[AWS Black Belt Online Seminar] Serverless モニタリング

ソリューションカットシリーズ

Solutions Architect 下川 賢介
2019/8/20

AWS 公式 Webinar
https://amzn.to/JPWebinar

過去資料
https://amzn.to/JPArchive

自己紹介

- 名前
  下川 賢介
- 所属
  アマゾン ウェブ サービス ジャパン 株式会社
  技術統括本部
  Serverless Specialist Solutions Architect

- 好きなAWSのサービス

AWS Lambda　Amazon API Gateway　Amazon CloudWatch

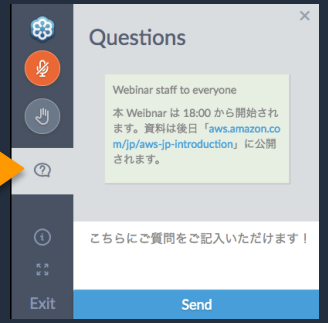
AWS Black Belt Online Seminar とは

「サービス別」「ソリューション別」「業種別」のそれぞれのテーマに分かれて、アマゾン ウェブ サービス ジャパン株式会社が主催するオンラインセミナーシリーズです。

質問を投げることができます！

- 書き込んだ質問は、主催者にしか見えません
- 今後のロードマップに関するご質問は
  お答えできませんのでご了承下さい

① 吹き出しをクリック
② 質問を入力
③ Sendをクリック



Twitter ハッシュタグは以下をご利用ください
#awsblackbelt

内容についての注意点

- 本資料では2019年8月20日時点のサービス内容および価格についてご説明しています。最新の情報はAWS公式ウェブサイト(http://aws.amazon.com)にてご確認ください。

- 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が東京リージョンを使用する場合、別途消費税をご請求させていただきます。

- AWS does not offer binding price quotes. AWS pricing is publicly available and is subject to change in accordance with the AWS Customer Agreement available at http://aws.amazon.com/agreement/. Any pricing information included in this document is provided only as an estimate of usage charges for AWS services based on certain information that you have provided. Monthly charges will be based on your actual use of AWS services, and may vary from the estimates provided.

本日のアジェンダ

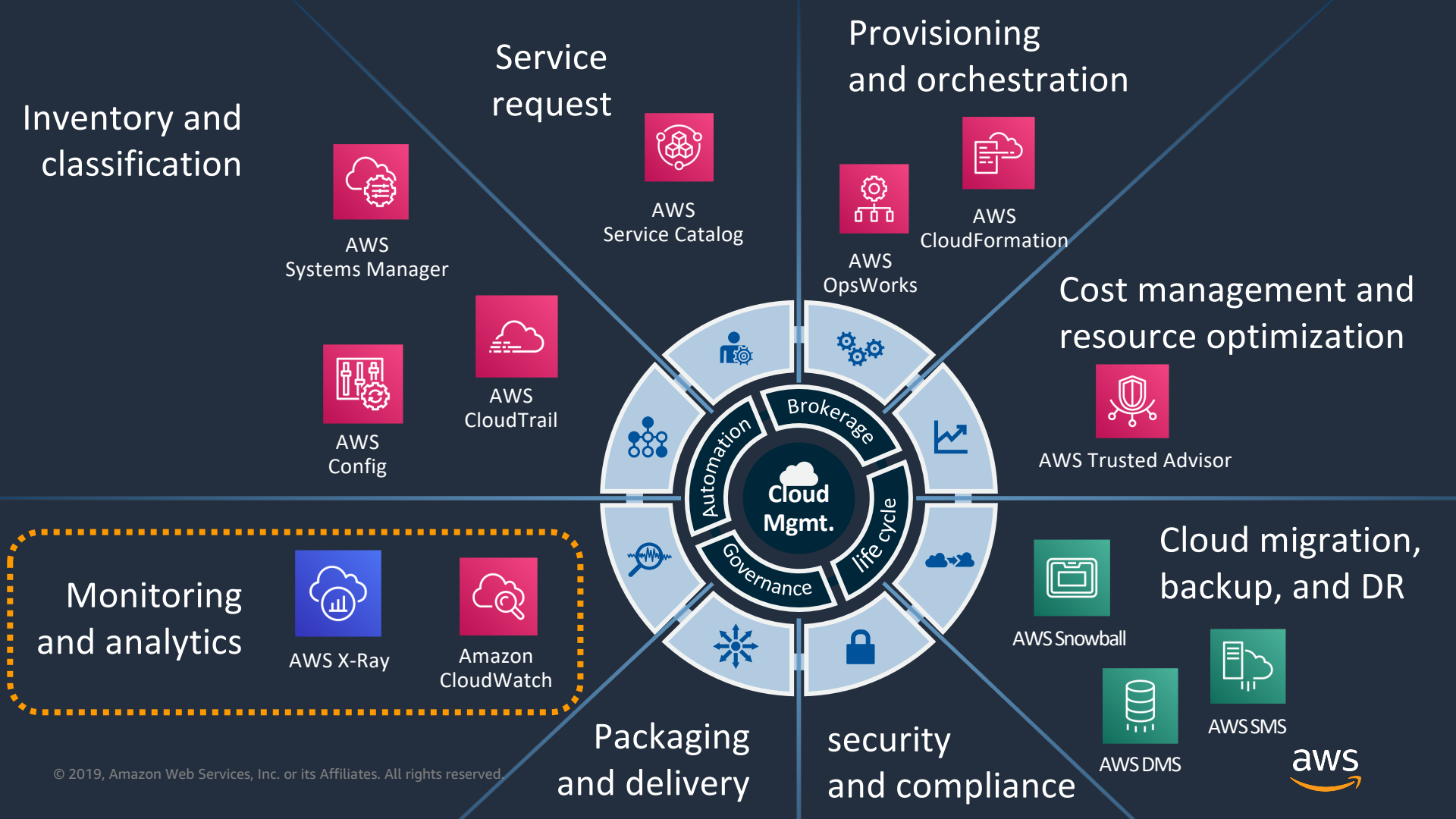
- モニタリングについて
- Metricsの理解
- Log運用と可視化
- Traceを使った可視化

モニタリングについて

モニタリングの目的

担当者のロールによって、モニタリングをする意味も大きく異なってきます。

- ユーザ体験の把握
- 監査対応
- 品質向上
- 不正アクセス検知
- コンプライアンス準拠
- コスト効率
- 性能劣化検知



多くの 3rdParty ツールとの連携が可能

aws

誰がモニタリングするのか

伝統的なシステムでは開発と運用の間に壁がある。
そんな壁はぶん投げてしまえ。
忘れてしまってもいい。Amazonでは不要だ。

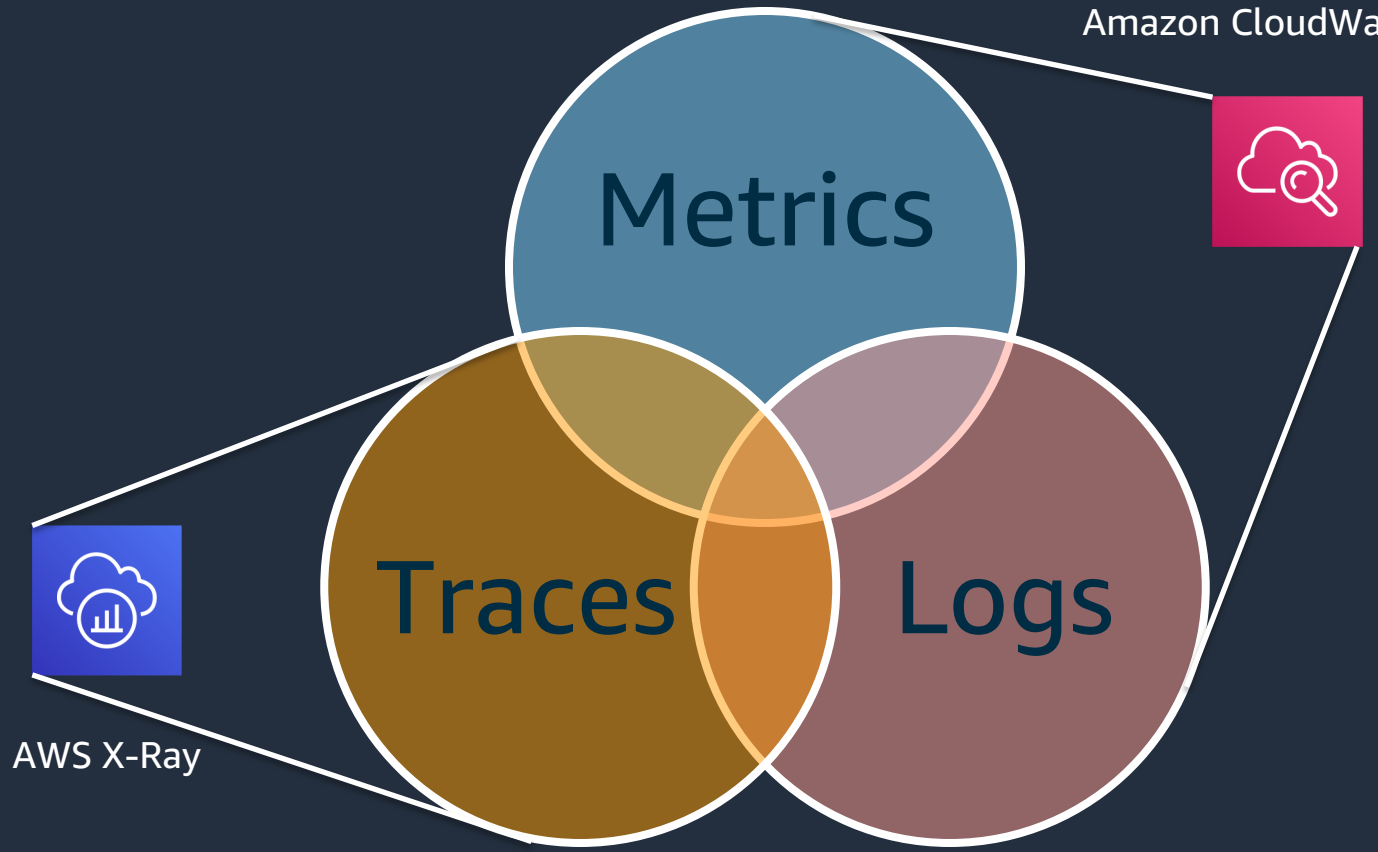
you build it, you run it

開発者も日々の運用に参加しなさい。
そうすればカスタマーのフィードバックが得られる。
これが、サービスの質を高めるのだ。

-- Werner Vogels in May 2006 / Amazon.com CTO

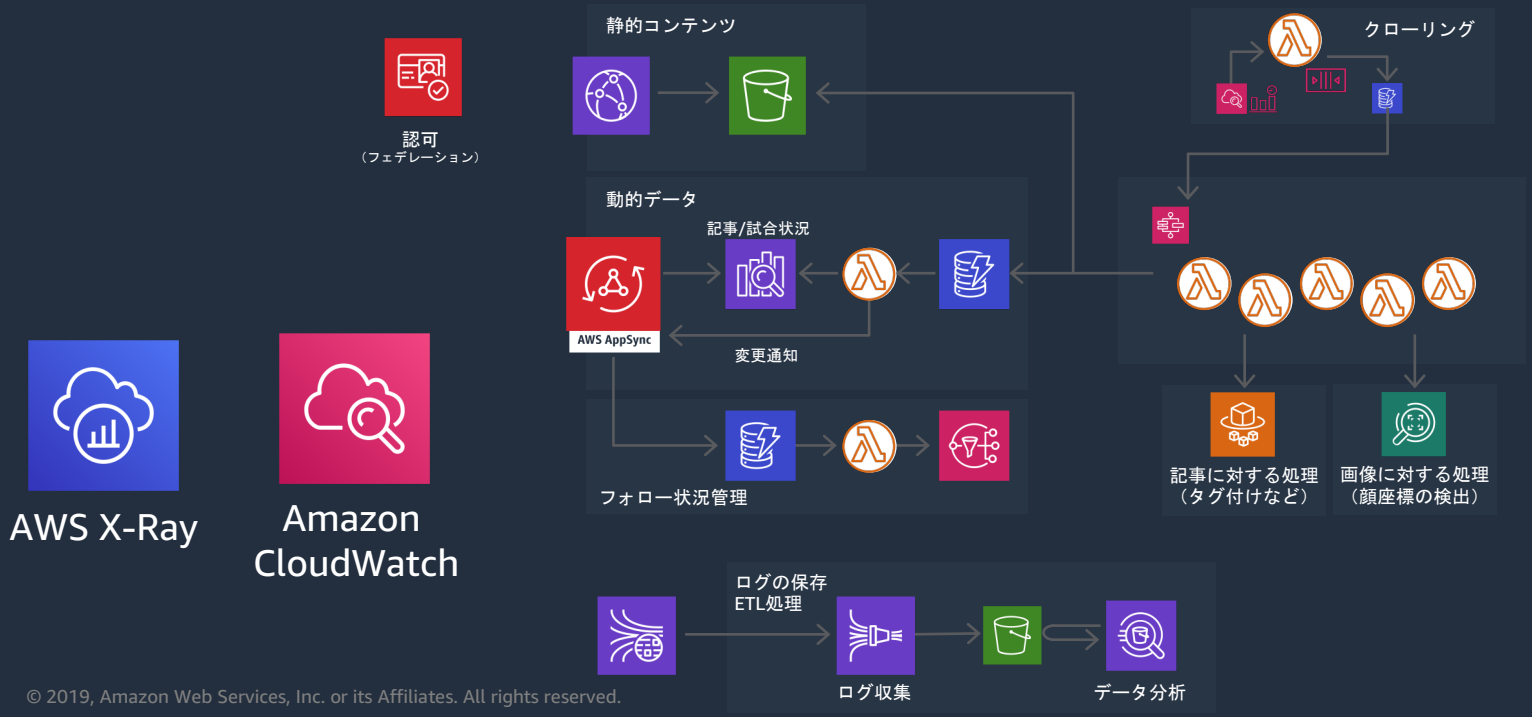
aws

モニタリングサービス

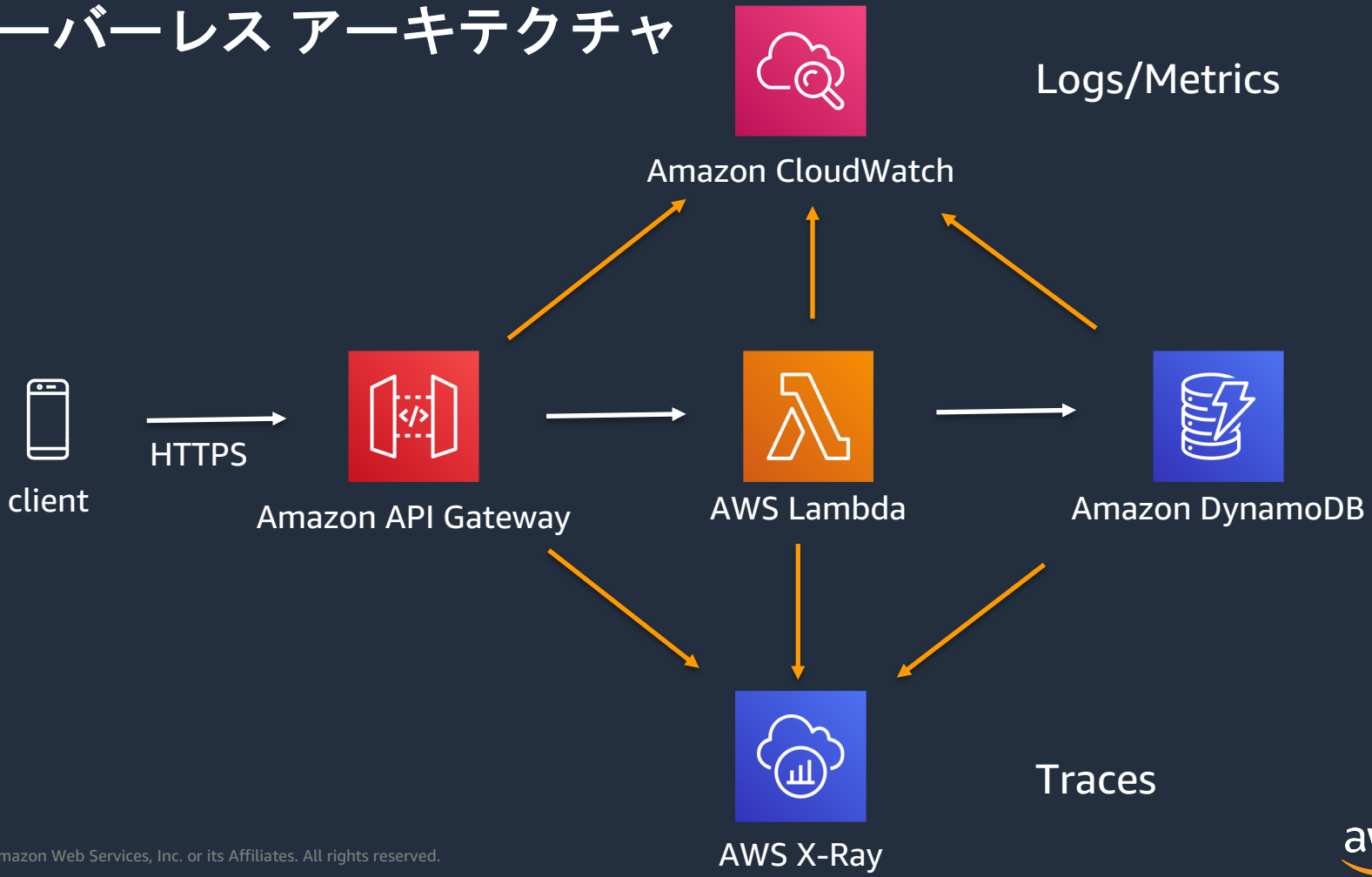


サーバーレスのモニタリング

サーバーレス アーキテクチャは、多くの場合分散しており、個々の分散パーツがモニタリングを必要とします。



サーバーレス アーキテクチャ




Metricsの理解

AWS Lambdaのメトリクス

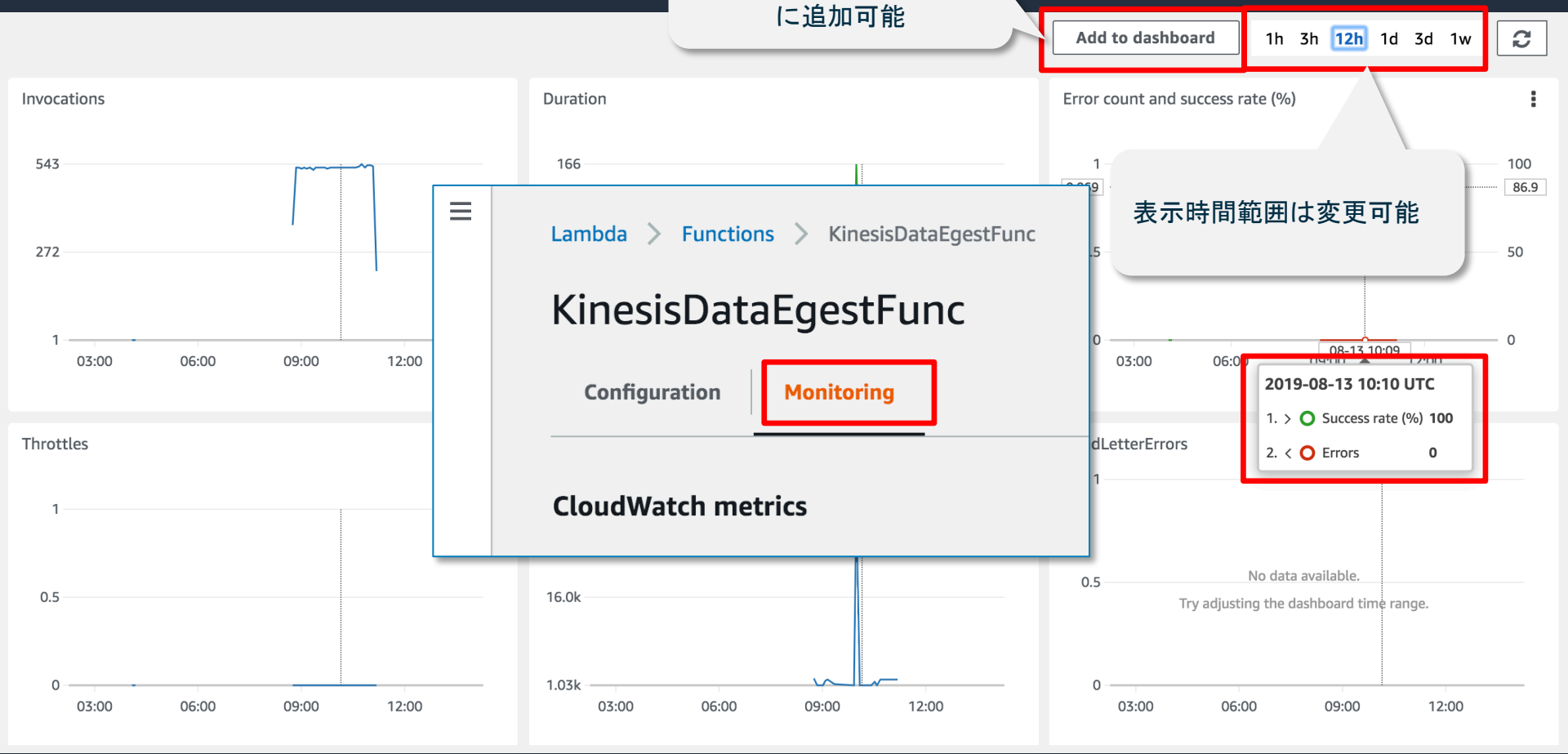
aws

AWS Lambda の Metrics

項目	単位	意味
Invocations	count	Lambdaの実行回数を計測したもの。=課金カウント **≠同時実行数**
Duration	ms	Lambdaの実行時間を計測。（課金は100msで切り上げる） **※コールドスタートのLambda起動までの時間は含まない**
Errors, Availability	count	Lambdaが正常終了しなかった回数を計測したもの Availabilityは%で表示される
Throttles	count	Throttleの発生回数 **アカウントにおける**Lambdaの同時起動数超過が発生している
IteratorAge	ms	ストリーム(Kinesis/DynamoDB)でのみ利用。バッチサイズ分取得した**レコード終端の時刻とLambdaがイベントとして受信した時刻差**で表示される
DeadLetterErrors	count	DLQを設定し、そのDLQへの書き込みが失敗すると増加する。

