[AWS Black Belt Online Seminar]

Let’s Dive Deep into AWS Lambda

Part1 & Part2

Solutions Architect 西谷 圭介

2019/04/09

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.

内容についての注意点

- 本資料では2019年4月1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。

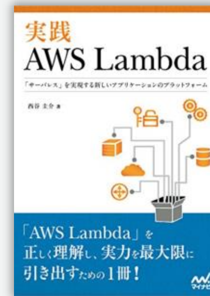
- AWS does not offer binding price quotes. AWS pricing is publicly available and is subject to change in accordance with the AWS Customer Agreement available at http://aws.amazon.com/agreement/. Any pricing information included in this document is provided only as an estimate of usage charges for AWS 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.

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

Who am I ?

Keisuke Nishitani

Manager, Specialist Solutions Architect
Amazon Web Service Japan K.K



ソーシャルは赤ドクロです

サーバーレスおじさん

アプリケーションよりなレイヤを担当しています

音楽が好きです、フジロッカーです

マンガも大好きです

@Keisuke69

Keisuke69

Keisuke69x

Keisuke69

Keisuke69

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

アジェンダ

サーバーレス概要

AWS Lambdaの基本

AWS Lambdaの使い方

ベストプラクティス/アンチパターン

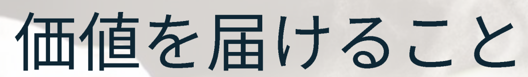
セキュリティ

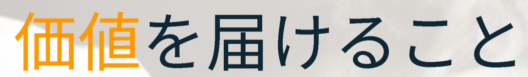
ユースケース・事例

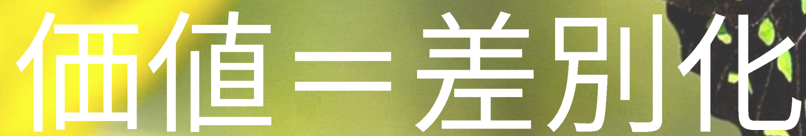
© 2019, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Developerの目的とは？

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


価値を届けること


価値を届けること



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

Photo credit: Ajith (അജിത്) via Visualhunt / CC BY-SA

Photo credit: Leo Hidalgo (@yompyz) via VisualHunt / CC BY

でもなかなかそうは行かない現実

サーバを所有すると

サーバのセットアップ
- OSのセットアップとネットワークなどの設定
- ランタイムやミドルウェアのセットアップ

キャパシティやスケーラビリティ管理

耐障害性を確保するための冗長化
- 複数台構成

セキュリティパッチの適用

差別化には繋がらない機能の実装
- スロットリング
- 認証・認可処理の実装

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

Undifferentiated Heavy Lifting

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

Photo credit: kizzzbeth via Visual hunt / CC BY

Developers need …

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

aws

More efficiency

More scalability

More agility

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

And…

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

No more servers

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

aws

アジェンダ

サーバーレス概要

AWS Lambdaの基本

AWS Lambdaの使い方

ベストプラクティス/アンチパターン

セキュリティ

ユースケース・事例

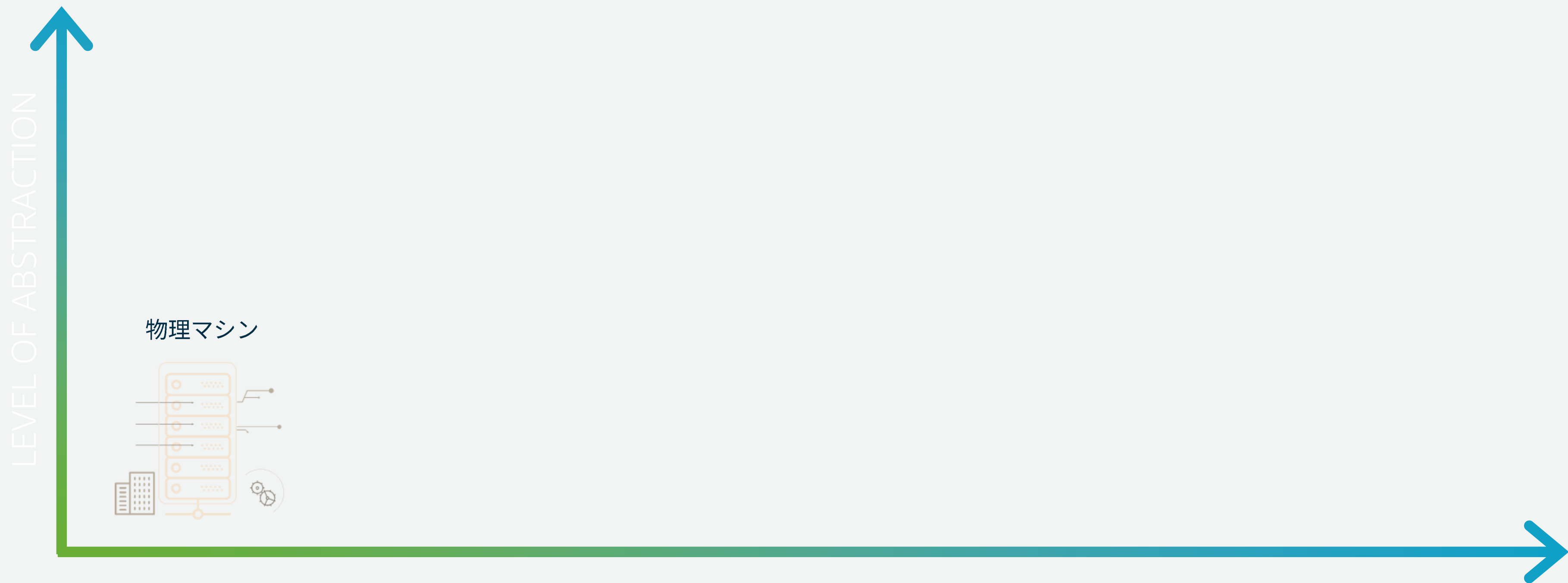
© 2019, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

サーバーレス概要

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

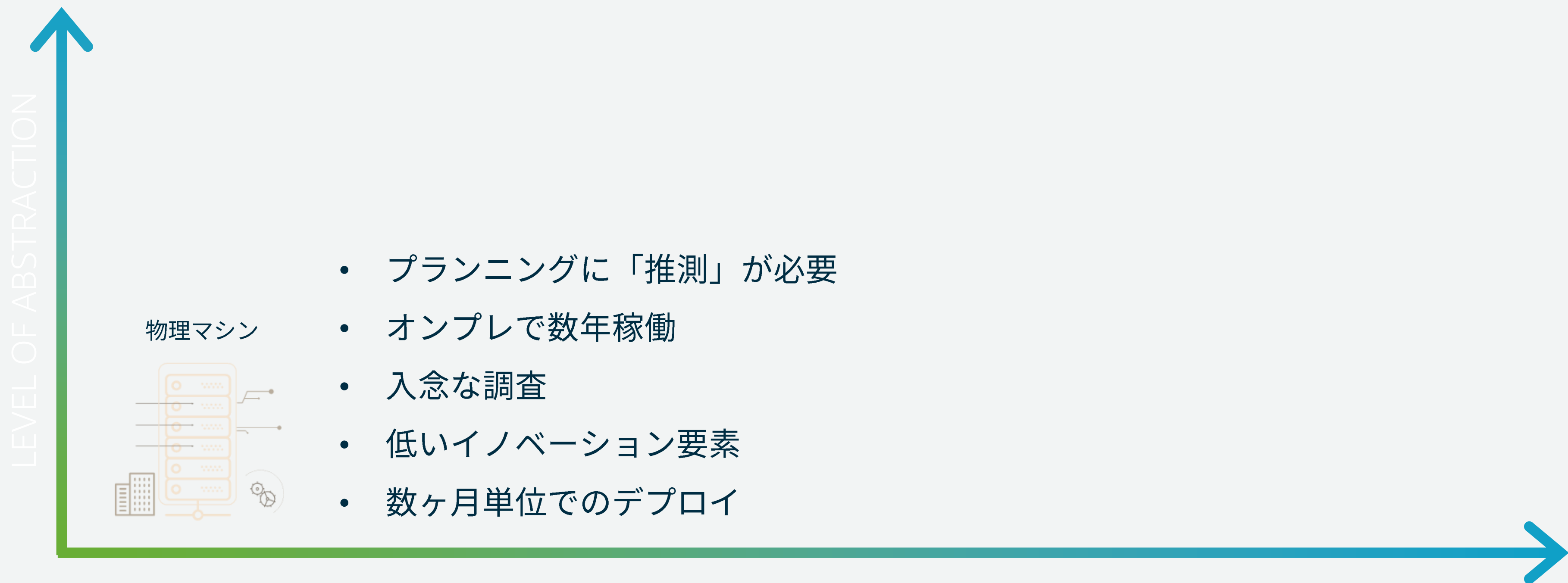
aws

Computingの進化



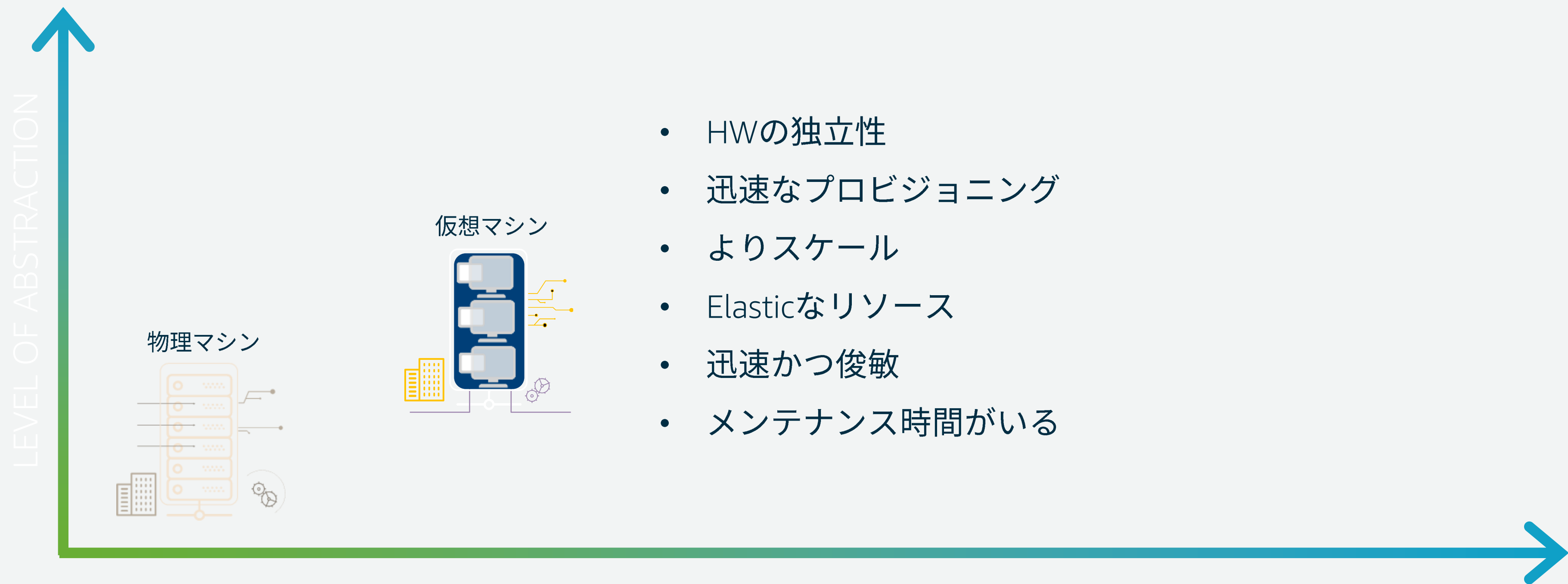
© 2019, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Computingの進化



