イチから理解するサーバーレスアプリ開発

エラー制御の勘所とモニタリング

@_kensh

© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Who am I?

Name

Kensuke Shimokawa

Company

Amazon Web Services Japan K.K.

Role

Serverless Specialist Solutions Architect

@_kensh

© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

本セッションについて

Serverlessな構成、AWS Lambdaの導入でサーバー管理はなくなるが、運用業務は発生します。
本セッションでは、サーバーレス のエラー制御・モニタリングを説明する

本セッションの対象
- AWS Lambda
- Amazon API Gateway

etc.

お話しないこと
- **Lambda@Edge のお話**
- AWS Fargate/Amazon Aurora Severlessのお話

© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

本日のアジェンダ

- AWS Lambdaのおさらい
- AWS Lambdaでのプログラミング
- 仕組みを理解しよう
- エラー制御について
- AWS Lambdaをモニタリング

© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

本日のアジェンダ

- AWS Lambdaのおさらい
- AWS Lambdaでのプログラミング
- 仕組みを理解しよう
- エラー制御について
- AWS Lambdaをモニタリング

© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

AWS Lambdaについて（おさらい）

インフラのプロビジョニング不要
管理不要

自動でスケール

価値に対する支払い

高可用かつ安全

© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

サポートされているランタイム言語

- Python 2.7, 3.6, 3.7, 3.8
- Node.js 10.x, 12.x
- .NET Core 2.1, 3.1(C# / PowerShell)
- Go 1.x
- Java 8, Java11
- Ruby 2.5, 2.7

サポートされていない言語は
Custom Runtimeを実装することで
利用可能

© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

AWS Lambdaのハンドラー

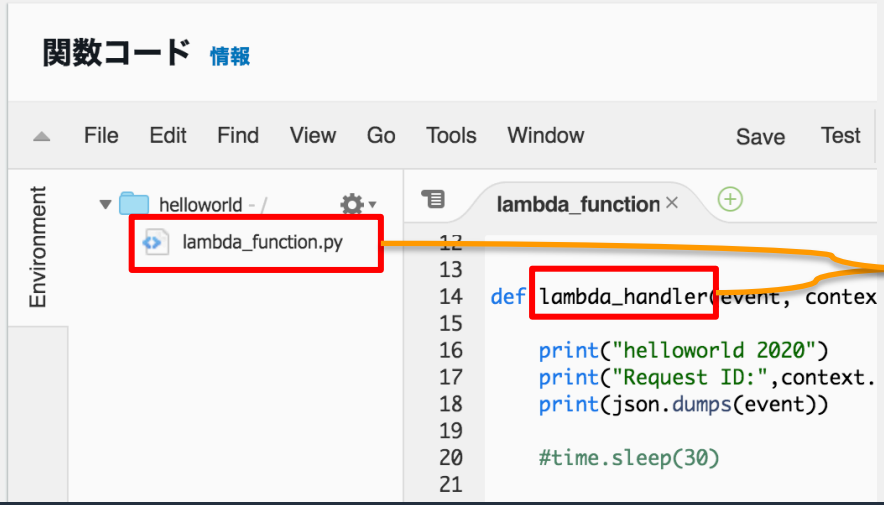
- 利用する言語の関数もしくはメソッドを指定し、実行時に呼び出すエントリポイントとする
- ハンドラーが呼び出されることで、Lambda関数のコードが実行開始される
- 呼び出しパラメータとして渡されるイベントデータを参照可能

Python3.7

```
import json

def lambda_handler(event, context):
    # 任意のコード
    return {
        'statusCode': 200,
        'body': json.dumps('Hello')
    }
```

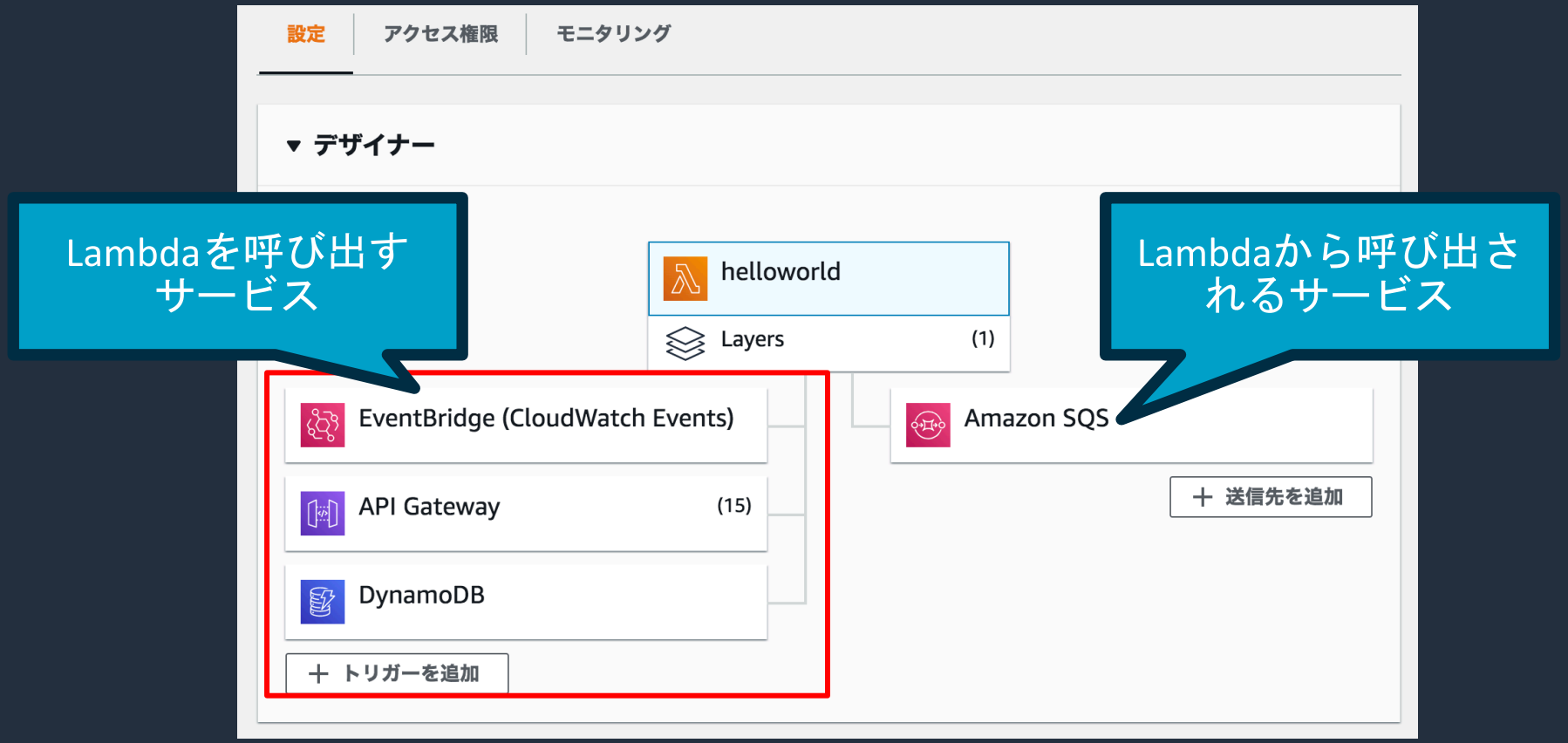
© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Management Console上の表示



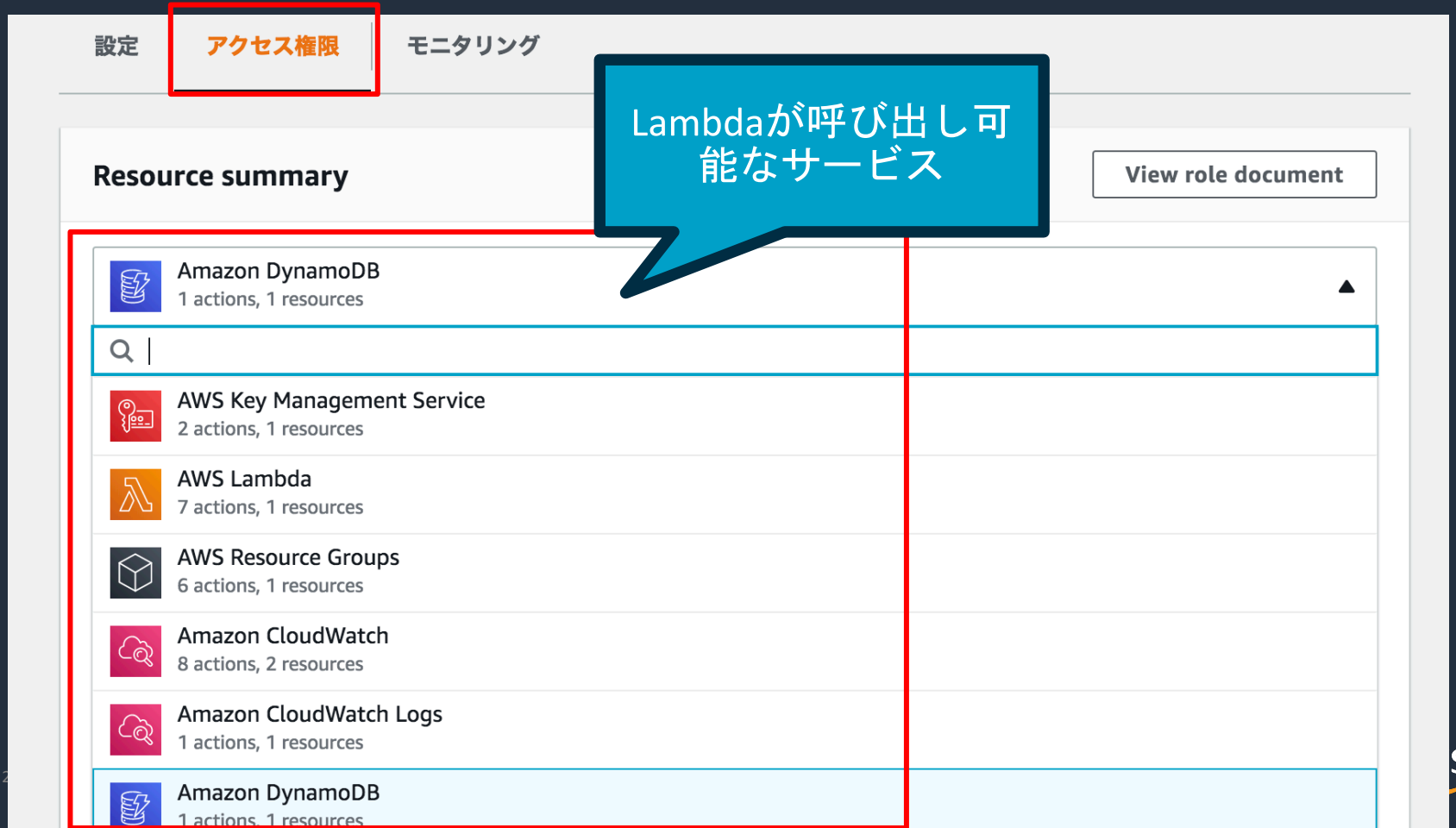
© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

他サービスとの連携



© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

他サービスとの連携


設定
アクセス権限
モニタリング
Lambdaが呼び出し可能なサービス
Resource summary
View role document
Amazon DynamoDB
1 actions, 1 resources
AWS Key Management Service
2 actions, 1 resources
AWS Lambda
7 actions, 1 resources
AWS Resource Groups
6 actions, 1 resources
Amazon CloudWatch
8 actions, 2 resources
Amazon CloudWatch Logs
1 actions, 1 resources
Amazon DynamoDB
1 actions, 1 resources

Amazon API Gateway から AWS Lambdaを呼び出した場合

Eventデータ （抜粋）


```
{
    "resource": "/helloworld",
    "path": "/helloworld",
    "httpMethod": "GET",
    "headers": {
        "Content-Type": "text/html",
        "Host": "eefsdfgr1.execute-api.ap-northeast-1.amazonaws.com",
        "User-Agent": "hey/0.0.1",
        "X-Forwarded-For": "11.23.222.132",
        "X-Forwarded-Port": "443",
        "X-Forwarded-Proto": "https"
    "queryStringParameters": null,
    "body": null
}
```


© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

本日のアジェンダ

- AWS Lambdaのおさらい
- AWS Lambdaでのプログラミング
- 仕組みを理解しよう
- エラー制御について
- AWS Lambdaをモニタリング

© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

基本的には言語のベストプラクティスに従う

python™

OpenJDK

node JS

.NET Core

© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

aws

AWS Lambdaのプログラミングの基本ルール

- ステートレス
  - 関数で保持したデータは揮発するので、保持したいデータ（ステート）は別のところで管理（Amazon DynamoDB, Amazon S3等）
- べき等
  - At least onceの原則に従うと、関数は複数回起動することがある
    - 例）Amazon S3から非同期に呼び出される場合のリトライ対応

      ※詳細は後ほど

© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

https://youtu.be/QvPgjEwgiew

[AWS Black Belt Online Seminar]

Let’s Dive Deep into AWS Lambda

Solutions Architect 西谷 圭介
2019/04/02

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

過去資料
https://amzn.to/JPArchive

© 2019, Amazon Web Services, Inc. or its Affiliates. All rights reserved. Amazon Confidential and Trademark

0:00 / 1:01:22

AWS LambdaでのLog出力の方法

- 基本的には、言語の標準出力機能をそのまま使えます。


```
print ("hello, world")
```


```
Console.WriteLine("hello, world");
```


```
System.out.println("hello, world");
```


```
console.log("hello, world");
```

© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

aws

AWS Lambdaでのログ用権限設定

設定 | アクセス権限 | モニタリング

Lambdaが呼び出し可能なサービス

Resource summary

View role document

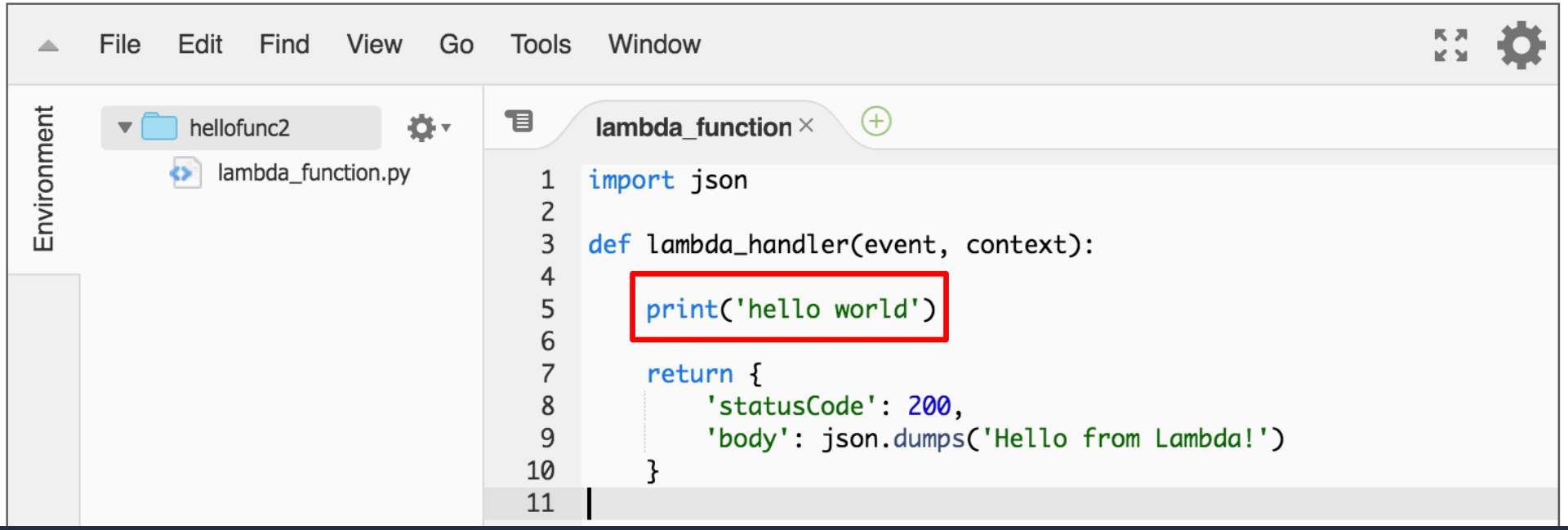
Amazon CloudWatch Logs
3 actions, 2 resources

To view the resources and actions that your function has permission to access, choose a service.

