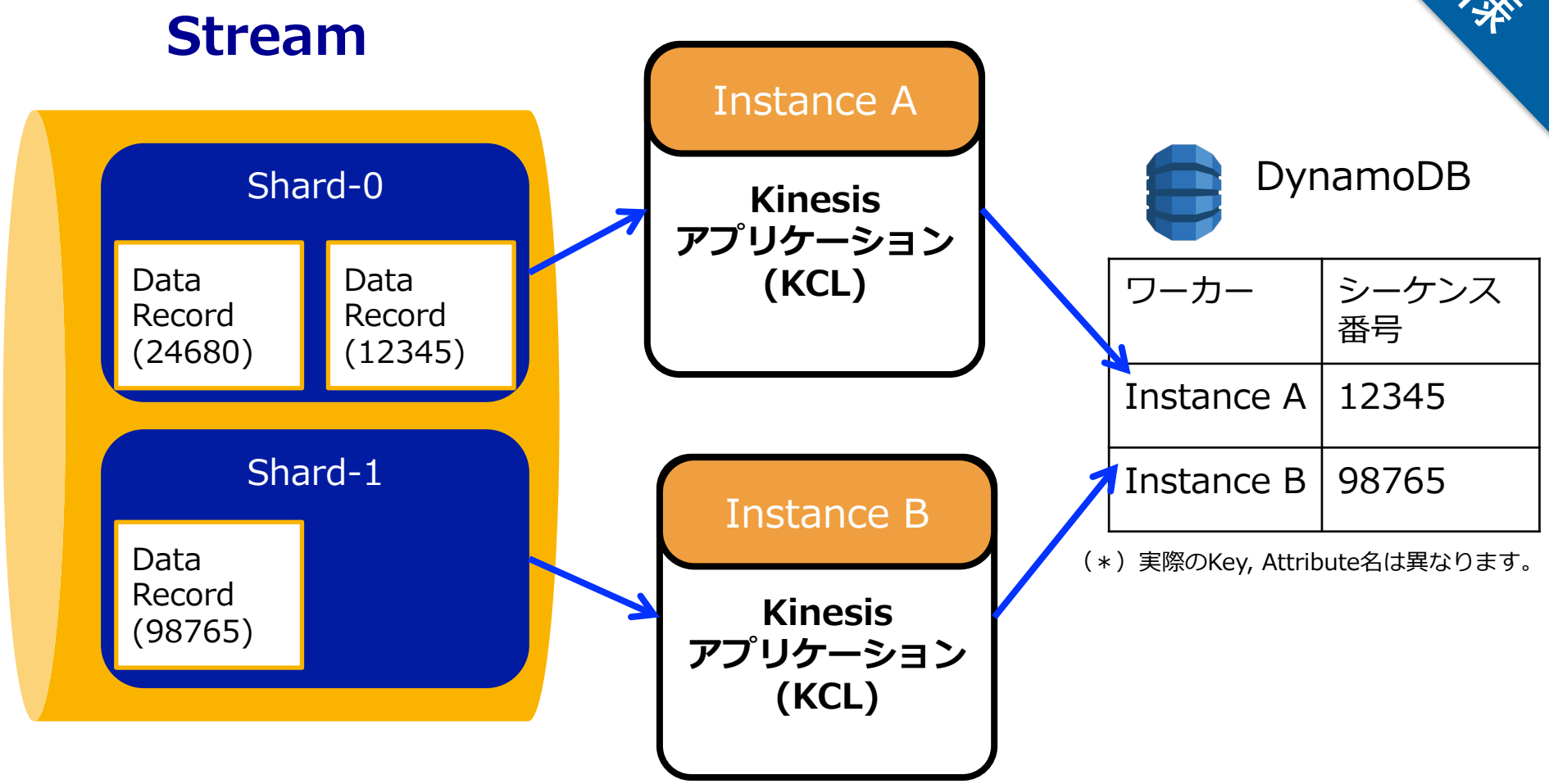


（＊）実際のKey, Attribute名は異なります。

1. Kinesis Client LibraryがShardからData Recordを取得
2. 設定された間隔でシーケンス番号をそのワーカーのIDをキーにしたDynamoDBのテーブルに格納
3. １つのアプリが複数Shardからデータを取得し処理を実行