© 2019, Amazon Web Services, Inc. or its Affiliates. All rights reserved.
aws

Computingの進化

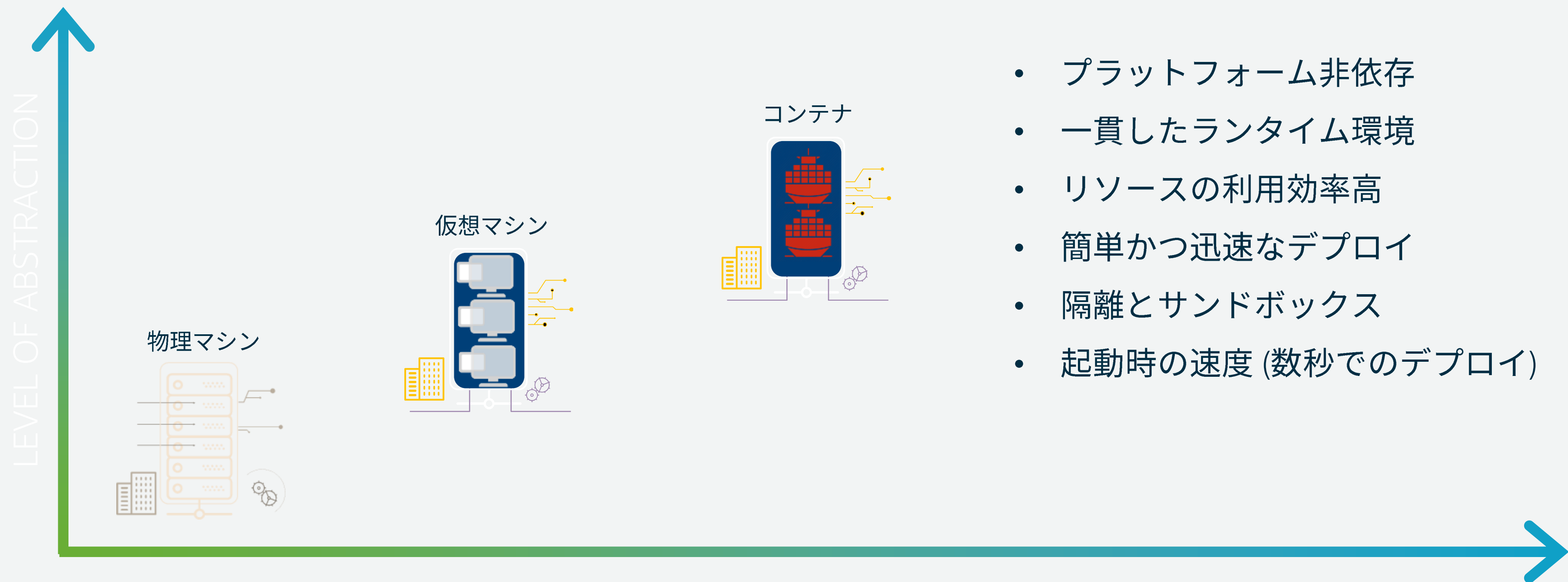


- HWの独立性
- 迅速なプロビジョニング
- よりスケール
- Elasticなリソース
- 迅速かつ俊敏
- メンテナンス時間がいる

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

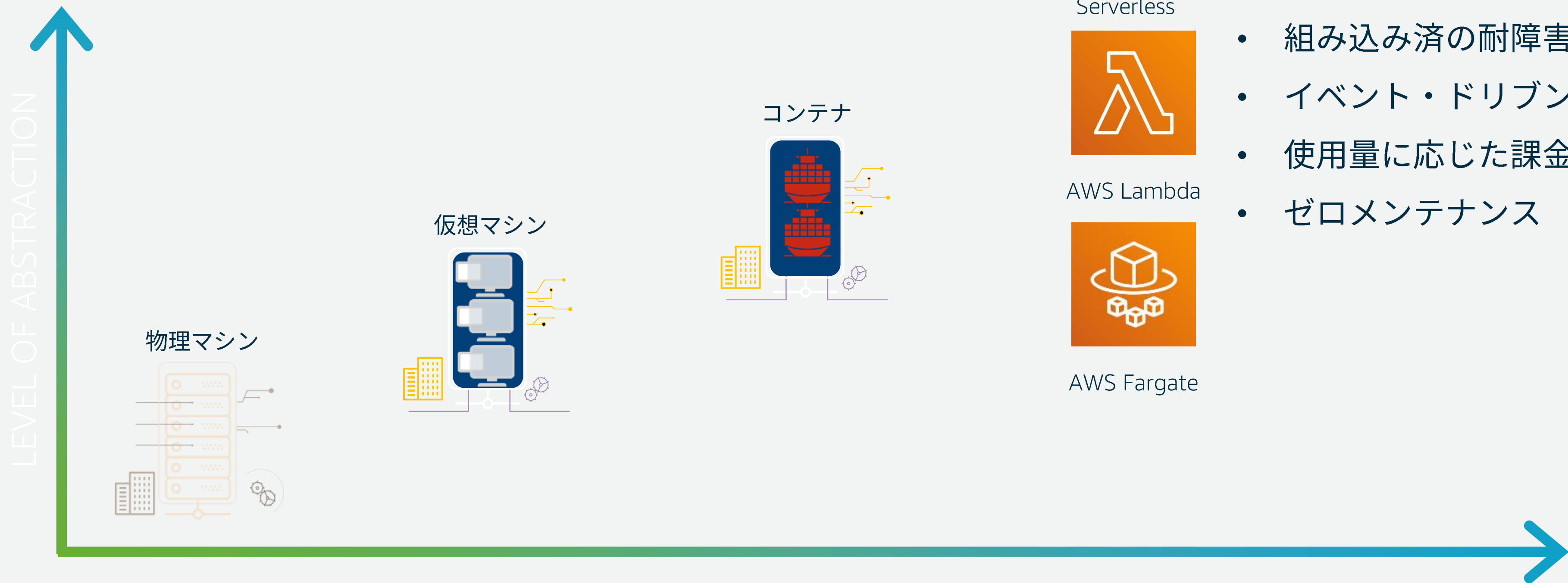
aws

Computingの進化



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

Computingの進化

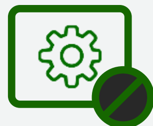


- 連続的なスケール
- 組み込み済の耐障害性
- イベント・ドリブン
- 使用量に応じた課金
- ゼロメンテナンス

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

aws

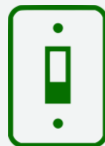
サーバーレスとは



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

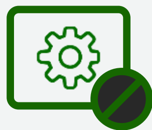


高可用かつ安全

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

aws

インフラのプロビジョニング、管理が不要



インフラのプロビジョニング不要
管理不要

事前にサーバなどのインフラのプロビジョニングが不要

インフラのキャパシティプランニングも不要

セキュリティパッチ適用といった保守作業不要

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

自動でスケール

自動でスケール

関数はリクエスト数に応じて自動的に起動数がコントロールされる

数件/月から数千件/秒までシームレスにスケール

スケーリングに基本的な上限はない

- ただし、アカウント単位で許可される数が決まっている

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

aws

価値に対する支払い

価値に対する支払い

事前に必要なコストが存在しない

リクエスト数ならびに処理の実行時間に対しての課金

- 実際の処理量に対する課金のみ

非常にコスト効率が良く、コストゼロからのスタートも可能

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

高可用かつ安全

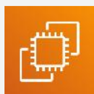
高可用かつ安全

レプリケーションと冗長性により
サービス本体およびサービスによって実行される Lambda関数の両方で高い可用性を実現するように設計されている

メンテナンス時間や定期的なダウンタイムはない

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

AWSのComputeサービス



Amazon EC2

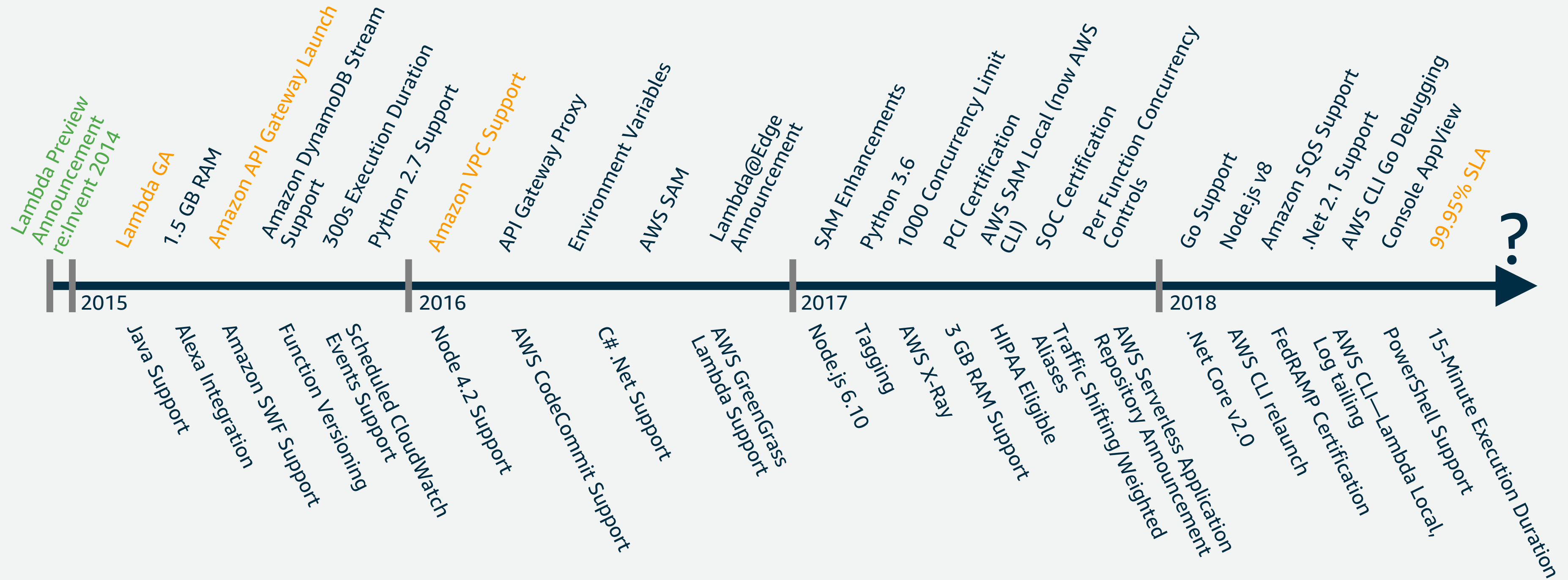
Container Services

AWS Lambda

スケールの単位	インスタンス	アプリケーション	関数
抽象化	ハードウェア	OS	ランタイム
使いドコロ	• OS、ネットワーク、ストレージのレベルで構成を制御したい • 好みのOSを利用したい • OS以上の全てを自分でコントロールしたい	• サーバを自分で構成して実行したい • アプリケーションの構成を制御したい • スケールを自分でコントロールしたい	• 必要なときだけコードの実行を行いたい • インフラの構成・管理を行いたくない