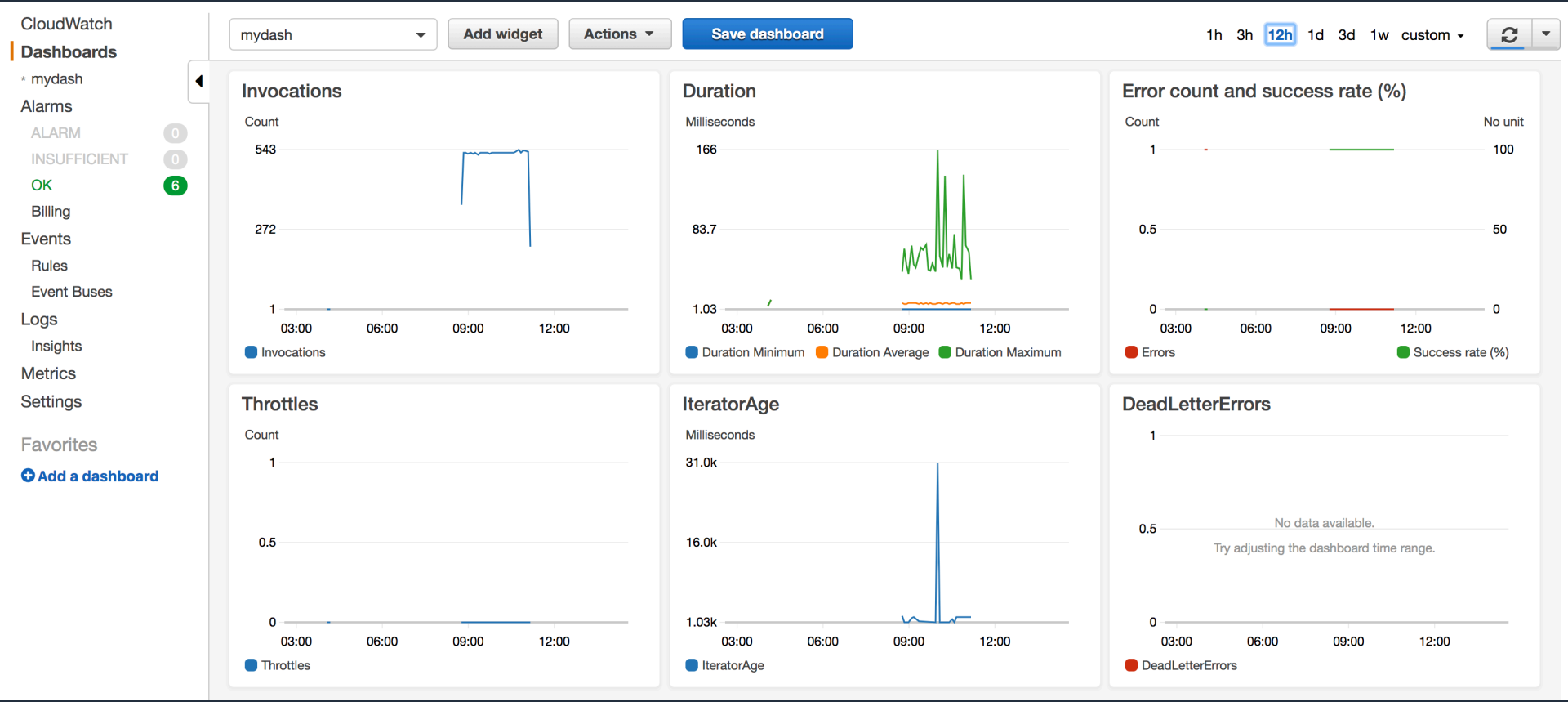
AWS Lambda の Metrics

Amazon CloudWatch Dashboards に追加可能



表示時間範囲は変更可能

Amazon CloudWatch Dashboardsに追加された Metrics



AWS Lambda の Metrics

項目	単位	意味
Invocations	count	Lambdaの実行回数を計測したもの。=課金カウント **≠同時実行数**
Duration	ms	Lambdaの実行時間を計測。（課金は100msで切り上げる） **※コールドスタートのLambda起動までの時間は含まない**
Errors, Availability	count	Lambdaが正常終了しなかった回数を計測したもの Availabilityは%で表示される
Throttles	count	Throttleの発生回数 **アカウントにおける**Lambdaの同時起動数超過が発生している
IteratorAge	ms	ストリーム(Kinesis/DynamoDB)でのみ利用。バッチサイズ分取得した**レコード終端の時刻とLambdaがイベントとして受信した時刻差**で表示される
DeadLetterErrors	count	DLQを設定し、そのDLQへの書き込みが失敗すると増加する。

AWS Lambdaの起動パターン

Invocations

RequestResponseベース

Amazon API Gateway

非同期Eventベース

Amazon Simple Storage Service(S3)

Amazon Simple Notification Service(SNS)

Stream Pollingベース

Amazon Kinesis Data Streams

Amazon DynamoDB Streams

Queue Pollingベース

Amazon Simple Queue Service (SQS)

aws

AWS Lambdaの起動パターン

Invocations

- RequestResponseベース

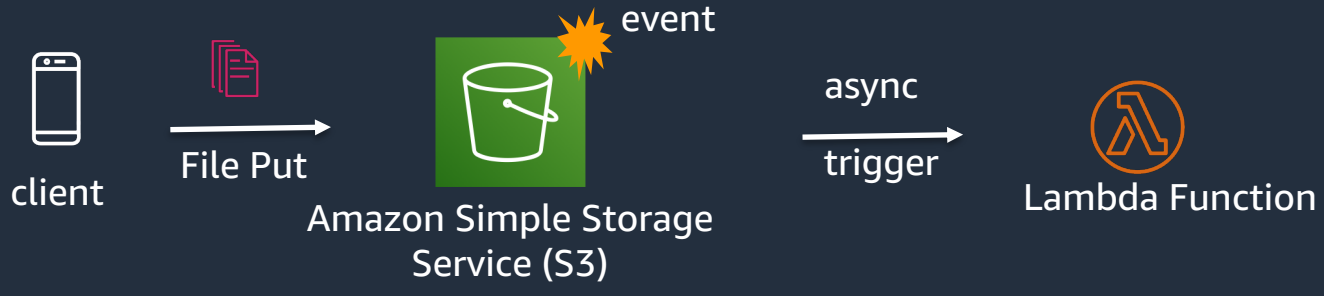
request数：Lambda実行数　＝　n：n
(実行数の比)

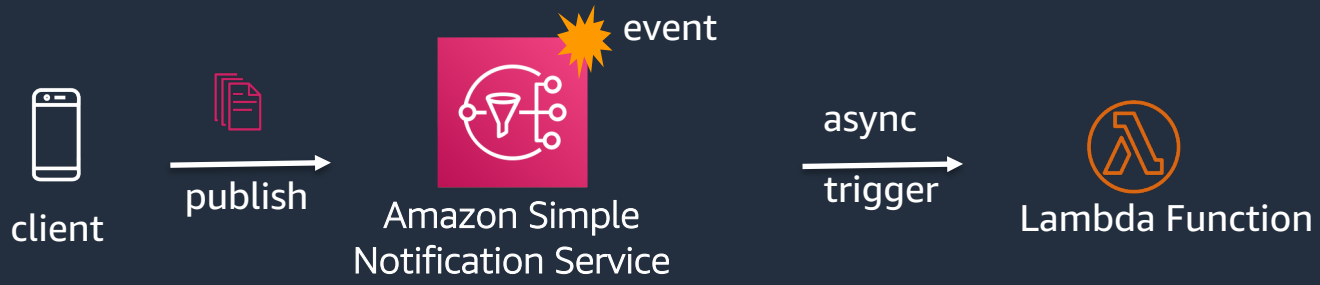
aws

AWS Lambdaの起動パターン

Invocations

❑ 非同期Eventベース





Event数：Lambda実行数　＝　n：n
(実行数の比)

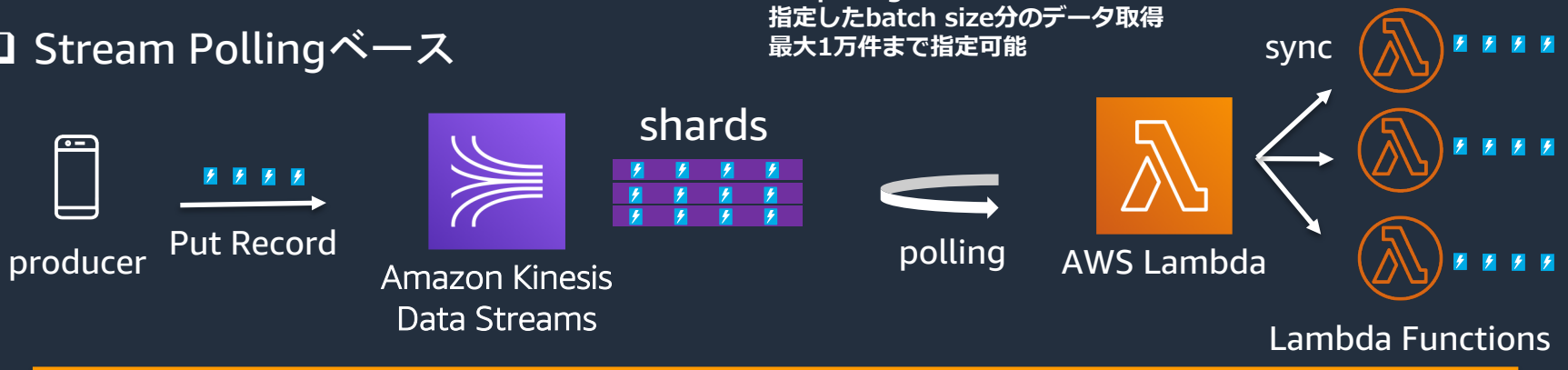
aws

AWS Lambdaの起動パターン

Invocations

❑ Stream Pollingベース

自動polling
指定したbatch size分のデータ取得
最大1万件まで指定可能



Record数：Lambda実行数 ＝ n：n/batch処理数平均 （実行数の比）
Shard数：Lambda同時実行数 ＝ m：m （同時実行数の比）

aws

AWS Lambdaの起動パターン

Invocations

❑ Queue Pollingベース

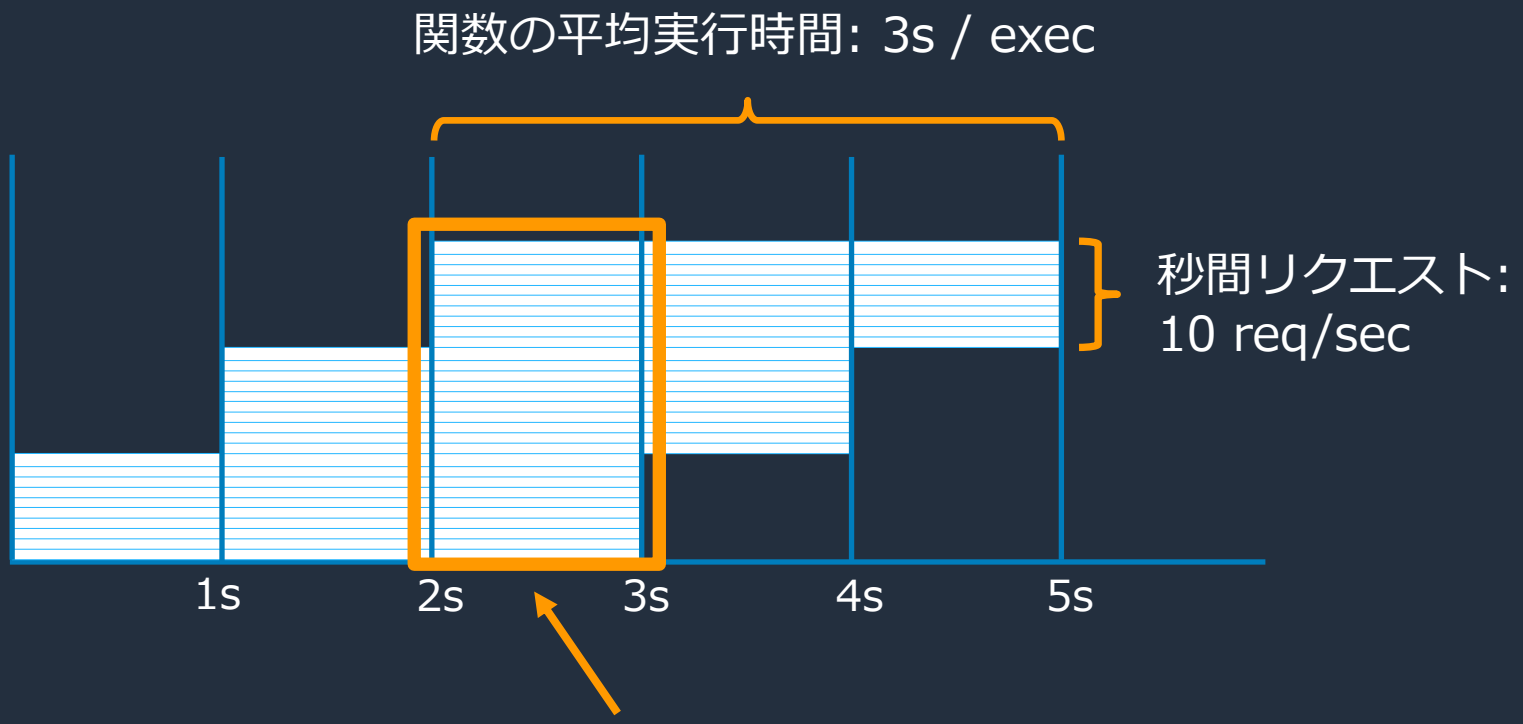
message数：Lambda実行数　＝　n：n /batch処理数平均　（実行数の比）
Queue内のmessage数に応じてLambda 起動数がauto scale（同時実行数）

AWS Lambda の Metrics

項目	単位	意味
Invocations	count	Lambdaの実行回数を計測したもの。=課金カウント **≠同時実行数**
Duration	ms	Lambdaの実行時間を計測。（課金は100msで切り上げる） **※コールドスタートのLambda起動までの時間は含まない**
Errors, Availability	count	Lambdaが正常終了しなかった回数を計測したもの Availabilityは%で表示される
Throttles	count	Throttleの発生回数 **アカウントにおける**Lambdaの同時起動数超過が発生している
IteratorAge	ms	ストリーム(Kinesis/DynamoDB)でのみ利用。バッチサイズ分取得した**レコード終端の時刻とLambdaがイベントとして受信した時刻差**で表示される
DeadLetterErrors	count	DLQを設定し、そのDLQへの書き込みが失敗すると増加する。

同時実行数とは

Throttles



同時実行数 = “同時”に実行されているタイミング
= 3 ✕ 10 = 30

aws

Throttlingの対策について

Throttles

Throttlingの対策としては

- アカウントの同時実行数の制限緩和を申請する
- SQSなどを用いて、非同期にリクエストを処理するようにアーキテクチャを変更する
- Lambda関数をチューニングして平均実行時間を下げる

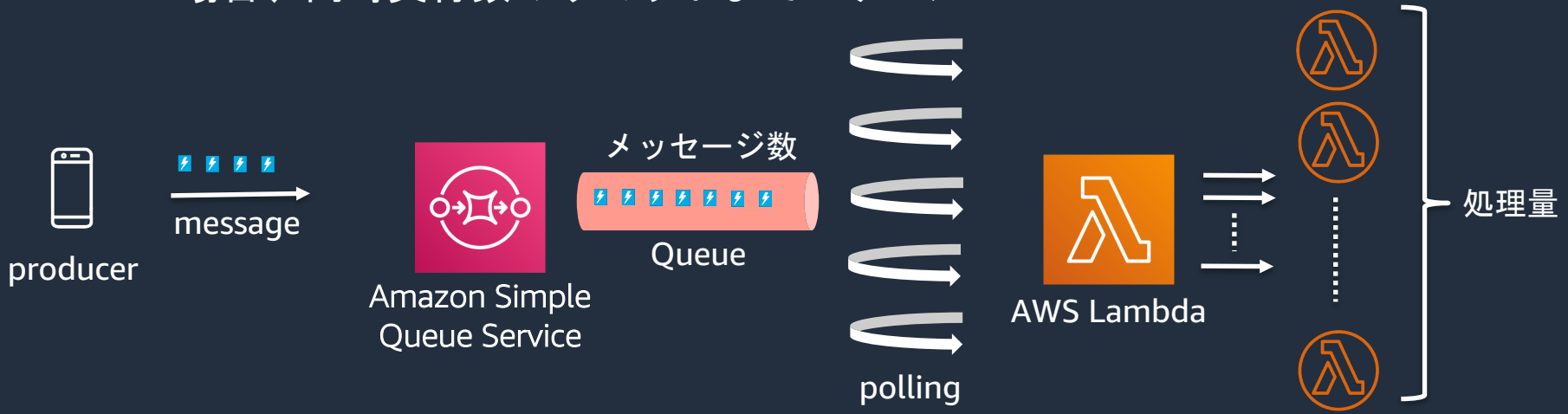
aws

Queue Pollingベースにおける注意事項

Throttles

メッセージ到達時の挙動

- 5つのパラレルロングポーリング接続を使用してSQSのQueueをポーリング
  - メッセージ数 > 処理量 の傾向が続き、最終的に処理が追いつかない場合、同時実行数のリミットまでスケール