Kinesis Client Libraryの動き

仕様

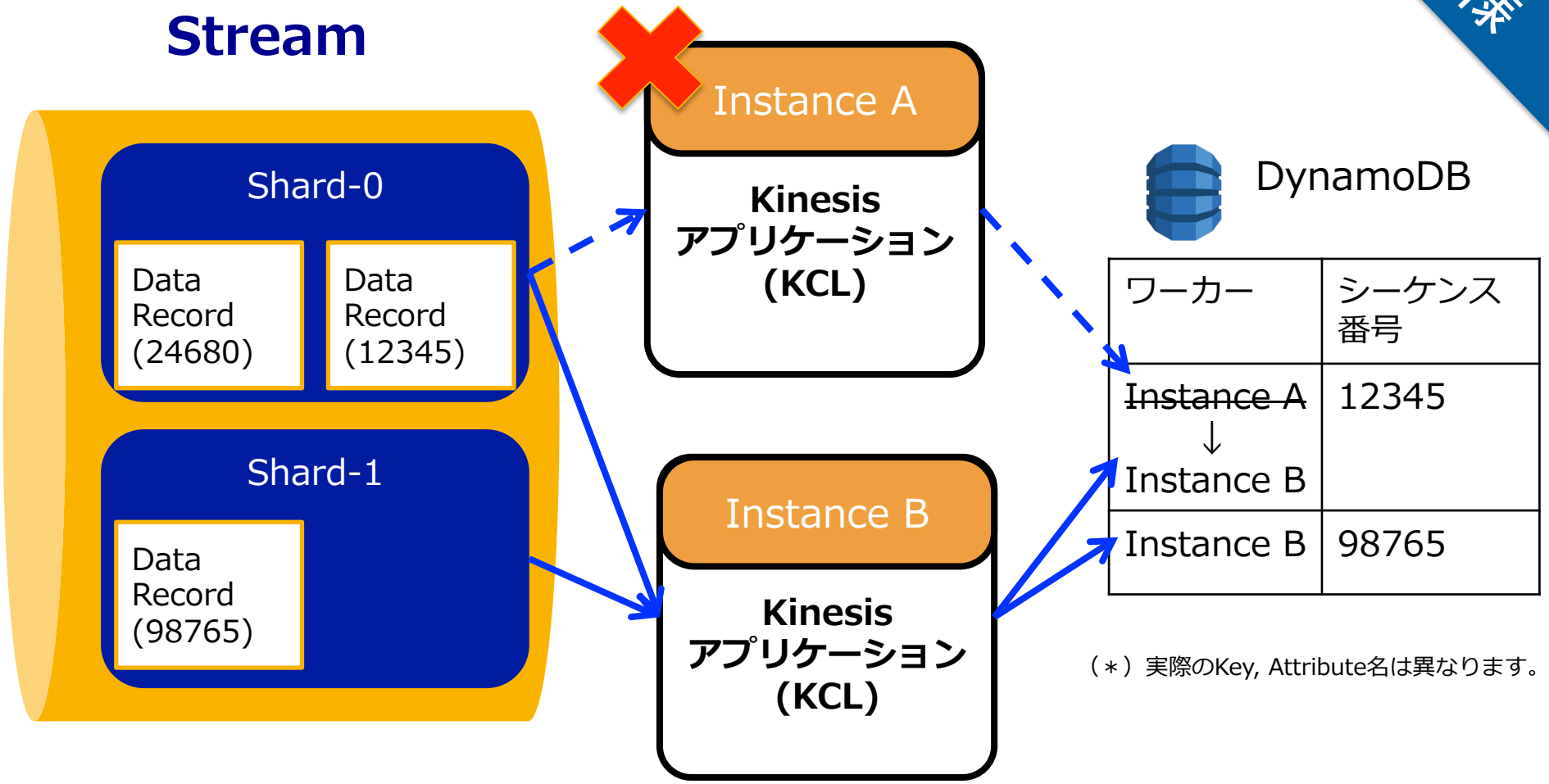


（＊）実際のKey, Attribute名は異なります。

1. 複数アプリを実行した場合は、負荷分散される

Kinesis Client Libraryの動き

仕様



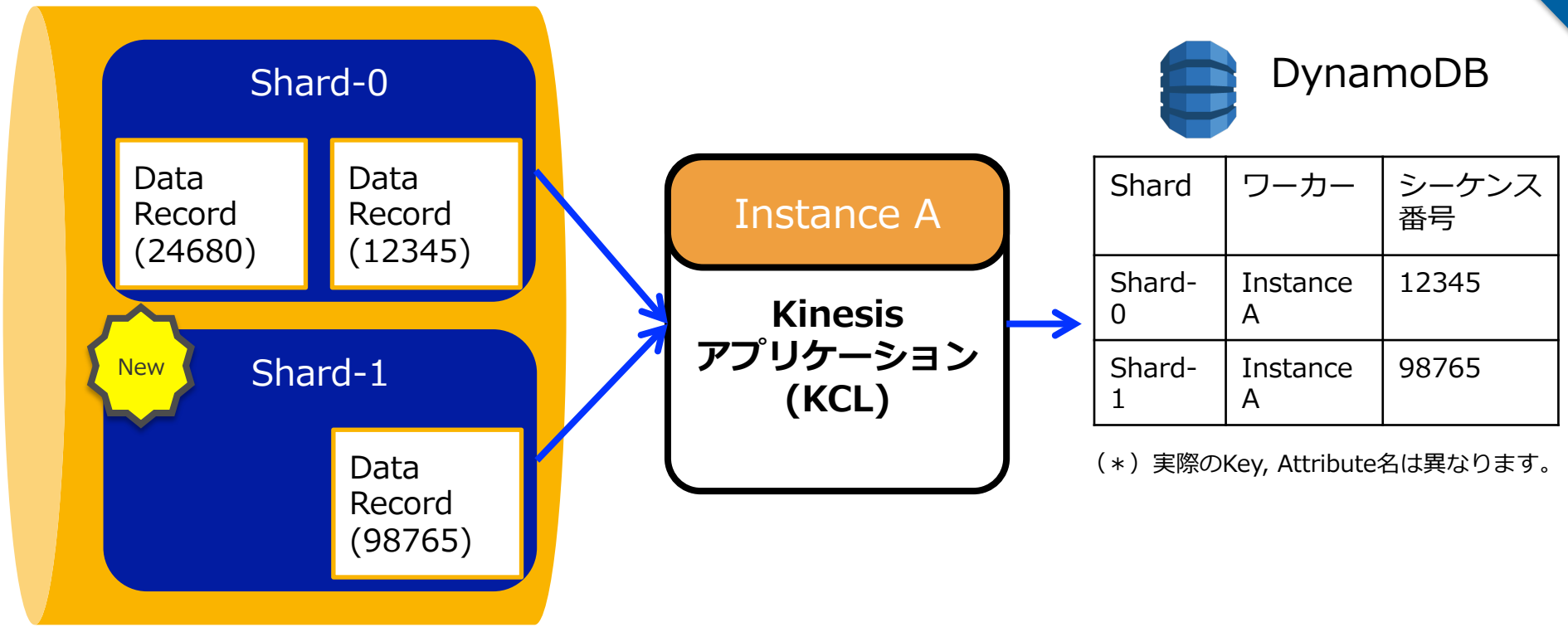
（＊）実際のKey, Attribute名は異なります。

Instance Aがデータ取得されない状況を検知し、Instance Bが、DynamoDBに格納されているシーケンス番号からデータ取得を行う