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

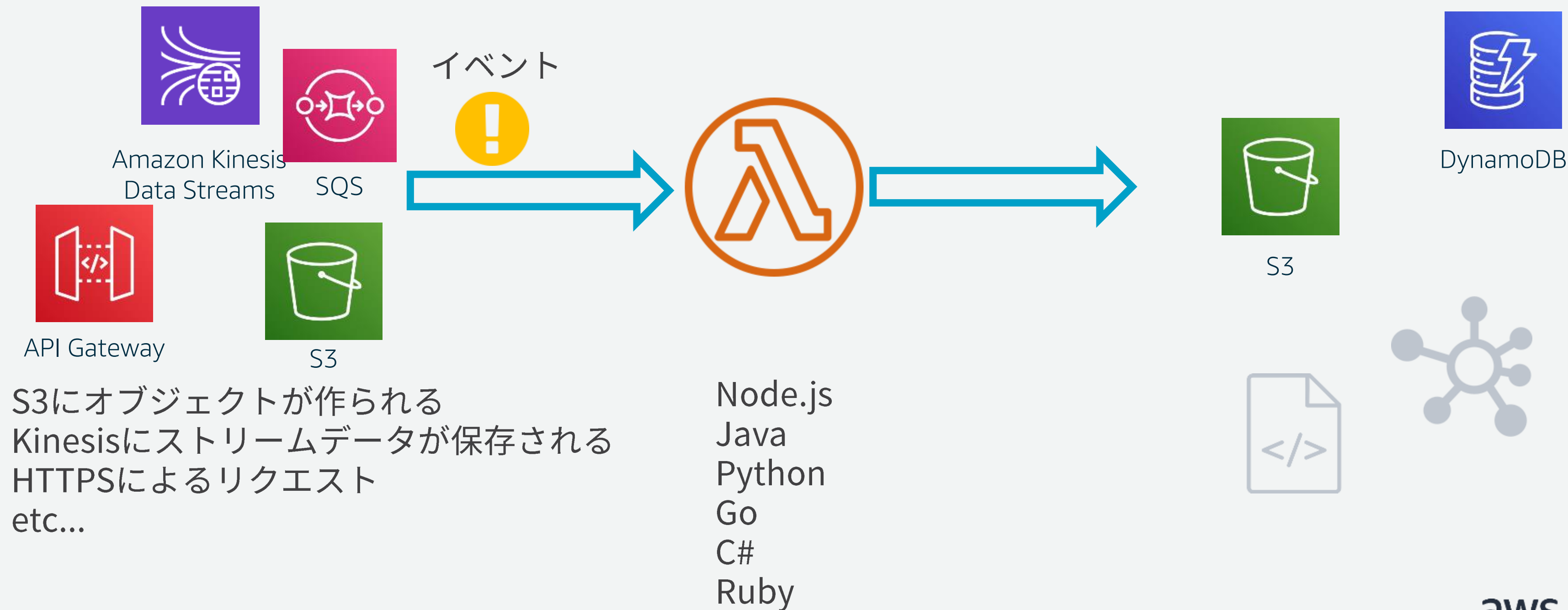
AWS Lambda release history



*As of October 2018, does not include region launches

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

サーバーレスなアプリケーションモデル



S3にオブジェクトが作られる
Kinesisにストリームデータが保存される
HTTPSによるリクエスト
etc…

Node.js
Java
Python
Go
C#
Ruby

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

広がり続ける組み合わせの選択肢

DEV TOOLS

AWS Cloud9
AWS Amplify
AWS SAM
AWS SAM CLI
AWS CodeStar
AWS CodeCommit
AWS CodeBuild
AWS CodeDeploy
AWS CodePipeline
AWS X-Ray

EVENT SOURCES

Amazon S3
Amazon API Gateway
Amazon Kinesis
Amazon DynamoDB
Amazon SQS
Amazon SNS
Amazon SES
AWS IoT
AWS AppSync
Amazon Lex
AWS CloudWatch Events

AWS Lambda

AWS Step Functions

DOWN STREAMS

Amazon S3
Amazon DynamoDB
Any AWS Services
External API

MONITORING

AWS CloudWatch
AWS CloudWatch Logs
AWS Config

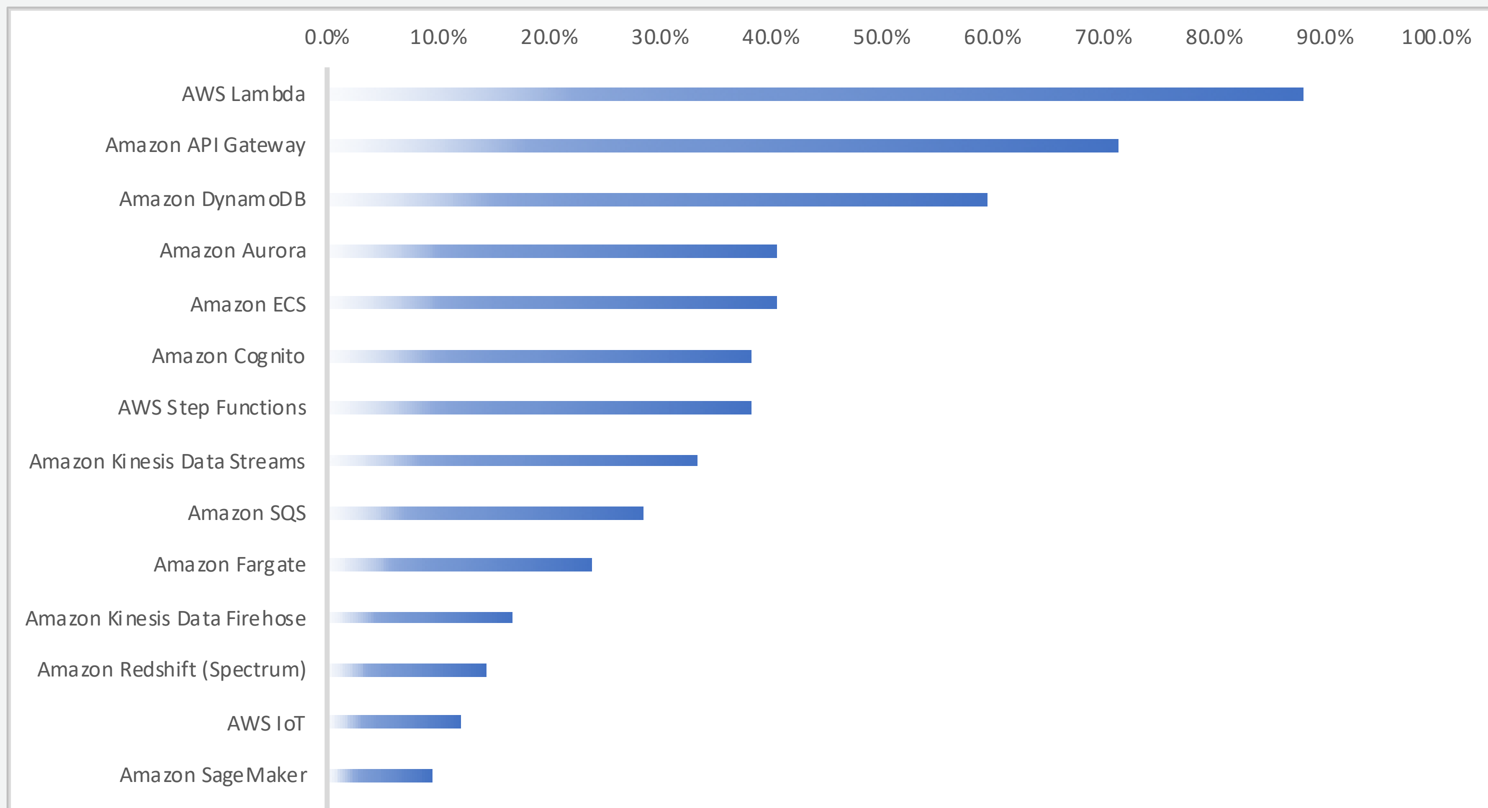
SECURITY

Amazon Cognito
AWS WAF
AWS IAM

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

aws

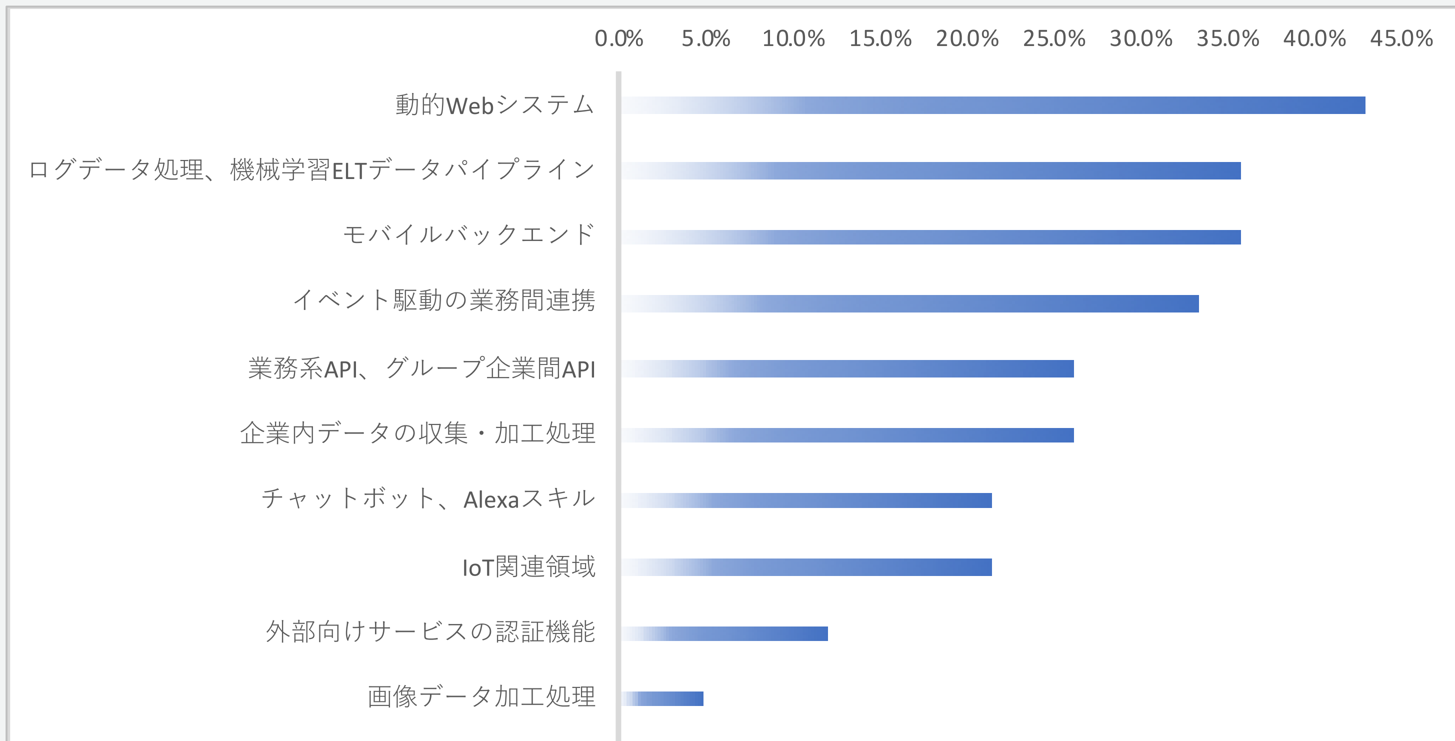
サーバーレス関心層の利用サービス



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

aws

サーバーレスの活用領域（これまでのアンケートに基づく）



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

アジェンダ

サーバーレス概要

AWS Lambdaの基本

AWS Lambdaの使い方

ベストプラクティス/アンチパターン

セキュリティ

ユースケース・事例

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

aws

AWS Lambdaの基本

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

aws

Lambda関数（ファンクション）

AWS Lambdaで実行するアプリケーション

- サポートされている言語、もしくはカスタムランタイムで用意した言語で記述

それぞれが隔離されたコンテナ内で実行される

- 1コンテナで複数イベントを同時に処理することはない

利用する言語の関数もしくはメソッドをハンドラーとして指定し、実行の際に呼び出す

- ハンドラー内では呼び出しの際にパラメータとして渡されるイベントのデータ（JSON形式）にアクセスすることが可能

コードは依存関係も含めてビルド、パッケージングしたうえでアップロード

- ZIP形式
- アップロードしたものはAmazon S3に保存され、実行時以外は暗号化される
- ユーザがS3にアップロードしてARNで指定することも可能

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

aws

サポートされている言語（2019年4月1日時点）

Python 2.7, 3.6, 3.7

Node.js 6.10, 8.10

.NET Core 1.0(C#), 2.0(C#), 2,1(C#/PowerShell 6.0)