アクション別 | リソース別

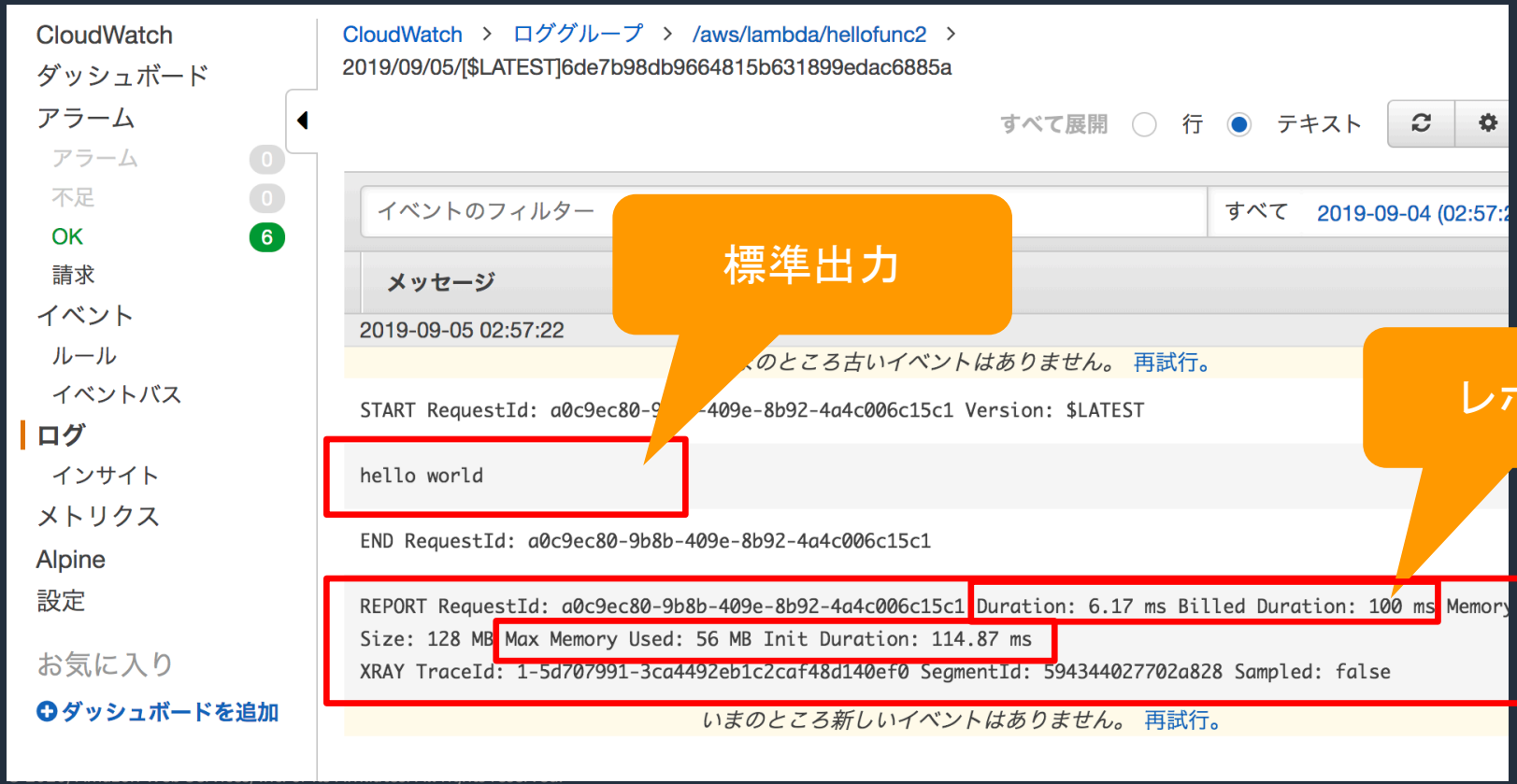
アクション	リソース
logs:CreateLogGroup	Allow: arn:aws:logs:ap-northeast-1:494319857181:*
logs:CreateLogStream	Allow: arn:aws:logs:ap-northeast-1:494319857181:log-group:/aws/lambda/binary:*

AWS Lambdaで標準出力



© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Amazon CloudWatch Logsでの表示



AWS LambdaでのLog出力の方法

- ただし、ログ運用のためにはLoggerなどログ出力ライブラリを用いる

Logの検索性を高めることで
後のLog解析に役に立つ

```
import logging
logger = logging.getLogger()
logger.setLevel(logging.INFO)

def lambda_handler(event, context):
    logger.debug('debug')
    logger.info('info')
    logger.warning('warn')
    logger.error('error')
    return {
        'statusCode': 200,
        'body': json.dumps('Hello from Lambda!')
    }
```

© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

aws

Loggerの出力例

- レベルタイプ
- タイムスタンプ
- リクエストID
- 標準出力

© 20

aws

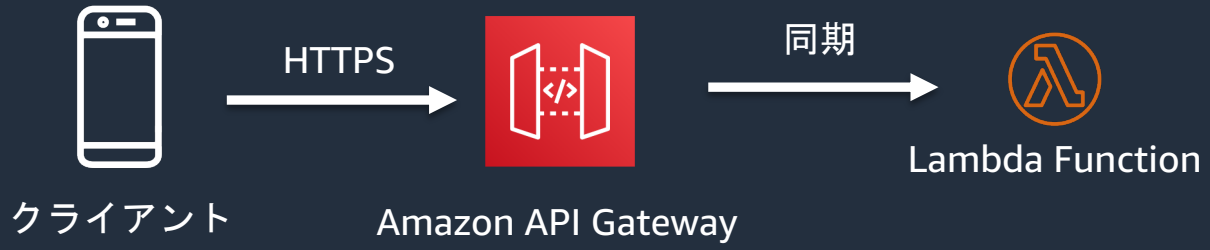
本日のアジェンダ

- AWS Lambdaのおさらい
- AWS Lambdaでのプログラミング
- 仕組みを理解しよう
- エラー制御について
- AWS Lambdaをモニタリング

© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

AWS Lambdaの仕組み

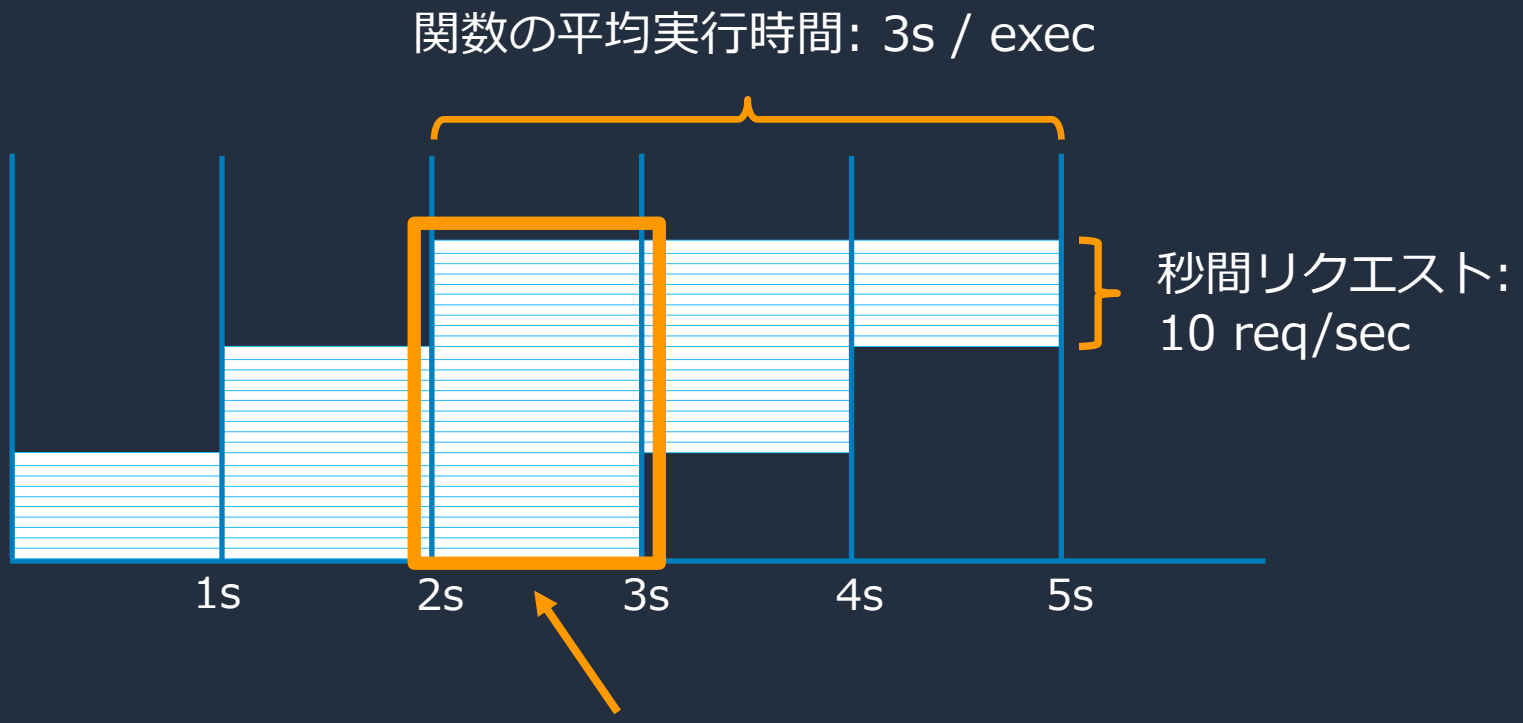
- RequestResponseベース



リクエスト数が伸びれば、Lambdaの同時実行数も伸びる

© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

同時実行数とは



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

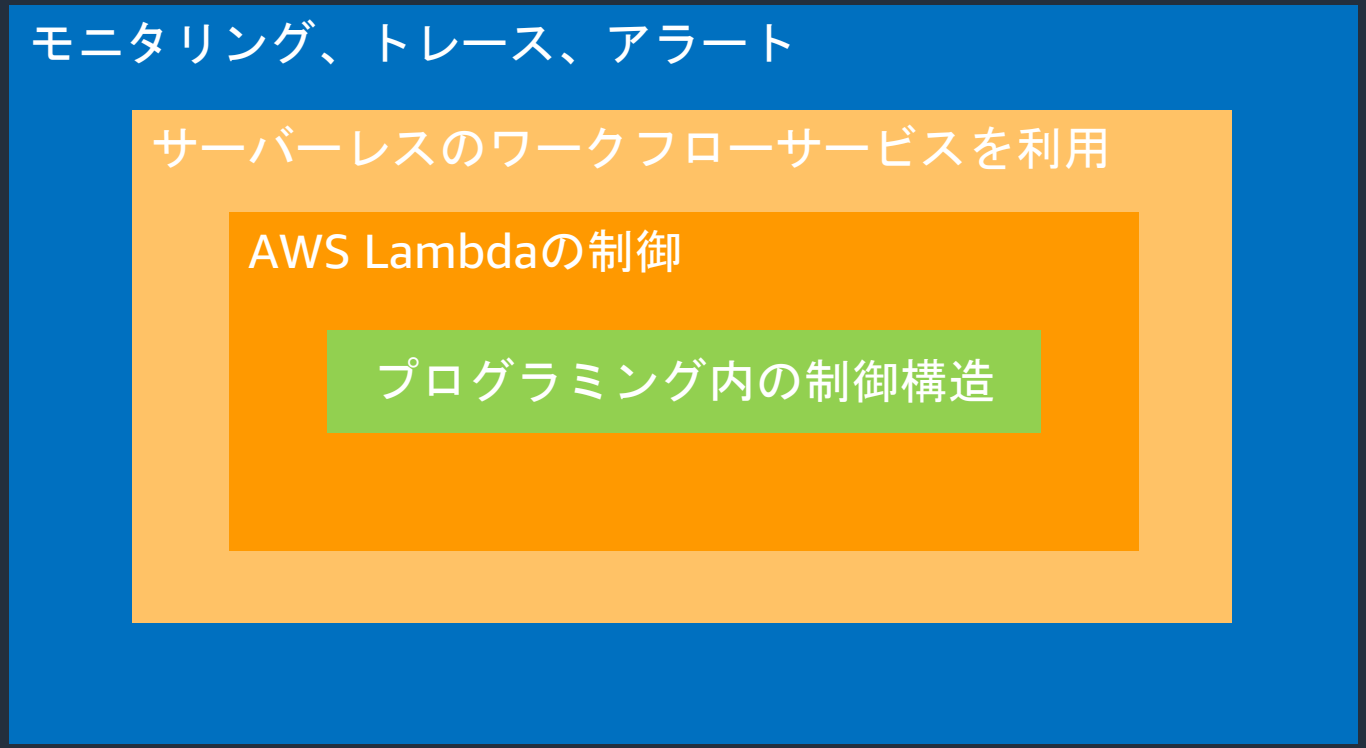
© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

本日のアジェンダ

- AWS Lambdaのおさらい
- AWS Lambdaでのプログラミング
- 仕組みを理解しよう
- エラー制御について
- AWS Lambdaをモニタリング

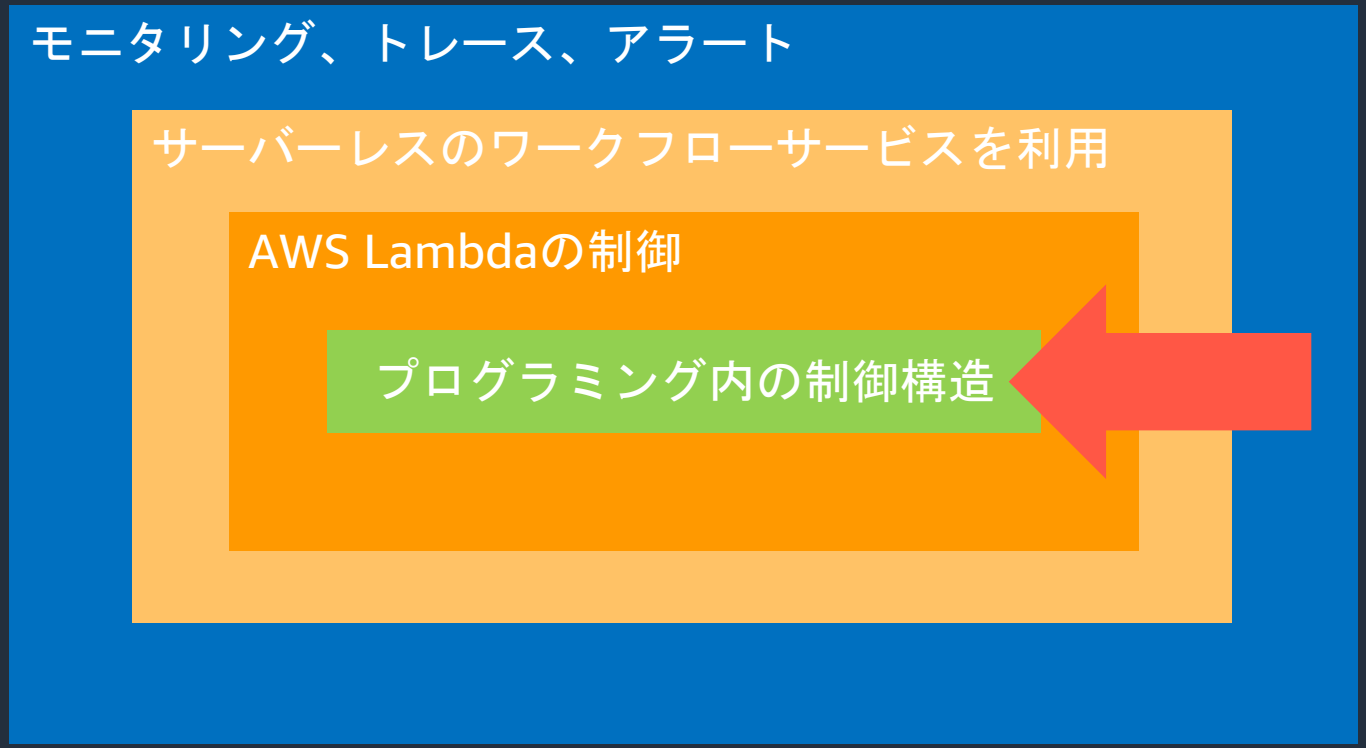
© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