Kinesis Client Libraryの動き–拡張性

仕様

Stream



Shard	ワーカー	シーケンス番号
Shard-0	Instance A	12345
Shard-1	Instance A	98765

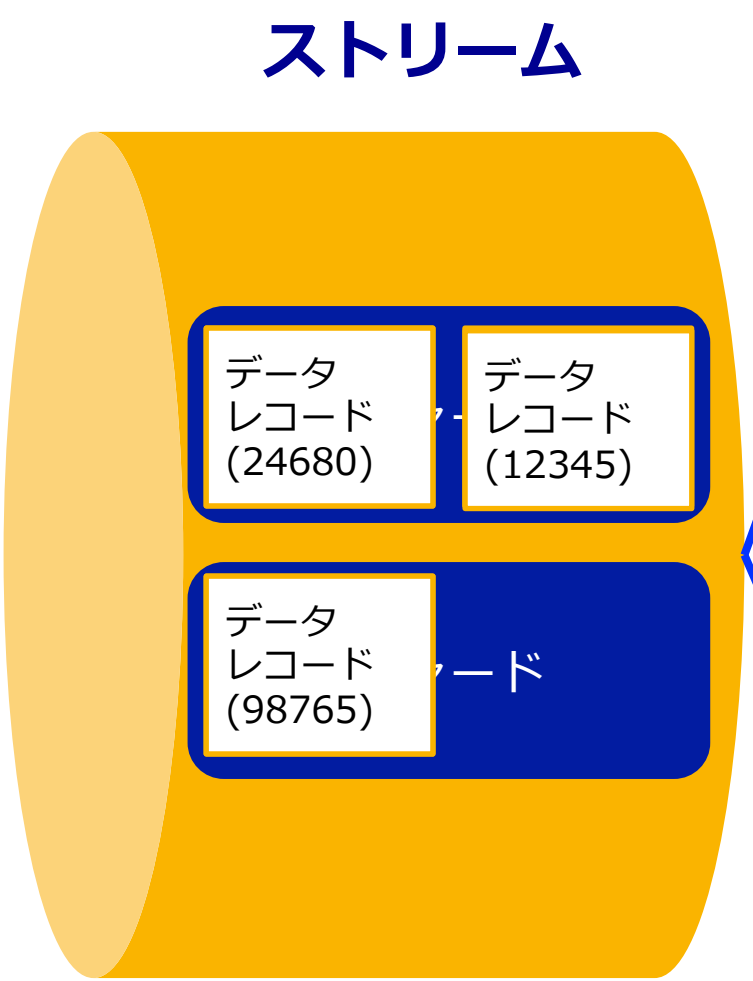
（＊）実際のKey, Attribute名は異なります。

Shard-1を増やしたことを検知し、データ取得を開始し、Shard-1のチェックポイント情報をDynamoDBに追加

目的に応じてKinesisアプリケーションを追加可能

仕様

DynamoDB



Instance A

アーカイブアプリ (KCL)