Go 1.x

Java 8

Ruby 2.5

サポートされていない言語はカスタムランタイムを実装することで利用可能

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

aws

Lambda関数 - 基本設定

メモリ

- 64MBごとに128MBから3008MBの間で設定可能
- 容量に応じてCPU能力なども比例
- メモリ容量が一定を超えると使用するコア数も増えるため、マルチコアを活用するようなコードを実装することでより効率的な処理が可能

タイムアウト

- Lambda関数の実行時間に関するタイムアウト
- 最大900秒（15分）まで

実行ロール

- 必要なAWSリソースへのアクセスを許可するIAMロール
- 指定されたIAMロールにそってLambda関数からAWSのリソースへのアクセスが許可される

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

aws

Lambda関数の制限

インバウンドネットワーク接続はブロックされる

アウトバウンドはTCP/IPとUDP/IPソケットのみ

ptraceシステムコールはブロックされる

TCP 25番ポートのトラフィックはブロックされる

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

aws

AWS Lambdaの実行環境

オペレーティングシステム: Amazon Linux

AMI: amzn-ami-hvm-2017.03.1.20170812-x86_64-gp2

Linux カーネル: 4.14.77-70.59.amzn1.x86_64

AWS SDK for JavaScript: 2.290.0

SDK for Python (Boto 3)

- Python 3.6: boto3-1.7.74 botocore-1.10.74
- Python 3.7: boto3-1.9.42 botocore-1.12.42

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

イベントソース

イベントの発生元となるAWSのサービスまたはユーザが開発したアプリケーション
- Lambda関数の実行をトリガーする

イベントソースには3タイプありそれぞれ特性が異なる
- ポーリングベース
  - さらにストリームベースとそれ以外がある
  - Lambdaがポーリングをして処理するデータがある場合にLambda関数を実行
- それ以外
  - Lambda関数はイベントソースから呼び出される

イベントソースによって呼び出しタイプが異なる
- したがってリトライの動きも異なる

AWSのサービスの場合、ポーリングベースのサービス以外は各サービスで呼び出すLambda関数の設定情報を保持する（イベントソースマッピング）

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

aws

サポートされるイベントソース（2019年4月1日時点）

イベントソース	呼び出しタイプ	備考
Amazon S3	非同期	
Amazon DynamoDB	同期	ポーリングベース（ストリームベース）
Amazon Kinesis Data Streams	同期	ポーリングベース（ストリームベース）
Amazon Simple Notification Service	非同期	
Amazon Simple Email Service	非同期	
Amazon Simple Queue Service	同期	ポーリングベース（ストリームベースではない）
Amazon Cognito	同期	
AWS CloudFormation	同期	
Amazon CloudWatch Logs	同期	
Amazon CloudWatch イベント	非同期	
AWS CodeCommit	非同期	
Schedlued Event(Amazon CloudWatch Events を使用)	非同期	
AWS Config	非同期	
Amazon Alexa	同期	
Amazon Lex	同期	
Amazon API Gateway	同期	
AWS IoT ボタン	非同期	
Amazon CloudFront	同期	
Amazon Kinesis Data Firehose	同期	

呼び出しタイプ

カスタムアプリケーションによる呼び出し、もしくはAWS CLIなどを用いての手動実行の場合に呼び出しタイプを指定できる

- イベントソースがAWSのサービスの場合はサービスごとに事前に決められており、変更はできない

非同期呼び出し

- InvocationTypeはEvent
- レスポンス内容はリクエストが正常に受け付けられたかどうかのみ

同期呼び出し

- InvocationTypeはRequestResponse
- 実行完了時にレスポンスが返ってくる。レスポンス内容はLambda関数内でセット

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

aws

Lambda関数のリトライ①

エラーの種類、イベントソース、呼び出しタイプによって異なる

ストリームベースではないイベントソース

- 同期呼び出し
  - エラー発生時にはレスポンスのヘッダにFunctionError が含まれる
  - パーミッション、Limit、関数コードや設定の問題によるエラーの場合は特定のステータスコードが返る。詳細は以下URLを参照 https://docs.aws.amazon.com/ja_jp/lambda/latest/dg/API_Invoke.html#API_Invoke_Errors
  - AWSサービスからの呼び出しの場合、その設定に従う
- 非同期呼び出し
  - 自動的に2回までリトライされ、その後イベントは破棄される
  - リトライには遅延がある
  - Dead Letter Queue（DLQ）を設定することで未処理のイベントをAmazon SQS キューまたは Amazon SNS トピックに移動させ確認が可能

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

Lambda関数のリトライ②

ポーリングベースでストリームベースのイベントソース

- データの有効期限が切れるまでリトライを行う
- 失敗したレコードの有効期限が切れるか処理が成功するまで、そのシャードからの読み込みはブロックされ新しいレコードの読み込みは行われない

ポーリングベースでストリームベースではないイベントソース

- そのバッチのメッセージはすべてキューに返り、Visibility Timeoutが過ぎればまた処理が行われ、その後成功すればキューから削除される
- 新しいメッセージの処理はブロックされない

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

VPCアクセス

Amazon RDSやAmazon ElasticacheなどVPC内のリソースへインターネットを経由せずにアクセス可能

VPC内リソースにアクセスさせたいLambda関数に対してVPCサブネットおよびセキュリティグループを指定

- 新規作成時だけでなく既存のものを後から変更することも可能
- Availbility Zone (AZ)ごとに1つ以上のサブネットを指定しておくのがおすすめ
  - AZ障害発生時やIPアドレス枯渇時にも別AZでLambda関数が実行可能になる

Elastic Network Interface(ENI)を利用して実現

- 作成・削除はLambdaによって完全にコントロールされる
- ENIには指定したサブネットのIPがDHCPで動的に割り当てられる
- 関数に割り当てるIAM Roleに”AWSLambdaVPCAccessExecutionRole”というポリシーをアタッチしておくこと

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

VPCアクセスの注意点

設定をしたタイミングからインターネットアクセスは不可となる

- パブリック IP アドレスは割り当てられない
- 必要な場合はNATインスタンスを用意する、もしくはAmazon VPC NATゲートウェイを利用すること

充分な数の ENI またはサブネット IP がない場合は、リクエスト数が増えた場合に失敗する

- 非同期呼び出しの場合、このエラーはCloudWatch Logsには記録されない
- コンソールで実行するなど、同期実行することでエラー応答は取得できる

必要なENIのキャパシティは以下の計算式でざっくりと計算可能
Projected peak concurrent executions * (Memory in GB / 3GB)

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

VPCアクセスにまつわるFAQ

プライベートIPアドレスを固定することは可能？

ENIはLambdaによって作成・削除が自動的に行われるため、任意のアドレスの指定はできません

グローバルIPアドレスを固定することは可能？

そもそもVPCアクセスを有効にした時点でそのままではインターネットアクセスは不可となる。ただし、Managed Nat Gatewayや独自のNATインスタンスを利用することでインターネット通信およびグローバルIPアドレスを固定することが可能

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

VPCアクセスにまつわるFAQ

アクセス元として特定Lambdaファンクションのみ許可したい
Lambda関数に割り当てたセキュリティグループをソースとする許可ルールをアクセス先リソースのセキュリティグループに追加することで実現できます

オンプレミスにあるリソースへアクセスしたい
AWS Direct ConnectやVPN経由でアクセスすることが可能です。

レイテンシへの影響は？
Lambda関数への初回アクセス時などENIの作成を伴う場合は10秒～60秒程度の時間を必要とします

ENIはLambda関数ごとに作成されるのか？
ENIは複数のLambda関数から共用されます

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

アクセス許可 - 実行ロール

Lambda関数がAWSのサービスやリソースにアクセスするためのアクセス許可

- 最低でもログ出力用にAmazon CloudWatch Logsへのアクセス許可が必要
- Lambda関数作成時に指定することで、呼び出されたときにLambda によってこのロールが引き受けられる
- AWS Identity and Access Management (IAM)を利用して作成
  - 信頼されたエンティティとして”AWS Lambda”を指定
  - アクセス許可は定義済みのもの、もしくはユーザが用意したものを指定

Lambdaのリソースを使用するためのアクセス許可を他のアカウントや AWS のサービスに付与するには、リソースベースのポリシーを使用

- Lambda関数、バージョン、エイリアス、レイヤーバージョンなど

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

アクセス許可 - 管理ポリシー

AWSLambdaBasicExecutionRole

- ログを CloudWatch にアップロードするためのアクセス許可。

AWSLambdaKinesisExecutionRole

- Amazon Kinesis データストリームまたはコンシューマーからイベントを呼び出すためのアクセス許可。

AWSLambdaDynamoDBExecutionRole

- Amazon DynamoDB ストリームからレコードを読み出すためのアクセス許可。

AWSLambdaSQSQueueExecutionRole

- Amazon Simple Queue Service (Amazon SQS) キューからメッセージを読み出すためのアクセス許可。

AWSLambdaVPCAccessExecutionRole

- 関数を VPC に接続するように Elastic Network Interface を管理するためのアクセス許可。

AWSXrayWriteOnlyAccess

- トレースデータを X-Ray にアップロードするためのアクセス許可。

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

アクセス許可 - リソースポリシー

Lambda関数およびレイヤー用にリソースベースのアクセス許可ポリシーをサポート

- リソースごとに他のアカウントに使用許可を付与することが可能
- AWSのサービスによる関数の呼び出しを許可することも可能

Lambda関数に対して、別の AWS アカウントへのアクセス許可を付与することも可能

- principalとして対象のAWSアカウント ID指定

IAM ポリシーでリソースと条件を指定することで、ユーザーのアクセス許可の範囲を制限

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

同時実行数

ある時点における実行中のLambda関数の数

セーフガードとしてアカウントに対してデフォルトでは1,000で制限されている

- 実績に応じて制限緩和申請することが可能

許可された同時実行数を越えてリクエストされた場合、スロットリングエラー (エラーコード: 429) が返却される

- 非同期呼び出しの場合、15～30 分程度はバーストとして許容されるがそれ以降はスロットリングの対象

アカウントに許可された値を上限として関数単位で任意の割合で割り振ることも可能

ConcurrentExecutionsとUnreservedConcurrentExecutionsのメトリクスで確認可能

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

同時実行数の見積もり

ポーリングベースかつストリームベース

- シャード数と同じ

ポーリングベースだがストリームベースではない

- 同時実行数までポーリングを自動的にスケールアップ
- Amazon SQSでは 5 つの同時関数呼び出しが最初のバーストでサポートされ、1 分ごとに 60 の同時実行呼び出しで同時実行が増加