aws

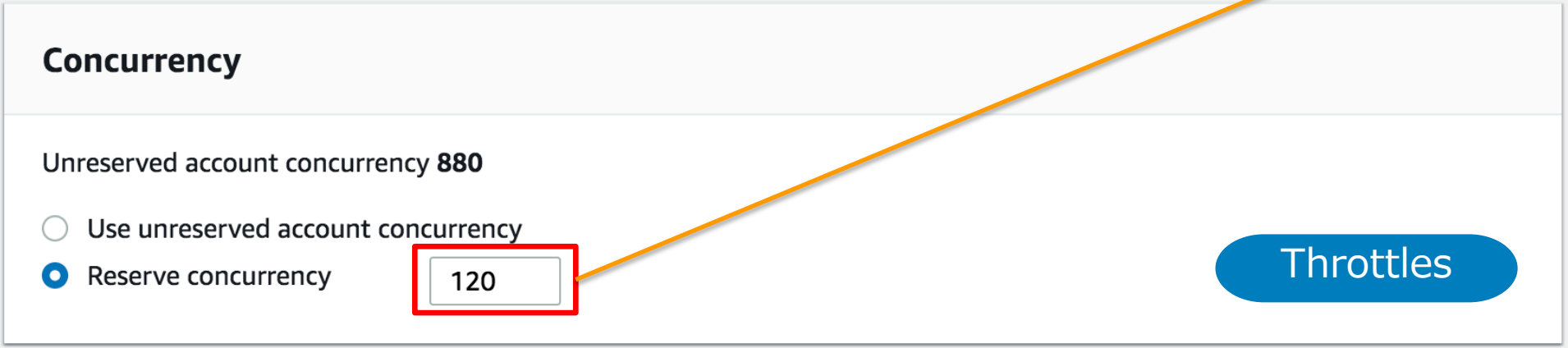
Queue Pollingベースにおける注意事項

自動スケール

- Lambda関数単位に同時実行数を制限しない場合
  - アカウントの上限までスケールする可能性
    - 他のLambda関数の起動を妨げる恐れがある

対処方法

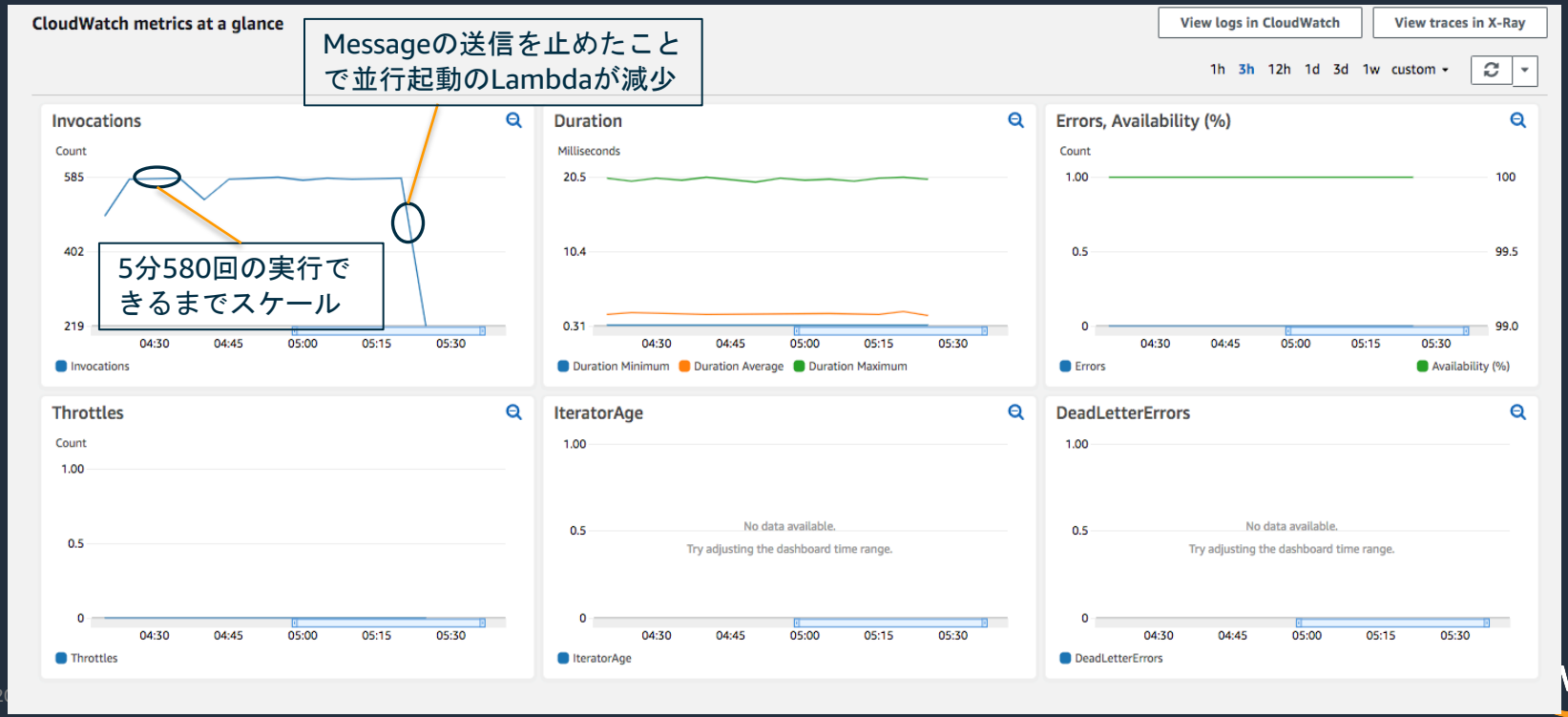
- 大規模に使用する際には同時実行数の設定も検討



Queue Pollingベースでの監視事項

Throttles

- 0.5秒間隔でMessageをQueueに送信
  （metricsは標準の5min）

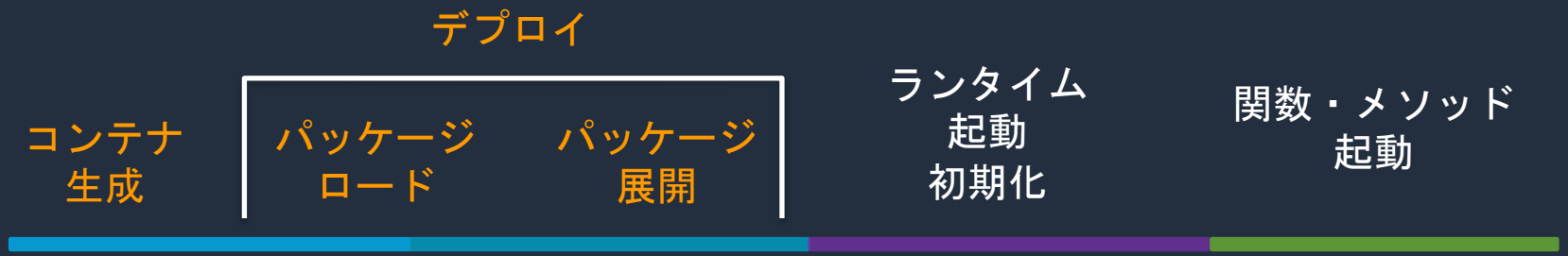


AWS Lambda の Metrics

項目	単位	意味
Invocations	count	Lambdaの実行回数を計測したもの。=課金カウント **≠同時実行数**
Duration	ms	Lambdaの実行時間を計測。（課金は100msで切り上げる） **※コールドスタートのLambda起動までの時間は含まない**
Errors, Availability	count	Lambdaが正常終了しなかった回数を計測したもの Availabilityは%で表示される
Throttles	count	Throttleの発生回数 **アカウントにおける**Lambdaの同時起動数超過が発生している
IteratorAge	ms	ストリーム(Kinesis/DynamoDB)でのみ利用。バッチサイズ分取得した**レコード終端の時刻とLambdaがイベントとして受信した時刻差**で表示される
DeadLetterErrors	count	DLQを設定し、そのDLQへの書き込みが失敗すると増加する。

Lambda Function Lifecycle

Duration



- コンテナ生成
- S3からのZIPダウンロード
- ZIPファイルの展開
- Durationには含まれない

aws

Lambda Function Lifecycle

Duration

- 各ランタイムの初期化処理
- グローバルスコープ処理
- Durationには含まれない

aws

Lambda Function Lifecycle

Duration

- ハンドラーで指定した関数/メソッドの実行
- Duration値は実行時間

aws

Lambda Function Lifecycle

Duration

VPC Lambda Function Lifecycle

Duration

- VPCを利用する場合だけ
- 10秒～30秒かかる
- Durationには含まれない

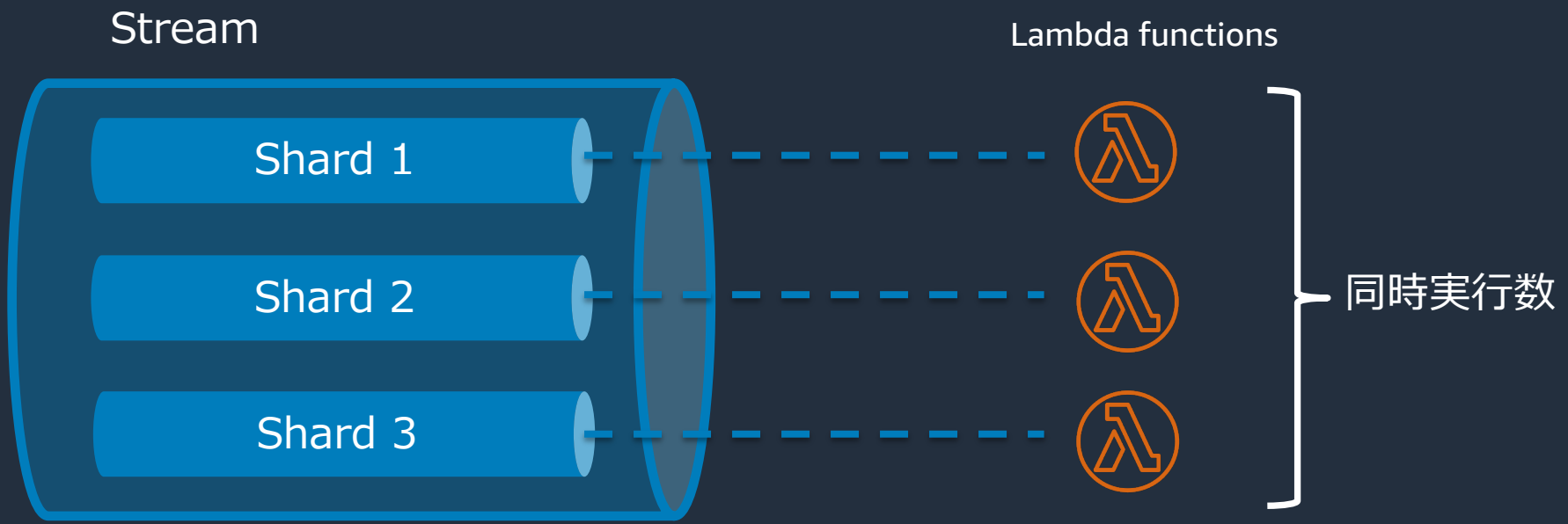
aws

AWS Lambda の Metrics

項目	単位	意味
Invocations	count	Lambdaの実行回数を計測したもの。=課金カウント **≠同時実行数**
Duration	ms	Lambdaの実行時間を計測。（課金は100msで切り上げる） **※コールドスタートのLambda起動までの時間は含まない**
Errors, Availability	count	Lambdaが正常終了しなかった回数を計測したもの Availabilityは%で表示される
Throttles	count	Throttleの発生回数 **アカウントにおける**Lambdaの同時起動数超過が発生している
IteratorAge	ms	ストリーム(Kinesis/DynamoDB)でのみ利用。バッチサイズ分取得した**レコード終端の時刻とLambdaがイベントとして受信した時刻差**で表示される
DeadLetterErrors	count	DLQを設定し、そのDLQへの書き込みが失敗すると増加する。

Stream PollingベースのLambda同時実行数

Errors



aws

Stream PollingベースのErrors Metrics監視

Errors

Q: AWS Lambda は Amazon Kinesis ストリームおよび Amazon DynamoDB ストリームからのデータをどのように処理しますか?

(snip)

あるレコードに対する呼び出しがタイムアウトになったり制限されたりした場合、またはその他のエラーが発生した場合、Lambda は成功するまで (またはレコードが 24 時間の有効期限切れになるまで) 次のレコードに移動しません。異なるシャードにあるレコード間の順序は保証されず、各シャードの処理は並行して行われます。

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

Stream Pollingベースでの監視・運用ポイント

Errors

Errors metricsが上がった場合は、処理がブロックされている恐れがある。

- Lambdaのプログラム改修
  - Streamの場合、Errorsを挙げないように設計/テストをすることが重要
    - CloudWatch Logsにエラーレベルで記録
    - SQSに異常レコードを登録して、Shardは進める

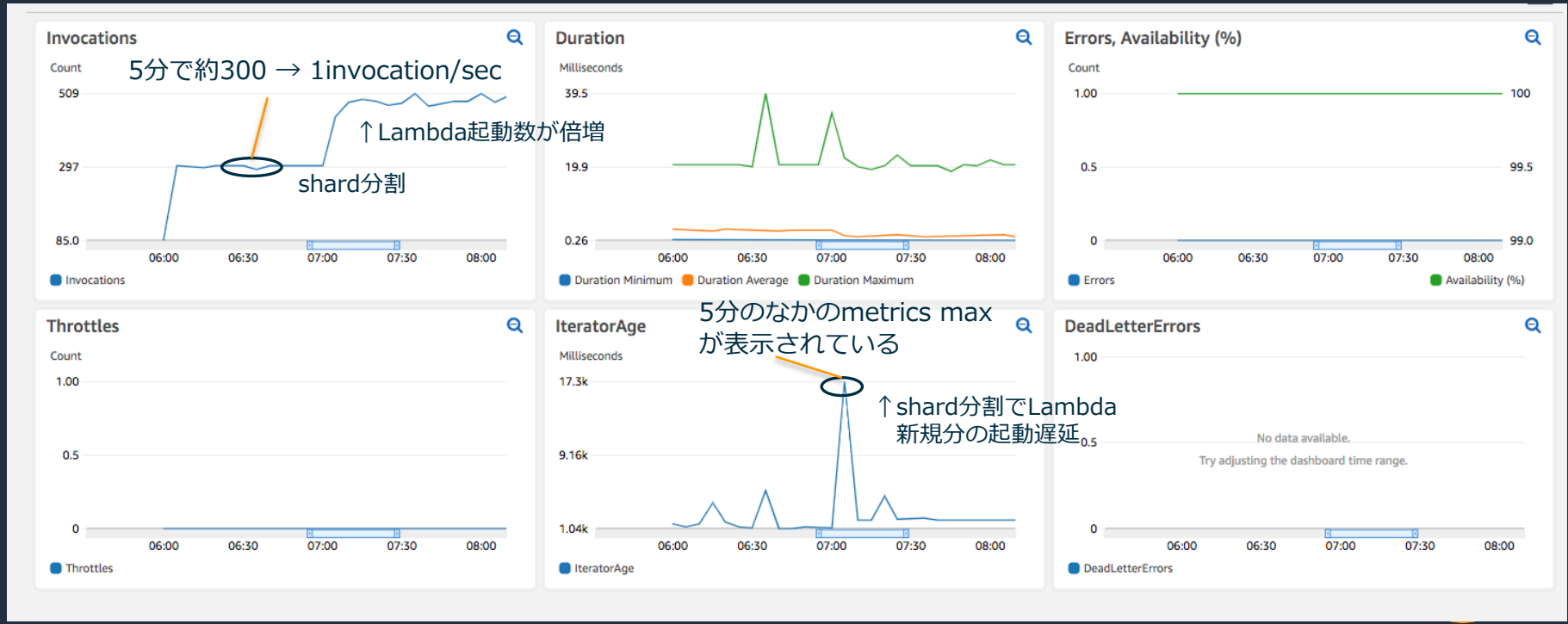
- LambdaをトリガーするStarting positionをTRIM_HORIZONから、 LATEST / AT_TIMESTAMPに変更（データを読まずに捨てることになるので注意）
  - TRIM_HORIZON – Shard内の最も古いレコードから読み込み
  - LATEST – Shard内の最新レコードから読み込み
  - AT_TIMESTAMP – 指定したtimestampポジションから読み込み

aws

Stream Pollingベースでの監視事項

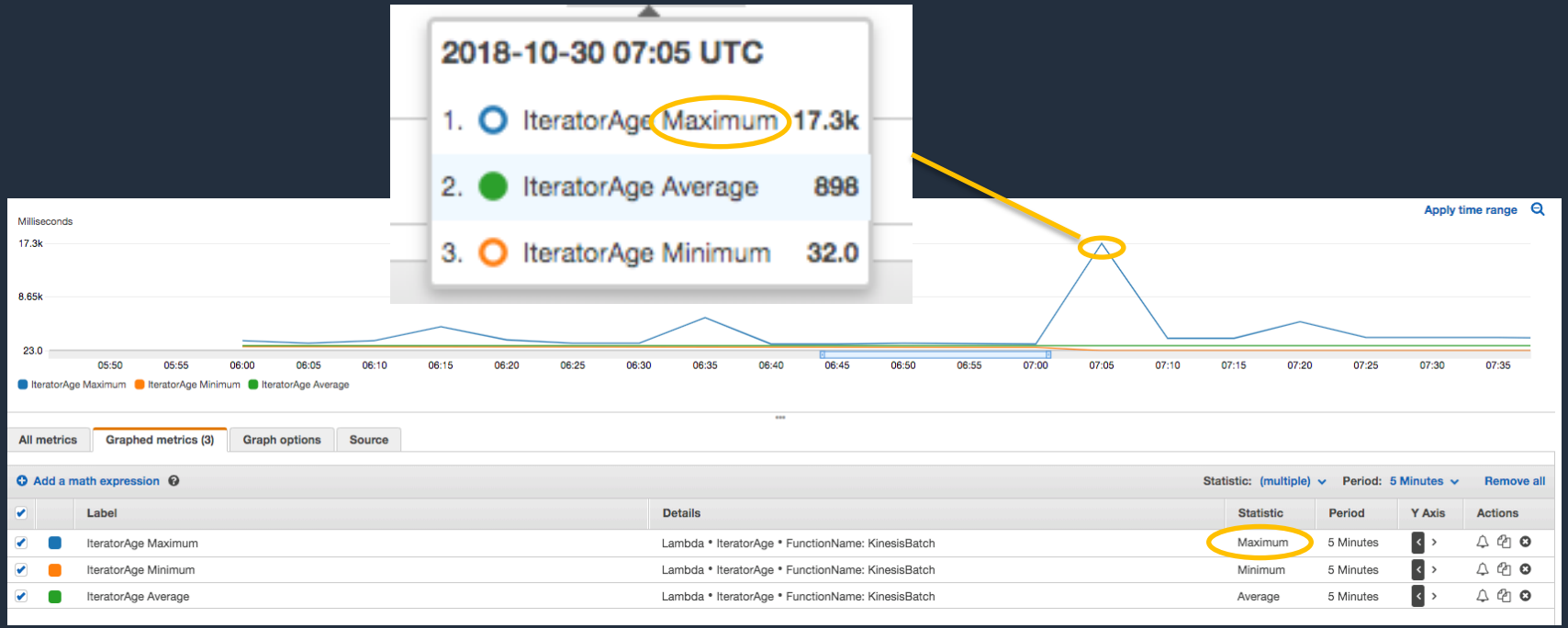
IteratorAge

1. 0.3秒間隔でrecordをKinesis Streamsへ送信
2. 途中でshard分割を実行



IteratorAgeをCloudwatch metricsでみると

IteratorAge



「5分のなかのmetrics Maximumが表示されている」

Stream Pollingベースでの監視・運用ポイント

IteratorAge

IteratorAgeの”Average”が大きくなっている場合は処理遅れが想定される。

- Lambda実行環境のスペック向上により実行時間の短縮を考える
  - メモリ増強など

- Kinesis Streamsのshardを分割
  - shardを分割することで並行処理数が増える
  - 東京リージョンの場合、1 streamあたり、200shards分割まで可能
    https://docs.aws.amazon.com/ja_jp/streams/latest/dev/service-sizes-and-limits.html

※デフォルトのShard上限数は、AWS リージョン 米国東部（バージニア北部）、米国西部 (オレゴン)、および 欧州 (アイルランド) で 500 Shardsです。その他のリージョンのデフォルトのシャード制限はすべて 200 Shardsです。

aws

AWS Lambda の Metrics

項目	単位	意味
Invocations	count	Lambdaの実行回数を計測したもの。=課金カウント **≠同時実行数**
Duration	ms	Lambdaの実行時間を計測。（課金は100msで切り上げる） **※コールドスタートのLambda起動までの時間は含まない**
Errors, Availability	count	Lambdaが正常終了しなかった回数を計測したもの Availabilityは%で表示される
Throttles	count	Throttleの発生回数 **アカウントにおける**Lambdaの同時起動数超過が発生している
IteratorAge	ms	ストリーム(Kinesis/DynamoDB)でのみ利用。バッチサイズ分取得した**レコード終端の時刻とLambdaがイベントとして受信した時刻差**で表示される
DeadLetterErrors	count	DLQを設定し、そのDLQへの書き込みが失敗すると増加する。

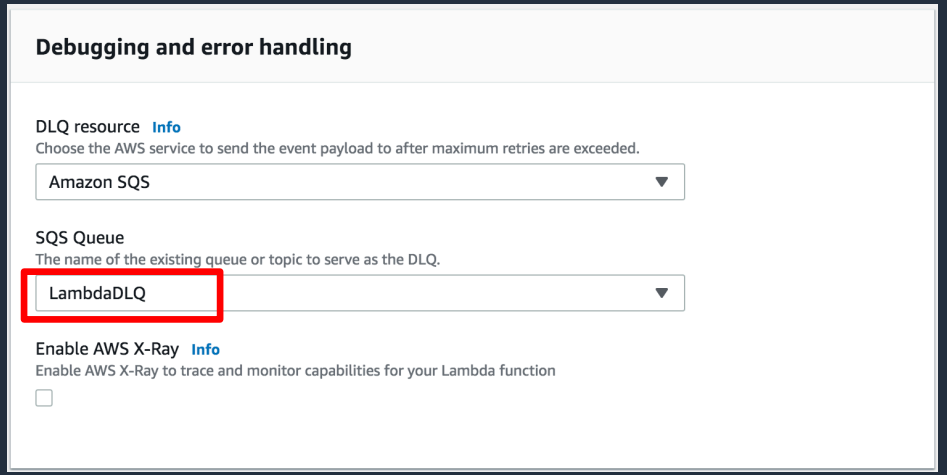
DeadLetterErrorsの監視

DeadLetterErrors

Dead Letter Queue(DLQ)とは

- 最大リトライ回数終了後に、LambdaのEventペイロードをSQSの標準Queue/SNSトピックに送信する機能
- 非同期イベントでLambdaが呼び出された場合に利用可

DeadLetterErrorsとは設定済みのDLQに書き込むことができない場合に、増加する



https://docs.aws.amazon.com/ja_jp/lambda/latest/dg/dlq.html

Metricsからは読み取れないこと

EventベースStreamベースのAWS Lambda

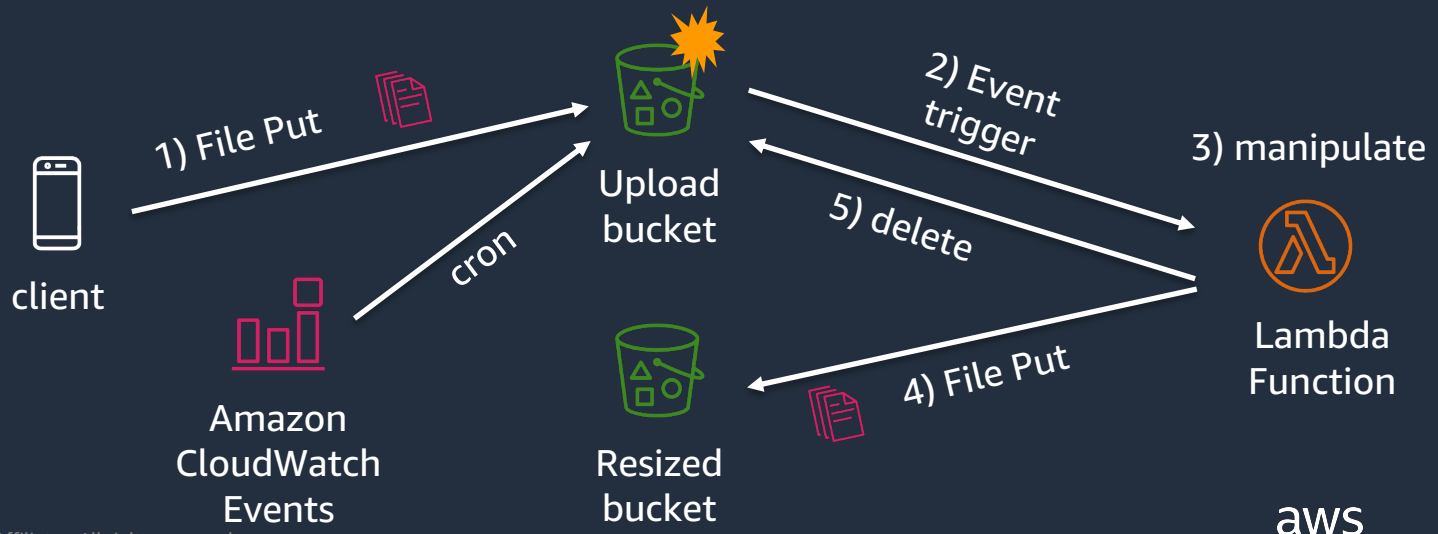
- At Least Once
  - AWS Lambdaは呼び出されれば最低一回は実行することを保証している。
    - 呼び出し側が呼び出さない
    - 呼び出し側が複数回呼び出す
  - この事象はAWS Lambdaでは監視/検知ができない
  - 全体の仕組みで検討する必要がある