Archive Table

Shard	ワーカー	シーケンス番号
Shard-0	Instance A	12345
Shard-1	Instance A	98765

各アプリ毎に別テーブルで管理される

Instance A

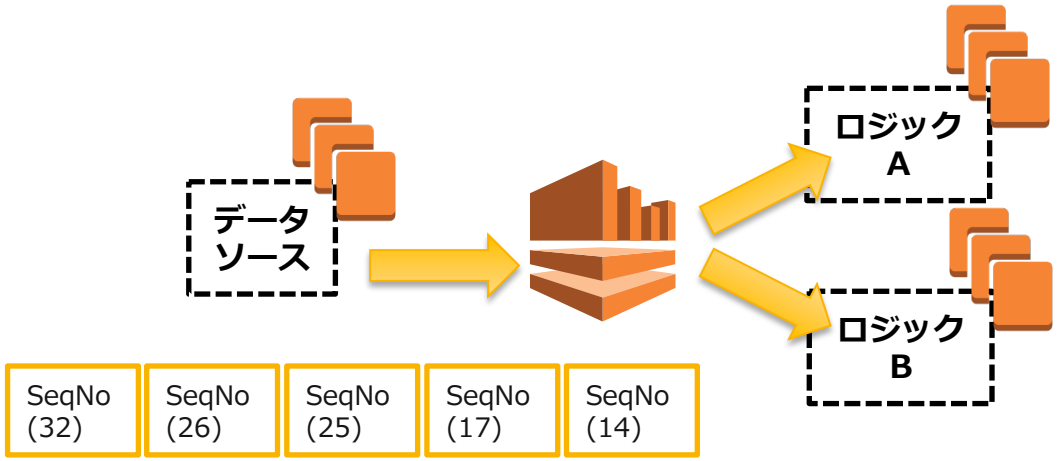
集計アプリ (KCL)

Calc Table

Shard	ワーカー	シーケンス番号
Shard-0	Instance A	24680
Shard-1	Instance A	98765

【前提】データ処理デザインパターン

- データ処理を行うアプリケーション側でリカバリーやロードバランシングを考慮した設計が必要
- Kinesisの特徴であるシリアル番号を利用しチェックポイントを打つことが重要
- １つのデータを複数のアプリケーションで利用できるためアプリケーション毎に追加・削除できる設計
- 本番データを用いて開発中のロジックの評価や複数のロジックを同じデータを用いて評価することが可能



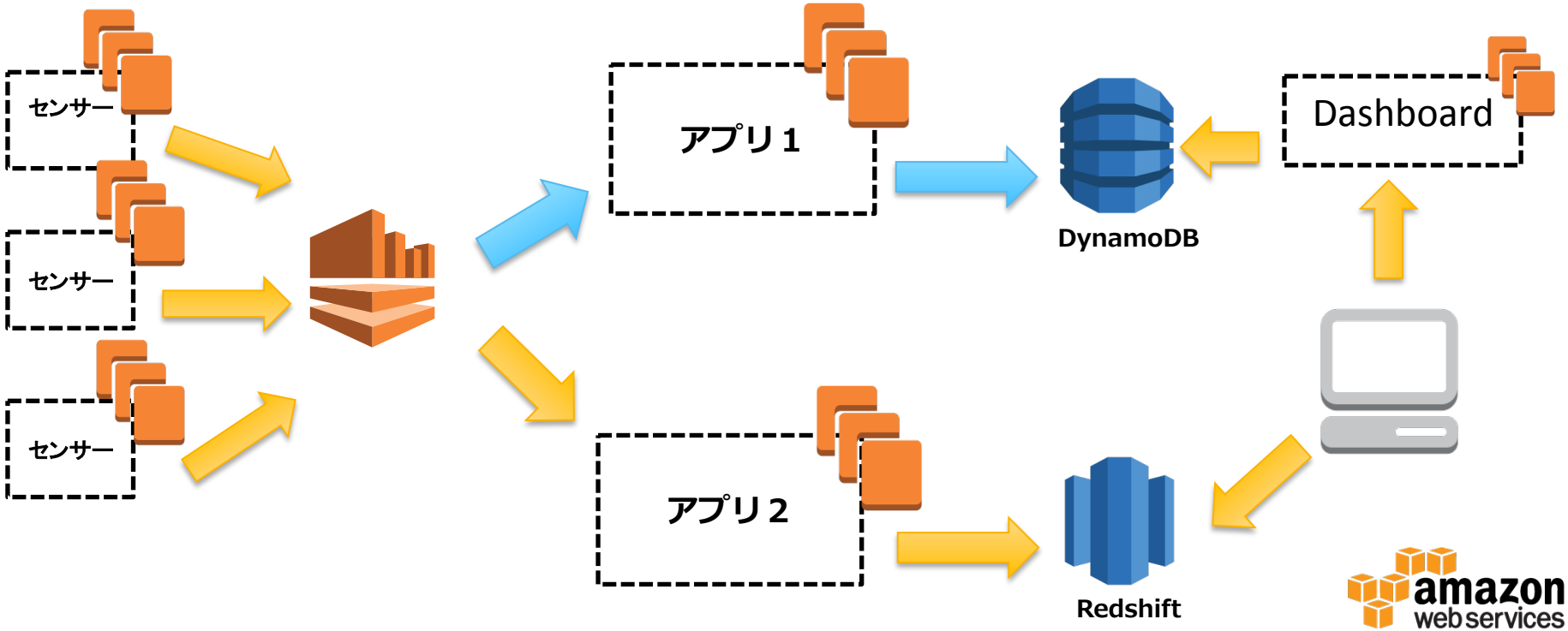
データ処理パターン分類

分類	ユースケース
データ処理パターン１ （基本パターン）	目的毎にKinesisアプリケーションを配置するパターン – Kinesisのデータ処理側の基本構成
データ処理パターン２ （Hadoop処理パターン）	KinesisでIngestされたデータをHadoopを用いてデータ処理するパターン – ETL、クレンジング、機械学習
データ処理パターン３ （Kinesis パイプラインパターン）	複数のKinesisをつなぎあわせてデータ処理するパターン – 複数コンポーネントを直列処理（フィルタして集計するなど）
データ処理パターン４ （Stormインテグレーションパターン）	KinesisでIngestされたデータをStormを用いてデータ処理するパターン – リアルタイムETL、機械学習
データ処理パターン５ （機械学習インテグレーションパターン）	オンラインとオフラインを組み合わせた機械学習インテグレーションパターン – 異常検知、リコメンデーション