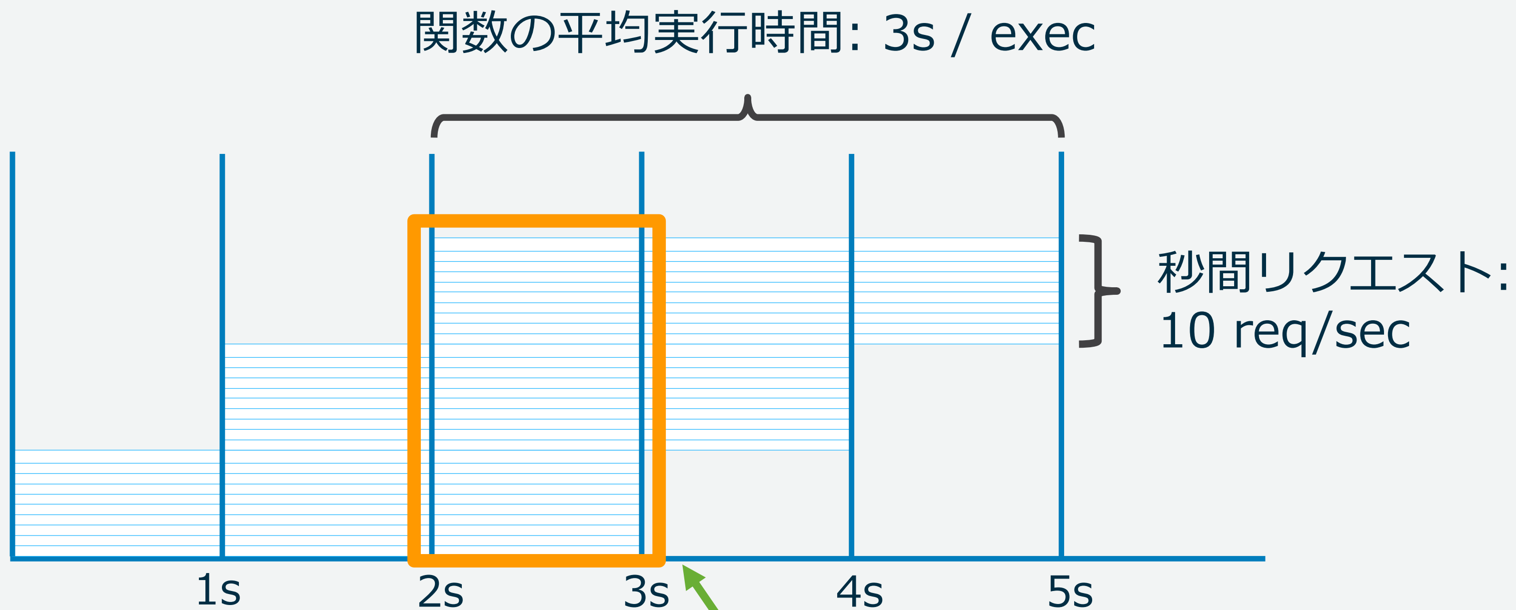
それ以外

- 以下の式にて算出
  秒間呼び出し回数×平均実行時間（秒）

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

aws

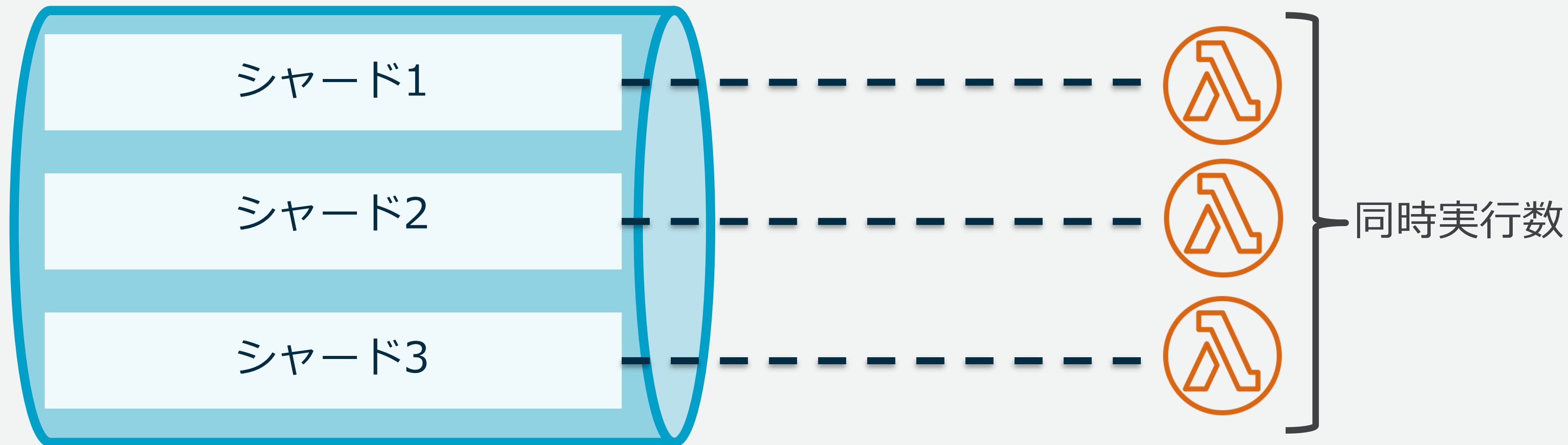
同時実行数



同時実行数
= “同時”に実行されているタイミング

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

同時実行数 - ストリームベースの場合



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

自動スケーリング

トラフィックの増加に応じて許可された同時実行数まで動的にスケール

初期レベルでバーストした後、負荷が落ち着くまでもしくは同時実行の制限に到達するまで1分ごとに500ずつ同時実行していく

- 初期値はリージョンによって異なり、東京リージョンは1000
- この値は制限緩和不可

仮に制限緩和を行って同時実行数が10,000まで引き上げられていたとしてもスパイク時にいきなり10,000実行されることはない

- 上記の初期レベルの値以上に”いきなり”スケールすることはない

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

aws

Lambda関数のライフサイクル

1. （ENIの作成）
2. コンテナの作成
3. デプロイパッケージのロード
4. デプロイパッケージの展開
5. ランタイム起動・初期化
6. 関数/メソッドの実行
7. コンテナの破棄

- 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

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

aws

Lambda関数のライフサイクル

1. （ENIの作成）
2. コンテナの作成
3. デプロイパッケージのロード
4. デプロイパッケージの展開
5. ランタイム起動・初期化
6. 関数/メソッドの実行
7. コンテナの破棄

- 指定されたランタイム
- S3からのダウンロードとZIPファイルの展開
- Durationには含まれない

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

aws

Lambda関数のライフサイクル

1. （ENIの作成）
2. コンテナの作成
3. デプロイパッケージのロード
4. デプロイパッケージの展開
5. ランタイム起動・初期化
6. 関数/メソッドの実行
7. コンテナの破棄

- 各ランタイムの初期化処理
- グローバルスコープの処理もこのタイミングで実行される
- Durationには含まれない

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

aws

Lambda関数のライフサイクル

1. （ENIの作成）
2. コンテナの作成
3. デプロイパッケージのロード
4. デプロイパッケージの展開
5. ランタイム起動・初期化
6. 関数/メソッドの実行
7. コンテナの破棄

- ハンドラーで指定した関数/メソッドの実行
- いわゆるDurationの値はここの実行時間

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

Lambda関数のライフサイクル

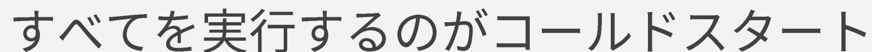
1. （ENIの作成）
2. コンテナの作成
3. デプロイパッケージのロード
4. デプロイパッケージの展開
5. ランタイム起動・初期化
6. 関数/メソッドの実行
7. コンテナの破棄

- 不要になったLambda関数のコンテナは破棄される

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

Lambda関数のライフサイクル

1. （ENIの作成）
2. コンテナの作成
3. デプロイパッケージのロード
4. デプロイパッケージの展開
5. ランタイム起動・初期化
6. 関数/メソッドの実行
7. コンテナの破棄



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

Lambda関数のライフサイクル

1. （ENIの作成）
2. コンテナの作成
3. デプロイパッケージのロード
4. デプロイパッケージの展開
5. ランタイム起動・初期化
6. 関数/メソッドの実行
7. コンテナの破棄

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

Lambda関数のライフサイクル

1. （ENIの作成）
2. コンテナの作成
3. デプロイパッケージのロード
4. デプロイパッケージの展開
5. ランタイム起動・初期化
6. 関数/メソッドの実行
7. コンテナの破棄

基本的に毎回同じ内容が実行される

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

Lambda関数のライフサイクル

1. （ENIの作成）
2. コンテナの作成
3. デプロイパッケージのロード
4. デプロイパッケージの展開
5. ランタイム起動・初期化
6. 関数/メソッドの実行
7. コンテナの破棄

基本的に毎回同じ内容が実行される

作成したコンテナを再利用して
1~6の処理を省略することで効率化
（ウォームスタート）

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

コールドスタートが起こる条件

簡単に言うと、利用可能なコンテナがない場合に発生

- そもそも1つもコンテナがない状態
- 利用可能な数以上に同時に処理すべきリクエストが来た
- コード、設定を変更した
  - コード、設定を変更するとそれまでのコンテナは利用できない

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

aws

コールドスタートが起こる条件

簡単に言うと、利用可能なコンテナがない場合に発生

- そもそも1つもコンテナがない状態
- 利用可能な数以上に同時に処理すべきリクエストが来た
- コード、設定を変更した
  - コード、設定を変更するとそれまでのコンテナは利用できない

裏を返せば安定的にリクエスト数が発生している場合はコールドスタートはほとんど発生しない

コンテナを維持するためのポーリングは多くの場合、無駄

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

aws

バックグラウンドプロセスの凍結と再開

関数の終了時に実行中のバックグラウンドプロセスがある場合、Lambdaはプロセスをfreezeさせ、次回、関数を呼び出した際に再開する

- ただし、コンテナが再利用される場合だけであり保証はされていない

この場合、バックグラウンドプロセスは残っていても処理は行われていない

- プロセス再作成のオーバーヘッドを減らせる

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

制限事項

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

制限事項（制限緩和可能）

リージョンごとに適用され、制限緩和申請が可能

- リクエストはサポートセンターコンソールから

リソース	デフォルトの制限
同時実行数	1000
関数とレイヤーストレージ	75GB

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

aws

制限事項（制限緩和不可）

リソース	制限
関数のメモリ割り当て	128 MBから3008 MBまで、64 MBごとに増加できます。
関数タイムアウト	900 seconds (15 分)
関数の環境変数	4 KB
関数リソースベースのポリシー	20 KB
関数レイヤー	5 layers
呼び出しペイロード (リクエストとレスポンス)	6 MB (同期) 256 KB (非同期)
デプロイパッケージサイズ	50 MB (zip 圧縮済み、直接アップロード) 250 MB (解凍、例: レイヤー) 3 MB (コンソールエディタ)
テストイベント (コンソールエディタ)	10
/tmp ディレクトリのストレージ	512 MB
ファイルの説明	1024
実行プロセス/スレッド	1024

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

料金

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

料金体系

リクエスト (全リージョン)

- 月間100万リクエストまでは無料
- 超過分は$0.20/100万リクエスト
(1リクエストあたり$0.0000002)

実行時間 (全リージョン)

- 100ms単位で課金となり、100ms以下は繰り上げで計算
- メモリー容量により単価および無料時間が異なる
例) メモリ128MBの場合おおよそ$ 0.000000208/100ms
- あくまでも単価としては1GBの関数を1秒間実行する場合の価格（$0.00001667）

かなり複雑なのでこちらも参考に

- http://qiita.com/Keisuke69/items/e3f79b50b6039175401b

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

アジェンダ

サーバーレス概要

AWS Lambdaの基本

AWS Lambdaの使い方

ベストプラクティス/アンチパターン

セキュリティ

ユースケース・事例

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

AWS Lambdaの使い方

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

aws

プログラミングモデル

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

aws

プログラミングモデルの基本

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、実行の際に呼び出すエントリポイントとなる
- ハンドラーを呼び出すことでLambda関数のコードが実行開始される
- 呼び出しの際にパラメータとして渡されるイベントのデータ（JSON形式）にアクセスすることが可能

コンテキスト

- ランタイムに関する情報が含まれ、ハンドラー内部からアクセス可
- コンテキストオブジェクトが 2 つめのパラメータとしてハンドラー関数に渡される
- コールバックを使用する言語の場合、コールバックメソッドの振る舞いを設定可能
  - デフォルト（true）は全ての非同期処理の完了を待ってレスポンス
  - false にするとcallback が呼び出された時点で即座に処理終了