EventベースのAWS Lambda

- S3がLambdaを呼び出さない
  https://docs.aws.amazon.com/ja_jp/lambda/latest/dg/with-s3-example-deployment-pkg.html

対処例

- Event発火用バケットと処理済みバケットを作成することで、Event発火漏れを検知する。



Event発火漏れ対策例

```
import boto3
s3_client = boto3.client('s3')

def lambda_handler(event, context):
    for record in event['Records']:
        bucket = record['s3']['bucket']['name']
        key = record['s3']['object']['key']
        download_path = '/tmp/{}{}'.format(uuid.uuid4(), key)
        s3_client.download_file(bucket, key, download_path)

        // do something to the image. (ex. thumbnail)

        s3_client.upload_file(download_path, 'upload-lambda-resized', key)
        s3_client.delete_object(Bucket=bucket, Key=key)
    return
```


aws

EventベースStreamベースのAWS Lambda

- 複数回呼び出す

対処例

- Lambda Codeでの冪等性の確保
    - DynamoDB ConditionExpression パターン
        1. 入力イベントの固有属性の値を取得。(購入 ID など)
        2. 固有値がTABLE に存在するかどうかを確認します。
            - 存在する場合 、(処理をSKIPし) Lambdaを正常に終了します
            - 存在しない場合、固有値をPUTして通常の実行を継続します。
        3. 関数の動作が正常に終了した場合、TABLEに固有値が含まれます。
        4. 関数の動作が異常に終了した場合、TABLEから固有値を消します。

https://aws.amazon.com/jp/premiumsupport/knowledge-center/lambda-function-idempotent/

複数回呼び出し対応のためのロック例

```
def lock(id) :
    try:
        dynamodb.put_item(
            TableName=table_name, Item={"id": {"S": id}},
            ConditionExpression='attribute_not_exists(id)'
        )
        return True
    except ClientError as e:
        if e.response['Error']['Code'] == 'ConditionalCheckFailedException’:
            return False
        else:
            raise
```


aws

AWS LambdaのMetrics監視のまとめ

Throttlesを観測した場合

- どのAWS Lambdaが大量に使っているかの特定
    - CloudWatchのアカウント単位と個別Lambda単位の使用状況を確認
        - アカウント単位で飽和している => Limit Increaseの申請
        - Lambda単位で飽和している => Reserve Concurrency設定変更

Errorsが増加している場合

- IAMでの権限設定漏れ
    - IAM Roleを他の関数と共有している場合、設定変更が行われたことを疑ってみる
- Queue/Streamは、batch sizeと全体処理時間とtimeout(min/sec)を確認
    - 単位処理時間 x batch size =全体処理時間 > timeout
    - これも処理が進まないパターーンになる

AWS LambdaのMetrics監視のまとめ

Dead Letter Queueが増え続ける

- 後段処理、連携システムのダウンなど、システム系としての障害の確認などを実施

リトライ処理の考慮

- 非同期型であれば1回の実行、2回のリトライが行われ、すべて失敗した場合に、（設定していれば） Dead Letter Queueへ送られる

Amazon API GatewayのMetrics

aws

Amazon API Gateway の Log・Traceの有効化

Amazon API Gateway の Log・Traceの有効化

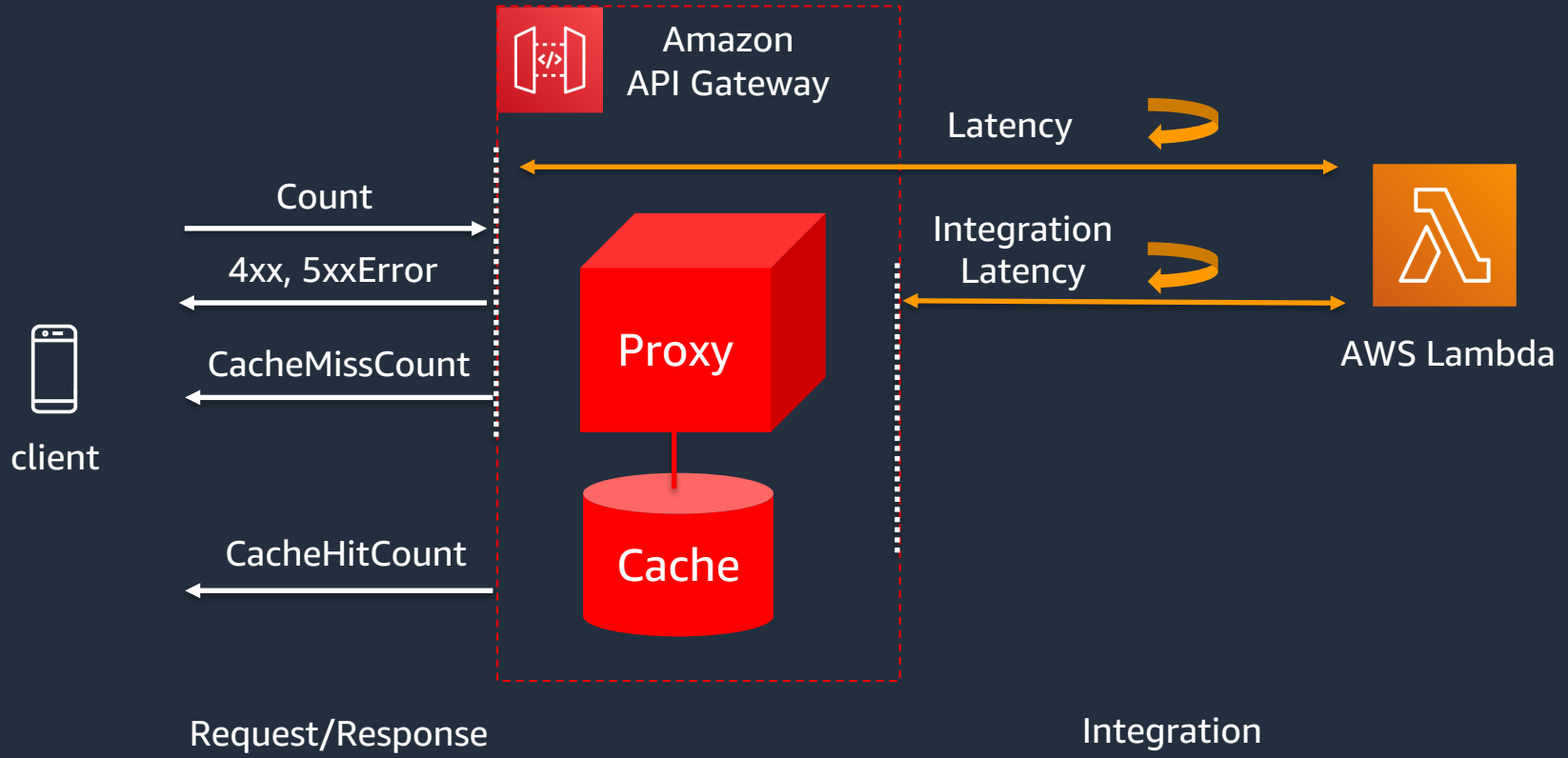
Amazon API Gateway の Log・Traceの有効化

Amazon API Gateway の Log・Traceの有効化

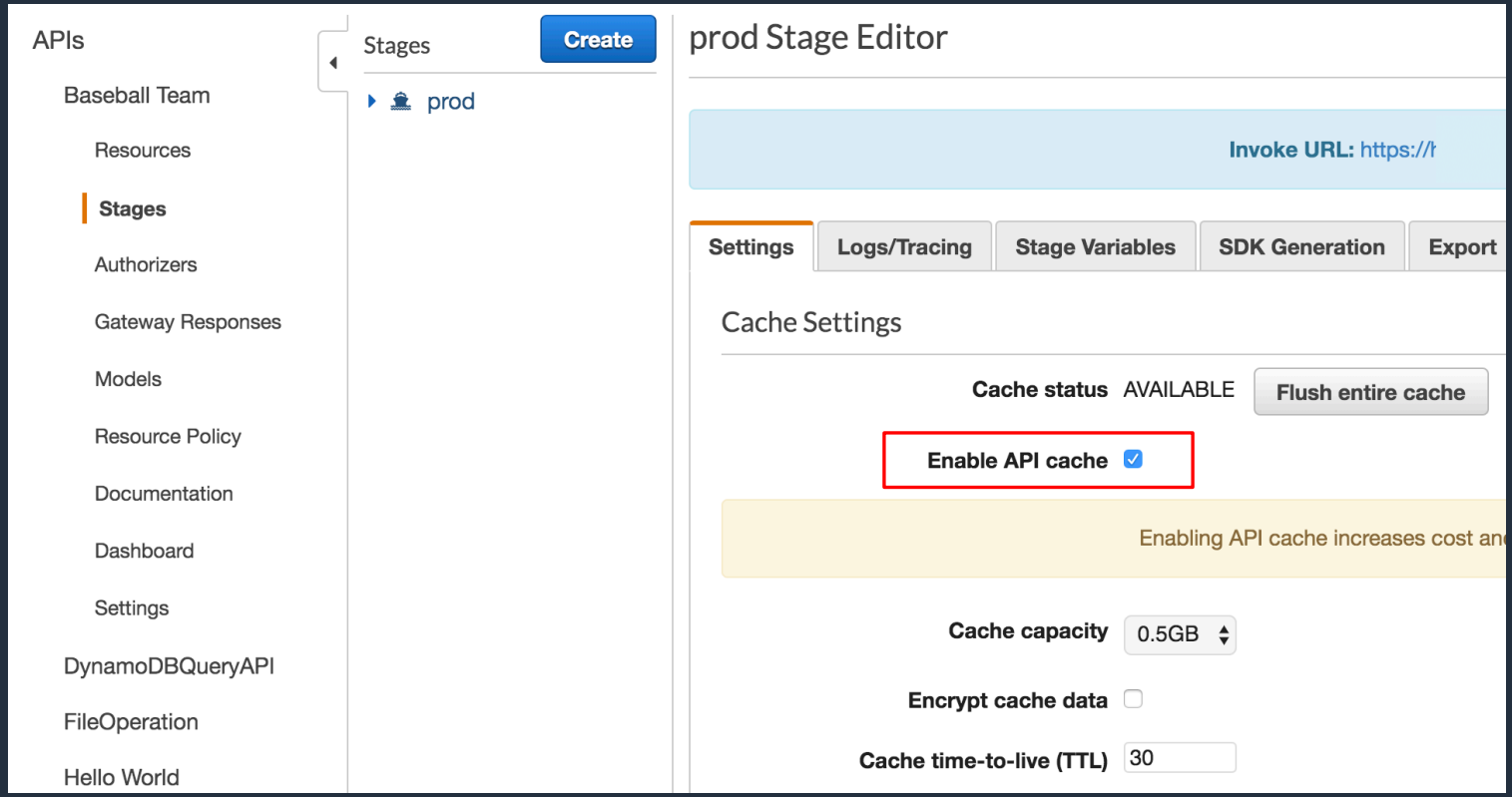
Amazon API Gateway の Metrics

項目	単位	意味
4xxError	count	Sumは4xxエラーの合計数、Averageは4xxエラー率
5xxError	count	Sumは5xxエラーの合計数、Averageは5xxエラー率
CacheHitCount	count	APIキャッシュからレスポンスした、 Sumはキャッシュヒットした数、Averageはキャッシュヒット率
CacheMissCount	count	キャッシュを有効にしているが、バックエンドから応答した数 Sumはキャッシュミスヒット数、Averageはキャッシュミス率
Count	count	APIのリクエスト数
IntegrationLatency	ms	API Gatewayが、バックエンドへリクエストして、バックエンドからレスポンスが返却されるまでの時間
Latency	ms	API Gatewayがクライントからリクエストを受け取り、クライントに返却するまでの時間。API Gatewayのオーバヘッドも含まれる

Amazon API Gateway の Metrics

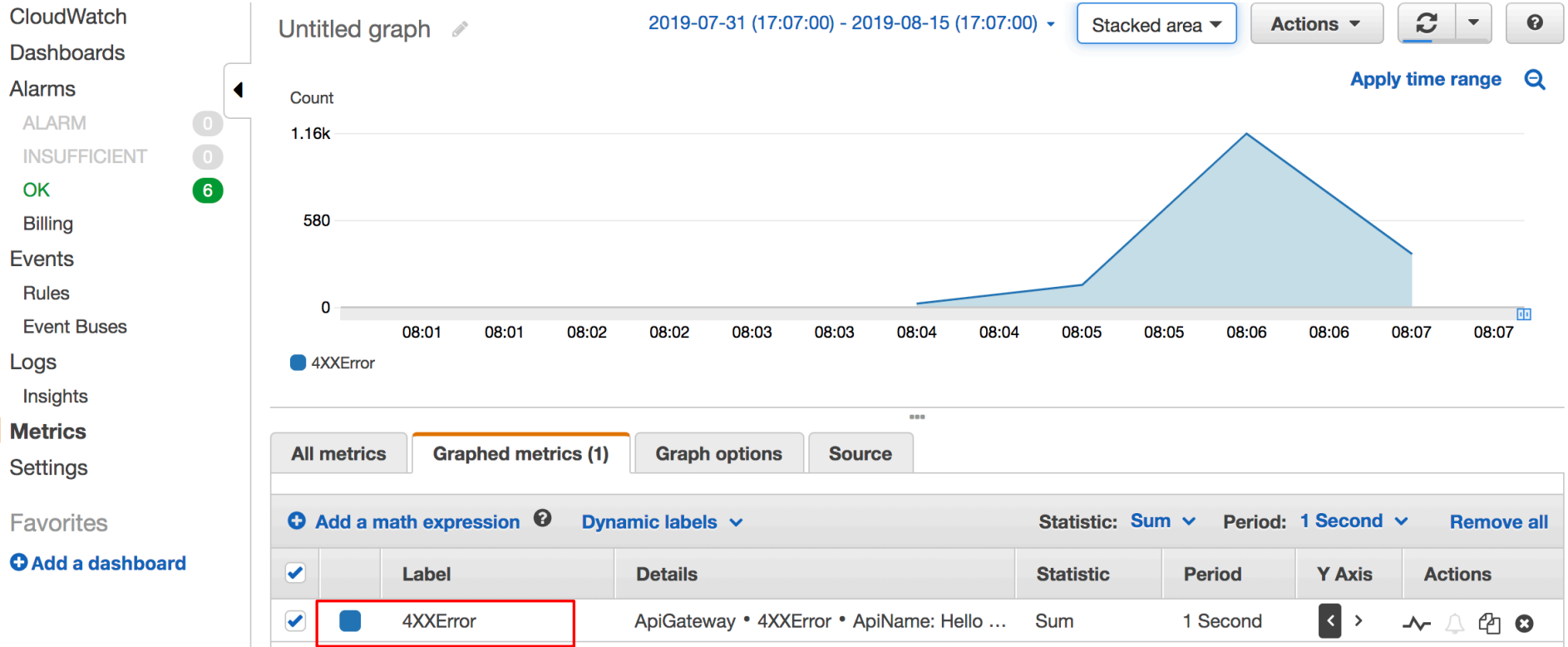


API GatewayのCache設定

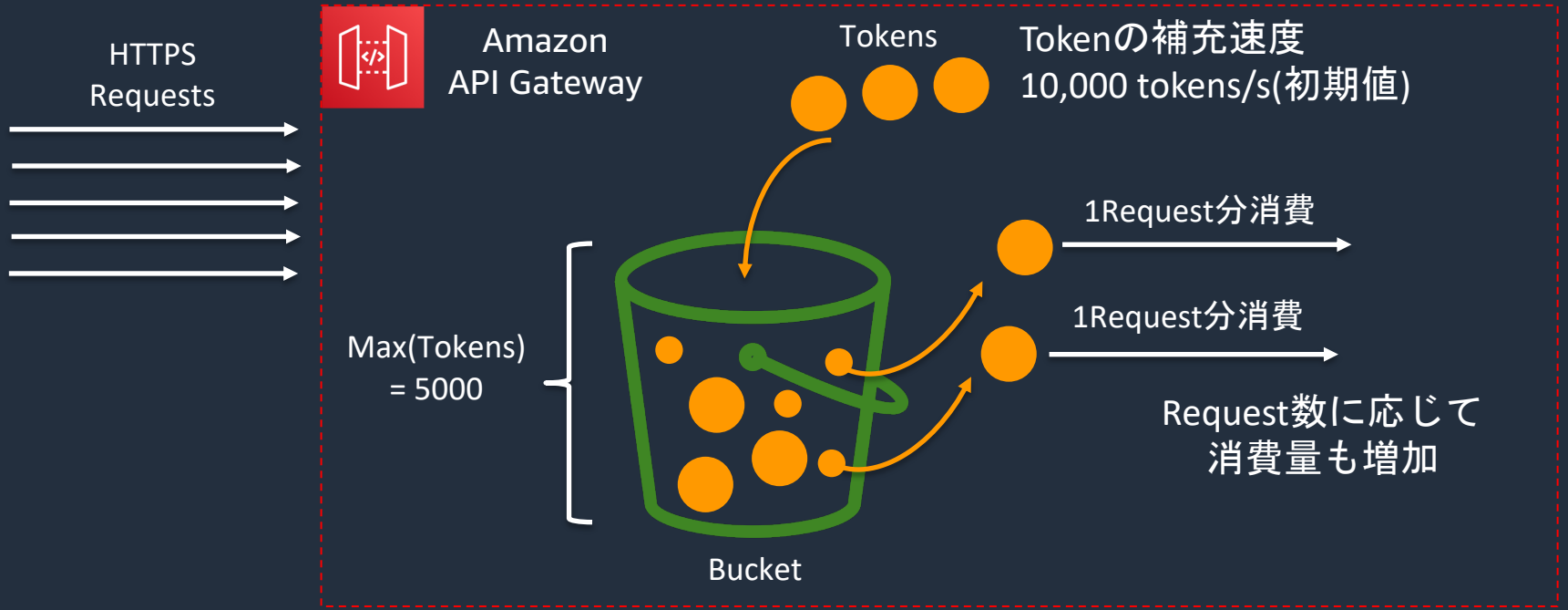


API Gatewayのスロットリング監視

MetricsとしてThrottlesが定義されていないためHTTP 429 のレスポンスを確認する (HTTP 429 = too many requests)



API Gatewayのスロットリングを理解する



バースト = BucketのTokenの最大量 = 5000/account (初期値)

https://docs.aws.amazon.com/ja_jp/apigateway/latest/developerguide/api-gateway-request-throttling.html

Amazon DynamoDBのMetrics

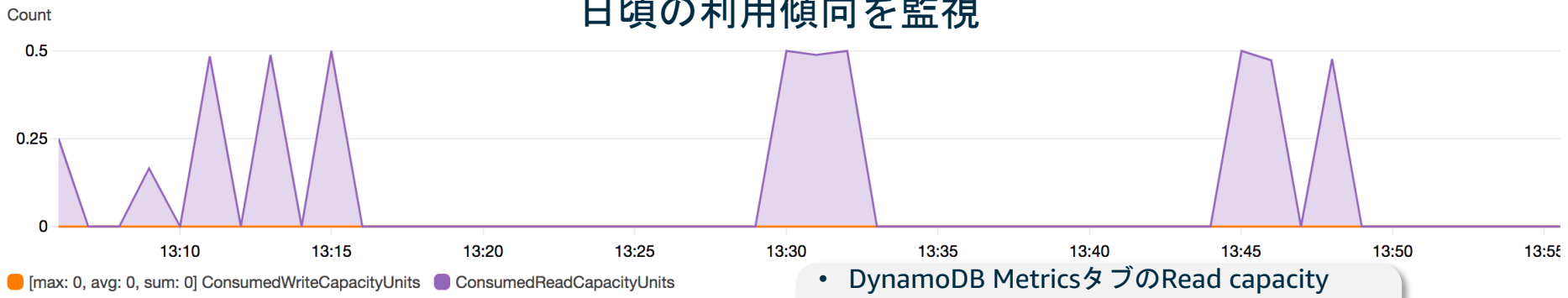
aws

Amazon DynamoDBの Metrics

項目	単位	意味
ConsumedReadCapacityUnits	Count	指定の期間内に消費された読み取り容量ユニット（RCU）の数
ConsumedWriteCapacityUnits	Count	指定の期間内に消費された書き込み容量ユニット(WCU)の数
ReadThrottleEvents	Count	テーブルまたはグローバルセカンダリインデックスのプロビジョニングされた読み取り容量ユニットを超える DynamoDB へのリクエスト
WriteThrottleEvents	Count	テーブルまたはグローバルセカンダリインデックスのプロビジョニングされた書き込み容量ユニットを超える DynamoDB へのリクエスト。
SystemErrors	Count	指定された期間中に HTTP 500 ステータスコードを生成する、DynamoDB または DynamoDB ストリームへのリクエスト。通常、HTTP 500 は内部サービスエラーを示します。

指定期間に消費された読み取り/書き込み容量ユニット監視

日頃の利用傾向を監視



- DynamoDB MetricsタブのRead capacity Unit/Secondは秒単位の表示
- CloudWatch ConsumedReadCapacityUnitsは分単位の表示(CloudWatchの設定に依存)

All metrics | Graphed metrics (2/5) | Graph options | Source

Add a math expression | Dynamic labels | Statistic: Averag | : 1 Minute | Remove all