エラー制御についてのパターン



© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

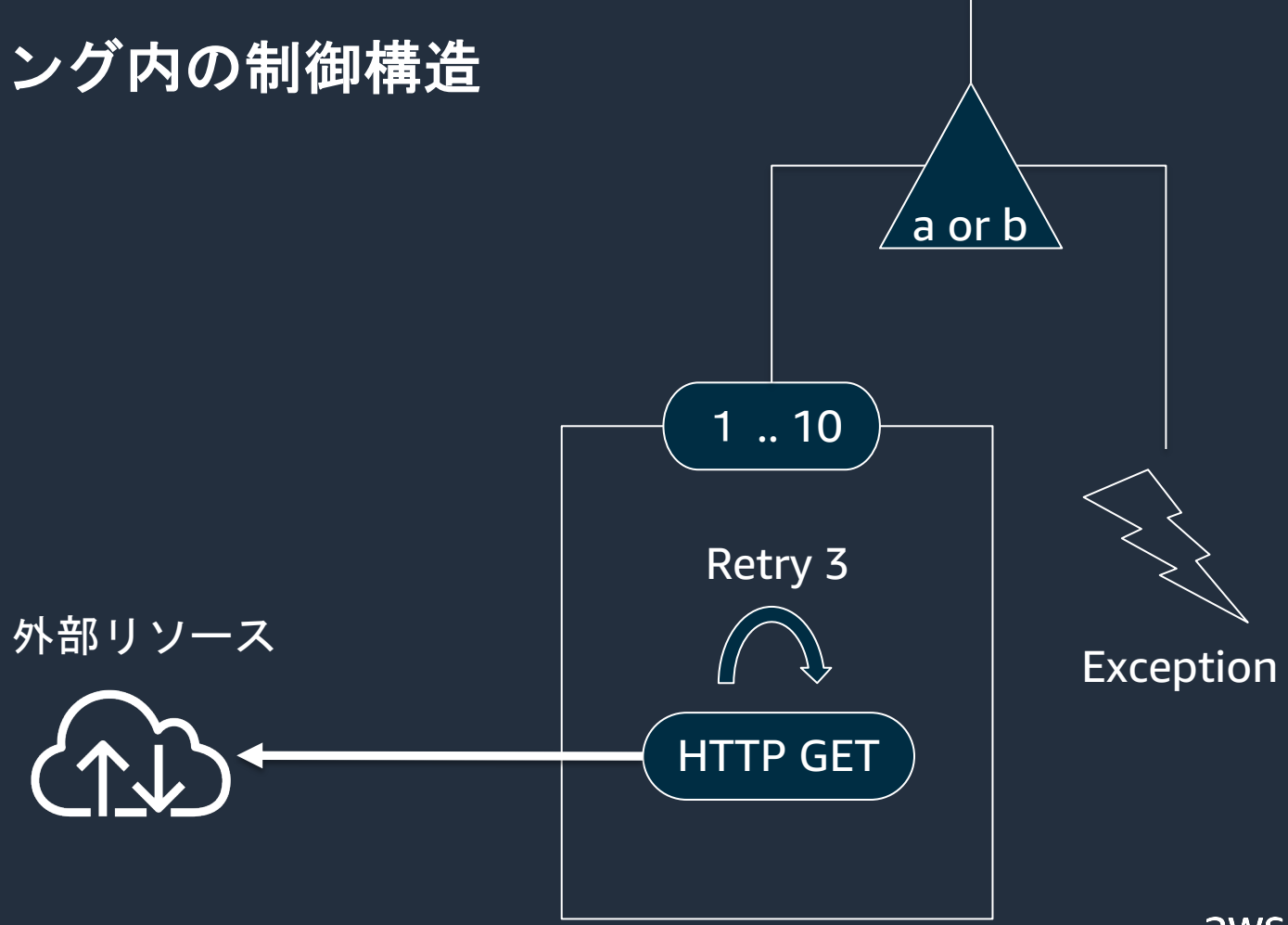
エラー制御についてのパターン



© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

プログラミング内の制御構造

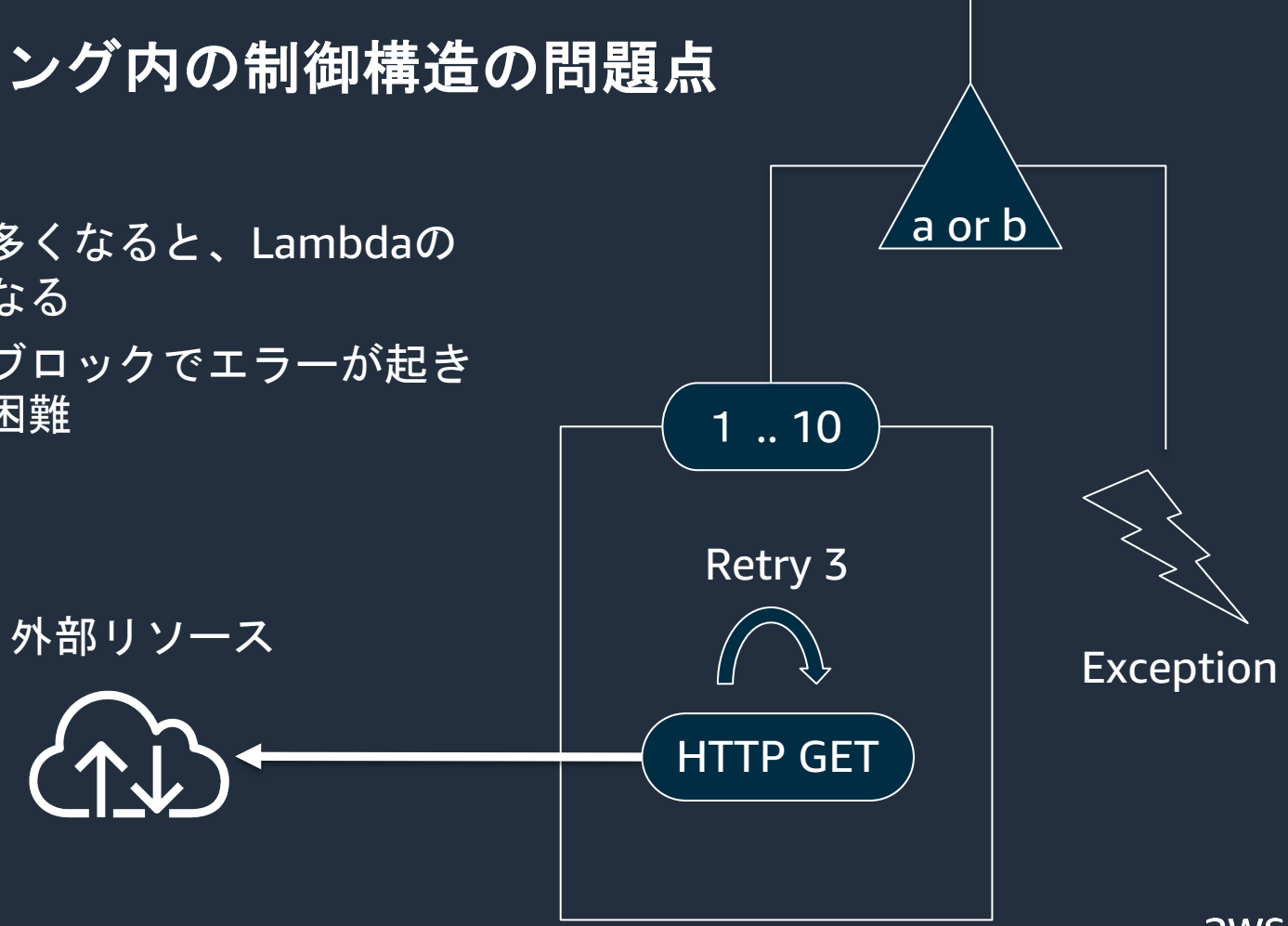
- 分岐
- 繰り返し
- 例外
- リトライ



© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

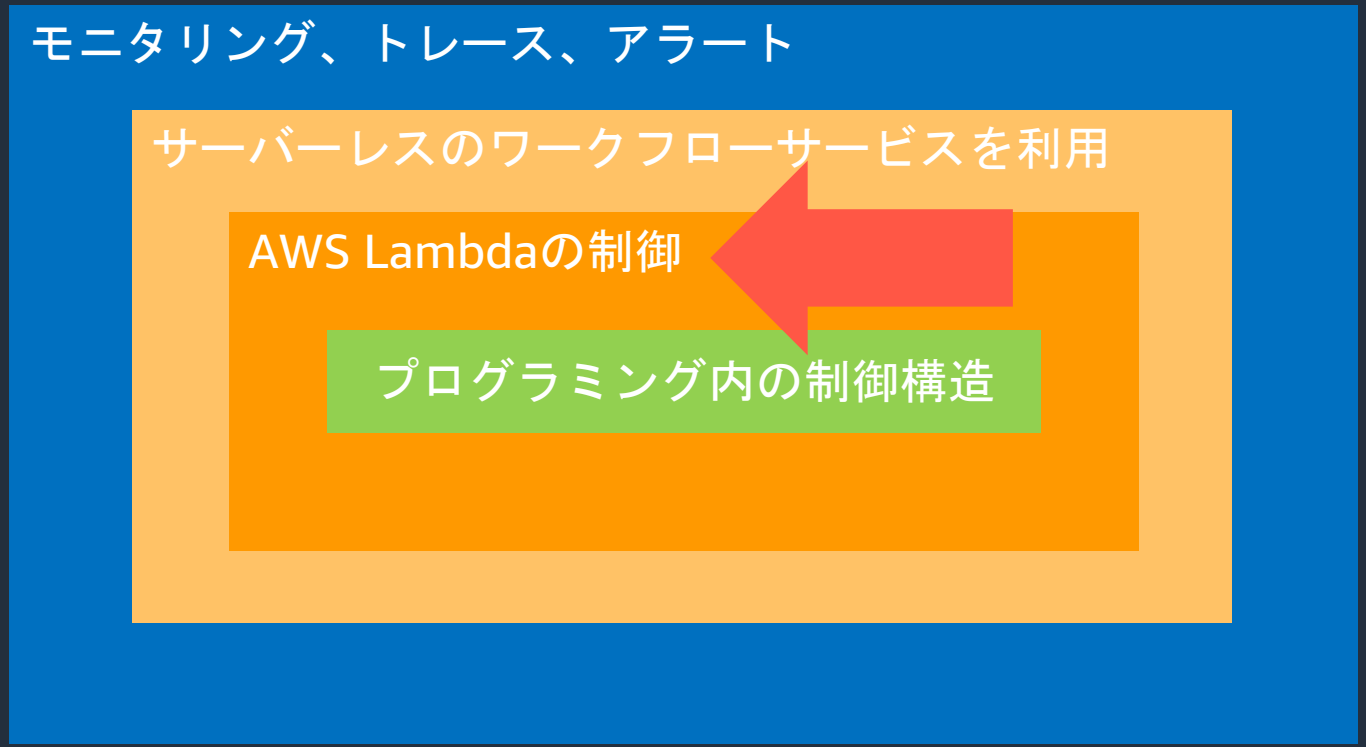
プログラミング内の制御構造の問題点

- コード量が多くなると、Lambdaの起動が遅くなる
- どのコードブロックでエラーが起きたか発見が困難



© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

エラー制御についてのパターン



© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

AWS Lambdaの非同期呼び出しについて

関数を間接的に呼び出す際、呼び出し元やリクエストが遭遇するすべてのサービスの再試行動作について把握する必要があります。これには、以下のシナリオが含まれます。

- **非同期呼び出し** – Lambda は、関数エラーを 2 回再試行します。関数にすべての受信リクエストを処理する十分なキャパシティがない場合、関数に送信されるまで、イベントはキューの中に数時間または数日間保持される可能性があります。正常に処理できなかったイベントを把握するために、デッドレターキューを設定できます。詳細については、「非同期呼び出し」を参照してください。

https://docs.aws.amazon.com/ja_jp/lambda/latest/dg/retries-on-errors.html

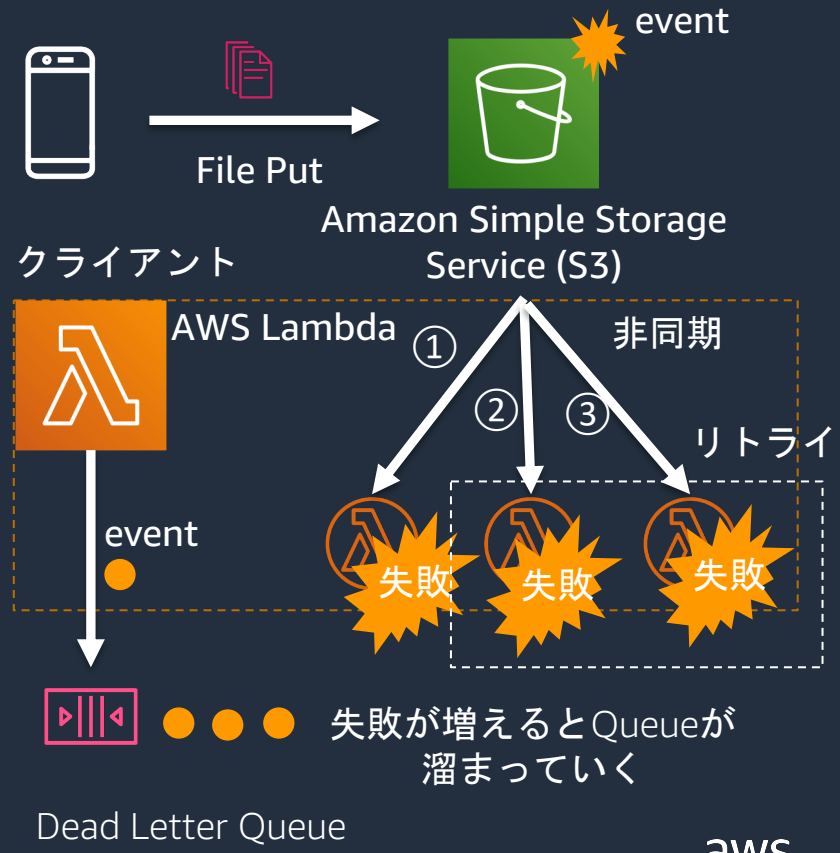
© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

aws

Amazon Simple Storage Service (S3)での非同期呼び出し例

- S3バケットにファイルをアップロード
- アップロードでイベント発生
- Lambda関数を呼び出すが、合計３回失敗(リトライ２回)