データ処理パターン１

- 目的毎にアプリケーションを構成するパターン
- それぞれのアプリの可用性・信頼性に合わせた設計
- DynamoDB、Redshift、S3などとのインテグレーションを容易にする Kinesis Connector Libraryを利用可能
  https://github.com/awslabs/amazon-kinesis-connectors

例：リアルタイムダッシュボード

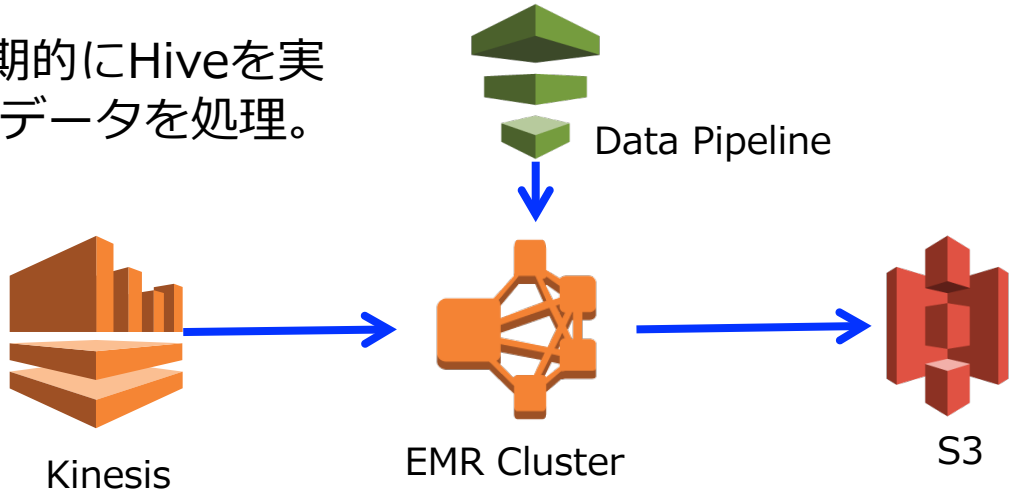


データ処理パターン２

- Hadoopを用いたETLパターン
- Kinesisに集積されたデータをHive、PigなどのHadoopツールを用いてETL処理（Map Reduce処理）が可能
- 別のKinesis Stream, S3, DynamoDB, HDFSのHive Tableなどの他のデータソースのテーブルとJOINすることなども可能
- Data pipeline / Crontabで定期実行することにより、定期的にKinesisからデータを取り込み、処理することが可能

構成例

DataPipelineで定期的にHiveを実行しKinesisにあるデータを処理。
結果をS3に格納

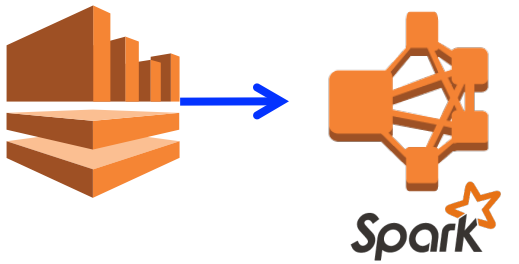


EMR AMI 3.0.4以上を用いることでKinesisインテグレーションが可能

amazon web services

Kinesis – Spark インテグレーション

- データ処理パターン２と同様
- Apache SparkをEMR上で実行するパターン
- Apache Hadoopより高速にMapReduceの処理が可能
- ブログ
  https://aws.amazon.com/articles/Elastic-MapReduce/4926593393724923