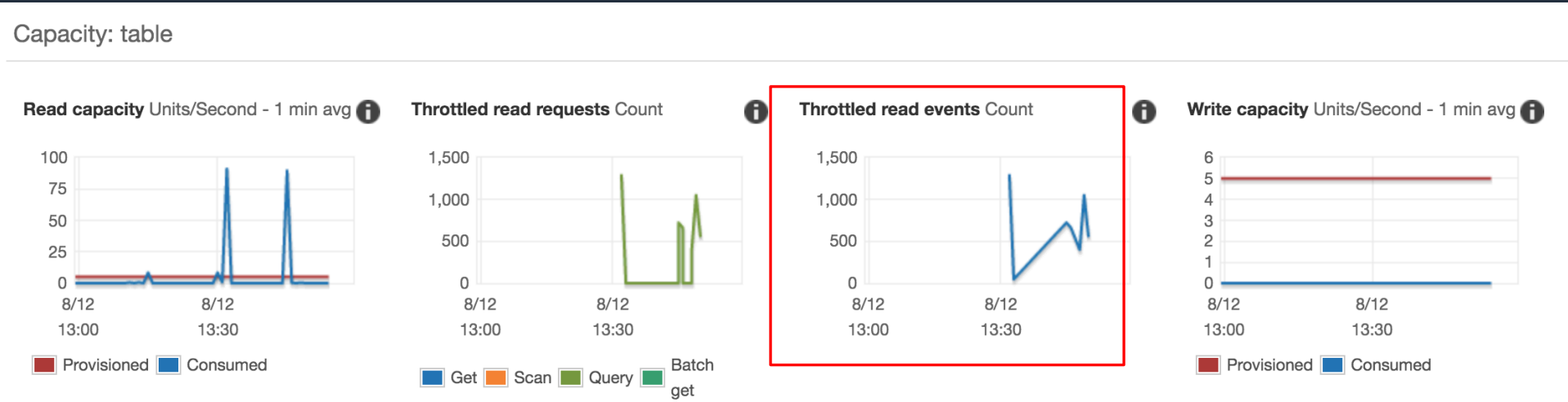
	Label	Details	Statistic		Y Axis	Actions
☐	[max: ${MAX}, avg: ${AVG}, su…	DynamoDB • ConditionalCheckFailedRequests • TableName…	Average			
☑	[max: ${MAX}, avg: ${AVG}, su…	DynamoDB • ConsumedWriteCapacityUnits • TableName: s…	Average	1		
☐	[max: ${MAX}, avg: ${AVG}, su…	DynamoDB • ProvisionedWriteCapacityUnits • TableName: …	Average	1 M te		
☐	[max: ${MAX}, avg: ${AVG}, su…	DynamoDB • ProvisionedReadCapacityUnits • TableName: …	Average	1 Mi ute		
☑	ConsumedReadCapacityUnits	DynamoDB • ConsumedReadCapacityUnits • TableName: s…	Average	1 Minute		

DynamoDBのスロットリング監視

Throttled read events/Throttled write eventsの監視

- テーブルにプロビジョニングされた読み取り/書き込み容量(RCU/WCU)を指定期間で超えたリクエストで、テーブルにプロビジョニングされたスループット制限をどのリクエストが超えたかを判断できる

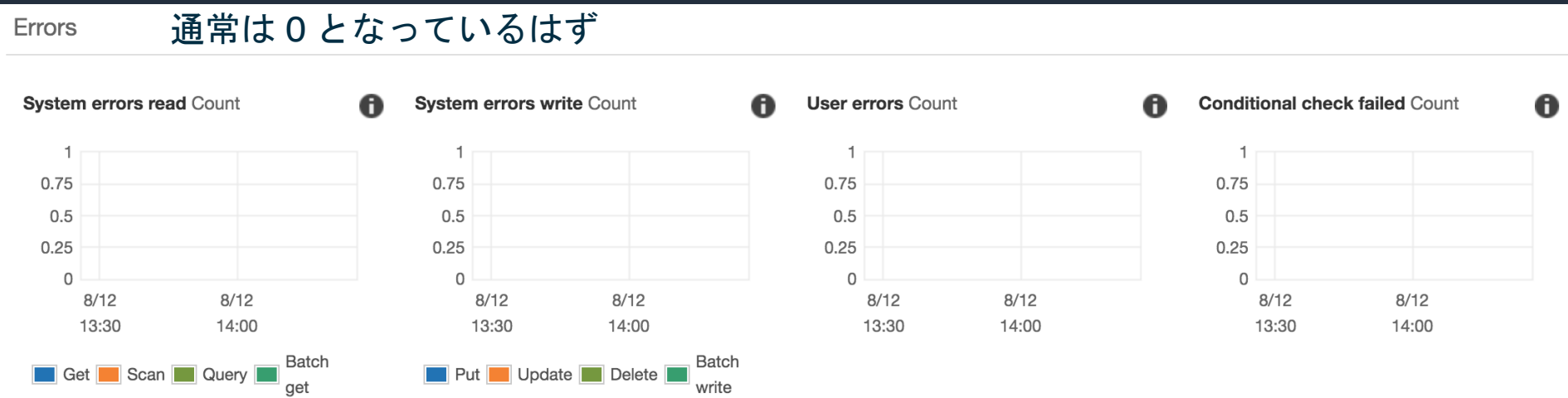
https://docs.aws.amazon.com/ja_jp/amazondynamodb/latest/developerguide/metrics-dimensions.html



DynamoDBのシステムエラー監視

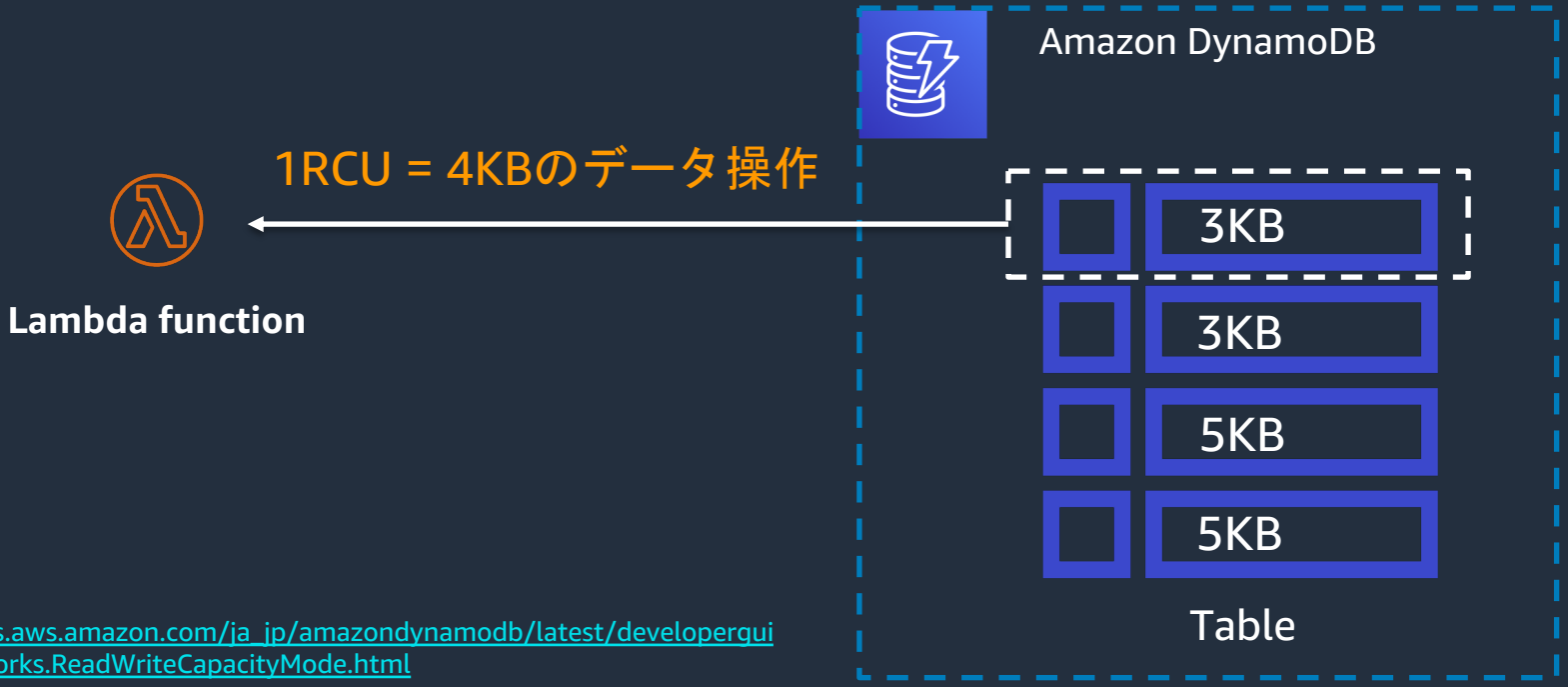
System errors read/writeを監視

- エラーが発生したかどうかを判断
  - 通常これは0であるべき値
- システムエラーの数 = HTTP Status 500 の数
  - DynamoDB / DynamoDB Streams へのエラーリクエスト数



APIとRead Capacity Unit(CU)の消費

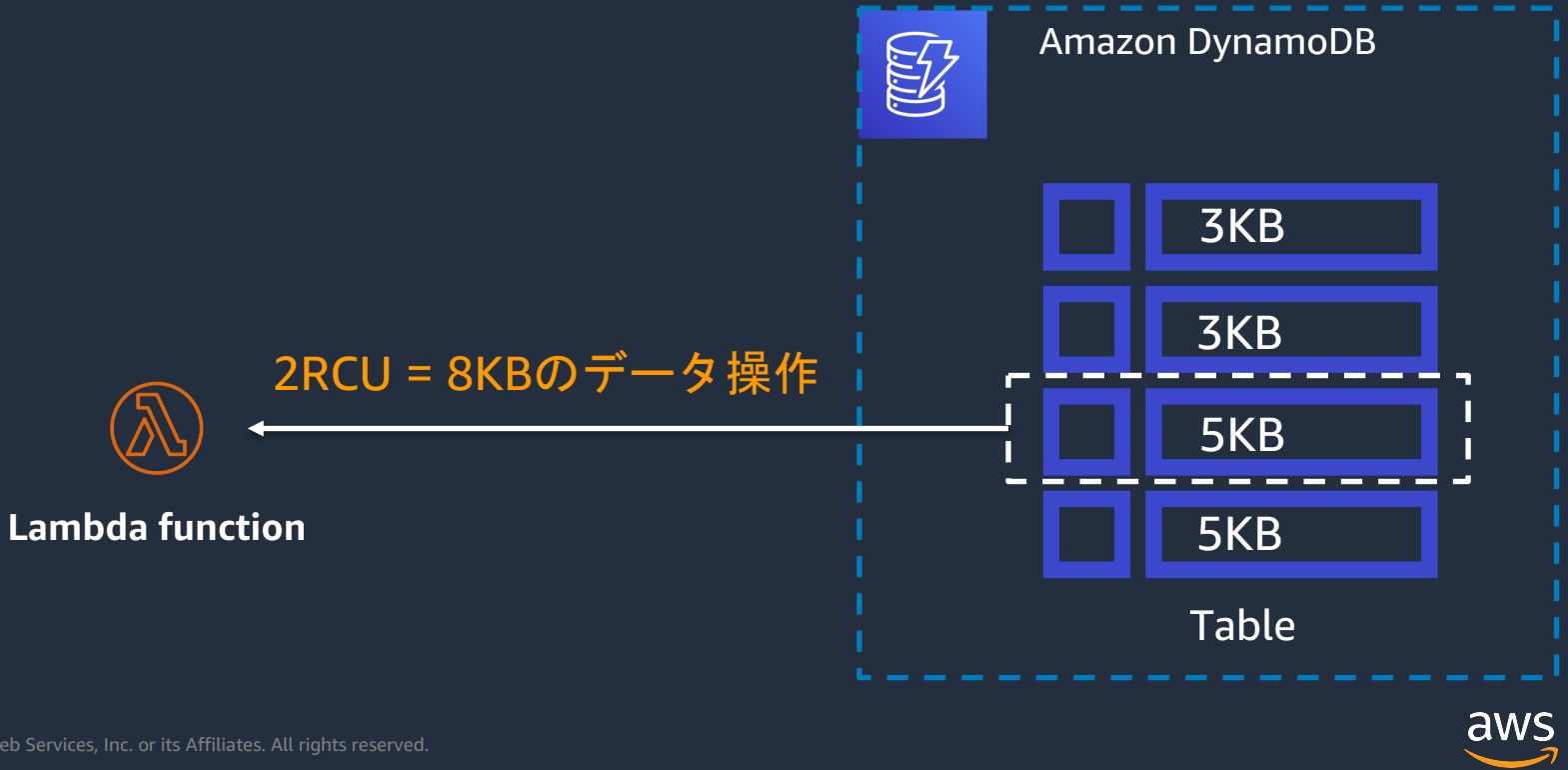
READの操作、query, scan, get_itemはリクエスト単位のデータサイズで算出



https://docs.aws.amazon.com/ja_jp/amazondynamodb/latest/developerguide/HowItWorks.ReadWriteCapacityMode.html

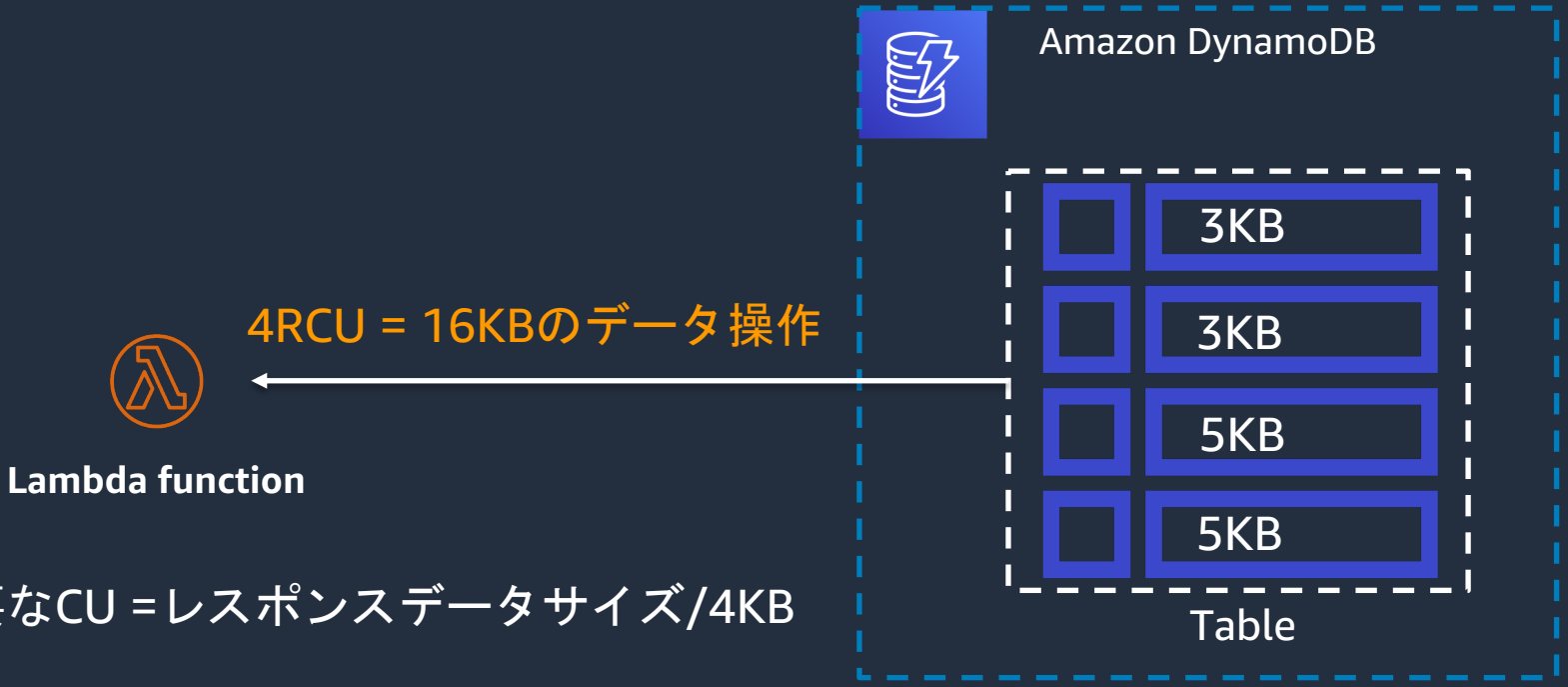
APIとRead Capacity Unit(CU)の消費

READの操作、query, scan, get_itemはリクエスト単位のデータサイズで算出



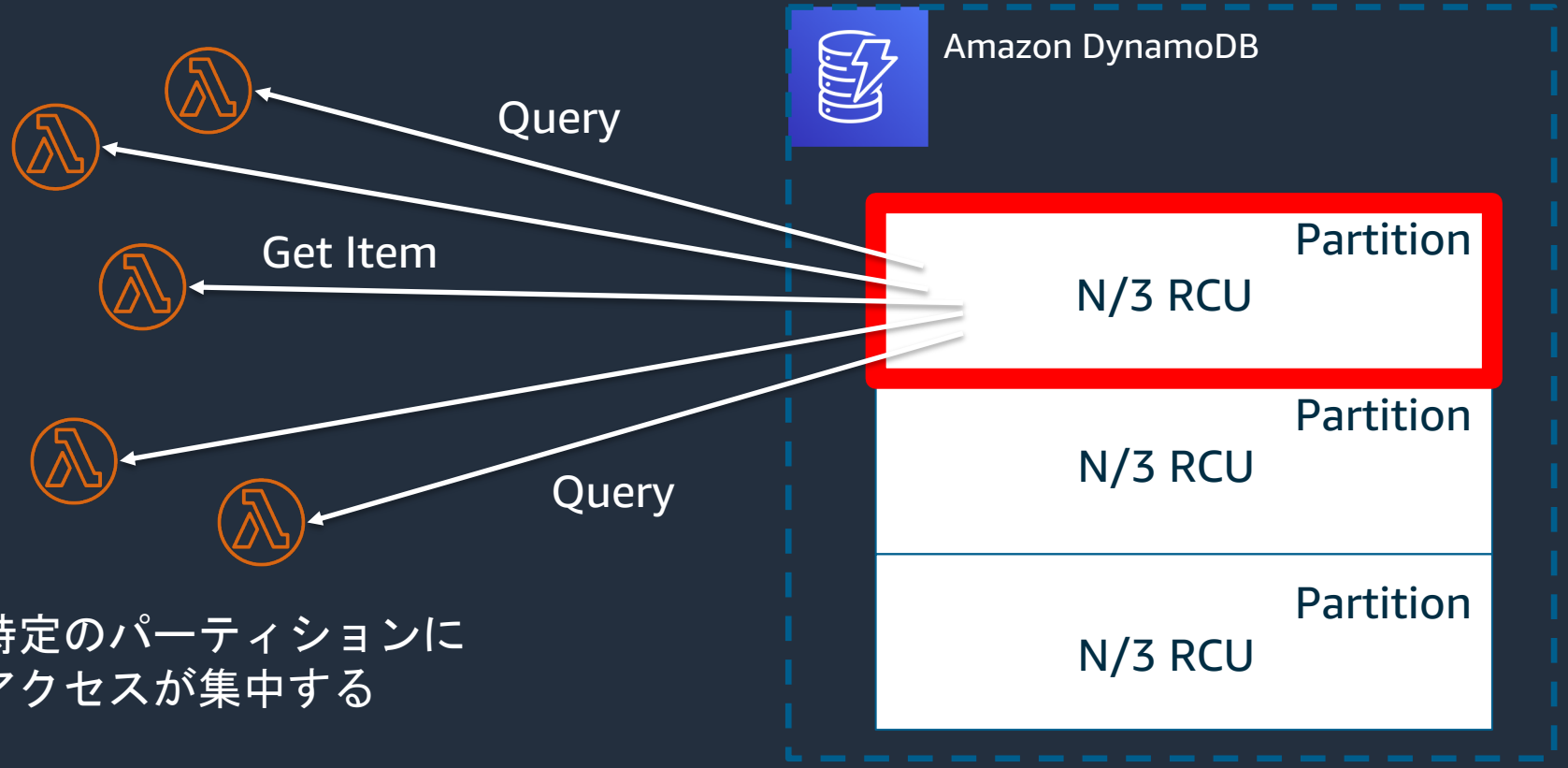
APIとRead Capacity Unit(CU)の消費

READの操作、query, scan, get_itemはリクエスト単位のデータサイズで算出



必要なCU =レスポンスデータサイズ/4KB

ホットパーティション とは



特定のパーティションに
アクセスが集中する

TTLの活用

TTLはCUを消費しない

- TTLを利用しない削除の場合は、削除対処のデータをquery/scanしてからdeleteするオペレーションとなり、瞬間的に大きなR/W CUを設定する必要がでたりなど運用上CUの管理が必要

- TTL期限が過ぎてすぐに削除されるわけではなく、48時間以内に削除となっているので、アプリケーション的な考慮は必要
    - queryでTTL切れのアイテムが取れる可能性がある

- TimeToLiveDeletedItemCountでTTLにより削除されたアイテム数をカウントすることも可能

https://docs.aws.amazon.com/ja_jp/amazondynamodb/latest/developerguide/TTL.html

aws

オートスケーリングを利用しているのにCU不足

オートスケールを利用しても、スパイクアクセスの完全対応が難しい

- 時刻によるスパイク
- イベントによるスパイク

APIコントロール（もしくはオートスケールと併用)が良い場合もある

オートスケールを設定したテーブルも初期設定は RCU/WCUともに5が初期値となるので、この点も注意

https://docs.aws.amazon.com/ja_jp/amazondynamodb/latest/developerguide/AutoScaling.CLI.html#AutoScaling.CLI.BeforeYouBegin