- DLQ (Dead Letter Queue)にイベントが送信される
- DLQは後ほど、バッチで処理するなど対応



© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

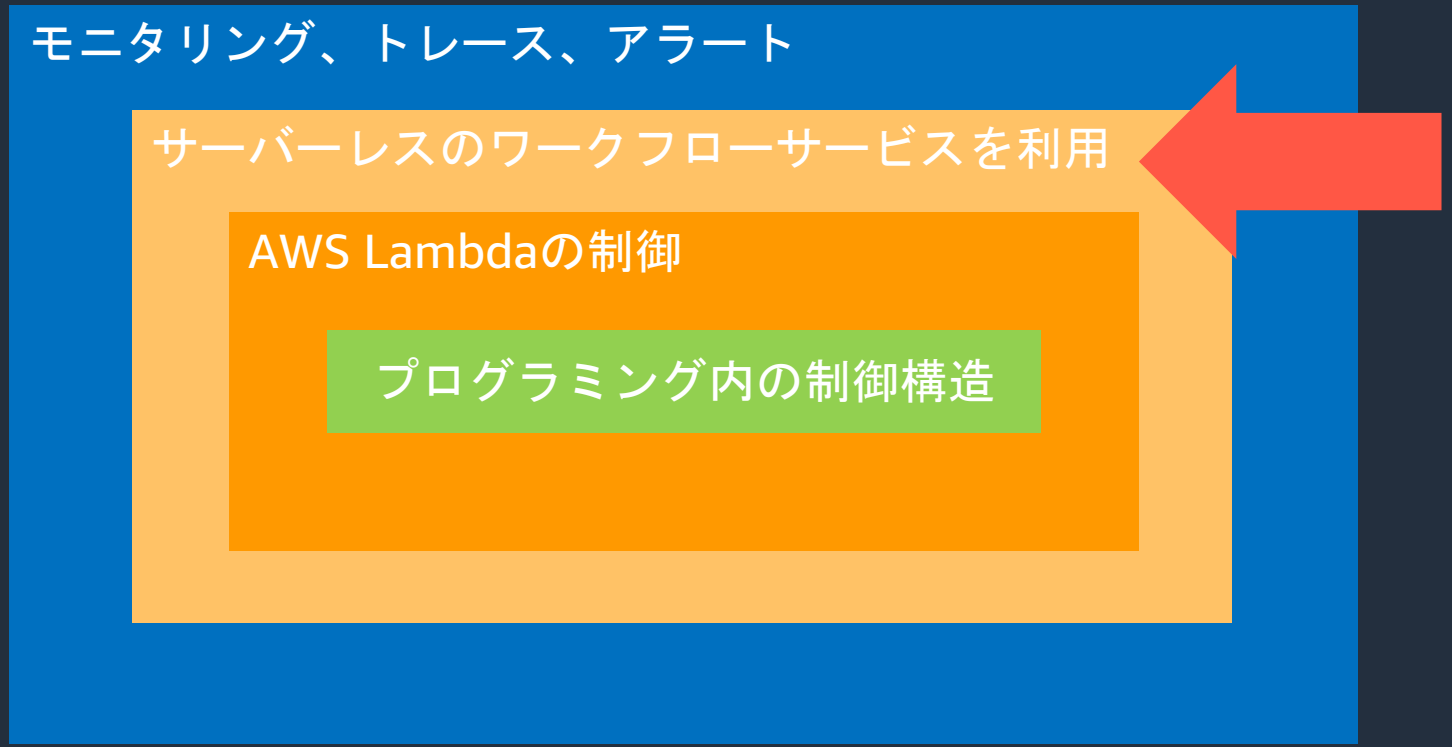
AWS Lambdaの制御

- 非同期実行のAWS Lambdaは関数失敗時に、最大2回リトライがされる。（初回含め3回）

- DLQ (Dead Letter Queue)は、そのリトライを尽くしてもエラーとなった場合に、未処理イベントが送信される先

© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

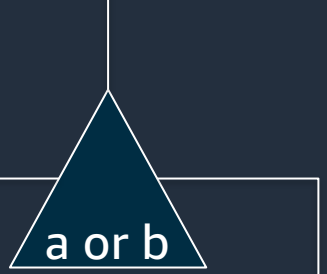
エラー制御についてのパターン



© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Lambda関数内でロジック実装

- 分岐
- 繰り返し
- 例外
- リトライ

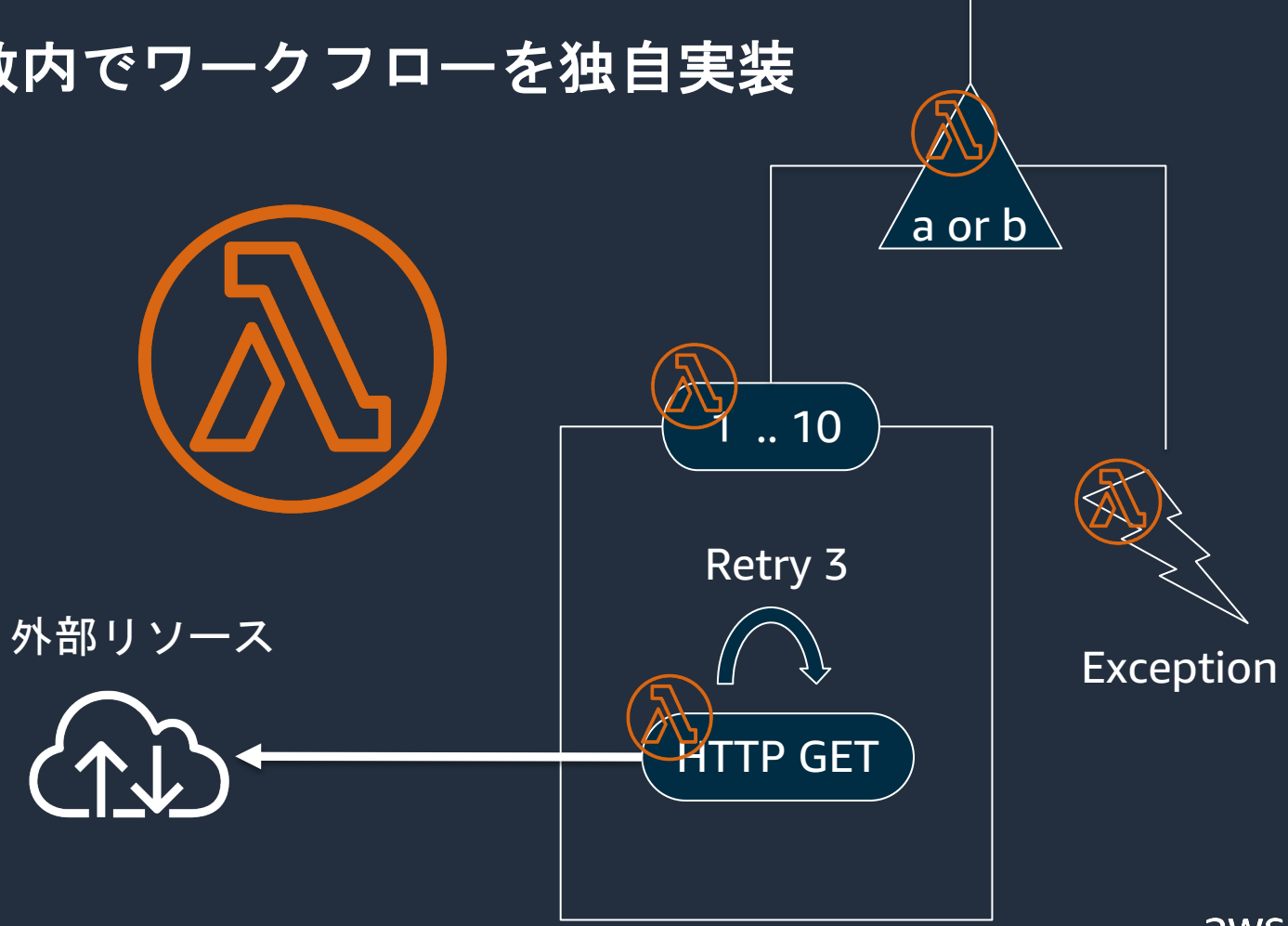


外部リソース

© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Lambda関数内でワークフローを独自実装

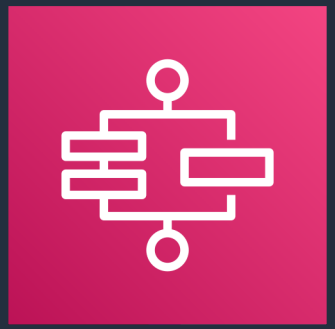
- 分岐
- 繰り返し
- 例外
- リトライ



© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

AWS Step Functionsでワークフローを実装

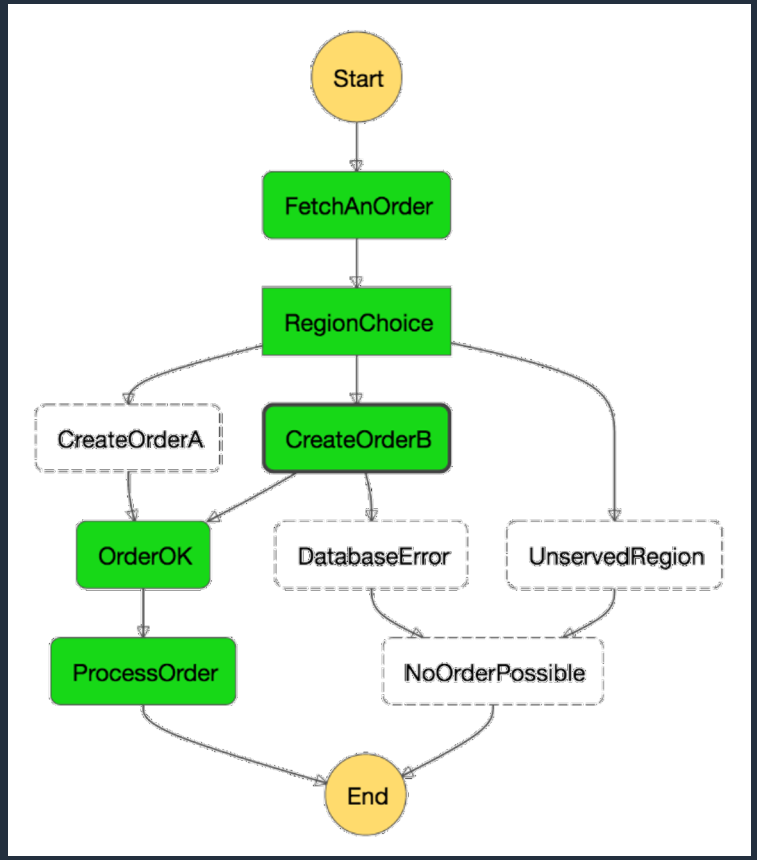
- 分岐
- 繰り返し
- 例外
- リトライ



© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

AWS Step Functions

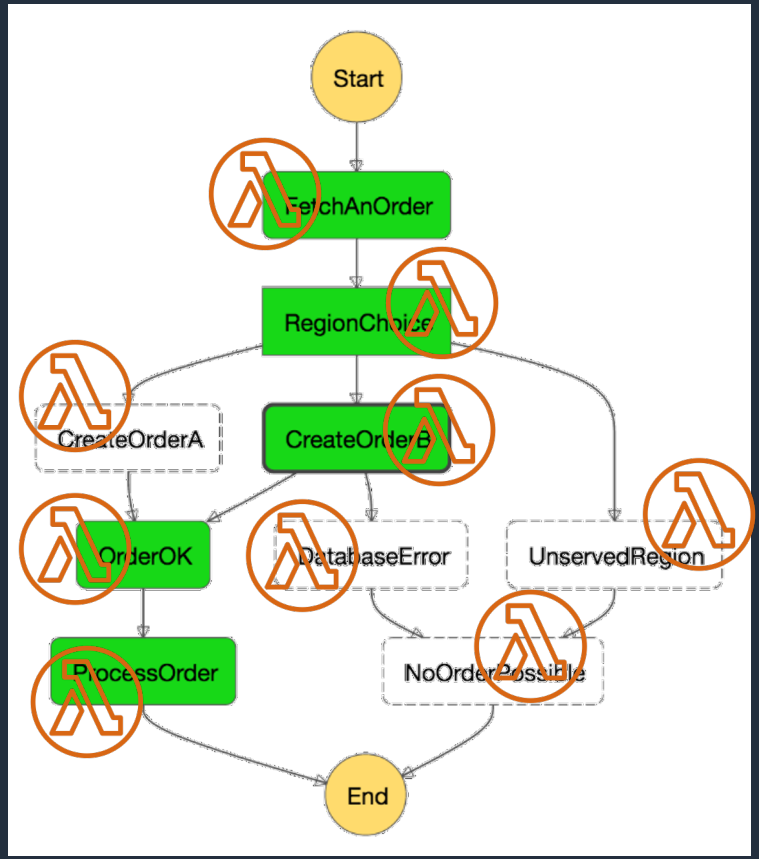
- 分散アプリケーション全体をオーケストレートできる
- Management Console から「**ワークフロー**」(右図) という形式で見やすく可視化できる
- 各ステップの実行履歴をログから追跡できる



© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

AWS Step Functions

- 分散アプリケーション全体をオーケストレートできる
- Management Console から「**ワークフロー**」(右図) という形式で見やすく可視化できる
- 各ステップの実行履歴をログから追跡できる



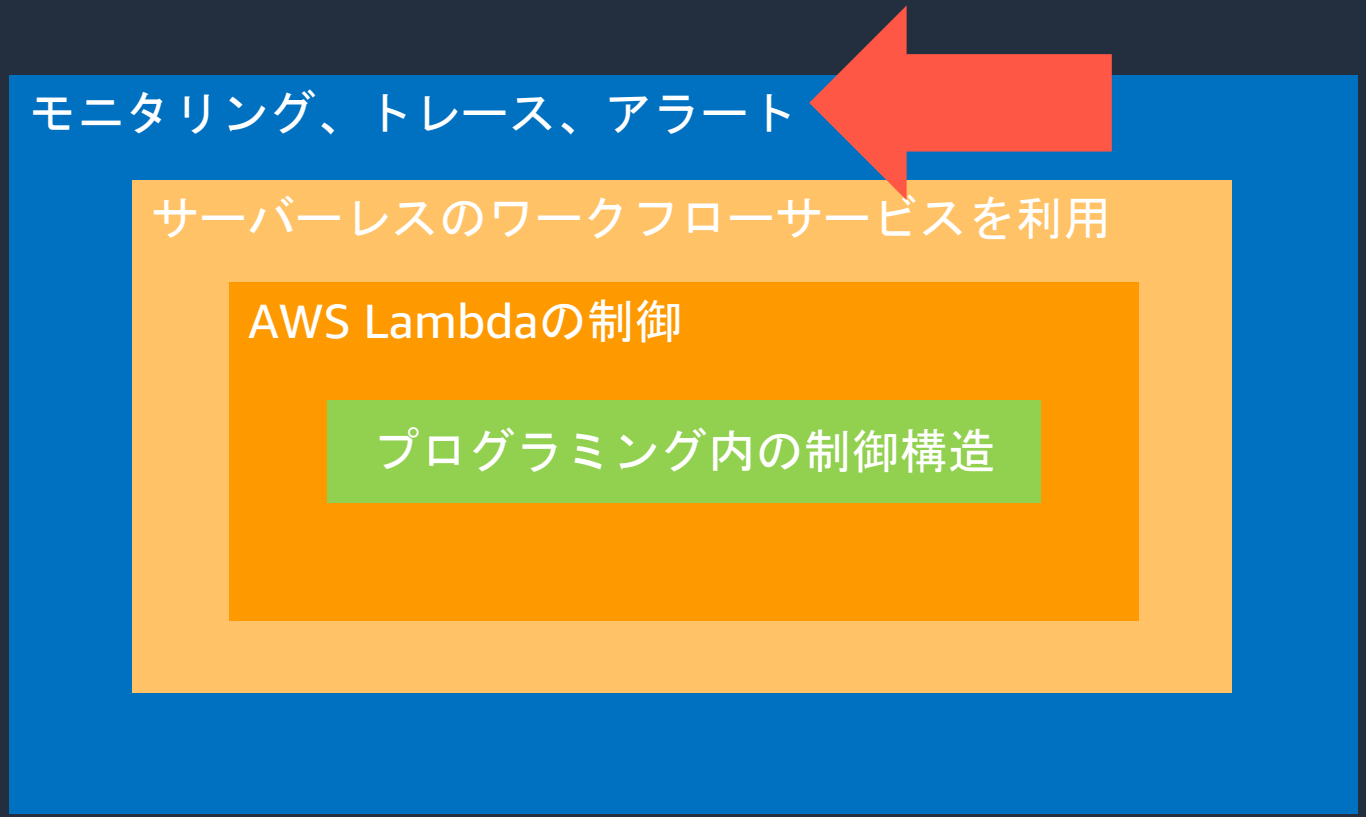
© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

ステートマシンから呼び出し可能な AWS のサービス

- AWS Lambda : Lambda 関数の実行
- Amazon Dynamo DB : 既存のアイテムの取得、新規アイテムの登録
- AWS Batch : ジョブの起動、ジョブ完了の待機
- Amazon ECS : ECS/Fargate タスクの実行
- Amazon SNS : SNS トピックへのメッセージ送信
- Amazon SQS : SQS キューへのメッセージ送信
- AWS Glue : Glue ジョブの実行
- Amazon SageMaker : トレーニングジョブ、トランスフォームジョブの起動

© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

エラー制御についてのパターン



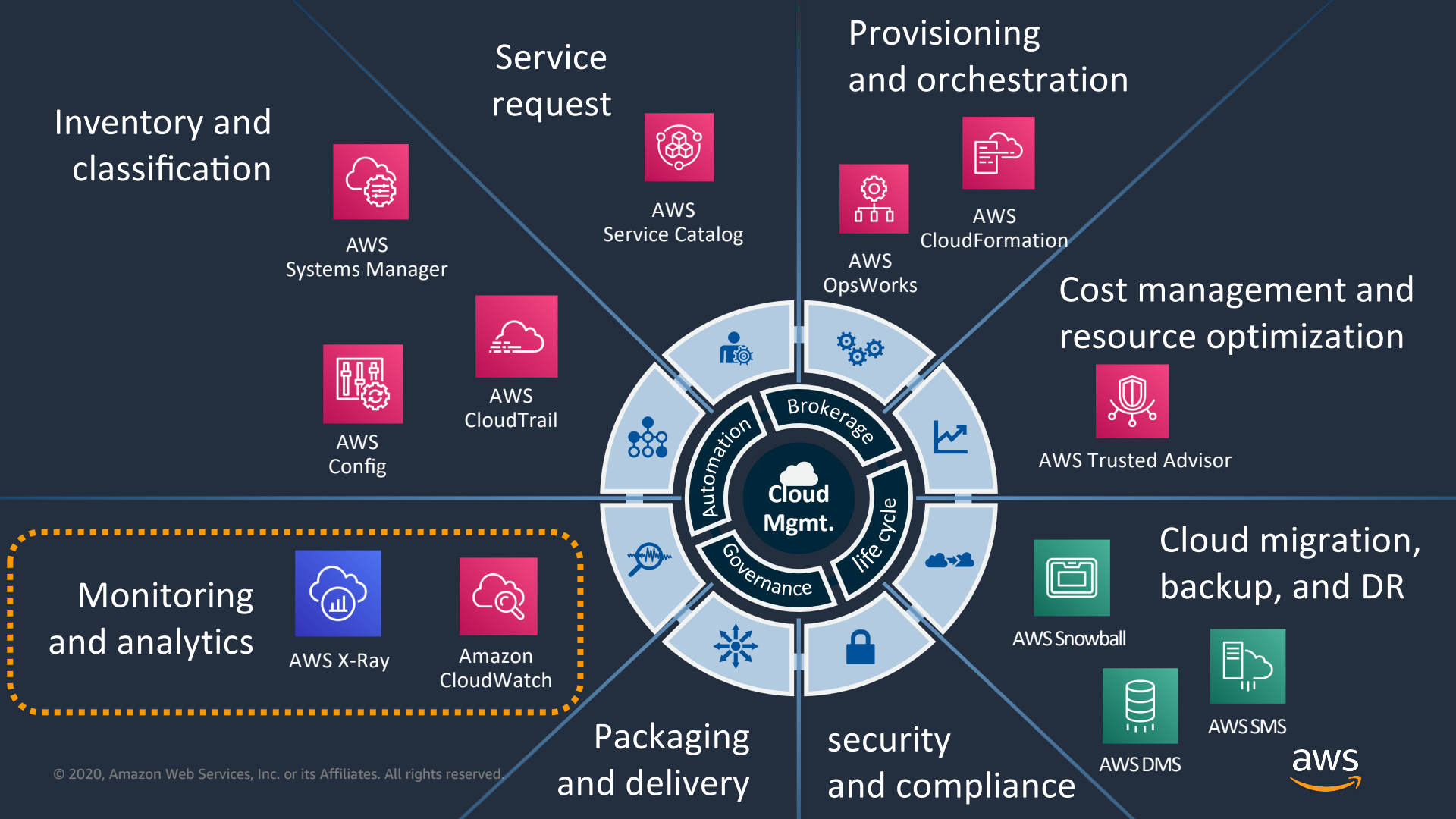
© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

本日のアジェンダ

- AWS Lambdaのおさらい
- AWS Lambdaでのプログラミング
- 仕組みを理解しよう
- エラー制御について
- AWS Lambdaをモニタリング

© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

モニタリングについて



© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

誰がモニタリングするのか

© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

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

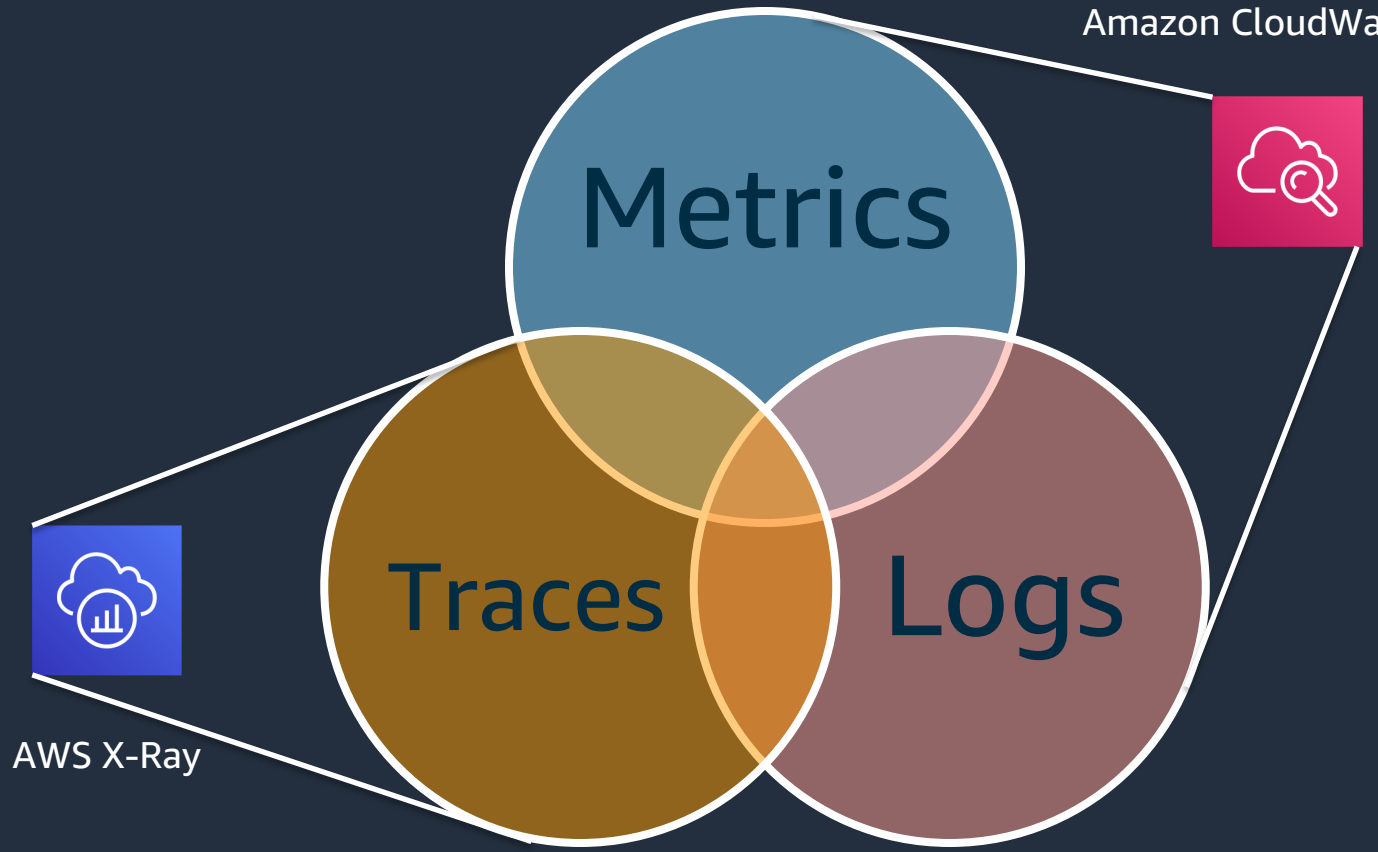
開発者も日々の運用に参加しなさい。
そうすればカスタマーのフィードバックが得られる。
これが、サービスの質を高めるのだ。