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

プログラミングモデルの基本

ロギング

- Lambda関数内にログ出力のステートメントを含めることができる
- これらのログはAWS CloudWatch Logsに書き込まれる
- 各言語ごとに特定のステートメントによるログエントリ作成が可能
- CloudWatch Logs の制限を受けるため、スロットリングによって失われることがあることに注意
  - 場合によっては、実行コンテキスト終了時に失われることがある

例外

- Lambda 関数として使用する言語によって正常終了方法は異なる
- 同様に実行中に発生したエラー（例外）を通知する方法も異なる
- Lambda関数を同期的に呼び出している場合、クライアントへとエラーがレスポンスされる

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

aws

プログラミングモデルの基本

Lambda 関数は、ステートレスにする必要がある

コンピューティングインフラストラクチャとの直接的な関連性はない

- 関数はリクエストのたびに同じコンピューティングインスタンスで実行されるとは限らない

ローカルファイルシステムへのアクセス、子プロセス、その他類似の生成物はリクエストの有効期限に限定される

永続化するには Amazon S3、Amazon DynamoDB、または別のクラウドストレージサービスなどへの保存が必要

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

aws

設計図

一般的なユースケース向けのサンプルコード集

- PythonとNode.js向けのみ提供

Lambda関数作成時に選択可能

設計図を利用して作成したLambda関数はあとから編集可能

- ひとまず設計図をベースに作成し、あとで自分用にカスタマイズ

同様の機能にServerless Application Repositoryというのもある（詳細は後述）

- 作成したサーバーレスアプリケーションを公開、共有するための機能
- 設計図はサービスチームによってメンテナンス、ユーザによる追加登録不可

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

aws

aws サービス リソースグループ sample バージニア北部 サポート

Lambda > 関数 > 関数の作成

関数の作成 情報

以下のいずれかのオプションを選択して、関数を作成します。

一から作成

シンプルな Hello World の例で開始します。

設計図の使用

一般的ユースケース用のサンプルコードと設定プリセットから Lambda アプリケーションを構築します。

Serverless Application Repository の参照

AWS Serverless Application Repository からサンプル Lambda アプリケーションをデプロイします。

設計図 情報

エクスポート

タグや属性によるフィルター、またはキーワードによる検索

< 1 2 3 4 5 6 7 ... 11 >

ユースケースを選択

kinesis-firehose-syslog-to-json

An Amazon Kinesis Firehose stream processor that converts input records from RFC3164 Syslog format to JSON.

nodejs6.10 · kinesis-firehose

logicmonitor-send-cloudwatch-events

Creates LogicMonitor OpsNotes for CloudWatch Events, thereby enabling correlation between events and performance data.

splunk-elb-application-access-logs-processor

Stream Application ELB access logs from S3 to Splunk's HTTP event collector

nodejs6.10 · splunk · elb · s3 · application-elb

フィードバック 日本語

© 2008 - 2019, Amazon Web Services, Inc. またはその関連会社。無断転用禁止。 プライバシーポリシー 利用規約

Lambda関数の実装

利用可能な言語で**普通に**実装する

言語ごとの方言はあるが基本的な構成や実装パターンに大きな違いはない

- ハンドラ
- コンテキスト
- ロギング
- 例外処理
- トレース

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

aws

Lambda関数の実装

各言語の細かい違いはドキュメント参照

Node.js
https://docs.aws.amazon.com/ja_jp/lambda/latest/dg/programming-model.html

Python
https://docs.aws.amazon.com/ja_jp/lambda/latest/dg/python-programming-model.html

Java
https://docs.aws.amazon.com/ja_jp/lambda/latest/dg/java-programming-model.html

Go
https://docs.aws.amazon.com/ja_jp/lambda/latest/dg/go-programming-model.html

C#
https://docs.aws.amazon.com/ja_jp/lambda/latest/dg/dotnet-programming-model.html

PowerShell
https://docs.aws.amazon.com/ja_jp/lambda/latest/dg/powershell-programming-model.html

Ruby
https://docs.aws.amazon.com/ja_jp/lambda/latest/dg/lambda-ruby.html

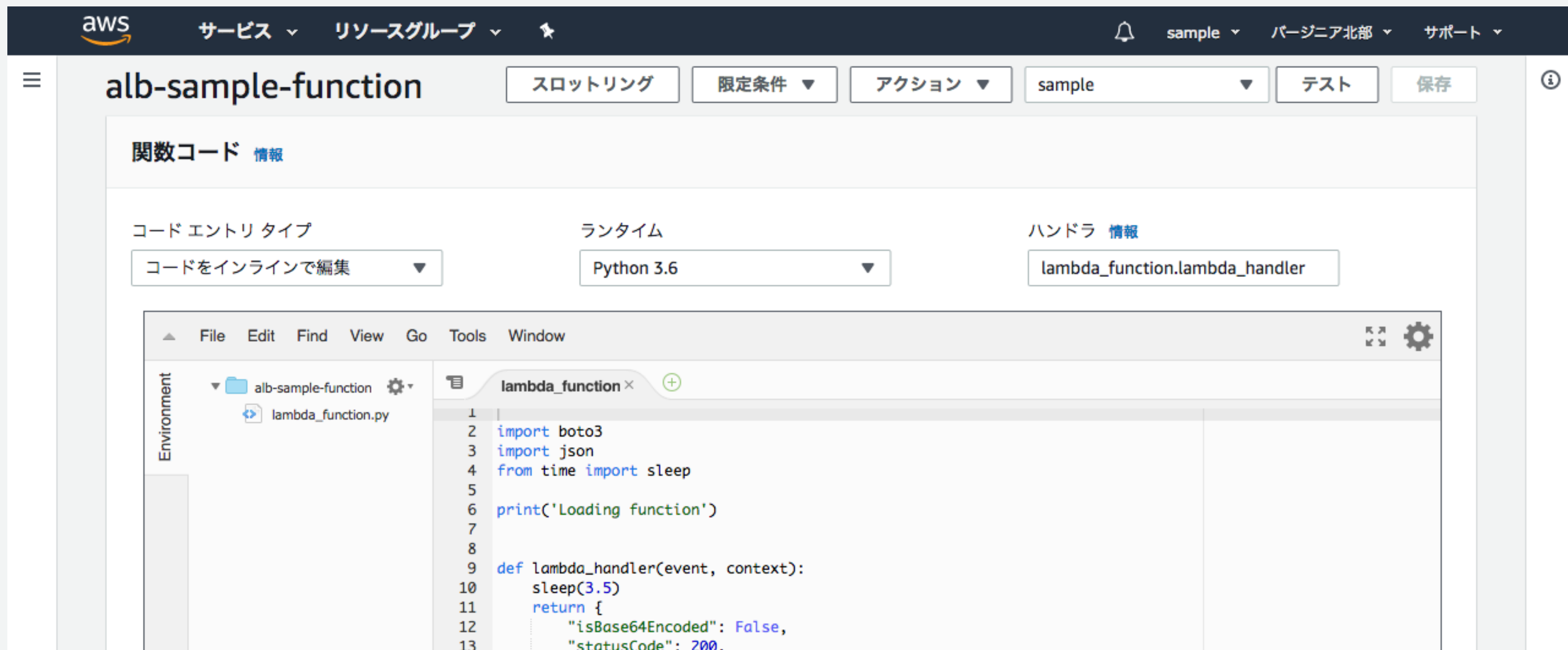
© 2019, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

コンソールエディタ

AWS Cloud9をベースにしたコンソールエディタが利用可能

- コンパイルが不要な言語のみ利用可能

マネージメントコンソール上でLambda 関数のコーディング、テスト、実行結果の表示が可能



デプロイパッケージ

コードと依存関係で構成される .zip または .jar ファイル

- フォルダを含めないフラットなzipファイルであること
- 依存関係の含め方は言語による
  例) Node.jsの場合、依存ライブラリはnode_modulesフォルダ内にインストールした上でこのフォルダも含めてzip化する
- 言語ごとの詳細
  https://docs.aws.amazon.com/ja_jp/lambda/latest/dg/deployment-package-v2.html

Lambda関数の実装コードとしてアップロード

コードファイルおよびデプロイパッケージを構成するすべての依存ライブラリに対してグローバルな読み取り権限が必要

- ないと実行がエラーになる

```
$ zipinfo test.zip
Archive:  test.zip
Zip file size: 473 bytes, number of entries: 2
-r--r--r--  3.0 unx        0 bx stor 17-Aug-10 09:37 exlib.py
-r--r--r--  3.0 unx      234 tx defN 17-Aug-10 09:37 index.py
2 files, 234 bytes uncompressed, 163 bytes compressed:  30.3%
```

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

aws

Lambda関数の設定

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

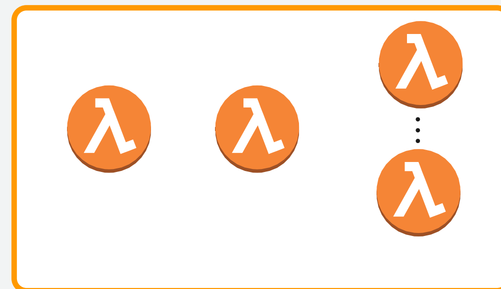
aws

同時実行数の管理

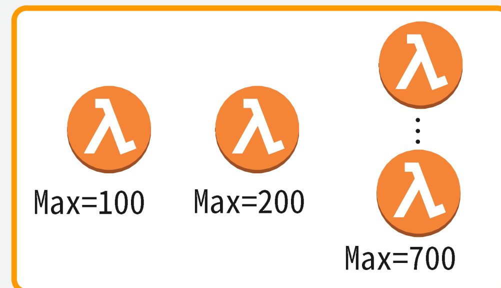
アカウント単位で許可された同時実行数をファンクション単位で任意に割当てることが可能

例：アカウントの上限が1000の場合

- 関数A：上限100
- 関数B：上限200
- その他：上限700(1000-100-200=700)



DBへの書き込みや外部APIの呼び出しなどダウンストリームの流量制御や、ENI/IPアドレスの利用量をコントロールしたい場合に便利



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

aws

aws サービス リソースグループ sample バージニア北部 サポート

alb-sample-function

スロットリング 限定条件 アクション sample テスト 保存

ネットワーク

Virtual Private Cloud (VPC) 情報
関数がアクセスする VPC を選択します。

非 VPC

デバッグとエラー処理

DLQ リソース 情報
最大再試行回数を超えた後にイベントペイロードを送信する AWS のサービスを選択します。

なし

アクティブトレースを有効にします 情報

同時実行数

予約されていないアカウントの同時実行 **900**

- 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 100

監査とコンプライアンス

この関数の呼び出しは、運用とリスクの監査、ガバナンス、およびコンプライアンスのために、AWS CloudTrail で記録できます。CloudTrail コンソールで作業を開始してください。

フィードバック 日本語

© 2008 - 2019, Amazon Web Services, Inc. またはその関連会社。無断転用禁止。

プライバシーポリシー 利用規約

環境変数

環境変数を利用することでコードの変更なく設定した値を渡すことが可能、

- 設定値などをコード内にハードコーディングする必要がなくなる
- DBの接続URLを本番用と開発用の2 種類のLambda 関数で切り替えたり、ファイルやログの出力先などの切り替えなどに
- キーと値のペアで定義

関数のコードからのアクセスは各言語でサポートされている環境変数へのアクセス方法がそのまま利用可能

- Node.js の場合、process.env を利用してアクセス可能

関数のバージョニング利用時は環境変数の情報もスナップショットに含まれる

Lambda関数から使用可能なあらかじめ定義された環境変数もある

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

環境変数の暗号化

キーと値はAWS Key Management Service（AWS KMS）で自動的に暗号化して保存され、必要に応じて復号される

- デフォルトではLambda 用のKMS サービスキーを利用して暗号化・復号化
- 暗号化されるのはデプロイプロセス中ではなくプロセス後
- 関数が呼び出されると自動的に復号化
- 環境変数に機密情報を保存する必要がある場合は、Lambda 関数をデプロイする前にその情報を暗号化することを強く推奨