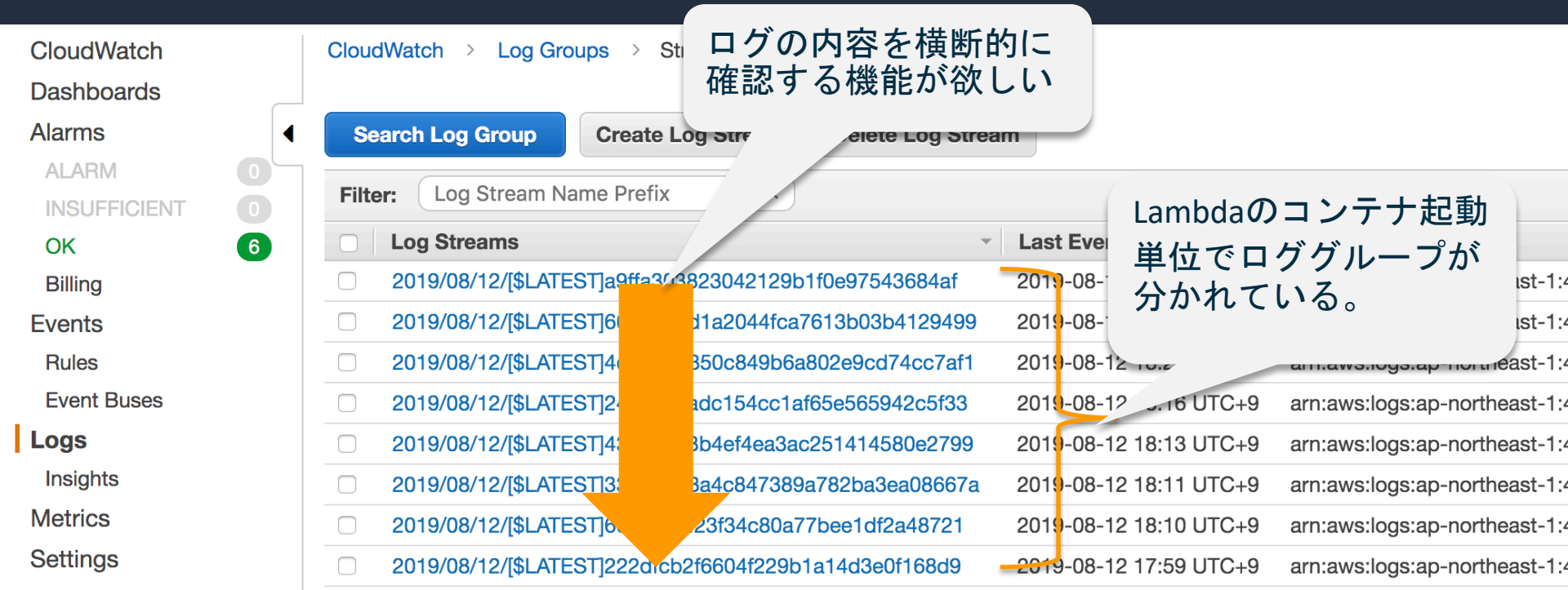

Log運用と可視化

Amazon CloudWatch でのログ管理

aws

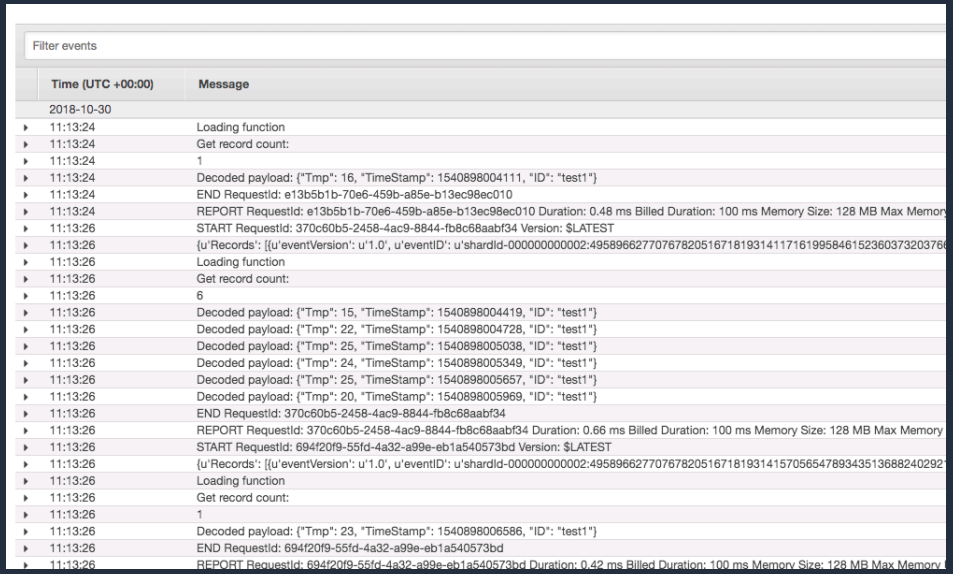
Amazon CloudWatch Logsの使い方を知る

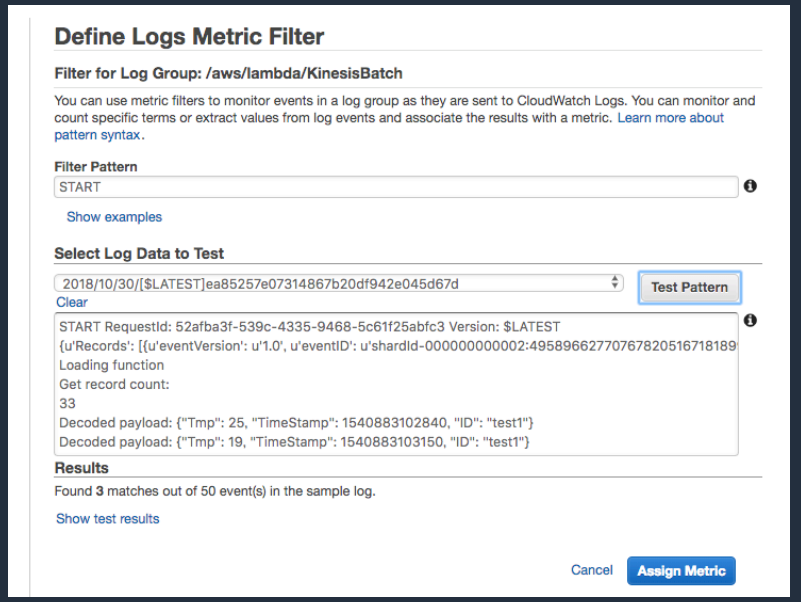
CloudWatchは簡単にLog出力でき、とても便利です。



フィルターパターンの利用例

- ログが正規化されていなくても 一致する文字列の検索も可能。
- アプリケーションが出力するローグフォーマットと利用ルールを明確化することでmetric filterも作成できる。
- printで出力するのではなく、logger関数などを利用してログ出力することがおすすめ





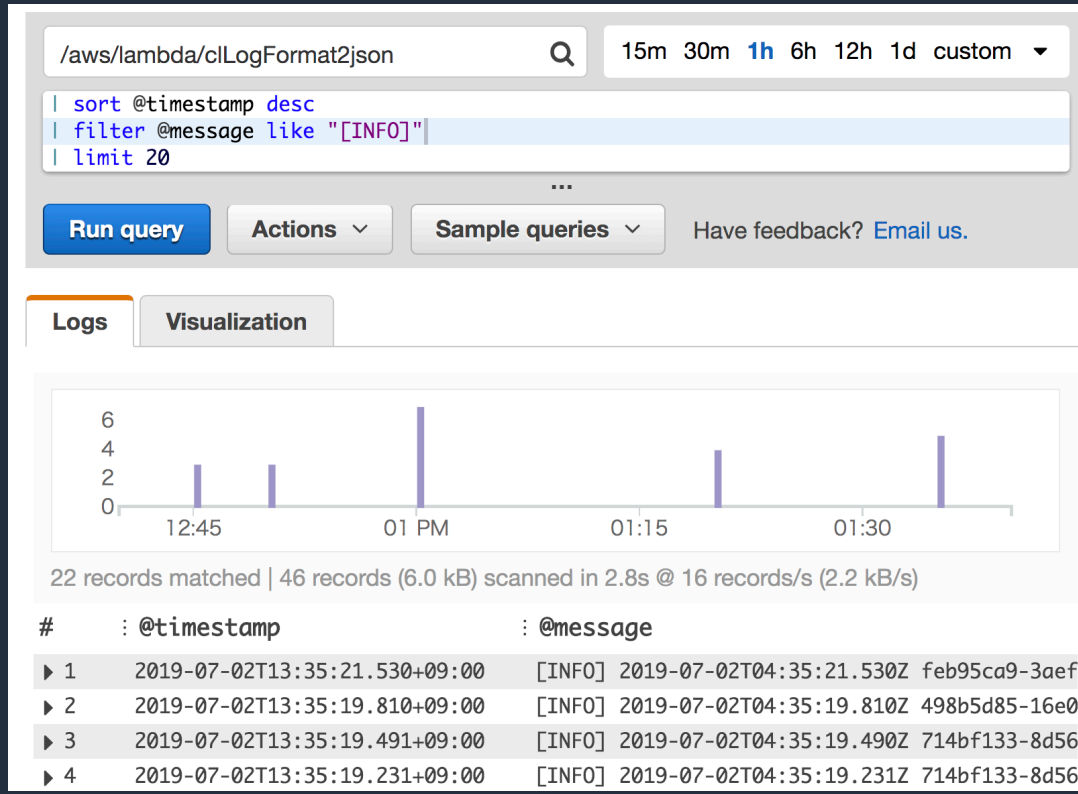
CloudWatch Logs filter pattern

filter patternの仕様

- 大文字、小文字は区別される
- 正規表現は利用できない
- 簡単なパターンマッチのみ利用可能
- JSON形式、スペース区切りは利用可能

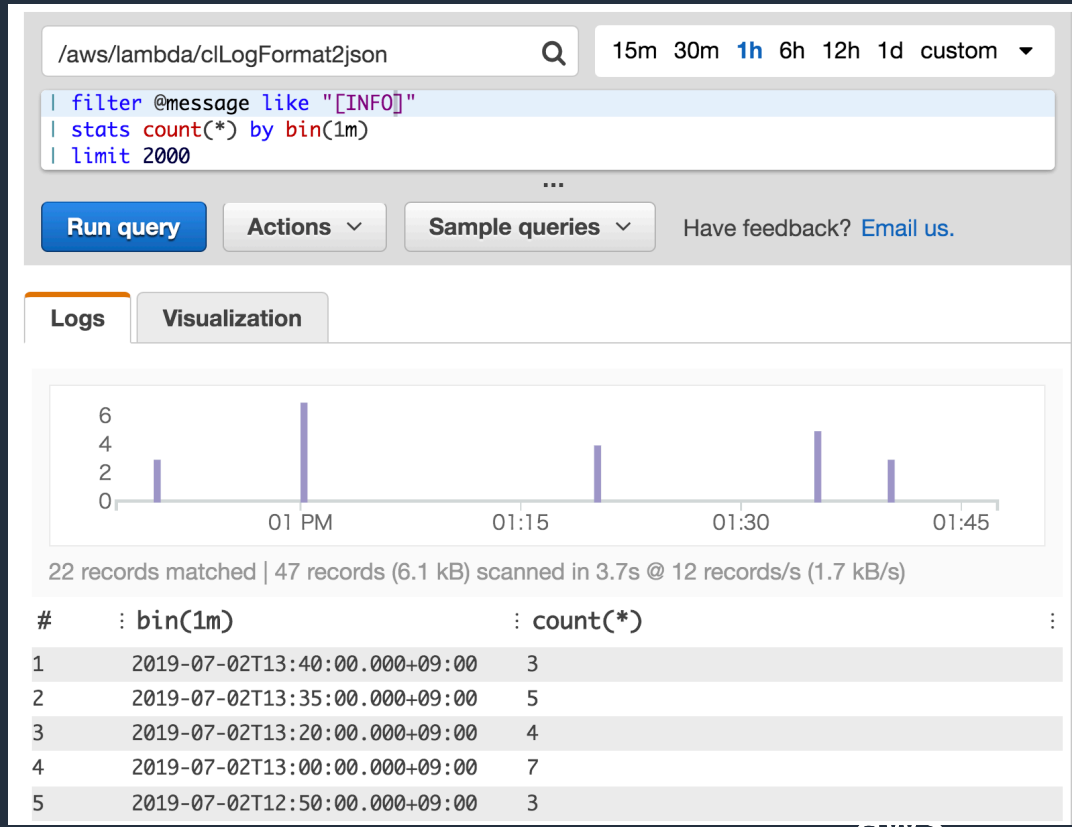
CloudWatch Logs Insightsの活用

```
fields @timestamp, @message
| sort @timestamp desc
| filter @message like "[INFO]"
| limit 20
```



CloudWatch Logs Insightsの活用

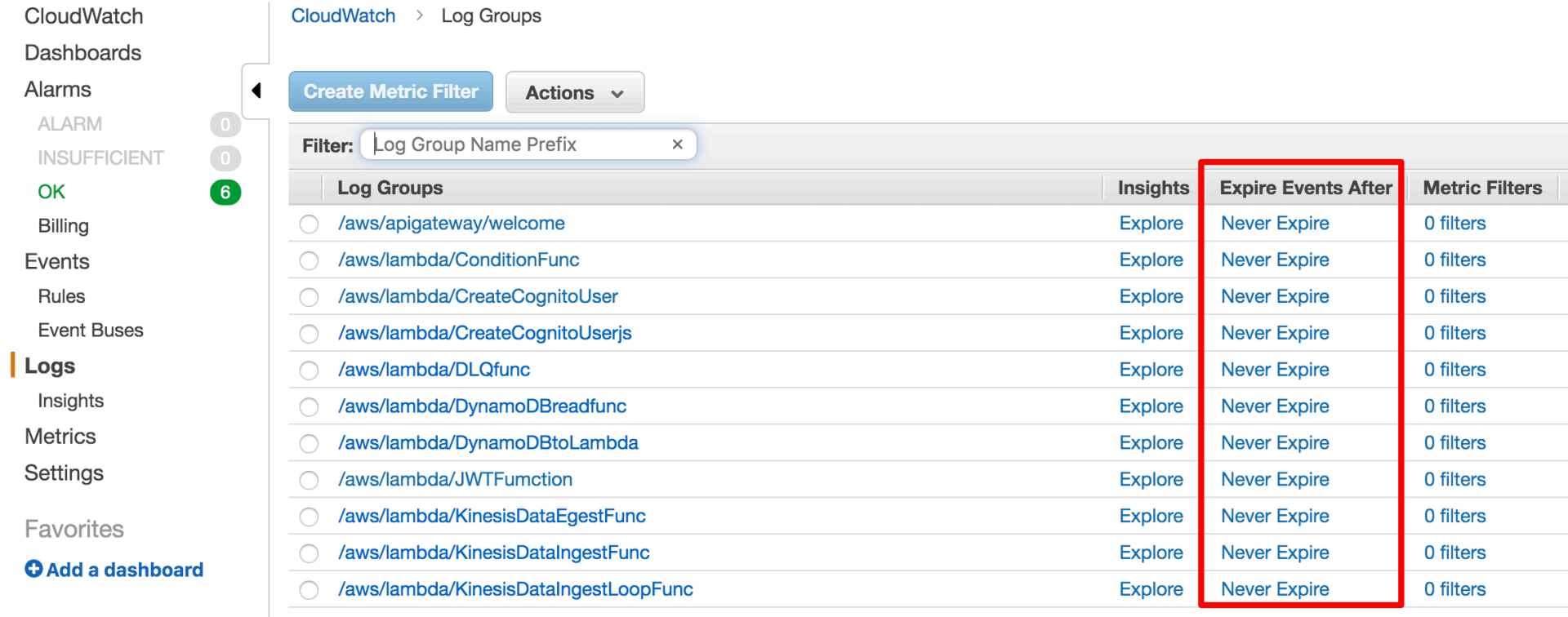
```
fields @timestamp, @message
| sort @timestamp desc
| filter @message like "[INFO]”
| stats count(*) by bin(1m)
| limit 2000
```



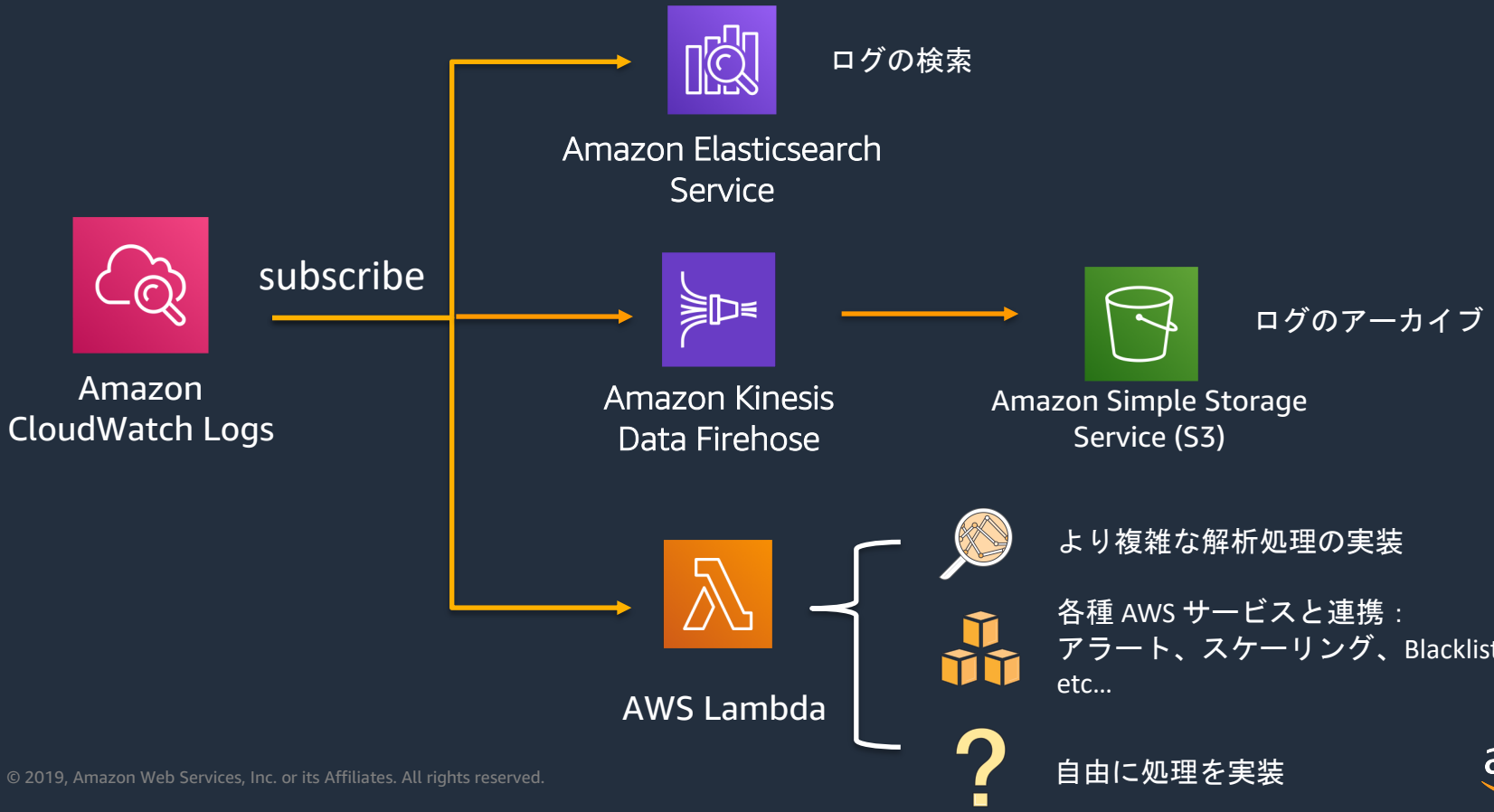
#	bin(1m)	count(*)
1	2019-07-02T13:40:00.000+09:00	3
2	2019-07-02T13:35:00.000+09:00	5
3	2019-07-02T13:20:00.000+09:00	4
4	2019-07-02T13:00:00.000+09:00	7
5	2019-07-02T12:50:00.000+09:00	3

ログ保管期間

- Lambdaのログ保管期間はデフォルトが無期限
- 必要な期間に圧縮することを忘れがち



他サービスに連携してログの活用



Amazon Elasticsearch Service との連携

aws

Dashboard
My domains
logsearch
Reserved instances

logsearch

Configure cluster | Modify access policy | Manage tags | Delete domain | Upgrade domain

Overview | Cluster health | Instance health | Indices | Logs | Upgrade history

Domain status	Active
Elasticsearch version	6.7
Endpoint	https://search-logsearc[redacted]vanbgdpy.ap-northeast-1.es.amazonaws.com
Domain ARN	arn:aws:es:ap-north[redacted]:domain/logsearch
Kibana	https://search-logsearc[redacted]vanbgdpy.ap-northeast-1.es.amazonaws.com/_plugin/kibana/
Availability zones	1
Instance type	t2.small.elasticsearch
Number of instances	1
Storage type	EBS
EBS volume type	General Purpose (SSD)
EBS volume size	10 GB
Encryption at rest	Disabled

Amazon CloudWatch Logs

CloudWatch
Dashboards
Alarms
ALARM 0
INSUFFICIENT 0
OK 4
Billing
Events
Rules
Event Buses
Logs
Insights
Metrics
Settings

Favorites
Add a dashboard

CloudWatch > Log Groups

Create Metric Filter | **Actions**

- Create log group
- Delete log group
- Export
- Export data to Amazon S3
- View all exports to Amazon S3
- Subscriptions
- Stream to AWS Lambda
- Stream to Amazon Elasticsearch Service
- Remove Subscription Filter

Filter: hello

Log Groups 1-1

Log Groups	...ic Filters	Subscriptions
helloworld_loggroup	...ers	None

Step 1: Choose Destination

Step 2: Configure Log Format and Filters

Step 3: Review

Step 4: Confirmation

Start Streaming helloworld_loggroup to Amazon Elasticsearch Service

You are about to start streaming data from your "helloworld_loggroup" log group to an Amazon Elasticsearch Service cluster. Any new log data sent to this log group will be sent to the cluster you choose.

Select account ◉ This Account

○ Another Account

Amazon ES cluster* logsearch

Lambda Function

CloudWatch Logs uses Lambda to deliver log data to Amazon ES. You must specify an IAM role that grants Lambda permission to make calls to Amazon ES. You can choose an existing role or create an IAM role that automatically has the required permissions. To deliver log data to another account, you must specify the Elasticsearch Domain ARN and Elasticsearch Endpoint of other account and ensure permissions are granted to be able to publish to that ARN.