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

© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

aws

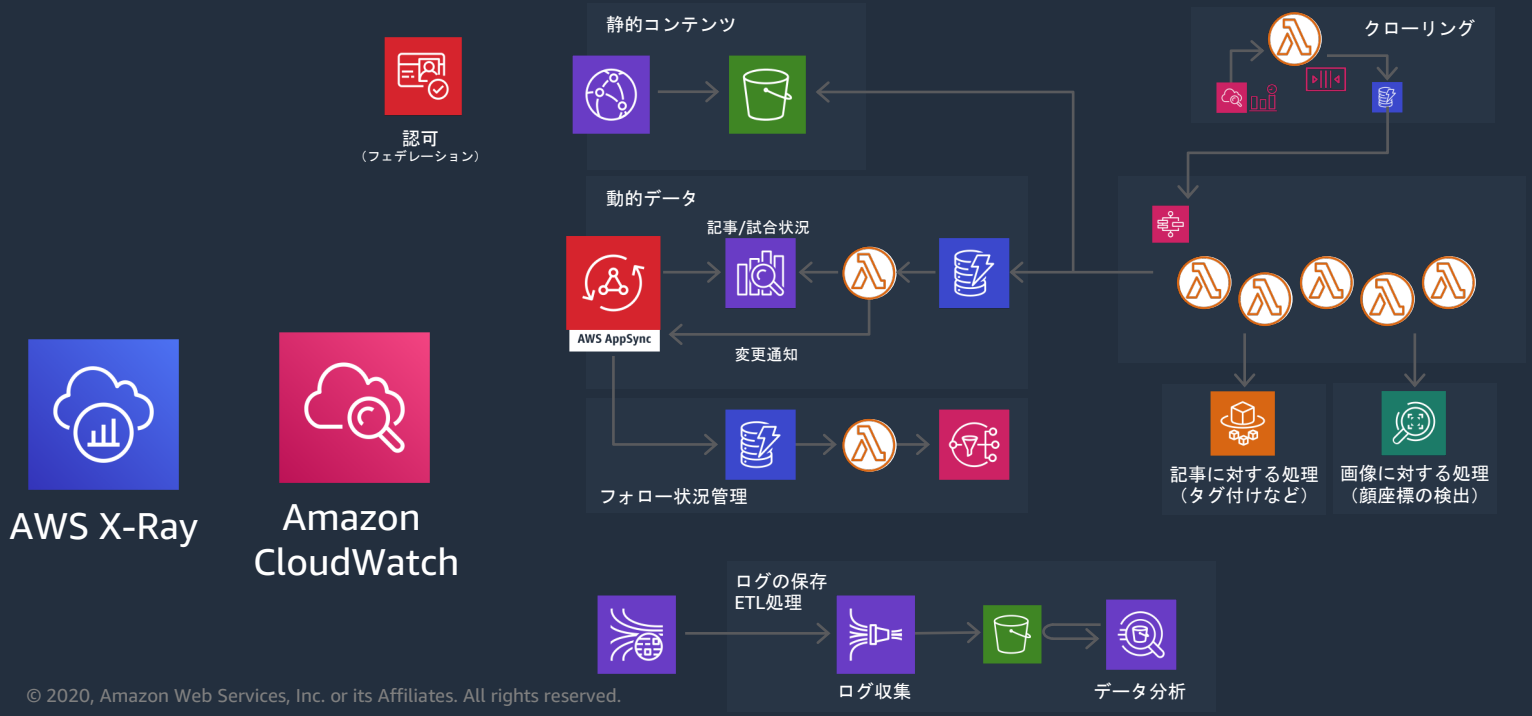
モニタリングサービス



© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

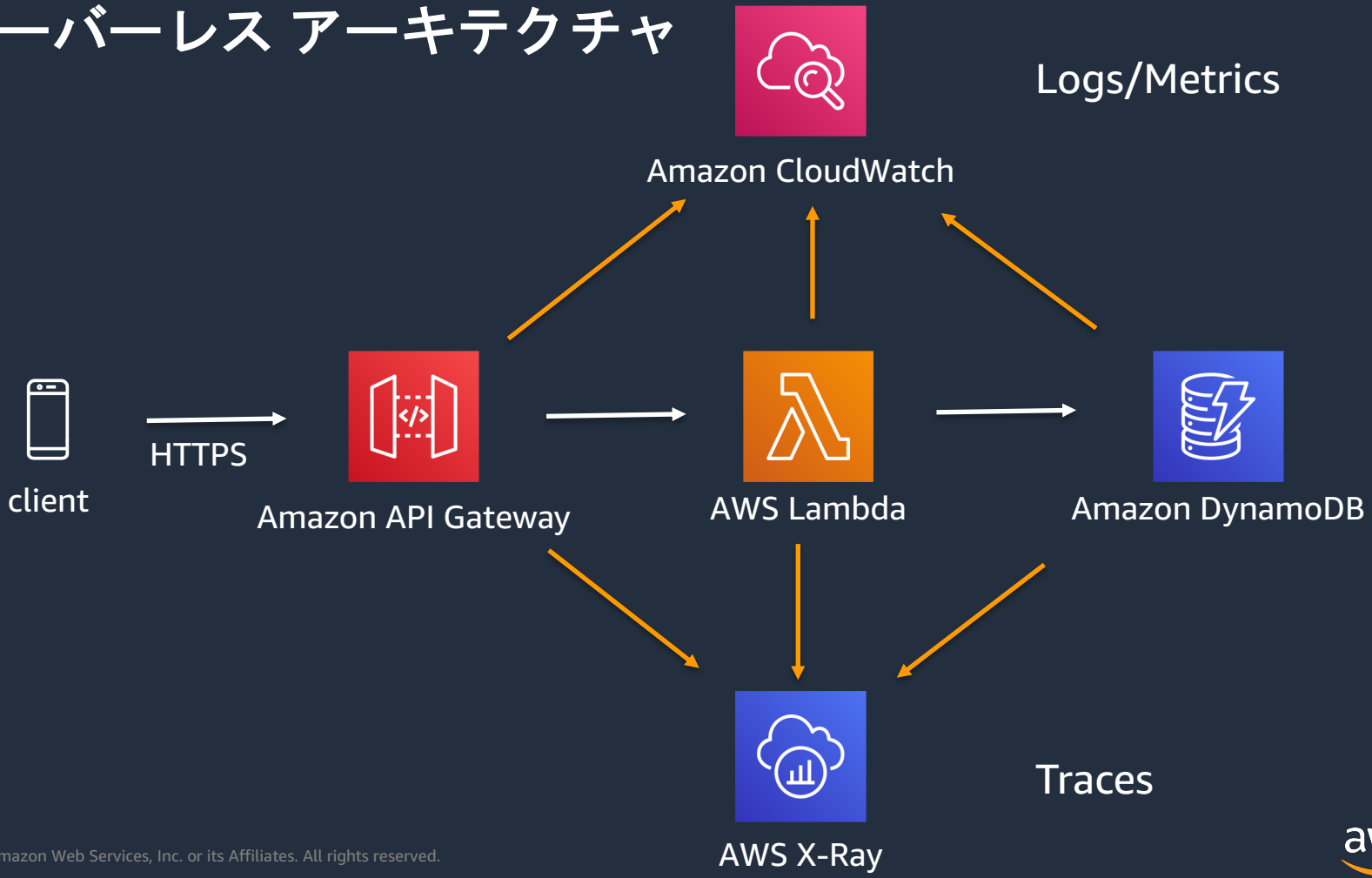
サーバーレスのモニタリング

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。



© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

サーバーレス アーキテクチャ



© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.


Metricsの理解

開発者にとってのモニタリングとは？

いま下記の質問にすぐに回答できますか？

- Lambdaの平均処理時間
- エラー回数
- deploy前後での実行時間差分

自分が作ったものがどう使われているのか/動いているのか？ 改善点が見えるようにしておくのが重要。

また、運用を人に託すのであれば何を見るのかを提示する責任を持つ。

© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

AWS Lambda Metrics

© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

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。

© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

AWS LambdaのManagement Consoleの再確認

実行結果: 成功 (ログ)

▼ 詳細

下のセクションに、関数の実行から返された結果が表示されます。

```
{
  "statusCode": 200,
  "body": "\"Hello from Lambda!\""
}
```

関数の実行結果

概要

コード SHA-256	リクエスト ID
CzZO0a9QX4MGEg46ZPAEHwsJDDpQI3ISw3/gYe4E7G0=	d1527f2e-75d1-404c-856f-b42bbda5d90b
所要時間	課金期間
5.65 ms	100 ms
設定済みリソース	使用中の最大メモリ
128 MB	56 MB Init Duration: 106.60 ms

実行情報の概要

ログ出力

下のセクションに、コード内のログ記録呼び出しが表示されます。これらはそれぞれ、CloudWatch ロググループ内でこの Lambda 関数に対応する単一行です。CloudWatch ロググループを表示するには、ここをクリックし、 をクリックしてください。

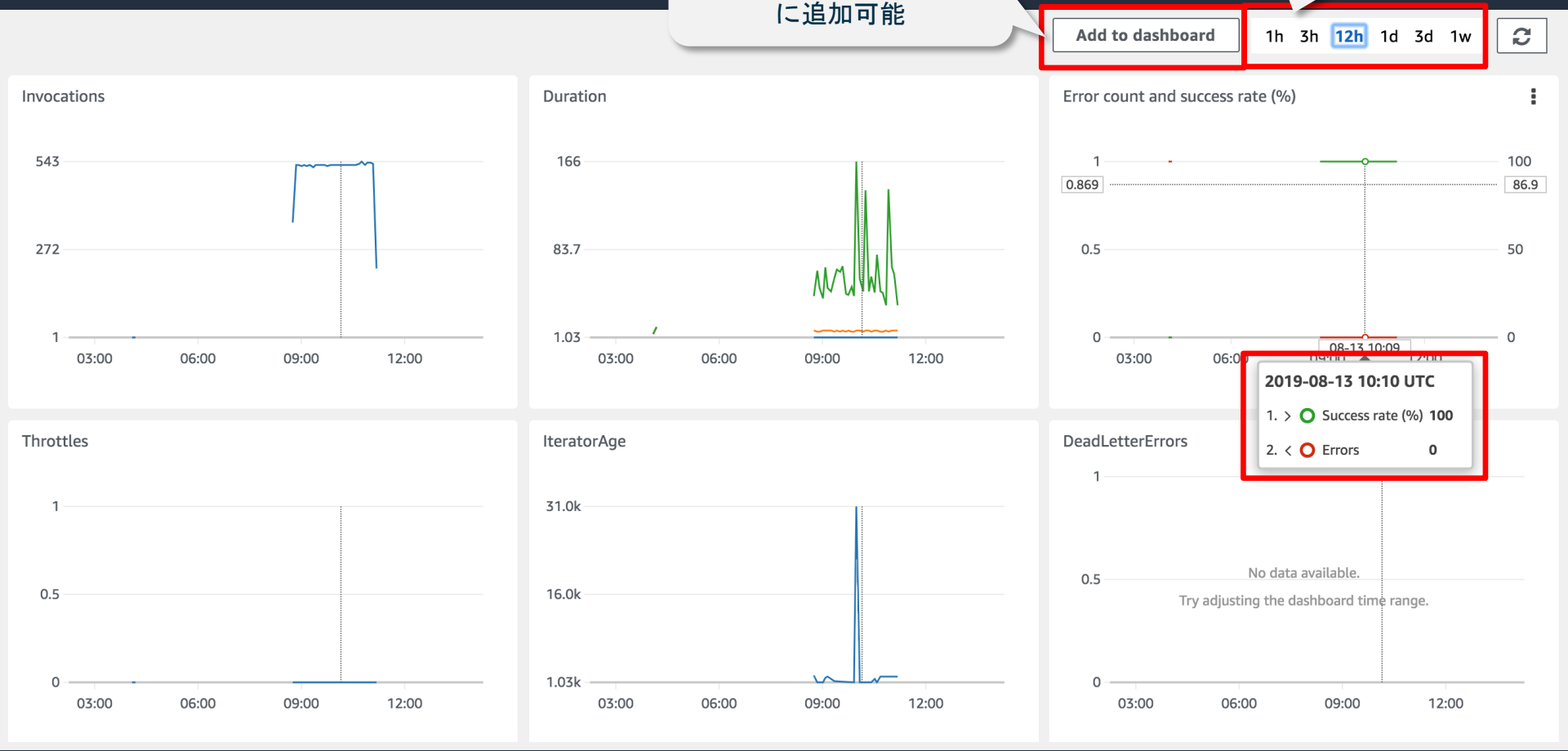
```
START RequestId: d1527f2e-75d1-404c-856f-b42bbda5d90b Version: $LATEST
[INFO]  2019-09-05T03:09:26.463Z        d1527f2e-75d1-404c-856f-b42bbda5d90b    info

[WARNING]       2019-09-05T03:09:26.463Z        d1527f2e-75d1-404c-856f-b42bbda5d90b    warn

[ERROR] 2019-09-05T03:09:26.463Z        d1527f2e-75d1-404c-856f-b42bbda5d90b    error

END RequestId: d1527f2e-75d1-404c-856f-b42bbda5d90b
REPORT RequestId: d1527f2e-75d1-404c-856f-b42bbda5d90b  Duration: 5.65 ms       Billed Duration: 100 ms Memory Size: 128 MB
Max Memory Used: 56 MB  Init Duration: 106.60 ms
```