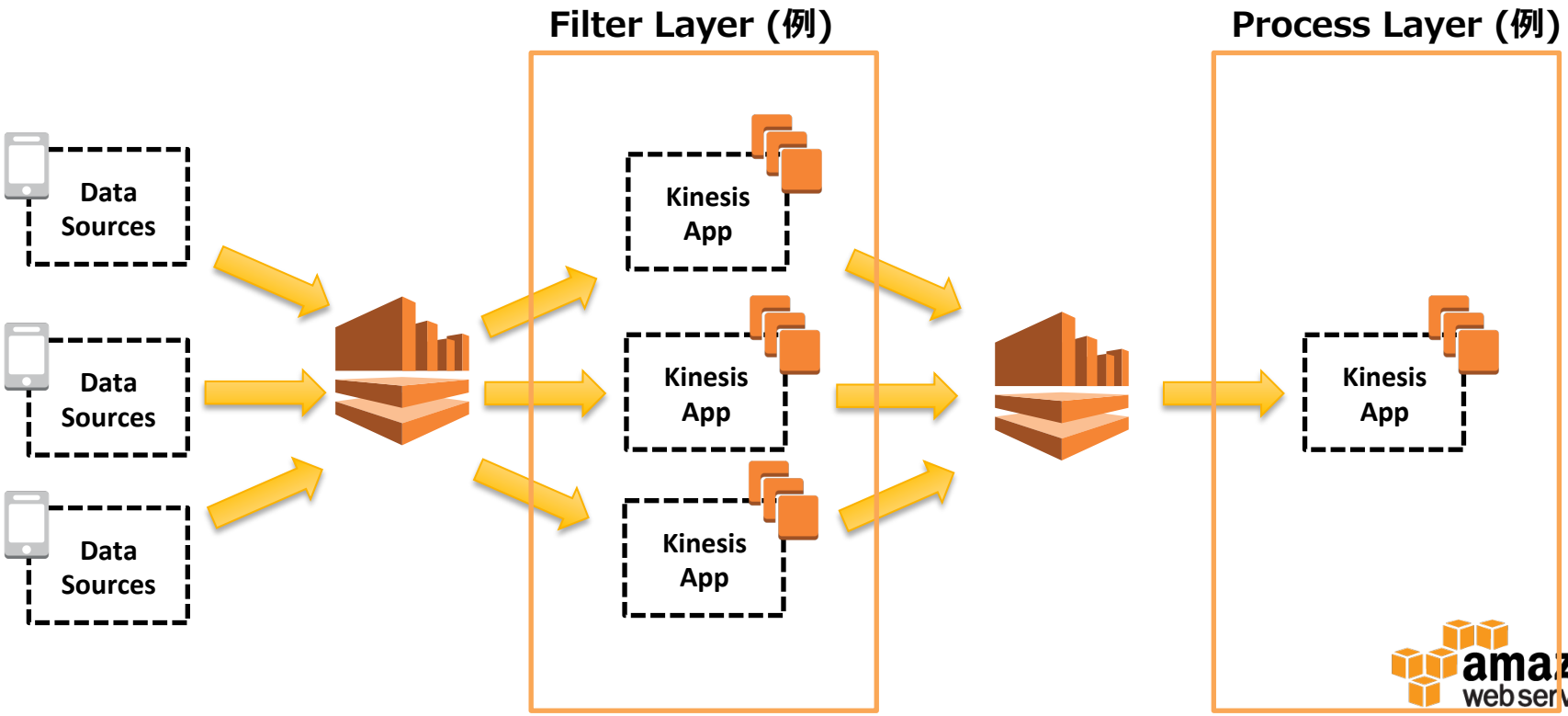
Kinesis　　EMR Cluster

WordCount Sample(抜粋)

```
def main(args: Array[String]) {
  val ssc = new StreamingContext(args(0), "KinesisWordCount",
                Seconds(30),System.getenv("SPARK_HOME"),
                Seq(System.getenv("SPARK_EXAMPLES")))
  var canalClient  = new AmazonKinesisClient(getCredentials());
  val describeStreamRequest = new DescribeStreamRequest().withStreamName(args(1));
  val describeStreamResult = canalClient.describeStream(describeStreamRequest);
  val shards = describeStreamResult.getStreamDescription().getShards().toList
  val stream = ssc.union(for (shard <- shards) yield (ssc.networkStream[String](new
                KinesisInputDStream("none",args(1),shard.getShardId()))))
  val hashTags = stream.flatMap(status => status.split(" "))
  val wordCount = hashTags.map((_, 1)).reduceByKeyAndWindow(_ + _, Seconds(60))
              .map{case (topic, count) => (count, topic)}
  wordCount.print
  ssc.start()
}
```