Lambda IAM Execution Role* LambdaElasticServiceRole

Cancel | Previous | Next | Start Streaming

Step 1: Choose Destination

Step 2: Configure Log Format and Filters

Step 3: Review

Step 4: Confirmation

Success

You have started streaming log data from your "helloworld_loggroup" log group to your "logsearch" Amazon Elasticsearch Service cluster.

If you have set an access policy for your Amazon Elasticsearch Service cluster that allows access from your browser, you can start interacting with your log data using the following links:

- Kibana 3
- Kibana 4
- Elasticsearch API

The following Lambda function transforms your incoming CloudWatch Logs data to index documents and submits them to your Elasticsearch cluster. You can modify this function to add custom data transformations (for example, converting IP addresses to geo-locations):

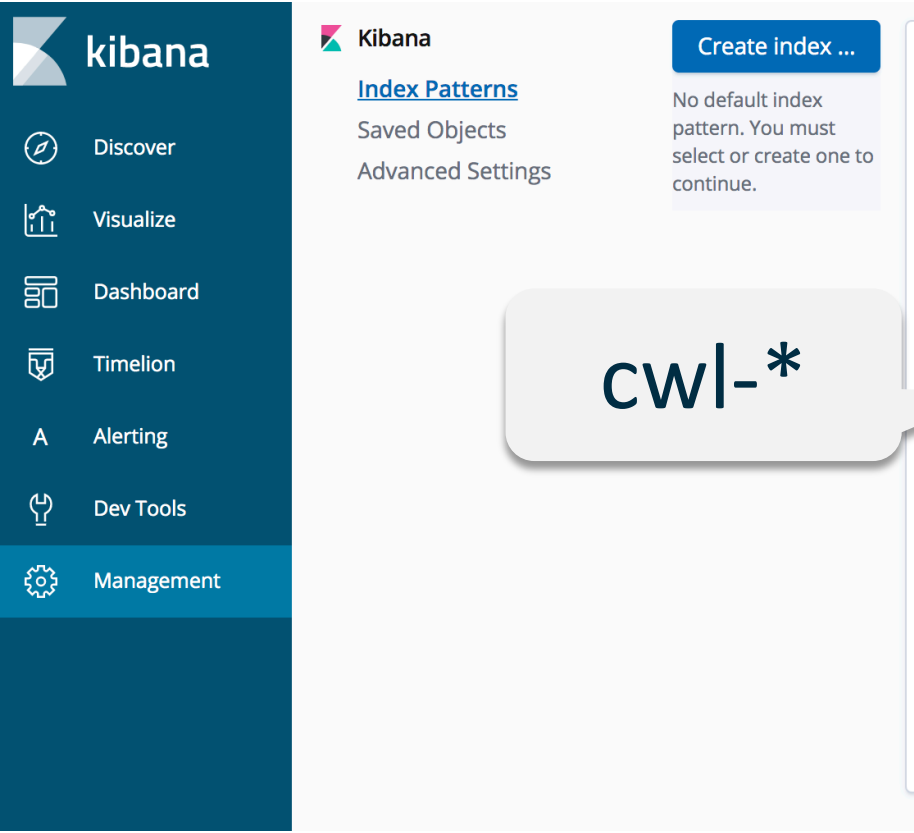
- LogsToElasticsearch_logsearch

If you are streaming any of the following AWS log data sources you can import sample Kibana 3 dashboards by downloading the relevant dashboard file below. Learn how to import dashboard files in Kibana 3.

- Amazon VPC Flow Logs
- AWS Lambda
- AWS CloudTrail

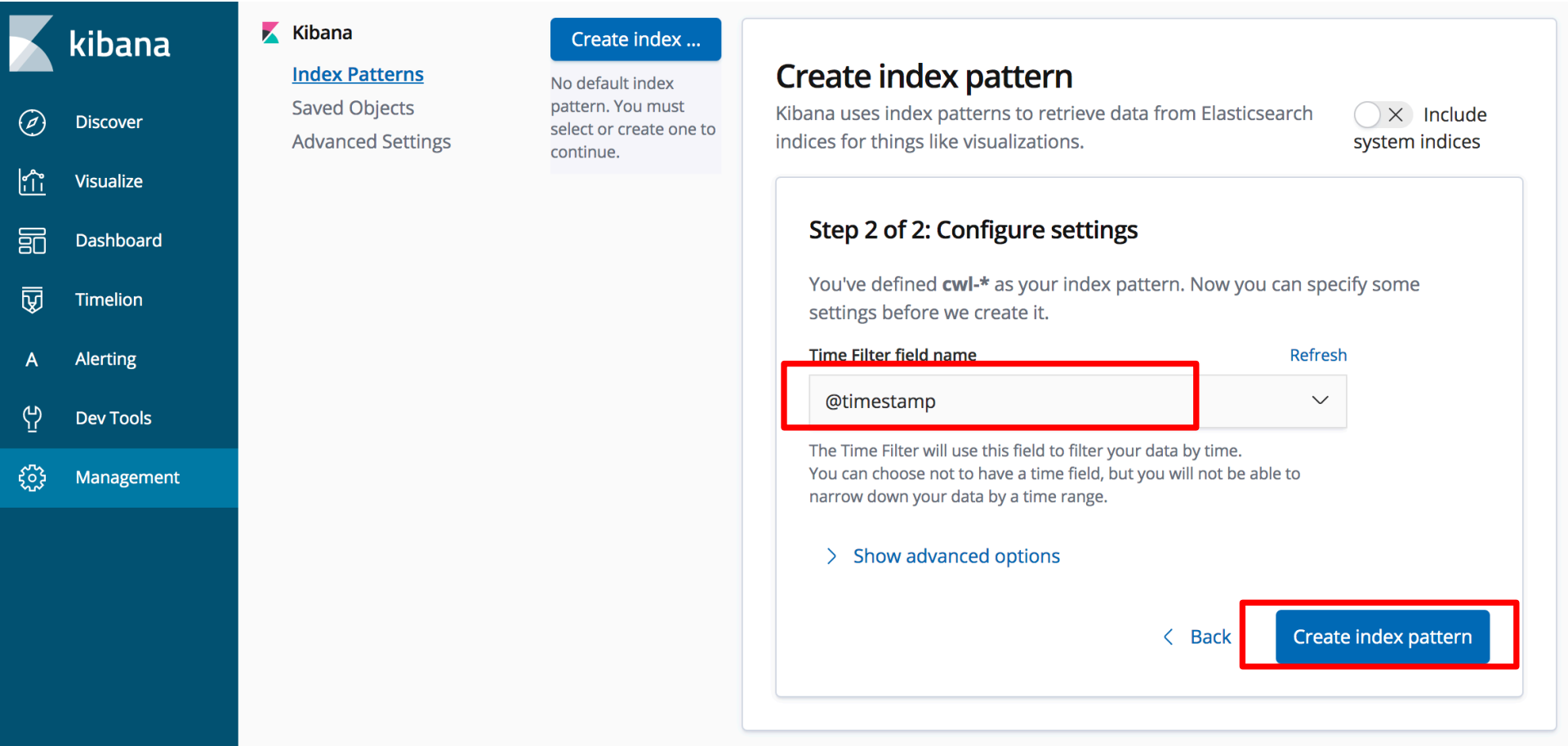
You can cancel the streaming of your log data to Amazon Elasticsearch Service any time by removing the subscription filter associated with your log group.

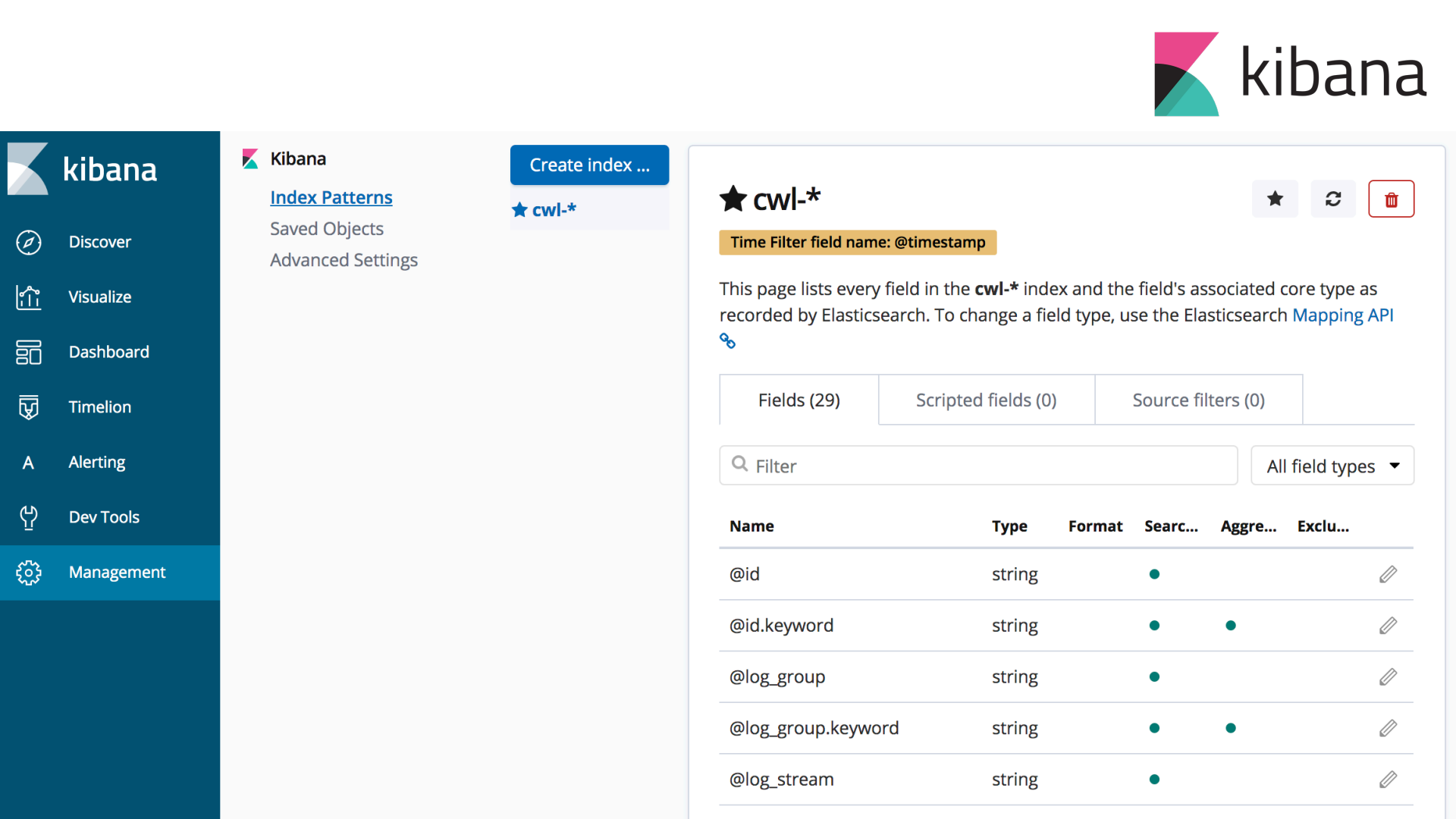
kibana



cwl-*

aws


kibana
Discover
Visualize
Dashboard
Timelion
Alerting
Dev Tools
Management
Kibana
Index Patterns
Saved Objects
Advanced Settings
Create index ...
No default index pattern. You must select or create one to continue.
Create index pattern
Kibana uses index patterns to retrieve data from Elasticsearch indices for things like visualizations.
Include system indices
Step 2 of 2: Configure settings
You've defined cwl-* as your index pattern. Now you can specify some settings before we create it.
Time Filter field name
Refresh
@timestamp
The Time Filter will use this field to filter your data by time.
You can choose not to have a time field, but you will not be able to narrow down your data by a time range.
Show advanced options
Back
Create index pattern


kibana
kibana
Discover
Visualize
Dashboard
Timelion
Alerting
Dev Tools
Management
Kibana
Index Patterns
Saved Objects
Advanced Settings
Create index ...
cwl-*
cwl-*
Time Filter field name: @timestamp
This page lists every field in the cwl-* index and the field's associated core type as recorded by Elasticsearch. To change a field type, use the Elasticsearch Mapping API
Fields (29)
Scripted fields (0)
Source filters (0)
Filter
All field types
Name
Type
Format
Searc...
Aggre...
Exclu...
@id
string
@id.keyword
string
@log_group
string
@log_group.keyword
string
@log_stream
string

kibana

50 hits New Save Open Share Inspect Auto-refresh August 11th 2019, 21:45:20.817 to August 11th 2019, 21:45:20.897

>_ 200 Options Refresh

Add a filter +

- Discover
- Visualize
- Dashboard
- Timelion
- Alerting
- Dev Tools
- Management

cwl-*

Selected fields

? _source

Available fields

t @id

t **@log_group**

Top 5 values in 50 / 50 records

helloworld_loggroup 100.0%

t @log_stream

t **@message**

Top 5 values in 50 / 50 records

13.231.21.130 - - [11... 2.0%

13.231.21.130 - - [11... 2.0%

13.231.21.130 - - [11... 2.0%

August 11th 2019, 21:45:20.817 - August 11th 2019, 21:45:20.897 — Auto

Time	_source
August 11th 2019, 21:45:20.841	@message: 13.231.21.130 - - [11/Aug/2019:12:45:20 +0000] "GET /helloworld HTTP/1.1" 200 20 e129c3fb-bc35-11e9-a4a8-71d913e77613 date: 11/Aug/2019:12:45:20 +0000 request: GET /helloworld HTTP/1.1 ident: - bytes: 20 e129c3fb-bc35-11e9-a4a8-71d913e77613 host: 13.231.21.130 authuser: - status: 200
August 11th 2019, 21:45:20.843	@message: 13.231.21.130 - - [11/Aug/2019:12:45:20 +0000] "GET /helloworld HTTP/1.1" 200 20 e12a121e-bc35-11e9-a4a8-71d913e77613 date: 11/Aug/2019:12:45:20 +0000 request: GET /helloworld HTTP/1.1 ident: - bytes: 20 e12a121e-bc35-11e9-a4a8-71d913e77613 host: 13.231.21.130 authuser: - status: 200

kibana

1 hit New Save Open Share Inspect Auto-refresh August 11th 2019, 21:45:20.817 to August 11th 2019, 21:45:20.897

>_ e12a121e Options Refresh

Add a filter +

- Discover
- Visualize
- Dashboard
- Timelion
- Alerting
- Dev Tools
- Management

cwl-*

Selected fields

? _source

Available fields

t @id

t **@log_group**

Top 5 values in 1 / 1 records

helloworld_loggroup 100.0%

t @log_stream

t **@message**

Top 5 values in 1 / 1 records

13.231.21.130 - - [11... 100.0%

t @owner

@timestamp

August 11th 2019, 21:45:20.817 - August 11th 2019, 21:45:20.897 — Auto

Time	_source
August 11th 2019, 21:45:20.843	bytes: 20 e12a121e-bc35-11e9-a4a8-71d913e77613 @message: 13.231.21.130 - - [11/Aug/2019:12:45:20 +0000] "GET /helloworld HTTP/1.1" 200 20 e12a121e-bc35-11e9-a4a8-71d913e77613 date: 11/Aug/2019:12:45:20 +0000 request: GET /helloworld HTTP/1.1 ident: - host: 13.231.21.130 authuser: - status: 200

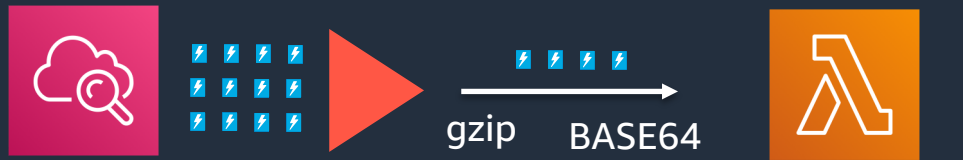
AWS Lambdaとの連携

aws

CloudWatch LogStreamをLambdaのEvent引数で受け取る

```
{
  "awslogs": {
    "data": "H4sIAAAAAAAAAI1SwYrbMBD9FaNr18mMJVmSb4
             Em6WFLC/FtNwTZmnUMTpzacrdLyL934qXsaaESQ
             (snip)
             AsHJKorPaoEWGu77ZDv104ZsjdV3/2g9dODDazOh
            wY81M8TuP8be6inGmc3lVB3G7721+9vgAVcwIAAA=="
  }
}
```

Event引数

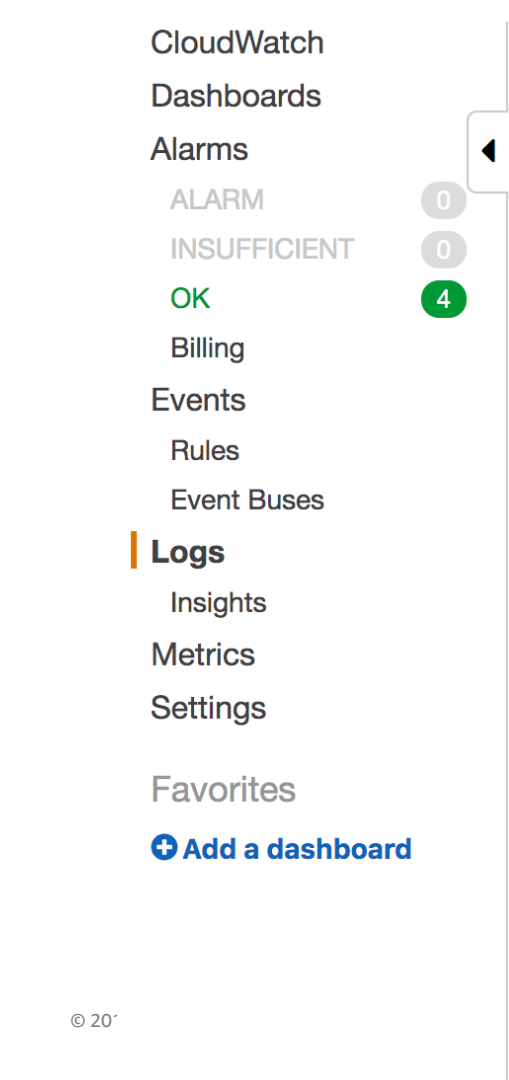


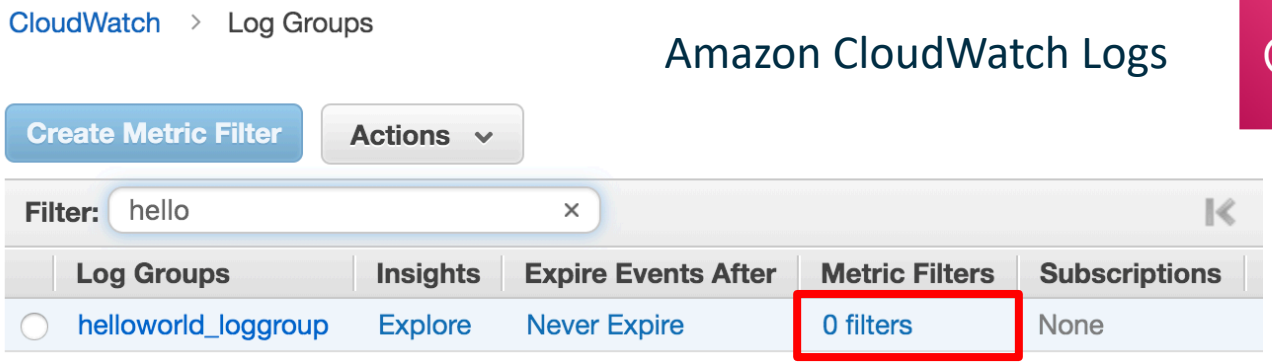
Amazon CloudWatch　filter　AWS Lambda

- LogStreamは、gzip圧縮され、BASE64エンコードされている
- 全streamを流すのではなく、フィルタする

aws

Amazon CloudWatch Logs





Step 1: Define Pattern

Step 2: Assign Metric

Define Logs Metric Filter

Amazon CloudWatch Logs

Filter for Log Group: helloworld_loggroup

You can use metric filters to monitor events in a log group as they are sent to CloudWatch Logs. You can monitor and count specific terms or extract values from log events and associate the results with a metric. Learn more about pattern syntax.

Filter Pattern

helloworld

Show examples

Select Log Data to Test

00cbfdb04daad5c1761d9890b1735a70

Test Pattern

Clear

```
13.231.21.130 - - [11/Aug/2019:12:45:20 +0000] "GET /helloworld HTTP/1.1" 200 20 e129eaec-bc35-11e9-b37
13.231.21.130 - - [11/Aug/2019:12:45:21 +0000] "GET /helloworld HTTP/1.1" 200 20 e150fb30-bc35-11e9-b37
13.231.21.130 - - [11/Aug/2019:12:45:21 +0000] "GET /helloworld HTTP/1.1" 200 20 e158754e-bc35-11e9-b37
13.231.21.130 - - [11/Aug/2019:12:45:21 +0000] "GET /helloworld HTTP/1.1" 200 20 e175e890-bc35-11e9-b37
13.231.21.130 - - [11/Aug/2019:12:53:58 +0000] "GET /helloworld HTTP/1.1" 200 20 157e1c37-bc37-11e9-b37
13.231.21.130 - - [11/Aug/2019:12:53:58 +0000] "GET /helloworld HTTP/1.1" 200 20 15859552-bc37-11e9-b37
13.231.21.130 - - [11/Aug/2019:12:53:58 +0000] "GET /helloworld HTTP/1.1" 200 20 159091e4-bc37-11e9-b37
```

Results

Found **8** matches out of 8 event(s) in the sample log.

Show test results

Cancel

Assign Metric

Step 1: Define Pattern

Step 2: Assign Metric

Create Metric Filter and Assign a Metric

Filter for Log Group: helloworld_loggroup

Log events that match the pattern you define are recorded to the metric that you specify. You can graph the metric and set alarms to notify you.