標準出力

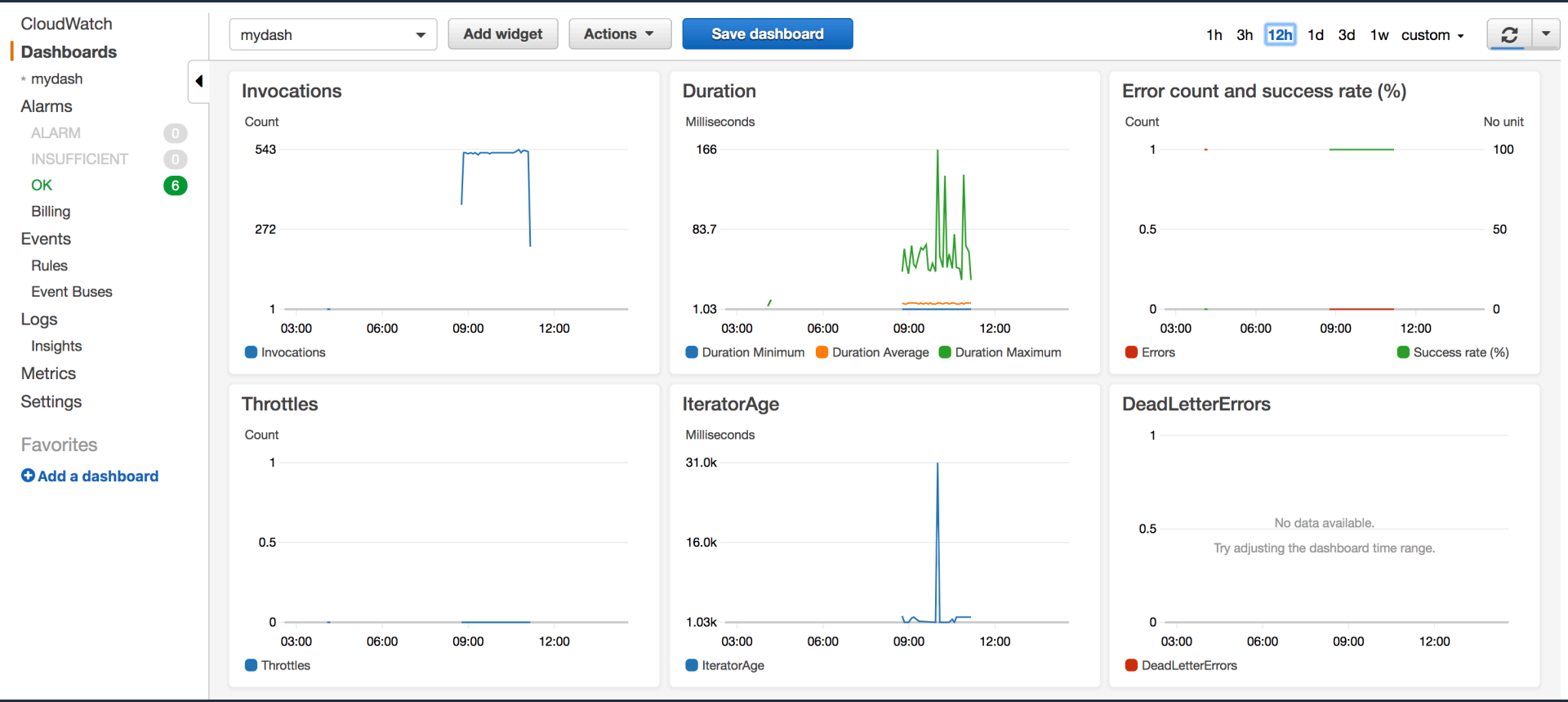
AWS Lambda の Metrics

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。

© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

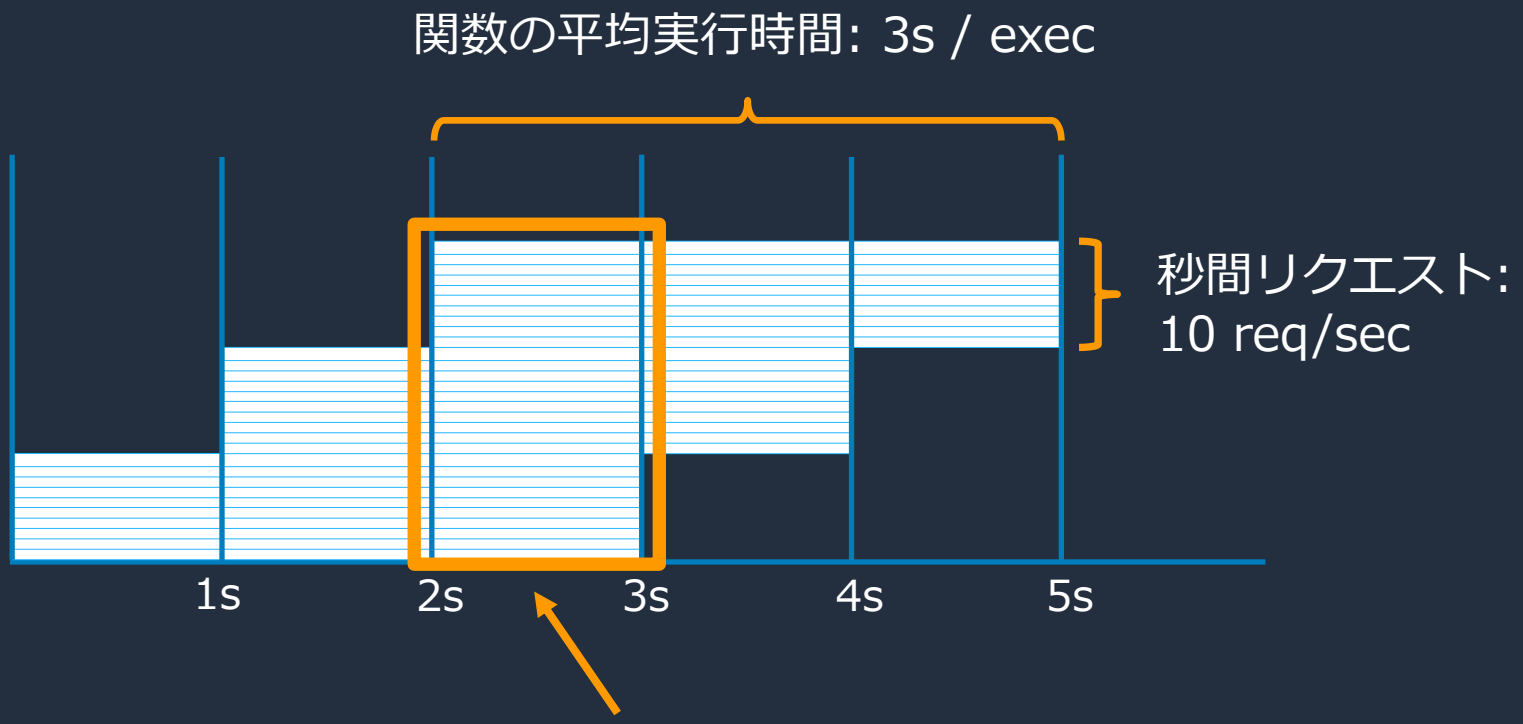
AWS Lambda の Metrics

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。

© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

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

© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

aws

AWS Lambdaの制限

AWS ドキュメント » AWS Lambda » 開発者ガイド » AWS Lambda の使用開始 » AWS Lambda の制限

AWS Lambda の制限

AWS Lambda では、関数の実行と保存に使用できるコンピューティングおよびストレージリソースの量が制限されます。以下の制限は、リージョンごとに適用され、引き上げることができます。引き上げをリクエストするには、サポートセンターコンソールを使用してください。

リソース	デフォルトの制限
同時実行数	1,000
関数とレイヤーストレージ	75 GB

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

© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Throttlingの対策について

Throttles

Throttlingの対策としては

- Lambda関数をチューニングして平均実行時間を下げる
- 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
- アカウントの同時実行数の制限緩和を申請する

© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

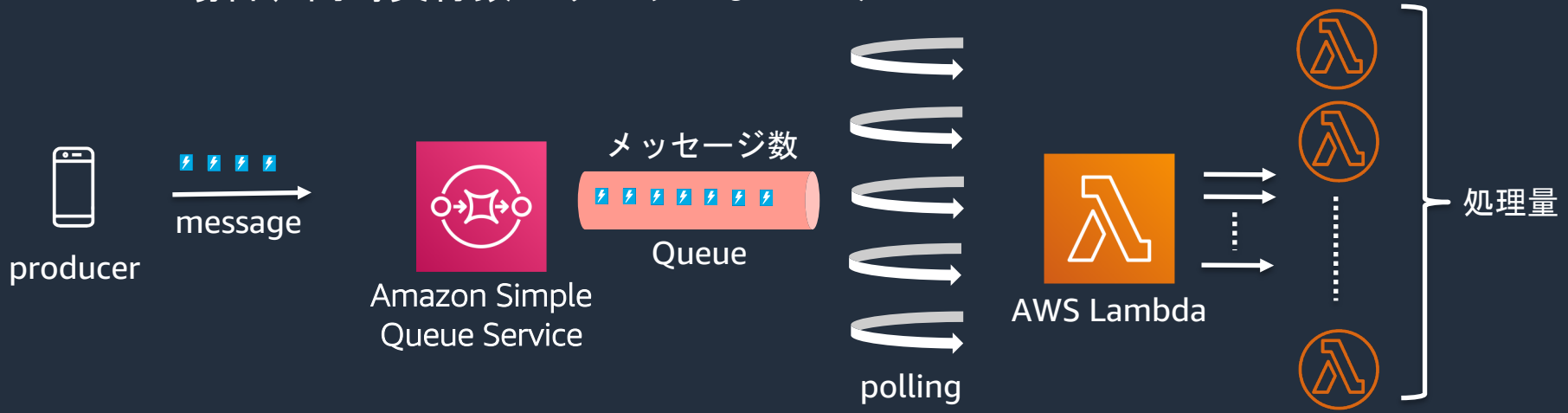
aws

SQS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



© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

aws

SQS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

同時実行数

予約されていないアカウントの同時実行 **880**

予約されていないアカウントの同時実行の使用

同時実行の予約

120

Throttles

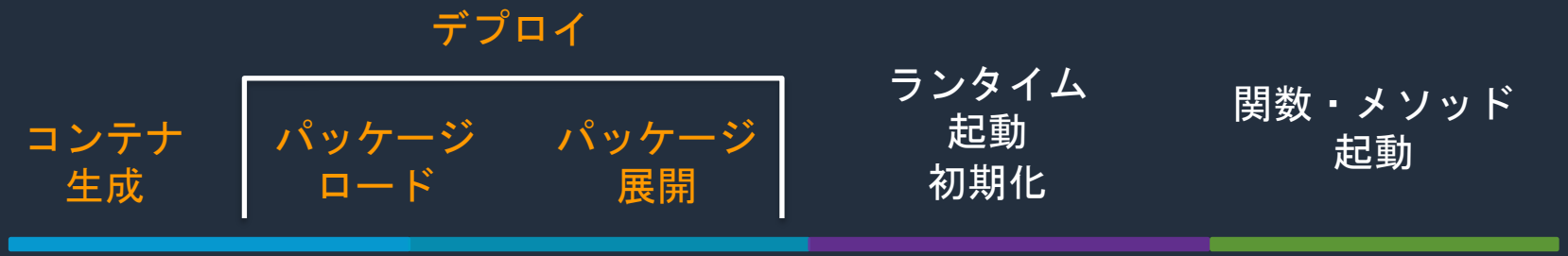
AWS Lambda の Metrics

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。

© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

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

© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

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

© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

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

© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

aws

Lambda Function Lifecycle

Duration

© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

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

© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

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

© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

aws

Amazon API GatewayのMetrics

© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

aws

Amazon API Gateway の Log・Traceの有効化

© 2020, Amazon Web Se

Amazon API Gateway の Log・Traceの有効化

© 2020, Amazon Web Se

Amazon API Gateway の Log・Traceの有効化

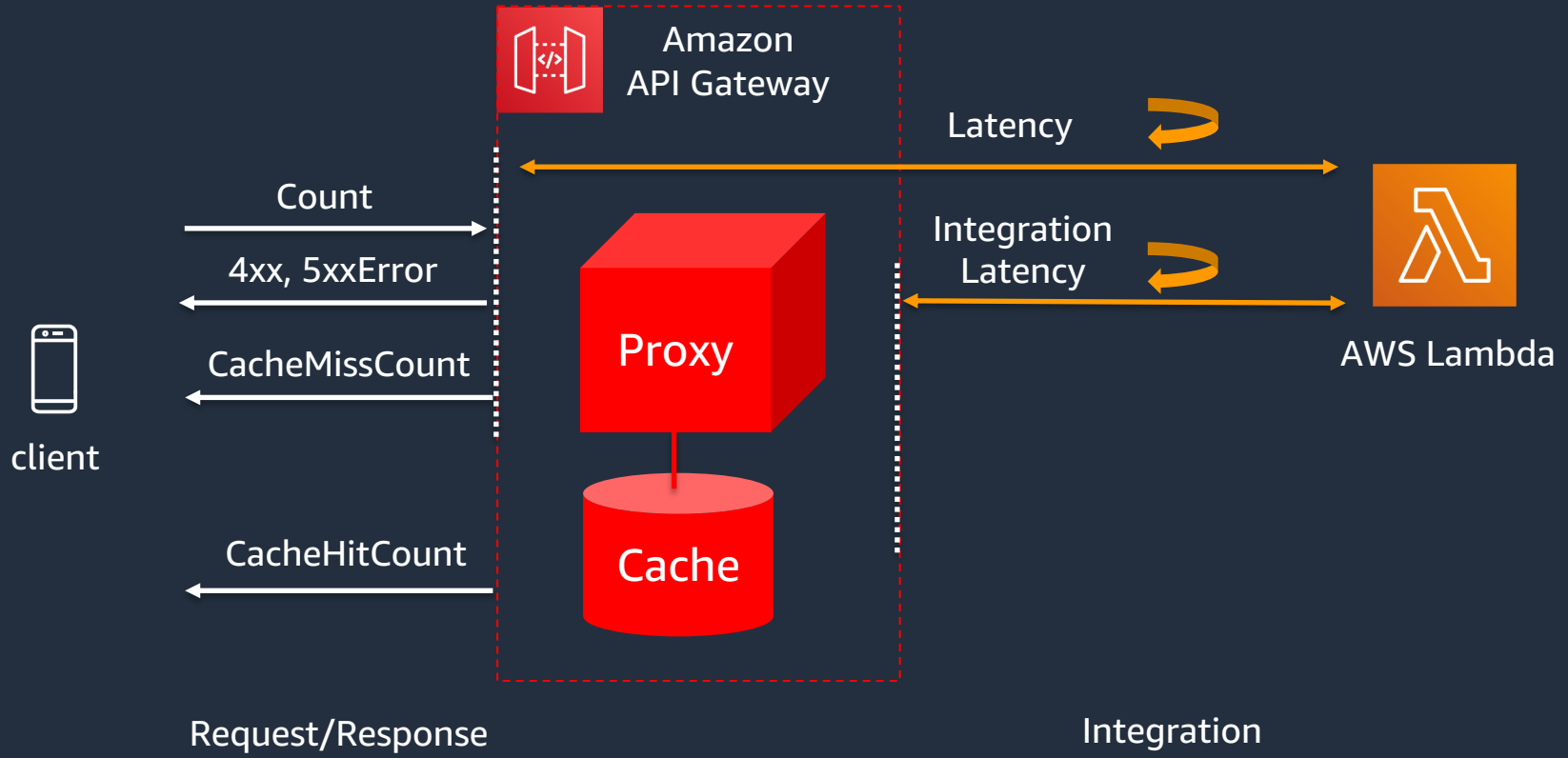
© 2020, Amazon Web Se

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

© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Amazon API Gateway の Metrics

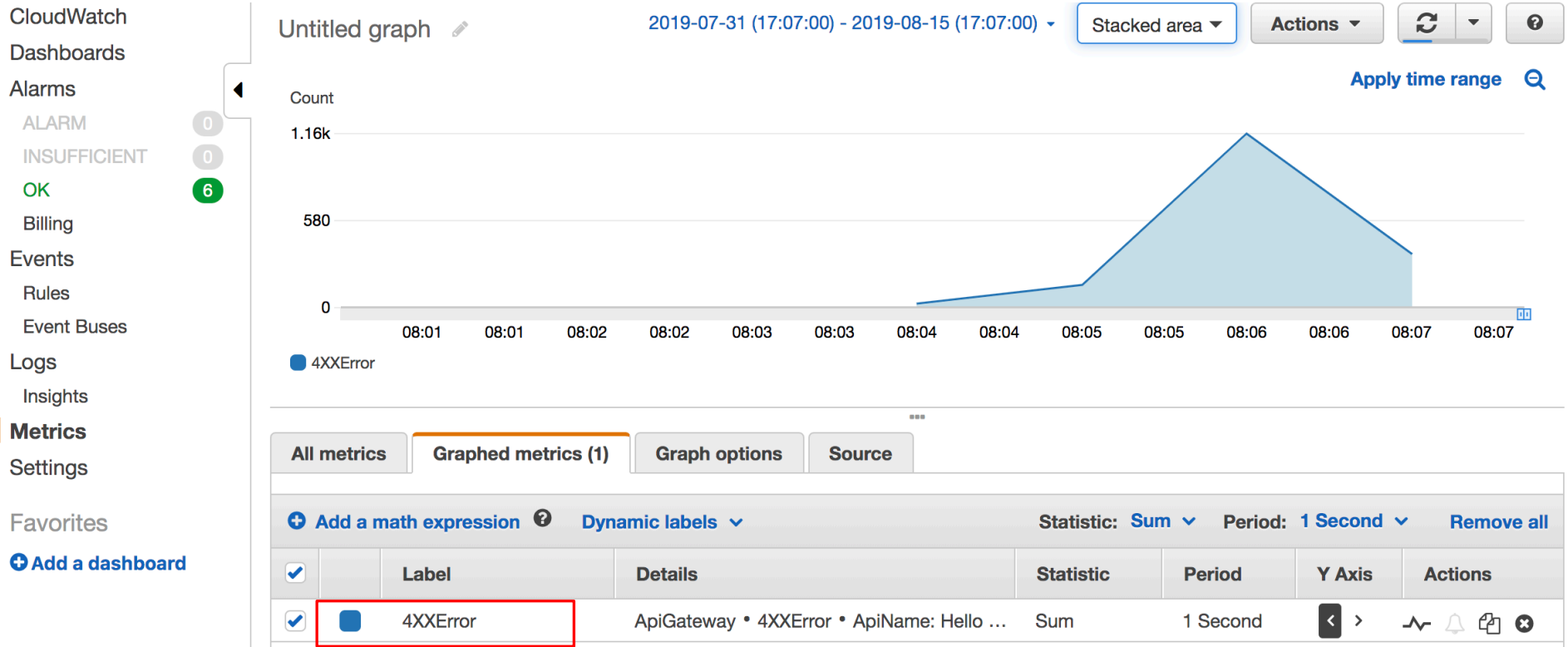


© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

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)