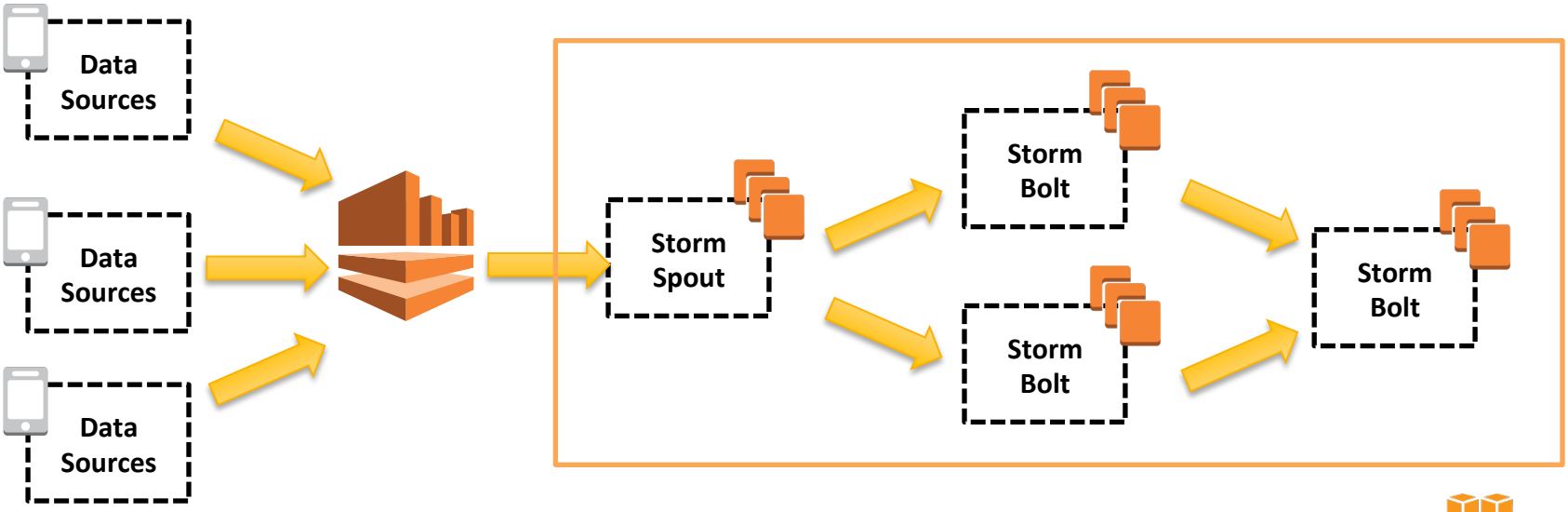
データ処理パターン３

- Kinesisをパイプラインとして連結するパターン
- FilterやMapReduceを多段Kinesisを用いて実現
- 最初のKinesisは、ピークトラフィックに対応しやすくするためにランダムな値をパーティションキーとしてセットし、平準化し、次のストリームを生成し、伝送する



データ処理パターン４

- Apache StormとインテグレーションすることによりリアルタイムETL、データプロセッシングパターン
- Boltをつなげることで高度なデータ処理をリアルタイムで分散処理が可能
- KinesisからApache Stormへのインテグレーションを容易にするためのSpoutを提供
  https://github.com/awslabs/kinesis-storm-spout