独自のサービスキーを利用することも可能

- より高い柔軟性が得られる
- 機密情報を環境変数としてセットする場合などに利用
- 独自のサービスキーを利用する場合はAWS KMS の料金が適用される
- 独自のサービスキーを利用する場合は追加でkms:Decryptを許可する必要あり

暗号化ヘルパー

- 機密情報を暗号化して保存可能
- 復号はLambda 関数内で実行
- 暗号化された環境変数の値をLambda関数内で復号化するためのサンプルも生成

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

aws
サービス
リソースグループ
sample
バージニア北部
サポート
alb-sample-function
スロットリング
限定条件
アクション
sample
テスト
保存
環境変数
環境変数を、関数コードからアクセス可能なキーと値のペアとして定義できます。これらは、関数コードを変更することなく構成設定を保存するのに便利です。詳細
DB_PASS
復号
コード
削除
S3_BUCKET_NAME
sample
暗号化
コード
削除
キー
値
暗号化
コード
削除
暗号化の設定
伝送中の暗号化のためのヘルパーの有効化 情報

伝送中に暗号化する AWS KMS キー
arn:aws:kms:us-east-1:530697723613:key/ae546a1a-66da-4a5c-8a15-dd60c29252cd

保管時に暗号化する AWS KMS キー 情報
保管時の環境変数を暗号化する AWS KMS キーを選択するか、Lambda が暗号化を管理するようにします。
(デフォルト) aws/lambda
カスタマーマスターキーの使用
フィードバック
日本語
© 2008 - 2019, Amazon Web Services, Inc. またはその関連会社。無断転用禁止。
プライバシーポリシー
利用規約

Lambda関数で利用可能な環境変数

キー	リザーブド	値
_HANDLER	あり	関数で設定されているハンドラの場所です。
AWS_REGION	あり	Lambda 関数が実行される AWS リージョン。
AWS_EXECUTION_ENV	あり	プレフィックス AWS_Lambda_ が付いたランタイム識別子です。たとえば、AWS_Lambda_java8 と指定します。
AWS_LAMBDA_FUNCTION_NAME	あり	関数の名前。
AWS_LAMBDA_FUNCTION_MEMORY_SIZE	あり	関数で使用できるメモリの量 (MB 単位)。
AWS_LAMBDA_FUNCTION_VERSION	あり	実行される関数のバージョン。
AWS_LAMBDA_LOG_GROUP_NAME AWS_LAMBDA_LOG_STREAM_NAME	あり	Amazon CloudWatch Logs グループの名前と関数のストリーム。
AWS_ACCESS_KEY_ID AWS_SECRET_ACCESS_KEY AWS_SESSION_TOKEN	あり	関数の実行ロールから取得されたアクセスキーです。
LANG	いいえ	en_US.UTF-8。これがランタイムのロケールです。
TZ	あり	環境のタイムゾーン (UTC)。実行環境は、システムクロックを同期するために NTP を使用します。
LAMBDA_TASK_ROOT	あり	Lambda 関数コードのパスです。
LAMBDA_RUNTIME_DIR	あり	ランタイムライブラリへのパス。
PATH	いいえ	/usr/local/bin:/usr/bin/:/bin:/opt/bin
LD_LIBRARY_PATH	いいえ	/lib64:/usr/lib64:$LAMBDA_RUNTIME_DIR:$LAMBDA_RUNTIME_DIR/lib:$LAMBDA_TASK_ROOT:$LAMBDA_TASK_ROOT/lib:/opt/lib
NODE_PATH	いいえ	(Node.js) /opt/nodejs/node8/node_modules/:/opt/nodejs/node_modules:$LAMBDA_RUNTIME_DIR/node_modules
PYTHONPATH	いいえ	(Python) $LAMBDA_RUNTIME_DIR.
GEM_PATH	いいえ	(Ruby) $LAMBDA_TASK_ROOT/vendor/bundle/ruby/2.5.0:/opt/ruby/gems/2.5.0.
AWS_LAMBDA_RUNTIME_API	あり	(カスタムランタイム) ランタイム API のホストおよびポート。

環境変数の利用における制限

キーとバリューの合計サイズが4KB以下

設定可能な環境変数のキー名には以下の制限がある

- 文字 [a-zA-Z] で始める必要あり
- 英数字とアンダースコア ([a-zA-Z0-9_])のみ
- AWS Lambda が予約する特定のキーセットのうち変更可能なもの
  例) Lang, LD_LIBRARY_PATHなど

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

バージョニング

ある一時点のLambdaファンクションをバージョンとして管理可能

- 新しいバージョンはいつでも発行可能で、各バージョンには一意のARNがある
- Lambdaファンクションの作成/更新時にpublishパラメータを追加する
- PublishVersionを実行することで明示的に発行することも可能
- バージョンの発行をするまでは$LATESTが唯一のバージョンとなる
- 一度発行すると構成も含めて一切変更不可
- 単純にバージョン番号がインクリメントする

```
exports.handler = function(event,context) {context.succeed("hi");}
```

Version:1

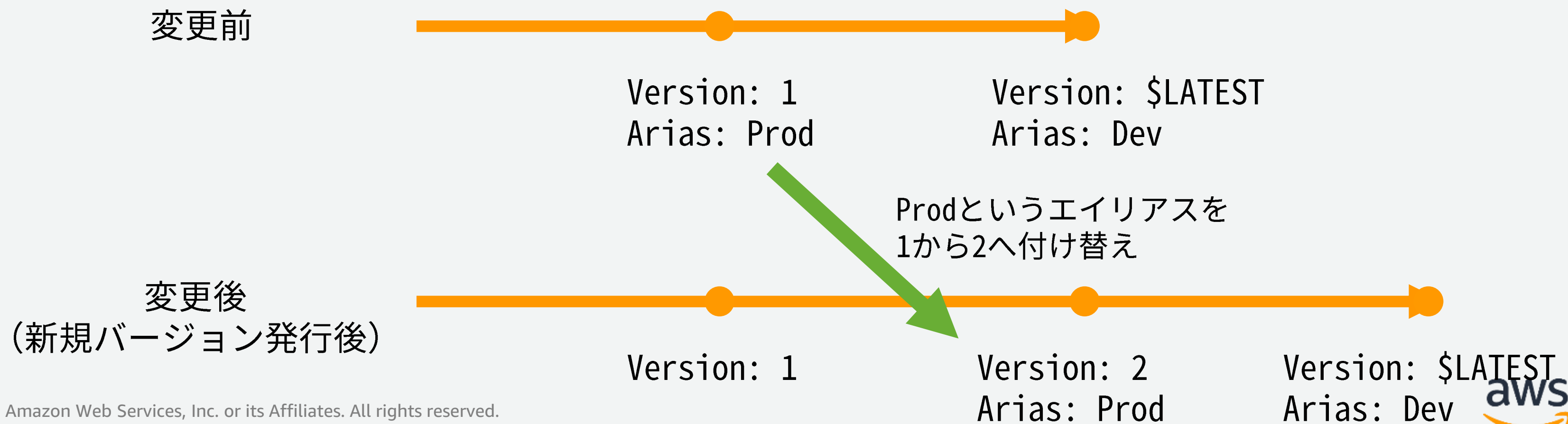
```
exports.handler = function(event,context) {context.succeed("bye");}
```

Version:2

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

エイリアス

- 特定バージョンに対するポインタのようなもの
- エイリアスを作成することでバージョン番号を把握していなくても指定バージョンを呼び出せる
- いつでも付け替え可能



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

エイリアスを利用したトラフィックの移行

通常、エイリアスが別の関数バージョンを指すように更新されると、インバウンドリクエストのトラフィックは更新後のバージョンにすぐ切り替わる

routing-configを利用することで2つのバージョン間でのリクエストトラフィックが転送される割合（%）を決められる

（例）

1. 2パーセントのみを新しいバージョンにルーティング
2. 残りの98 パーセントは元のバージョンにルーティング
3. 新しいバージョンが成熟して安定すれば、必要に応じて比率を徐々に変更
4. エイリアスを更新し、すべてのトラフィックを新しいバージョンにルーティング

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

aws

ファンクションの指定方法

バージョン発行前、最新バージョンを指定する場合:

- FunctionName
- FunctionName:$LATEST

特定バージョンを指定する場合

- FunctionName:1
- FunctionName:2

エイリアスで指定する場合:

- FunctionName:production
- FunctionName:v1_2_3_4

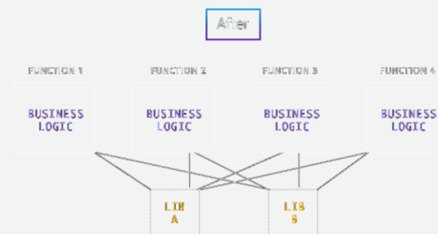
© 2019, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

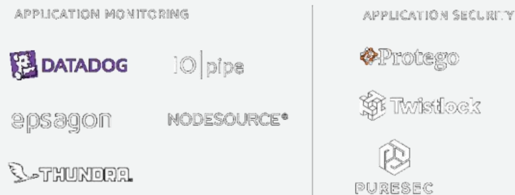
Lambda Layers

様々なLambdaファンクションで共通利用するコンポーネントを個別に持つのではなく、Lambda Layerとして定義し参照することができるように

責務を分けることができるようになるため、プログラマはよりビジネスロジックに集中できるように

Layerをシェアするエコシステム





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

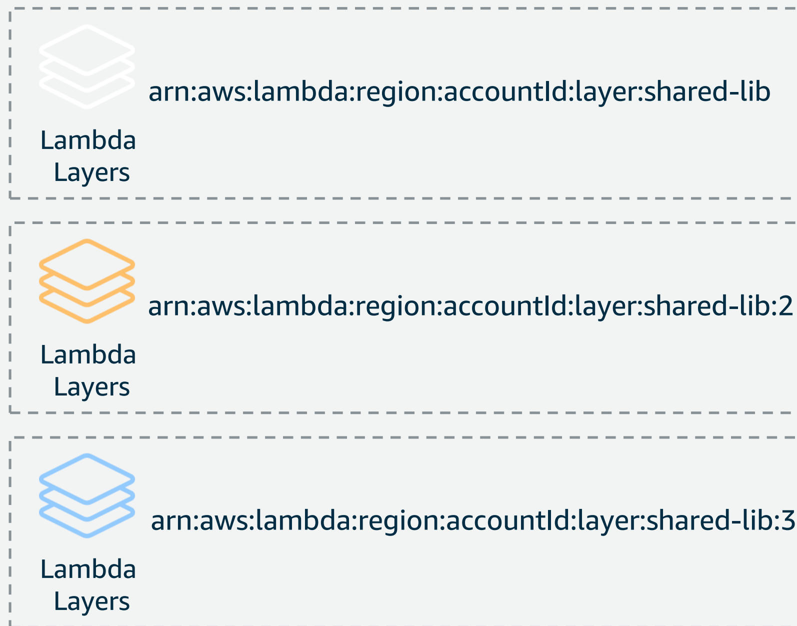
Lambda Layersの利用

共通コンポーネントをZIPファイルにしてLambda Layerとしてアップロード

Layersはimmutableであり、アップデート管理のためにバージョニングする場合も

バージョンが削除されたり、利用権限が剥奪された場合、それを利用していたファンクションは稼働し続けるが新規作成は不可

5つのLayersまで利用可能で、カスタムランタイムをLayerとすることも可能



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

aws Services Resource Groups

sample N. Virginia Support

Lambda > Layers

Layers (0)

Create layer

Filter by attributes or search by keyword

< 1 >

Layer name	Latest version	Layer version ARN
There is no data to display.		

Feedback English (US)

© 2008 - 2018, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

Create layer

Layer configuration

Name

sample-layer

Description

Description

Code entry type

Upload a .zip file

For files larger than 10 MB, consider uploading using Amazon S3.