Log運用と可視化

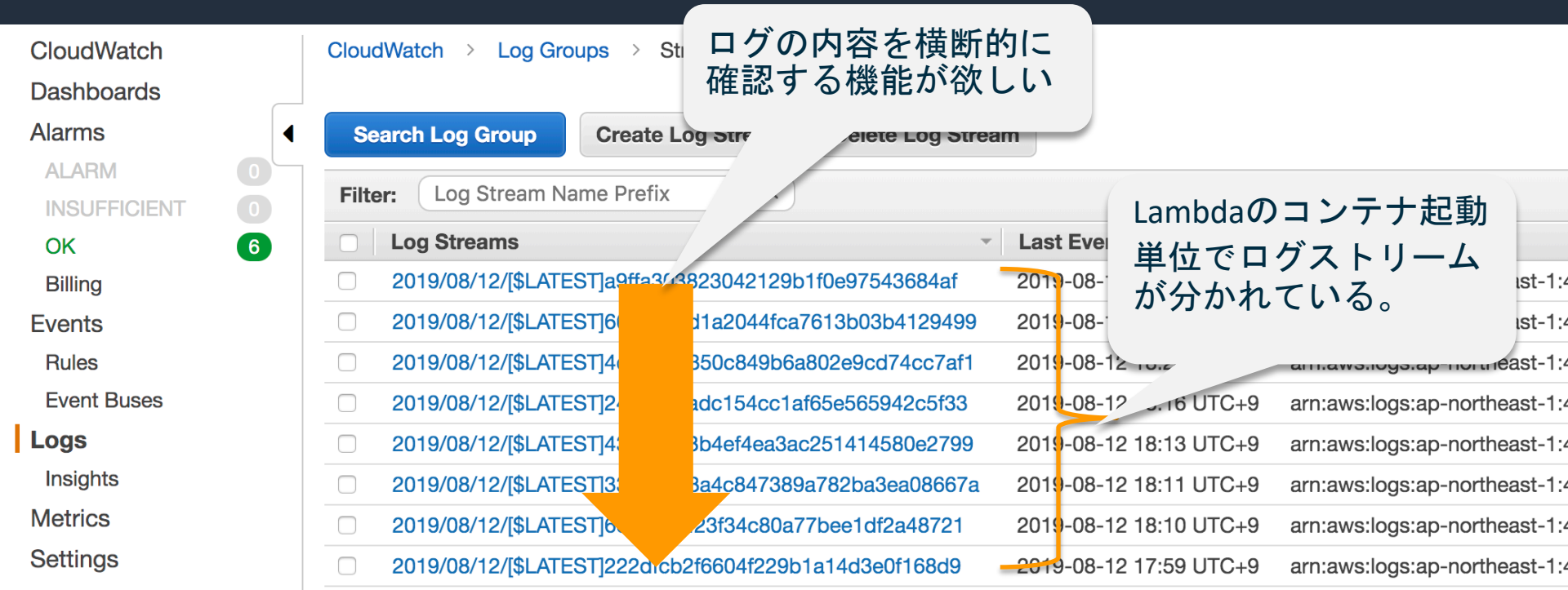
Amazon CloudWatch でのログ管理

© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

aws

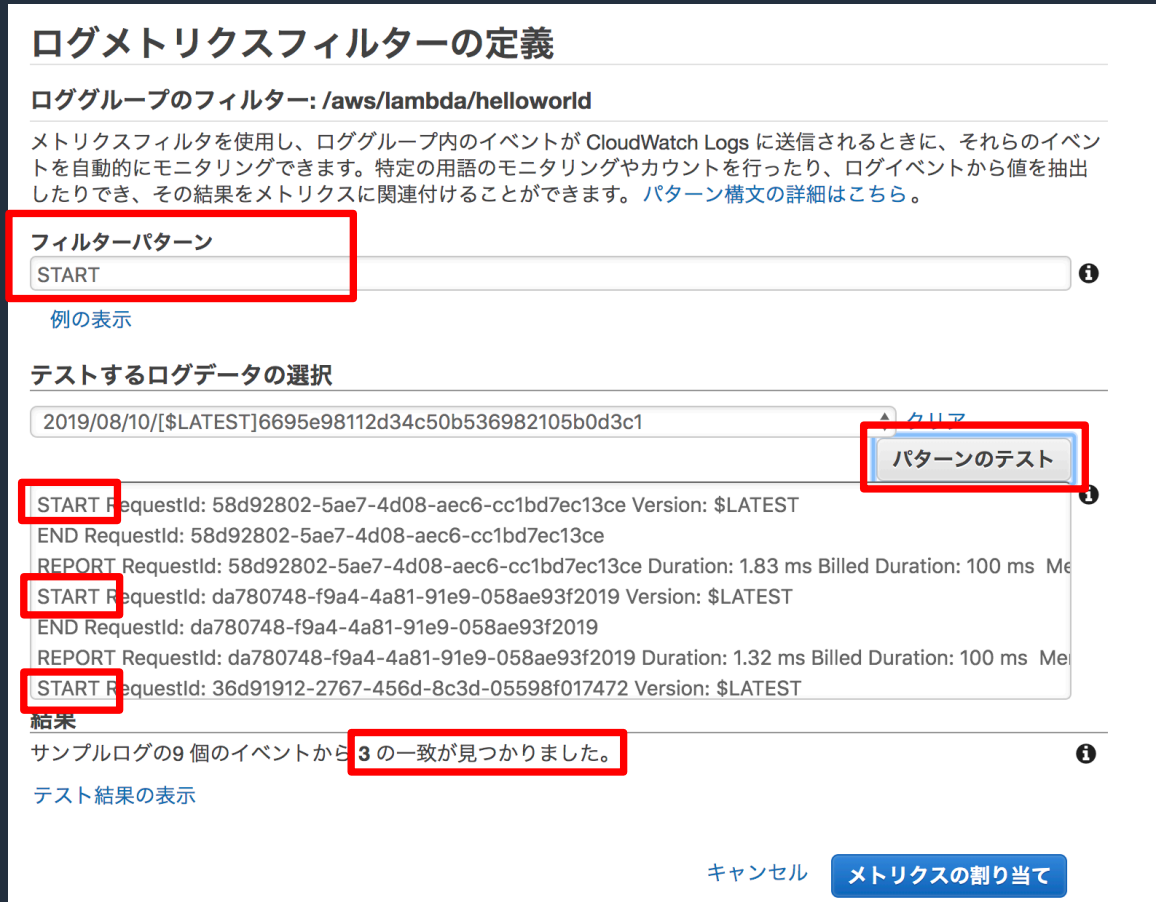
Amazon CloudWatch Logsの使い方を知る

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

- 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メトリックスの割り当てもできる。
- 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



© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

CloudWatch Logs filter pattern

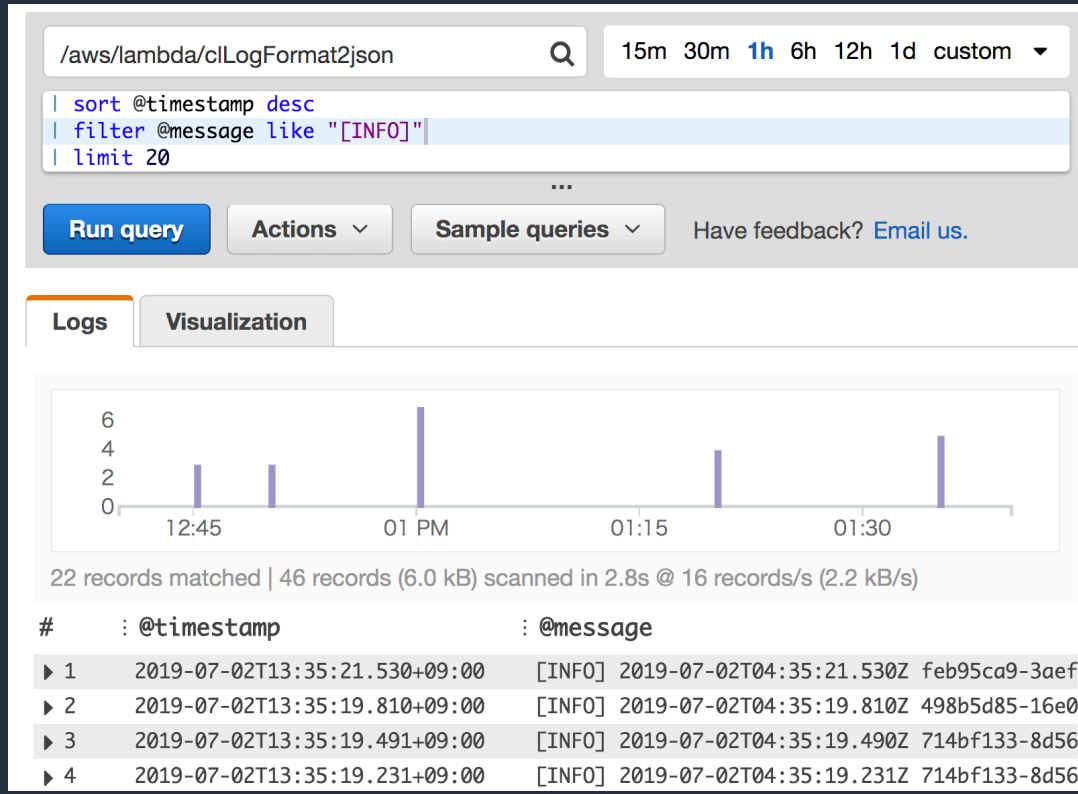
filter patternの仕様

- 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

© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

CloudWatch Logs Insightsの活用

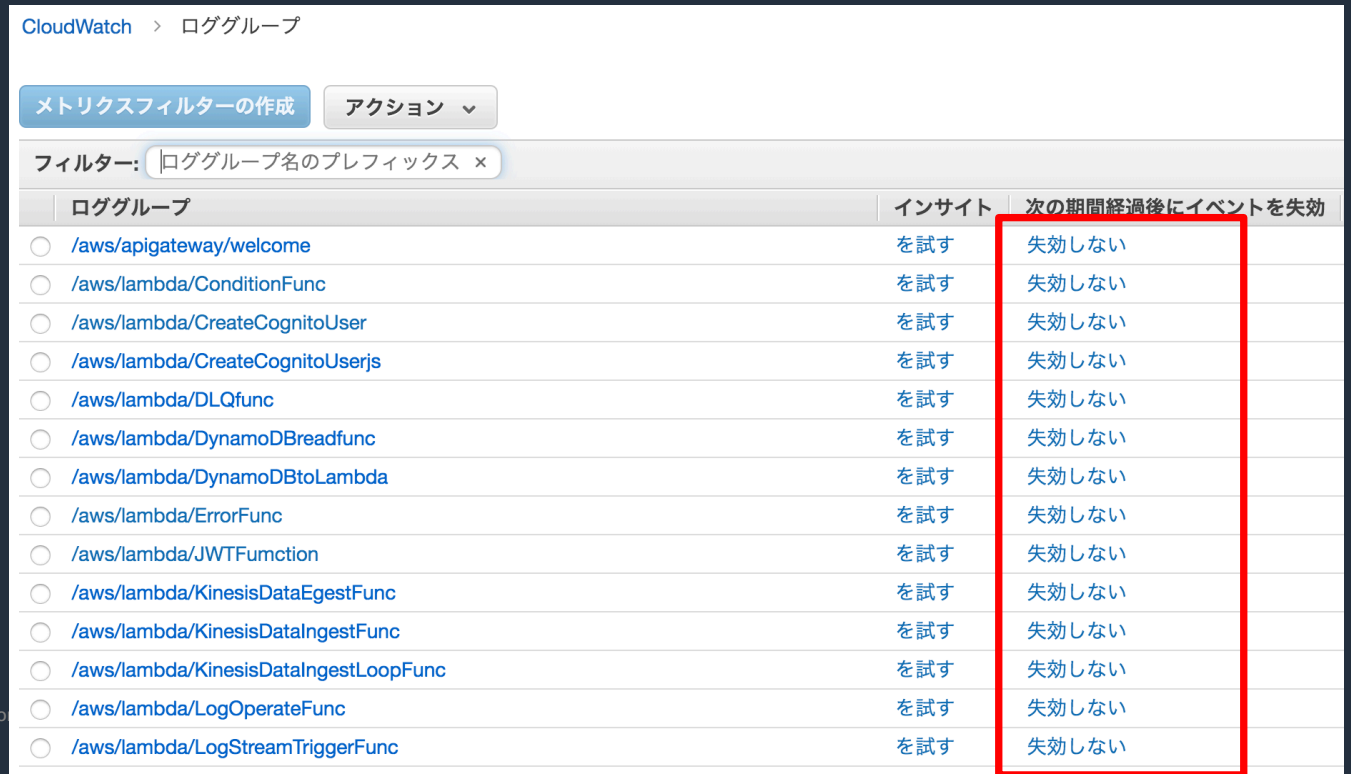
```
fields @timestamp, @message
| sort @timestamp desc
| filter @message like "[INFO]"
| limit 20
```



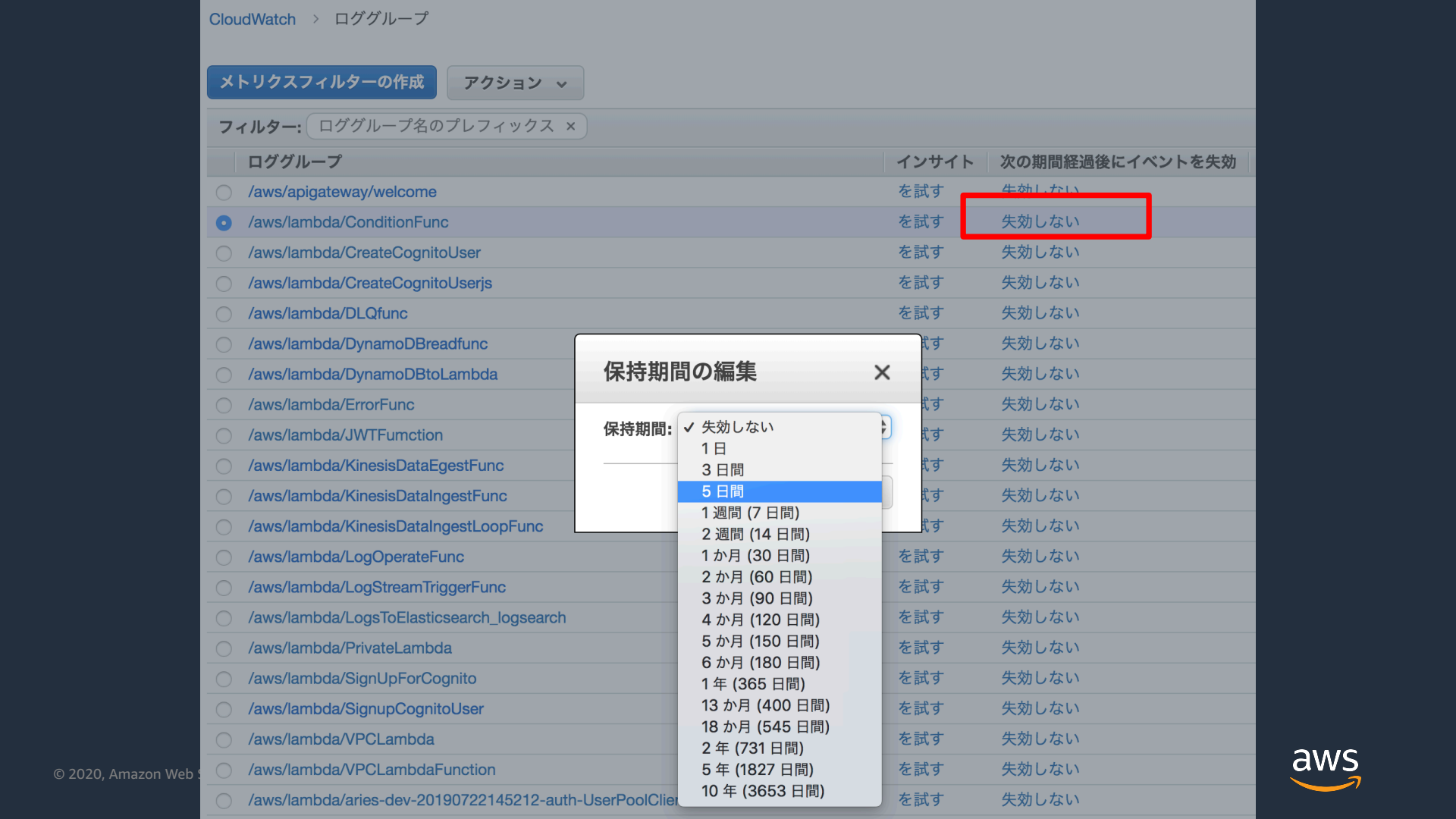
© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

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

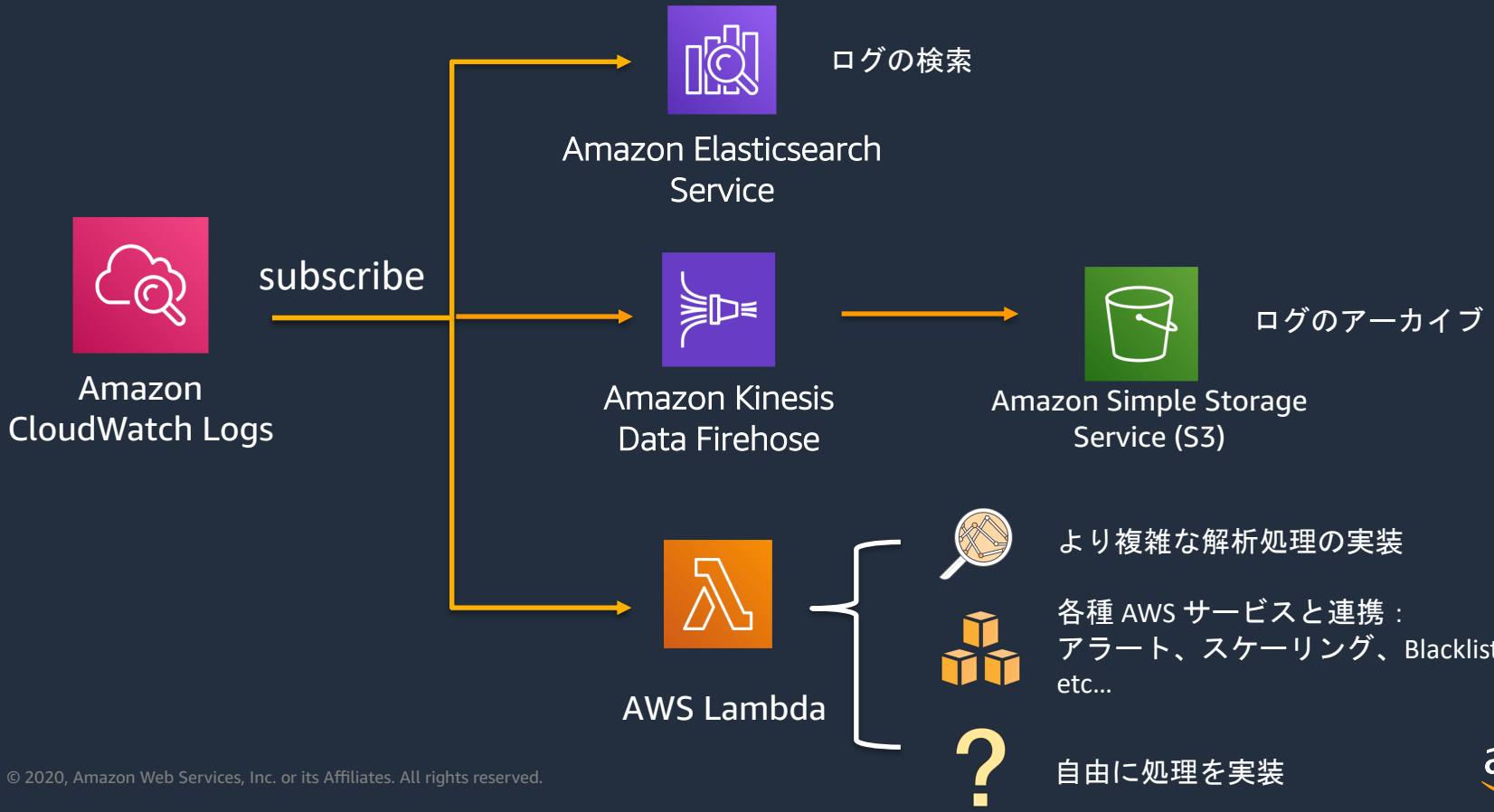


© 2020, Amazo



© 2020, Amazon Web

他サービスに連携してログの活用



© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.