Filter Name: helloworld

Filter Pattern: helloworld

Metric Details

Metric Namespace: LogMetrics Create new namespace

Metric Name: helloworld_metric

Show advanced metric settings

Cancel Previous Create Filter

Create function Info

AWS Lambda

Choose one of the following options to create your function.

Author from scratch

Start with a simple Hello World example.

Use a blueprint

Build a Lambda application from sample code and configuration presets for common use cases.

Browse serverless app repository

Deploy a sample Lambda application from the AWS Serverless Application Repository.

Blueprints Info

Export

Add filter

1

Keyword : cloudwatch-logs-process-data

cloudwatch-logs-process-data

A real-time consumer of log events ingested by an Amazon CloudWatch Logs log group.

nodejs · logs · cloudwatch

Cancel Configure

CloudWatch Logs trigger

Remove

Log group

Please select the CloudWatch Logs log group that serves as the event source. Log Events sent to the log group will trigger your Lambda function with the contents of the logs received.

helloworld_loggroup

Filter name

Choose a name for your filter.

helloworld

Filter pattern

Enter an optional filter pattern.

Lambda will add the necessary permissions for Amazon CloudWatch Logs to invoke your Lambda function from this trigger. Learn more about the Lambda permissions model.

Enable trigger

Enable the trigger now, or create it in a disabled state for testing (recommended).

Function code Info

Code entry type

Edit code inline

Runtime

Node.js 8.10

File Edit Find View Go Tools Window

Environment

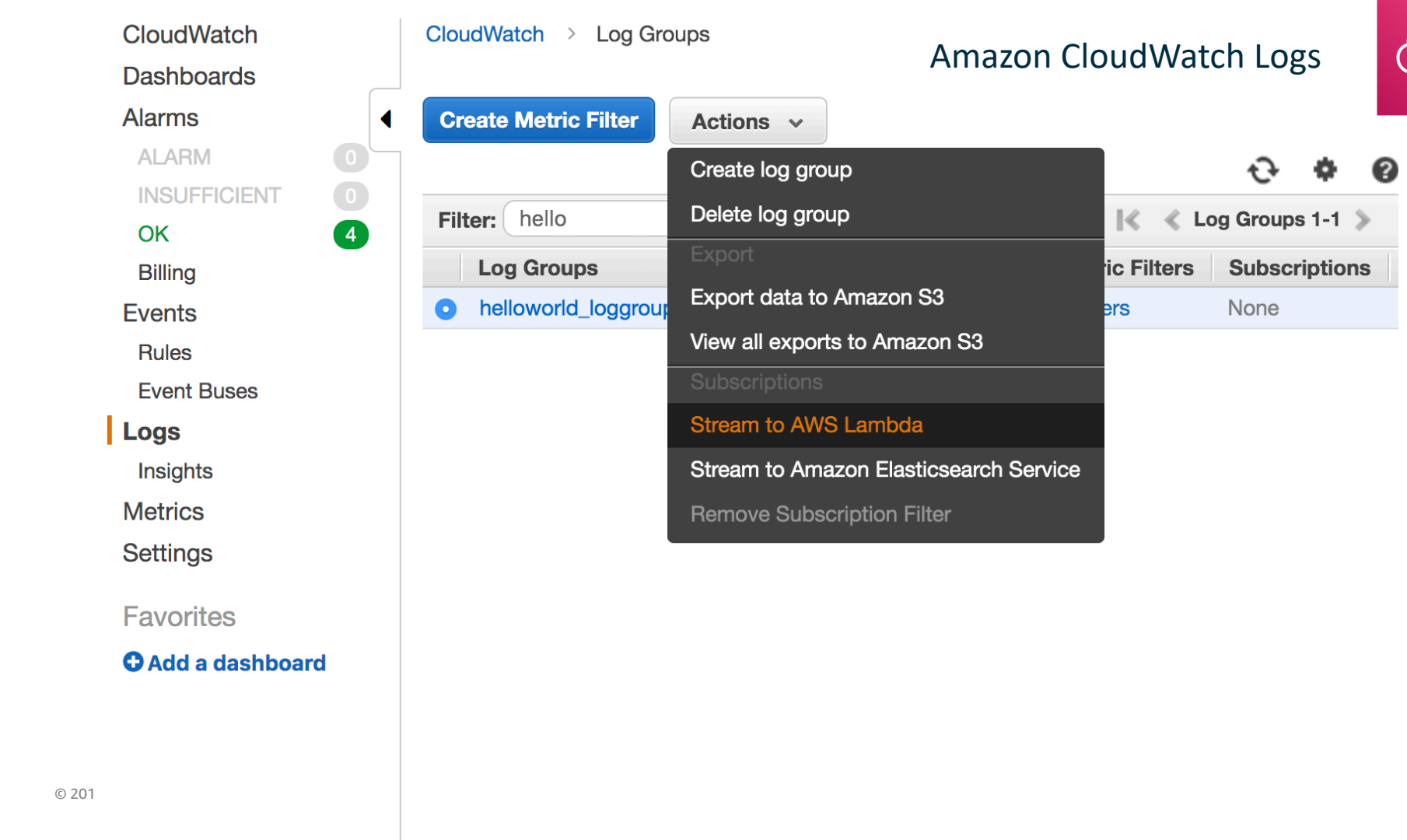
LogStreamTriggerFu

index.js

index.js

```
1  const zlib = require('zlib');
2
3  exports.handler = async (event, context) => {
4      const payload = new Buffer(event.awslogs.data, 'base64');
5      const parsed = JSON.parse(zlib.gunzipSync(payload).toString('utf8'));
6      console.log('Decoded payload:', JSON.stringify(parsed));
7      return `Successfully processed ${parsed.logEvents.length} log events.`;
8  };
9
```


Amazon CloudWatch Logs



Step 1: Choose Destination

Step 2: Configure Log Format and Filters

Step 3: Review

Lambda Function

Choose the Lambda function that should execute when a log event matches the filter you are going to specify. Learn more about Lambda functions.

Lambda Function * LogStreamTriggerFunc

Warning

Streaming large amounts of CloudWatch Logs data to other destinations might result in high usage charges. We recommend that you create a Budget in the Billing and Cost Management console. For more information, see Managing Your Costs with Budgets.

Cancel | Previous | Next | Start Streaming

Step 1: Choose Destination

Step 2: Configure Log Format and Filters

Step 3: Review

Review & Start Streaming to Amazon Lambda Service

CloudWatch Logs will stream data from the log group "helloworld_loggroup" to the Lambda function "LogStreamTriggerFunc".

Choose Destination

Edit

Lambda Function Name LogStreamTriggerFunc

Configure Log Format and Filters

Edit

Subscription Filter Name LambdaStream_LogStreamTriggerFunc

Filter Pattern [host, ident, authuser, date, request, status, bytes]

Cancel | Previous | Next | Start Streaming

Filter events

Message

2019-08-10 07:37:48

```
START RequestId: 7cab1ff5-6ea8-4a5c-afb0-a1ca7fe8a40f Version: $LATEST


2019-08-10T07:37:48.810Z 7cab1ff5-6ea8-4a5c-afb0-a1ca7fe8a40f Decoded payload:
{
    "messageType": "DATA_MESSAGE",
    "owner": [redacted]
    "logGroup": "helloworld_loggroup",
    "logStream": "05d6d898fc9676b8468c3fe74ded9382",
    "subscriptionFilters": [
        "LambdaStream_LogStreamTriggerFunc"
    ],
    "logEvents": [
        {
            "id": "34910091598381004365381727989893135091847801358698938368",
            "timestamp": 1565422647880,
            "message": "27.0.3.145 - - [10/Aug/2019:07:37:27 +0000] \"GET /helloworld HTTP/1.1\" 200 20 b401dcd7-bb41-11e9-9070-6dac7fc19459",
            "extractedFields": {
                "date": "10/Aug/2019:07:37:27 +0000",
                "request": "GET /helloworld HTTP/1.1",
                "ident": "-",
                "bytes": "20 b401dcd7-bb41-11e9-9070-6dac7fc19459",
                "host": "27.0.3.145",
                "authuser": "-",
                "status": "200"
            }
        }
    ]
}

END RequestId: 7cab1ff5-6ea8-4a5c-afb0-a1ca7fe8a40f
```


Traceを使った可視化

AWS X-Ray

リクエスト実行状況の確認

アプリケーションを構成する
個々のサービスやリソースの
実行結果ステータスを集計し、
アプリケーションの実行状況を
エンドツーエンドで確認可能

アプリケーションの問題の検出

アプリケーションの実行状況に
ついての関連する情報を収集し、
問題の根本原因を調査可能

AWSとの連携

Amazon EC2, Amazon ECS,
AWS Lambda, AWS Elastic Beanstalk
と連携

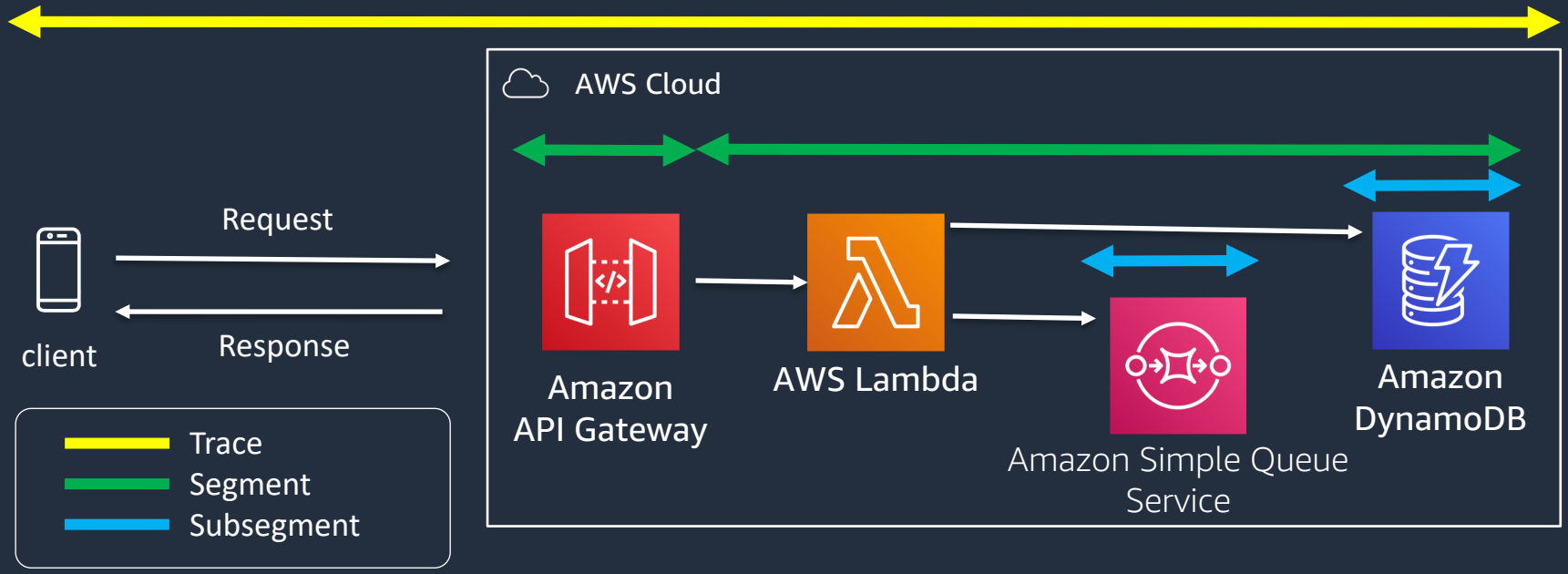
アプリケーションのパフォーマンス向上

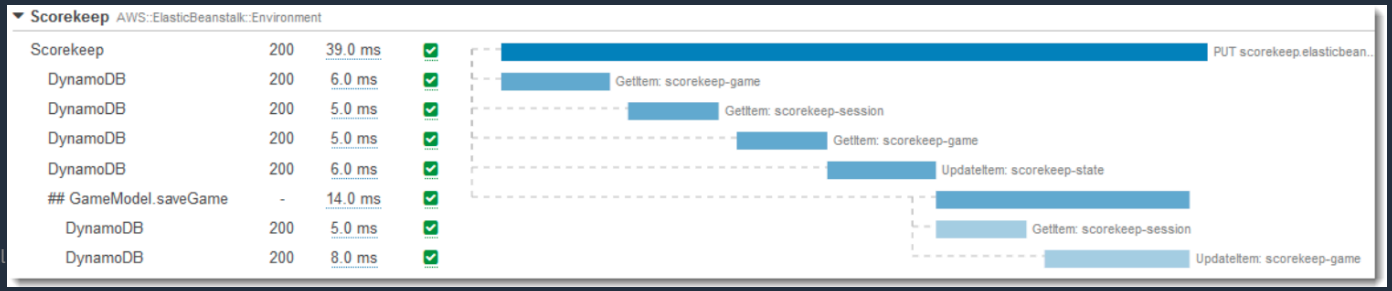
サービスやリソースの関係をリアルタイムで
表示し、レイテンシ増加やパフォーマンス低下
などのボトルネックを特定可能

さまざまなアプリケーション向けの設計

非同期のシンプルなイベント呼び出し、
３層のウェブアプリケーション、数千の
サービスから構成される複雑なマイクロ
サービスも分析可能

AWS X-Ray コンポーネント

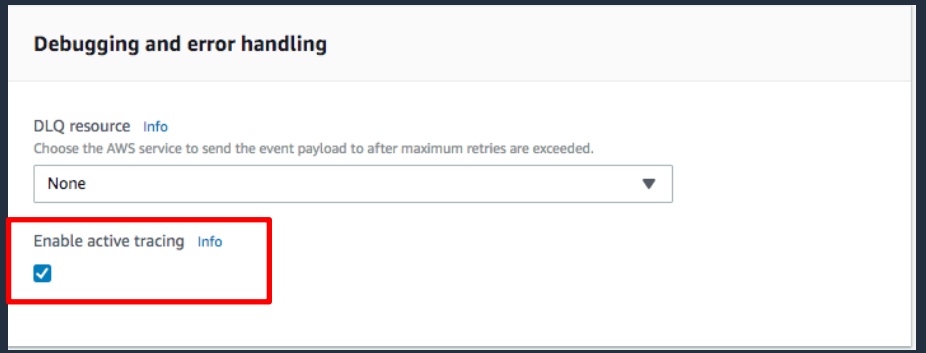




AWS LambdaでX-Rayを利用するためには、、

Lambda用パッケージにX-Ray SDKを追加し、アクティブトレースをONにする。
IAM roleも必要、Managed policyが用意されている

- Pythonの場合
  - Python 2.7, Python3.6以降
- Node.jsの場合
  - Node.js 4.3以降
- Javaの場合
  - Java8以降
- Goの場合
  - Go1.7以降
- .NETの場合
  - .NET Core 2.0以降



CLIであれば、
--tracing-config オプション

実装例(Python)

```
from aws_xray_sdk.core import xray_recorder
from aws_xray_sdk.core import patch_all

patch_all()
def main(event, cotext):
    xray_recorder.begin_segment('main segment')
    (main処理)
        xray_recorder.begin_subsegment('sub segment')
        (処理A)
        xray_recorder.end_subsegment('sub segment')
    xray_recorder.end_segment
    return
```


aws

実装例(Python)

```
from aws_xray_sdk.core import xray_recorder
from aws_xray_sdk.core import patch_all
```

patch_all()

対応しているライブラリにパッチ適用
pythonの場合
-botocore, boto3
-requests
-sqlite3
-mysq-connector-python
など

patch_allではなく patch('boto3')などとすることも可能

実装例(Python)

```
from aws_xray_sdk.core import xray_recorder
from aws_xray_sdk.core import patch_all

patch_all()
def main(event, cotext):
    xray_recorder.begin_segment('main segment')
    (main処理)
        xray_recorder.begin_subsegment('sub segment')
        (処理A)
        xray_recorder.end_subsegment('sub segment')
    xray_recorder.end_segment
    return
```

処理全体のセグメントを定義

実装例(Python)

```
from aws_xray_sdk.core import xray_recorder
from aws_xray_sdk.core import patch_all

patch_all()
def main(event, cotext):
    xray_recorder.begin_segment('main segment')
    (main処理)
        xray_recorder.begin_subsegment('sub segment')
        (処理A)
        xray_recorder.end_subsegment('sub segment')
    xray_recorder.end_segment
    return
```

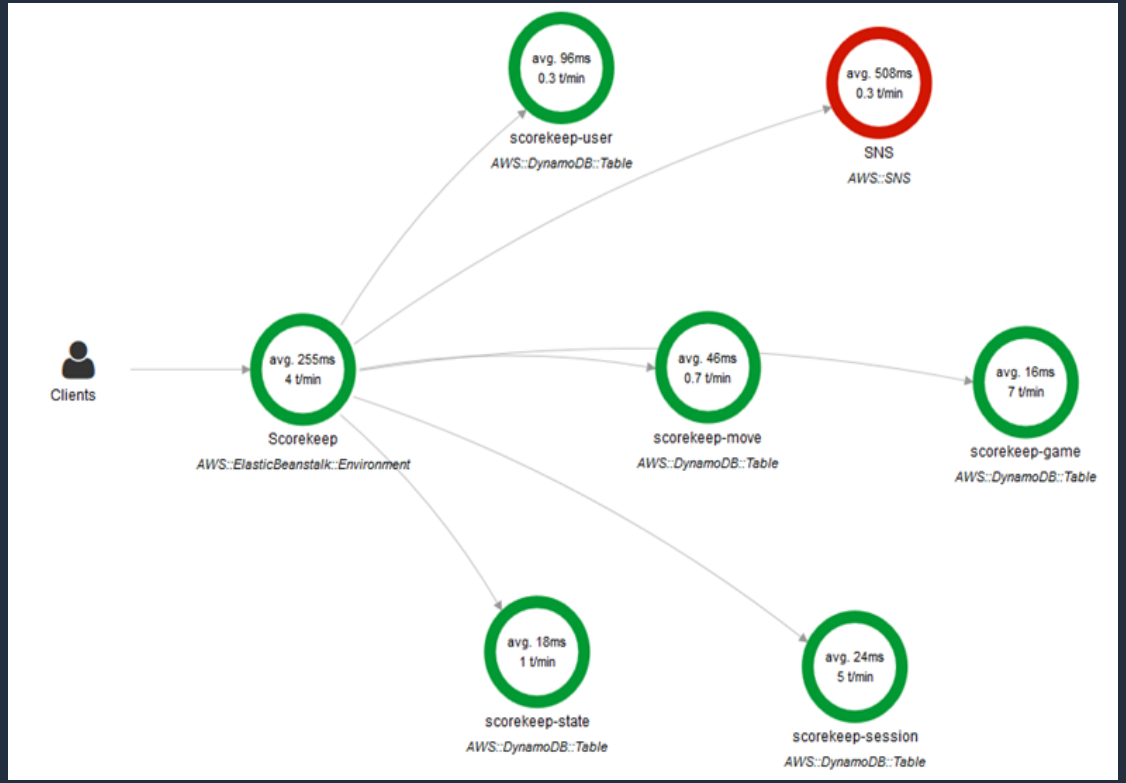
処理の中でサブセグメントを作ることも可能

サービスグラフ

各ノードの呼び出しの結果を色で分類、割合を円グラフに（サービスマップ）

グリーン　成功した呼び出し
レッド　5xx errors
イエロー　4xx errors
パープル　429 Too Many Requests (スロットリングエラー)

- 平均レイテンシ (ms)
- トレース数 (trace/min)
- サービス名
- サービスの分類




まとめ

まとめ

AWS サーバーレスサービスに関わる考え方やMetricsをご紹介してきました。

- アプリケーション開発、運用においては一般的な知識と経験はサーバーレス でも活用できます
- ログ設計なども通常のアプリケーション設計と変わらない
  - CloudWatch Logsや3rd partyツールなどの監視ツールも同様
- サーバーレス サービスの制約や仕様を理解し、今までのアプリケーション開発経験と組み合わせることで従来のアプリケーションよりも、柔軟なアプリケーション開発/運用もできる可能性がある

Q&A

お答えできなかったご質問については
AWS Japan Blog 「https://aws.amazon.com/jp/blogs/news/」にて
後日掲載します。

8 月の Black Belt Online Seminar 配信予定

https://amzn.to/JPWebinar

~~08/06 (火) 12:00-13:00 AWS Glue~~
~~08/13 (火) 12:00-13:00 実践的 Serverless セキュリティプラクティス~~
~~08/14 (水) 18:00-19:00 AWS Serverless Application Model~~
08/20 (火) 12:00-13:00 Serverless モニタリング
08/21 (水) 18:00-19:00 AWS AppSync
08/28 (水) 18:00-19:00 Amazon Aurora with PostgreSQL Compatibility






DEV DAY
TOKYO

今年もやります！イノベーションをリードするDeveloperのための秋の祭典「**AWS DevDay Tokyo 2019**」
- 今おさえておくべき技術領域を網羅し、新たな時代に活躍するDeveloperとなるための知識とスキルを身に着ける2日間
- 豪華スピーカーによるゼネラルセッション
- 50にのぼるセッションとワークショップで構成。CFP枠を拡大し、より実践的なコンテンツへ

日程：2019年10月3日（木）ー　4日（金）
会場：神田明神ホール（東京都千代田区 神田明神内）
主催：アマゾン ウェブ サービス ジャパン株式会社
協賛：インテル株式会社

今年の会場は「神田明神」

aws

Save the dateはこちらへ
https://amzn.to/devdaytokyo2019

AWS の日本語資料の場所「AWS 資料」で検索

https://amzn.to/JPArchive

AWS Well-Architected 個別技術相談会

毎週”W-A個別技術相談会”を実施中

- AWSのソリューションアーキテクト(SA)に対策などを相談することも可能
- 申込みはイベント告知サイトから

(https://aws.amazon.com/jp/about-aws/events/)

AWS イベント で[検索]

ご視聴ありがとうございました

AWS 公式 Webinar
https://amzn.to/JPWebinar

過去資料
https://amzn.to/JPArchive