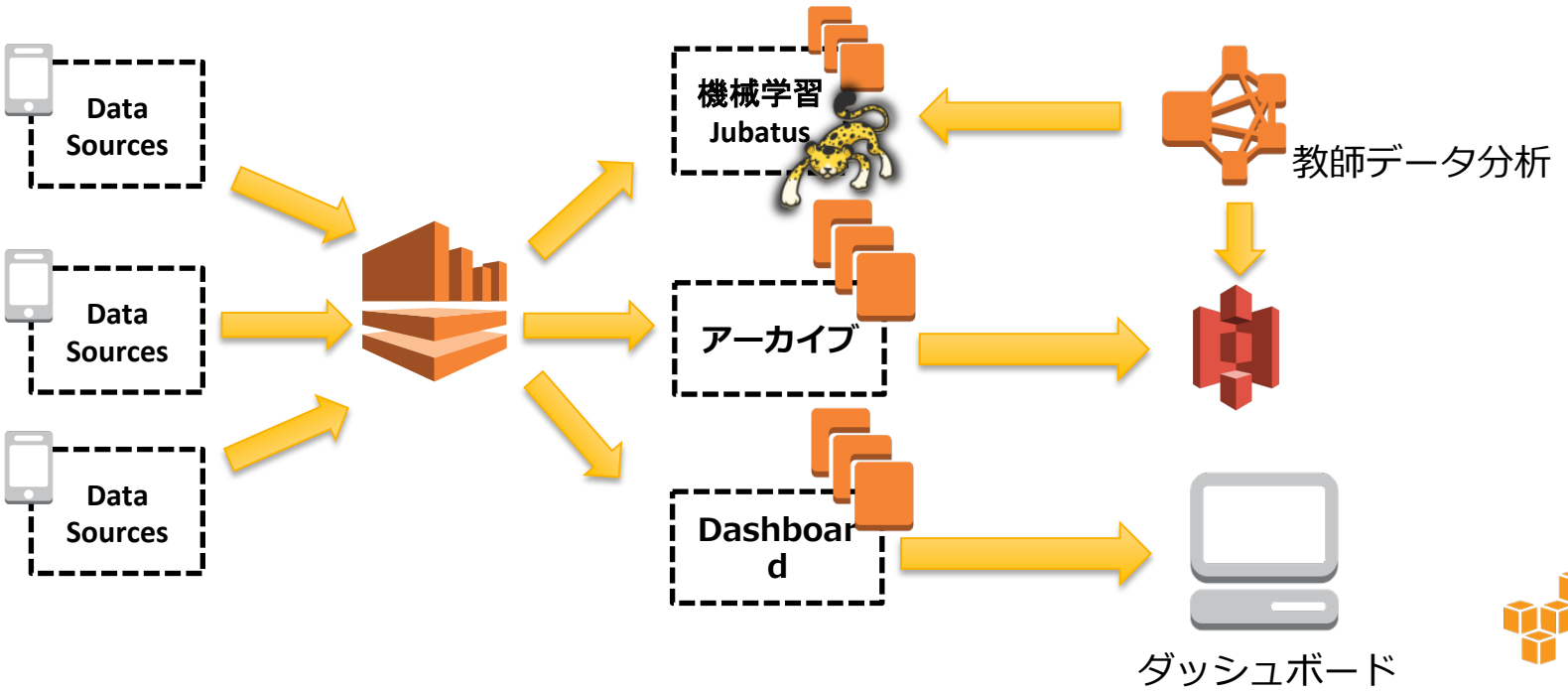


amazon web services

データ処理パターン５

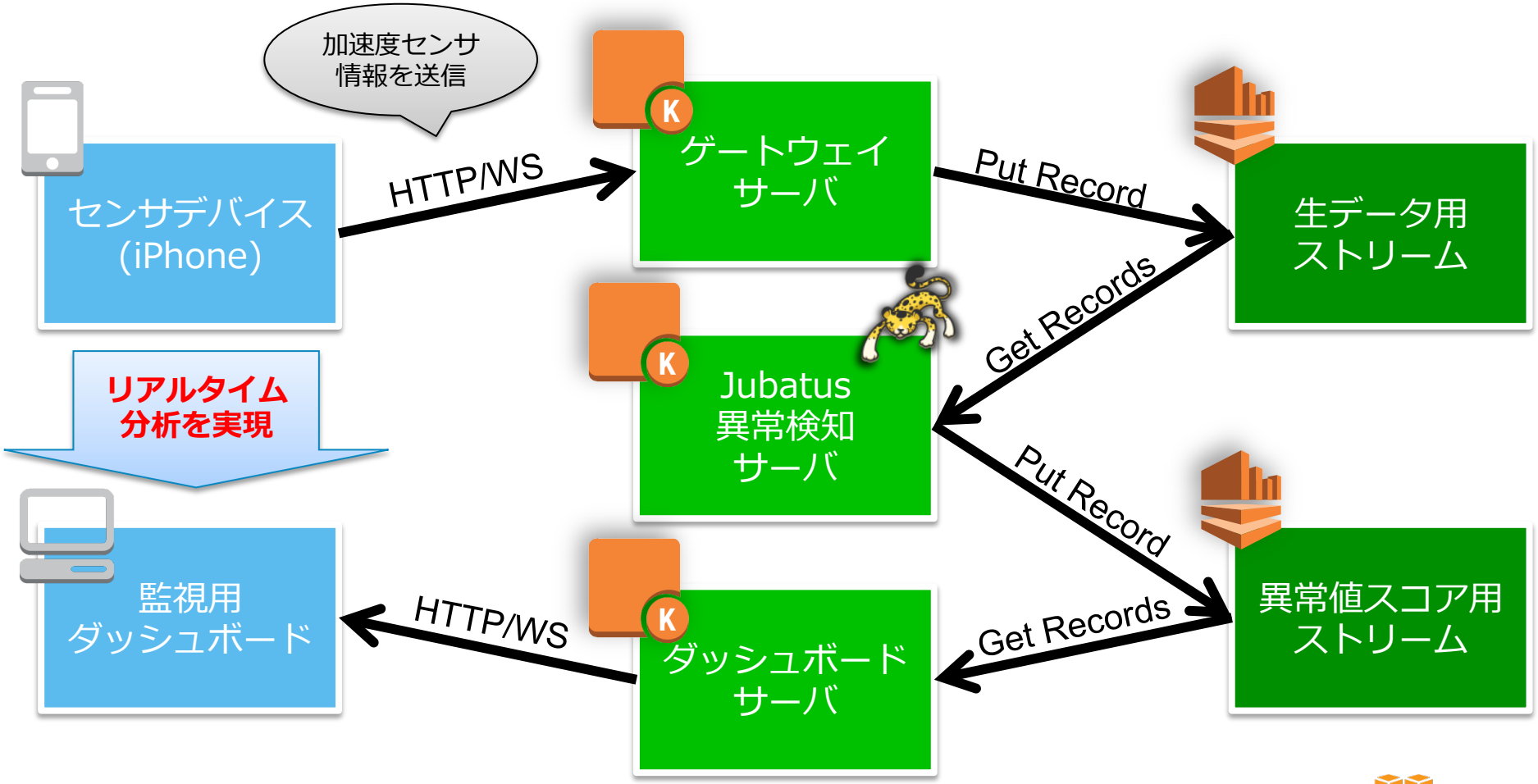
- ストリーミングデータの分類、異常検知などを機械学習を用いて実行
- 機械学習への教師データの反映を定期的に実行
- 機械学習器は、パターン２（Spark）、パターン３の上に動作させることも可能

例：オンライン機械学習（Jubatus）の例



デモ

- 異常検知とアーカイブを目的としたKinesisアプリをEC2上で実行
- ダッシュボードで検知状態を確認



まとめ

クラウドのメリットをフルに活かす

- IoTが産み出す、より多くのデータを収集・活用することで、ビジネス価値がうまれる
- AWSを活用することで、データを効率よく収集し、分析することが実現できる
- 特に、１つのデータ・ソースをアジリティ高く試すことができるアーキテクチャが重要

参考資料

- Amazon Kinesis API Reference
  - http://docs.aws.amazon.com/kinesis/latest/APIReference/Welcome.html
- Amazon Kinesis Developer Guide

  http://docs.aws.amazon.com/kinesis/latest/dev/introduction.html
- Amazon Kinesis Forum

  https://forums.aws.amazon.com/forum.jspa?forumID=169#

amazon web services