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

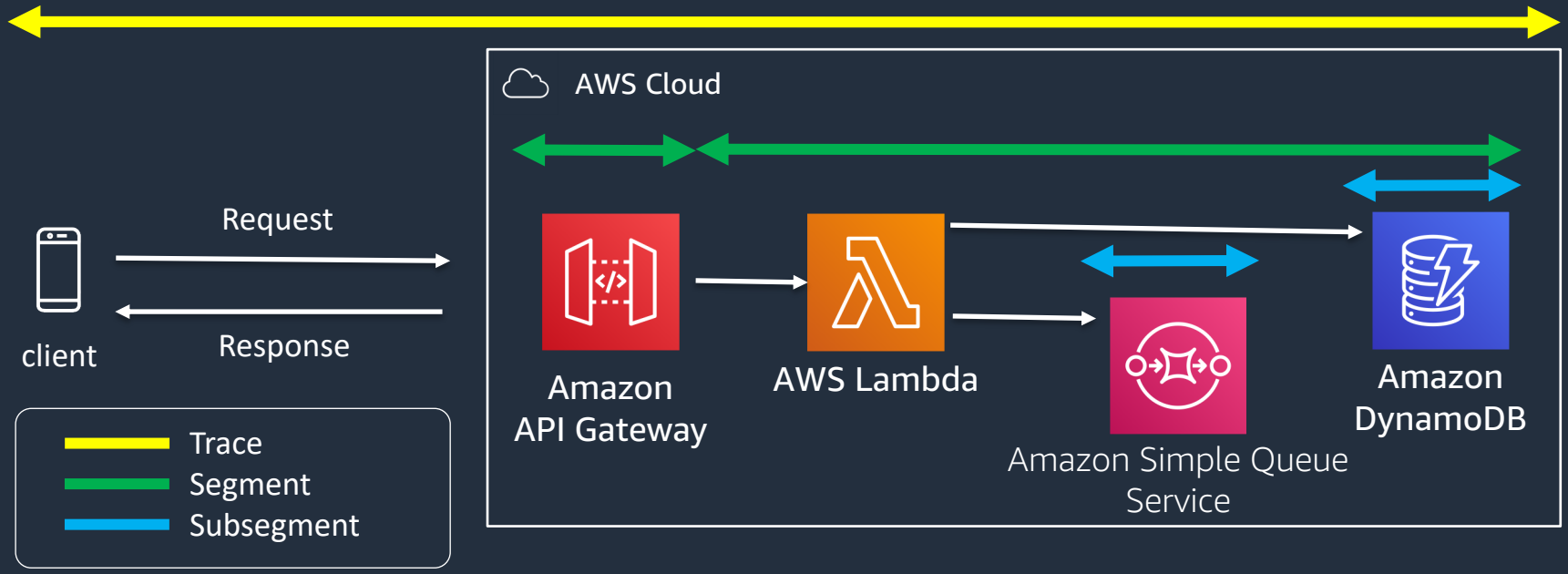
サービスやリソースの関係をリアルタイムで
表示し、レイテンシ増加やパフォーマンス低下
などのボトルネックを特定可能

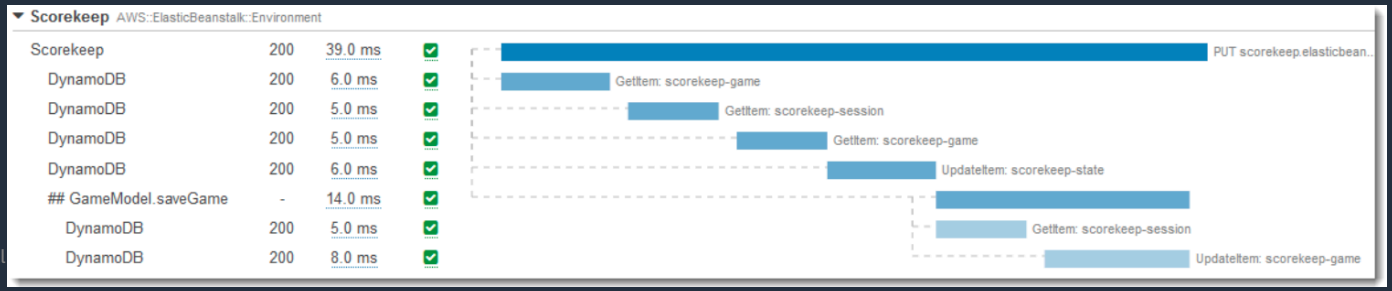
さまざまなアプリケーション向けの設計

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

© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

AWS X-Ray コンポーネント



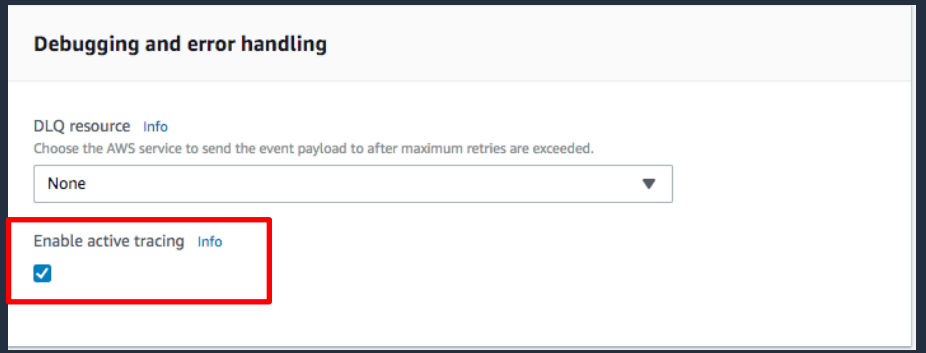


© 2020, Amazon Web Services, Inc. or its Affiliates. All

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

© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

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
```

© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

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

© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

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

© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

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

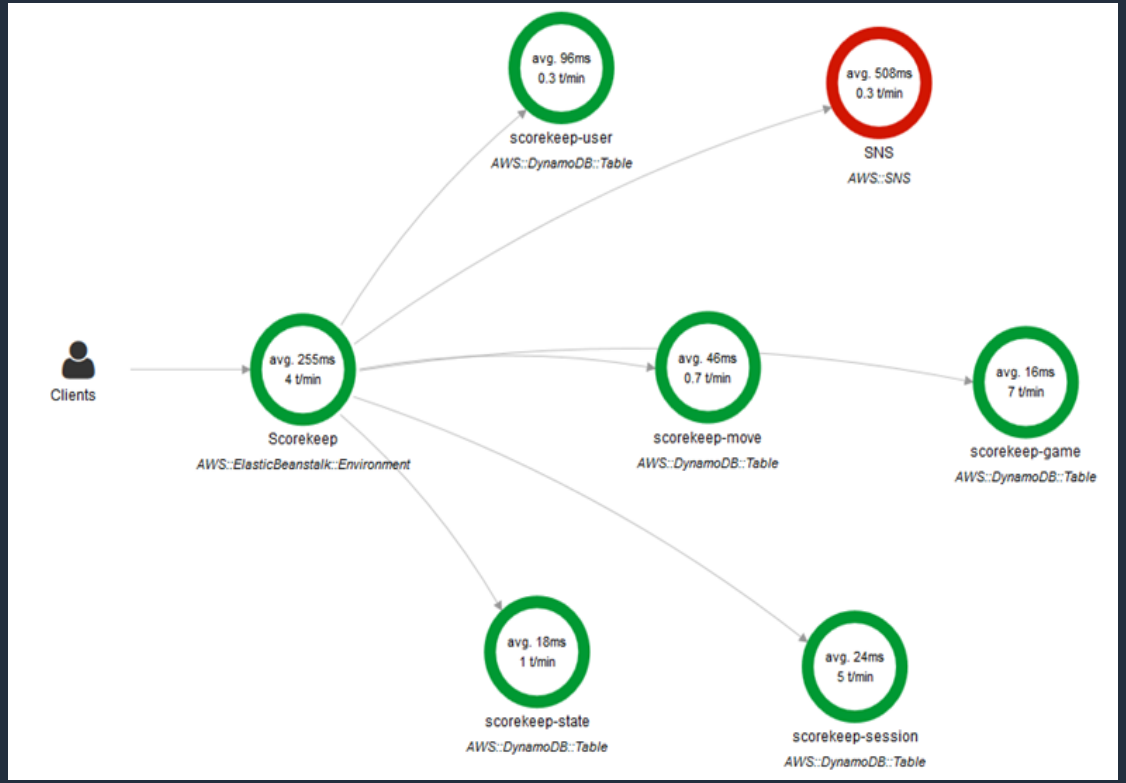
© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

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

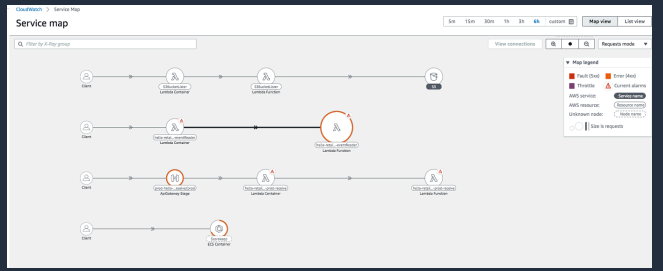


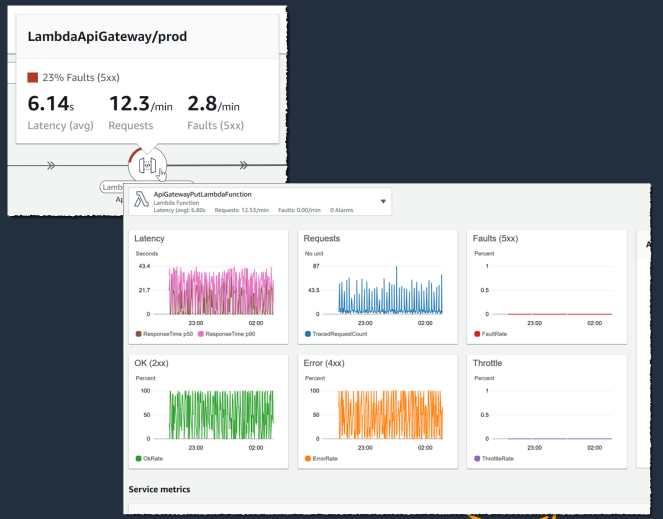
© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Amazon CloudWatch ServiceLensを発表

NEW

- アプリケーションの健全性、パフォーマンスなどを視覚化し分析を容易にするための機能
- CloudWatchメトリクスとログ、X-Rayからのトレース情報を結びつけてアプリとその依存関係をビジュアライズする
- メトリクスデータとトレースデータを可視化することでシステム全体を俯瞰し問題箇所を特定、原因を掘り下げることが容易に
- 東京リージョンを含むX-Rayが利用可能なリージョンで利用可能。ServiceLens自体は無料だがX-Rayのトレース費用が発生する

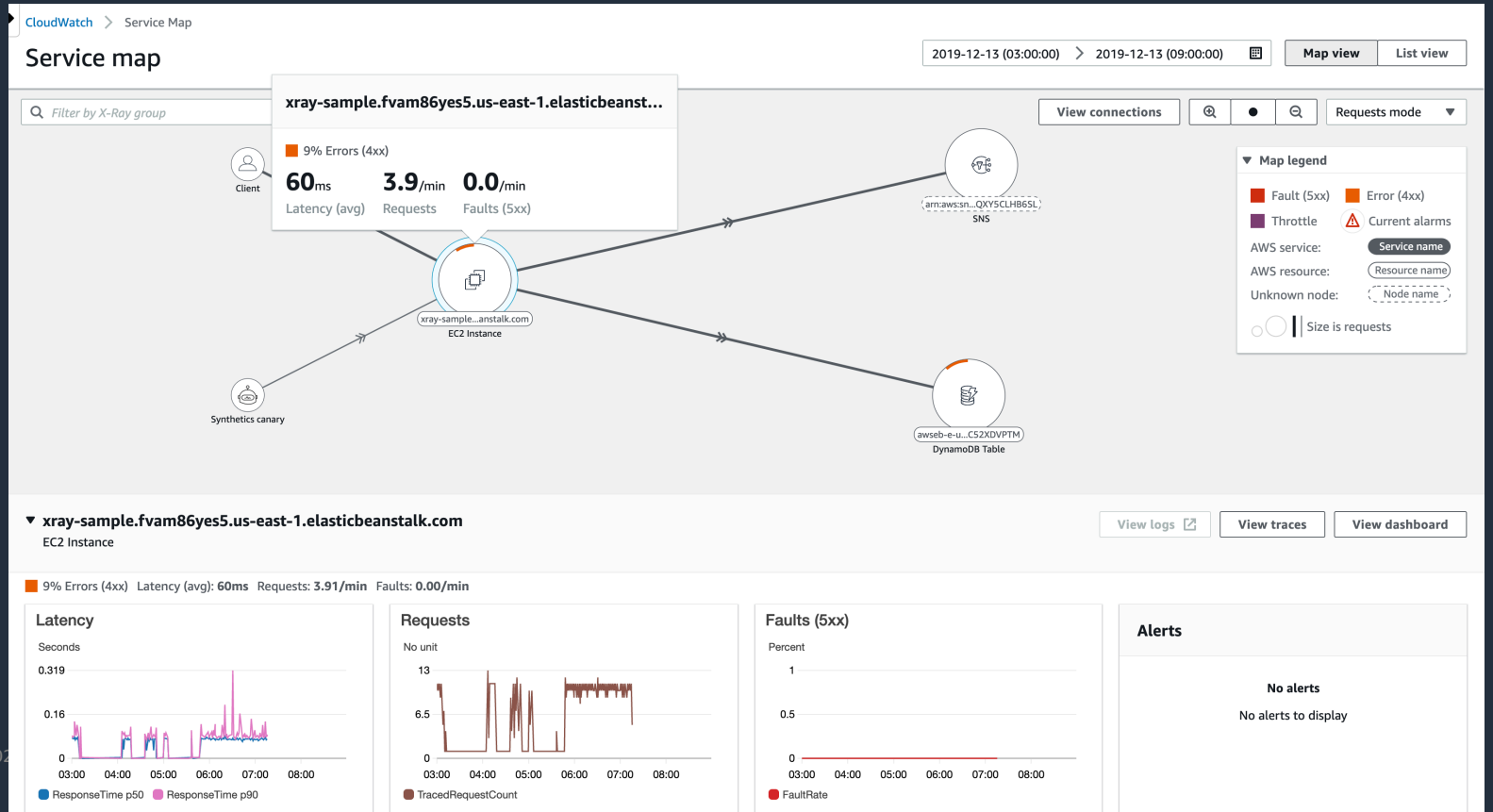




© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

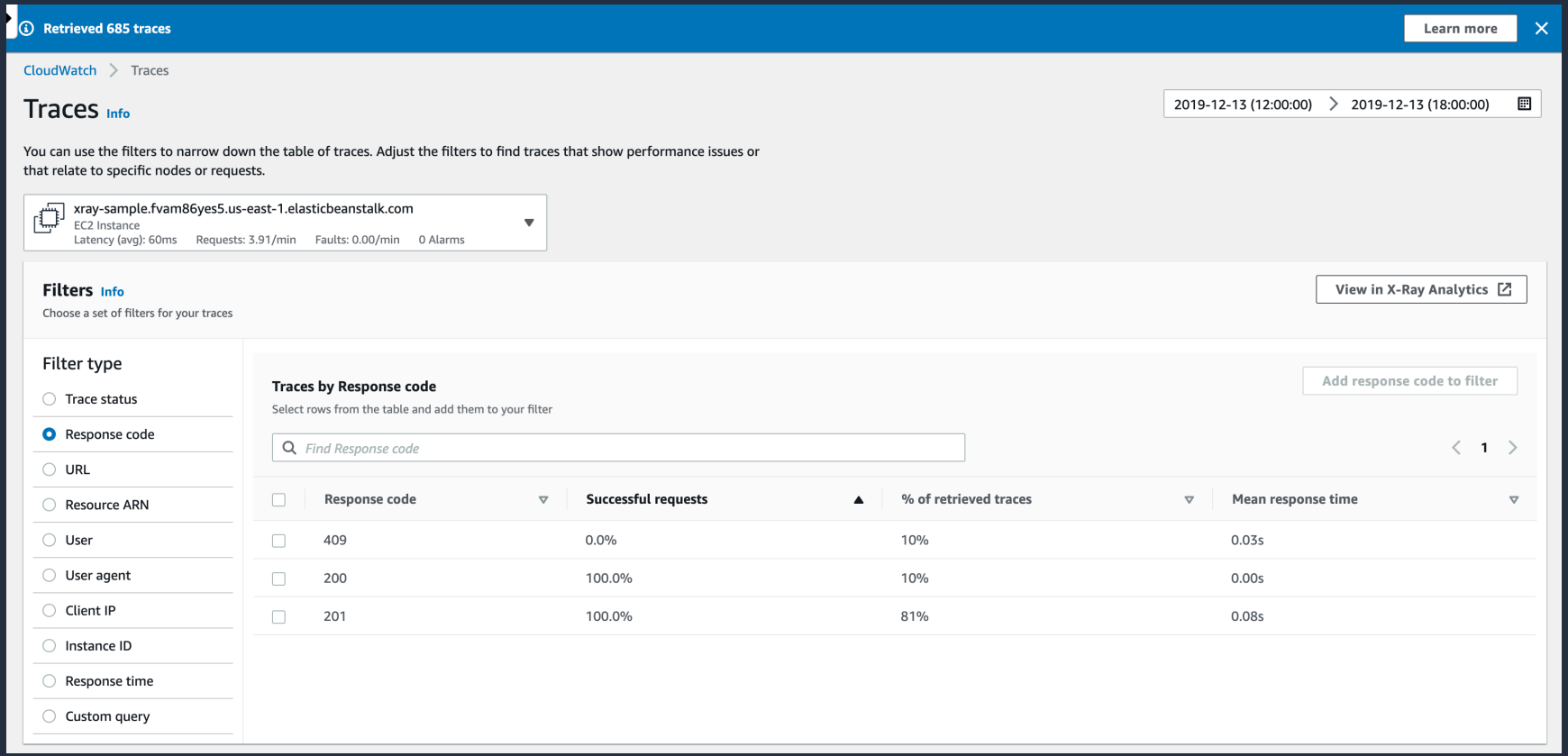
Service Lens (1/2) Service map

XRayのサンプルを us-east-1 (N.Virginia) で実行することで ServiceLensの動作を確認できます
https://docs.aws.amazon.com/ja_jp/xray/latest/devguide/xray-gettingstarted.html



© 202

Service Lens (2/2) Traces



© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.


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方法やログ出力などは、これまでと変わらない知識と経験がサーバーレス でも活用できます
- ログフォーマットなども通常のアプリケーション設計と変わらない
  - CloudWatch Logsが利用できる
- 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ができる。

© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

ありがとうございました

@_kensh

© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.