Compatible runtimes Info

Select all compatible runtimes for your layer.

Select runtimes

Python 2.7 ✕ Python 3.6 ✕ Python 3.7 ✕

 © 2008 - 2018, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Add layer to function

Layer selection

Select an existing AWS-vended layer or layer in your account, or provide a layer that has been shared with you. You can connect a maximum of 5 layers to a function.

- (•) Select from list of runtime compatible layers
- ( ) Provide a layer version ARN

Select from list of runtime compatible layers

Layer: sample-layer

Version: 1

Cancel | Add

© 2008 - 2018, Amazon Web Services, Inc. or its affiliates. All rights reserved.

aws Services Resource Groups sample N. Virginia Support

Lambda > Functions > alb-sample-function

ARN - arn:aws:lambda:us-east-1:530697723613:function:alb-sample-function

alb-sample-function

Throttle | Qualifiers | Actions | sample | Test | Save

Configuration | Monitoring

Designer

Add triggers

Choose a trigger from the list below to add it to your function.

- API Gateway
- AWS IoT
- Alexa Skills Kit
- Alexa Smart Home
- Application Load Balancer
- CloudFront

alb-sample-function

Unsaved changes

Layers (1)

Application Load Balancer

Add triggers from the list on the left

Amazon CloudWatch Logs

Resources that the function's role has access to appear here

Feedback English (US)

© 2008 - 2018, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

Lambda Layersの注意点

/optの下に指定した順番で展開される

- 特定のモジュールのバージョンを上書きすることも可能
- 同じパスや、同名ファイルを使うと意図せず上書きされることに注意

Layerは1つの関数で5個まで

- デプロイパッケージのサイズ制限は変わらず、関数あたり非圧縮で250MBまで
- Layerへの登録はzip or S3で登録のみ

/optの下に各言語の依存を解決する固有のパスがある

- Python – python, python/lib/python3.7/site-packages (site directories)
- Node.js – nodejs/node_modules, nodejs/node8/node_modules (NODE_PATH)

https://docs.aws.amazon.com/lambda/latest/dg/configuration-layers.html

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

aws

タグ付け

Lambda関数にタグを使用することで「グループ化とフィルタリング」、「コスト配分」が可能

- グループ化とフィルタリング
  タグを適用することで、特定のアプリケーションまたは請求部門に含まれる Lambda 関数のリストを隔離
- コスト配分
  Lambda のタグ付けのサポートは AWS の請求と統合されているため、請求書を動的カテゴリに明細化して関数を特定のコストセンターにマッピング可能。コスト配分が請求明細レポートで明示されるため、より簡単に AWS のコストを分類して追跡可能

コンソールとCLIのどちらからでも可能

© 2019, Amazon Web Services, Inc. or its Affiliates. All rights reserved.
aws

aws サービス リソースグループ sample バージニア北部 サポート

alb-sample-function

スロットリング 限定条件 アクション sample テスト 保存

タグ

タグを使用して関数をグループ化し、フィルタリングできます。タグは、大文字と小文字が区別されるキーと値のペアから構成されます。詳細

キー	値	
service	prod:serviceA	削除
キー	値	削除

実行ロール

関数のアクセス権限を定義するロールを選択します。カスタムロールを作成するには、IAM コンソールに移動します。

既存のロールを使用する

既存のロール

この Lambda 関数で使用するために作成した既存のロールを選択します。このロールには、Amazon CloudWatch Logs にログをアップロードするアクセス権限が必要です。

lambda_basic_execution

IAM コンソールで lambda_basic_execution ロールを表示します。

基本設定

説明

メモリ (MB) 情報

作成する関数には、設定したメモリに比例する CPU が割り当てられます。

512 MB

タイムアウト 情報

0 分 10 秒

フィードバック 日本語

© 2008 - 2019, Amazon Web Services, Inc. またはその関連会社。無断転用禁止。

プライバシーポリシー 利用規約

デッドレターキュー（DLQ）

非同期実行時に2 回のリトライにも関わらず処理が正常に終了しなかったときにそのイベントを指定されたAmazon SQS のキューもしくはAmazon SNS のトピックへと送信可能

イベントを失うことなく後続の処理で扱うことができるため、より信頼性の高いシステムが開発可能に

関数単位で設定可能であり、全ての非同期呼び出しで利用可能

- コードのロジックによるエラーだけでなくスロットリングされた場合もそのイベントを保存

振り分け先に応じて実行ロールにアクセス許可を追加する必要あり

- Amazon SQS: SendMessage
- Amazon SNS: Publish

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

DLQのメッセージ内容

ペイロードには元のイベントのペイロードがそのまま格納される

メッセージが DLQで許容された最大サイズを超過しているなどによってDLQに書き込めない場合、そのイベントは削除され、DeadLetterErrors メトリクスに記録される

メッセージ属性には以下の情報が含まれる

名前	タイプ	値
RequestID	文字列	一意のリクエスト ID
ErrorCode	数値	3 桁の HTTP エラーコード
ErrorMessage	文字列	エラーメッセージ (1 KB に切り捨て)

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

aws

Custom Runtimes

Linuxで互換のランタイムを持ち込むことで、任意の言語をLambda関数を記述できるように

新たにリリースされたRuntime APIによって可能に

AWSからはC++/RustをOSSでリリース

https://github.com/awslabs/aws-lambda-cpp-runtime

https://github.com/awslabs/aws-lambda-rust-runtime

パートナーからはPHP/Erlang/Elixir/COBOLなどが提供される予定

使用時は関数のランタイムをprovidedに設定

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

aws

Runtime bootstrap

ファンクションに“bootstrap”と呼ばれる実行ファイルを含める必要がある

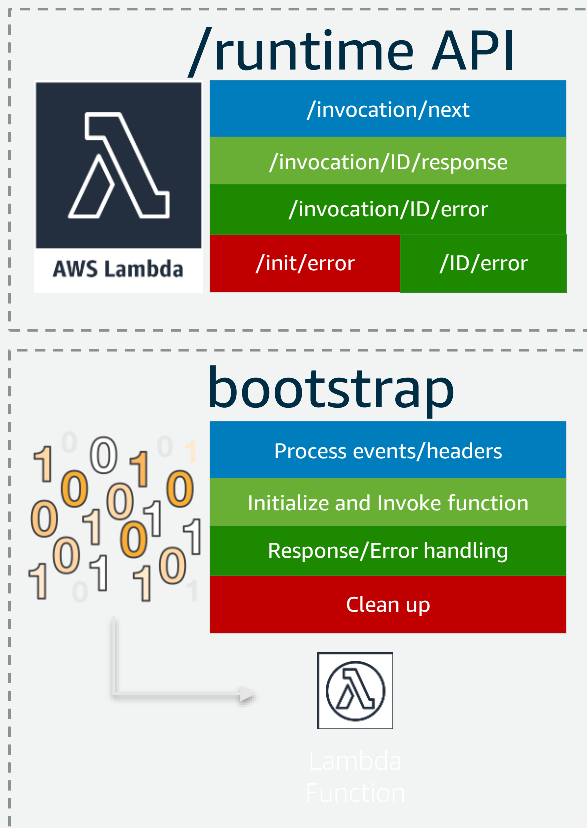
- デプロイパッケージに含めてもLayerとして登録してもいい

bootstrap

- Runtime HTTP APIと実行するファンクションとの間でブリッジとして振る舞う
- レスポンスとエラーハンドリング、contextの作成とファンクション実行を管理する
- Lambdaファンクション実行時にデプロイパッケージ、もしくは指定されたLayer内にbootstrapという名前を実行する
- 存在しなければエラーとなる

Layerとして登録すればRuntime管理者とRuntime利用者の責務を分離できる

- 利用者はRuntimeを気にせず実装すればよい



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

aws

Custom Runtimeの作り方

Custom Runtimeのエントリポイントはbootstrapという名前の実行ファイル

bootstrap内では初期化処理を実行したのち、ファンクションの実行処理を実装する

- Initialization Tasks
- Processing Tasks

Processing Taskはループしてイベントを待ち受ける必要がある

Initialization Tasksは請求時間とタイムアウトの対象

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

aws

Initialization Tasks

環境変数の読み取り（例）

- _HANDLER
- ファンクションで設定されたハンドラのロケーション。通常ファイル名.メソッド
- LAMBDA_TASK_ROOT
  - ファンクションのコードを含むディレクトリ
- AWS_LAMBDA_RUNTIME_API
  - Runtime APIのホスト名とポート

関数の初期化

- ハンドラファイルを読み込み、グローバルスコープの処理を実行する
- SDKクライアントやデータベース接続などの静的リソースを一度作成し、複数の呼び出しに再利用する

エラーハンドリング

- エラー発生時は初期化エラーAPIをコールし、即座に終了させる

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

aws

Processing Tasks ①

イベントの取得

- next invocation APIをコールして次のイベントを取得
- レスポンスボディにはイベントデータが含まれる

トレーシングヘッダの伝播

- APIレスポンスに含まれるLambda-Runtime-Trace-IdヘッダからX-Rayのトレーシングヘッダを取得
- _X_AMZN_TRACE_IDに値をセット

コンテキストオブジェクトの作成

- 環境変数のコンテキスト情報およびAPI レスポンスのヘッダーでオブジェクトを作成

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

Processing Tasks ②

関数のハンドラ呼び出し

- イベントとコンテキストオブジェクトをハンドラに渡す

レスポンスのハンドリング

- Invocation response APIをコールしてレスポンスをポスト

エラーハンドリング

- エラー発生時はinvocation errorをコール

クリーンアップ

- 使われないリソースをリリースし、他サービスへとデータ送信するもしくは次のイベントを処理する前の追加処理を実行する

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

aws

Example

```
#!/bin/sh

set -euo pipefail

# Initialization - load function handler
source $LAMBDA_TASK_ROOT/"$(echo $_HANDLER | cut -d. -f1).sh"

# Processing
while true
do
  HEADERS="$(mktemp)"
  # Get an event
  EVENT_DATA=$(curl -sS -LD "$HEADERS" -X GET "http://${AWS_LAMBDA_RUNTIME_API}/2018-06-
01/runtime/invocation/next")
  REQUEST_ID=$(grep -Fi Lambda-Runtime-Aws-Request-Id "$HEADERS" | tr -d '[:space:]' | cut -d: -
f2)

  # Execute the handler function from the script
  RESPONSE=$($(echo "$_HANDLER" | cut -d. -f2) "$EVENT_DATA")

  # Send the response
  curl -X POST "http://${AWS_LAMBDA_RUNTIME_API}/2018-06-
01/runtime/invocation/$REQUEST_ID/response"  -d "$RESPONSE"
done
```

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

Runtime Interface

カスタムランタイムのためのHTTP API

環境変数 “AWS_LAMBDA_RUNTIME_API” でエンドポイントを取得

API	パス	メソッド	
Next Invocation	/runtime/invocation/next	GET	呼び出しイベントを取得します。レスポンス本文には、呼び出しのペイロードが含まれます。
Invocation Response	/runtime/invocation/*AwsRequestId*/response	POST	呼び出しのレスポンスを Lambda に送信します
Invocation Error	/runtime/invocation/*AwsRequestId*/error	POST	関数からエラーが返ると、ランタイムはエラーを JSON ドキュメント形式に変換し、それを呼び出しエラーパスに投稿します。
Initialization Error	/runtime/init/error	POST	エラーメッセージを初期化エラーパスに投稿します。

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

aws

Custom Runtimeの使いどころ

Lambdaがサポートしている言語の新しいバージョンを待てない場合
- 新バージョンをCustom Runtimeとして用意し、サポートされたのちに移行

Lambdaがサポートを打ち切った言語のバージョンを使い続けたい場合

AWS Lambdaでサポートされていない言語を利用したい場合

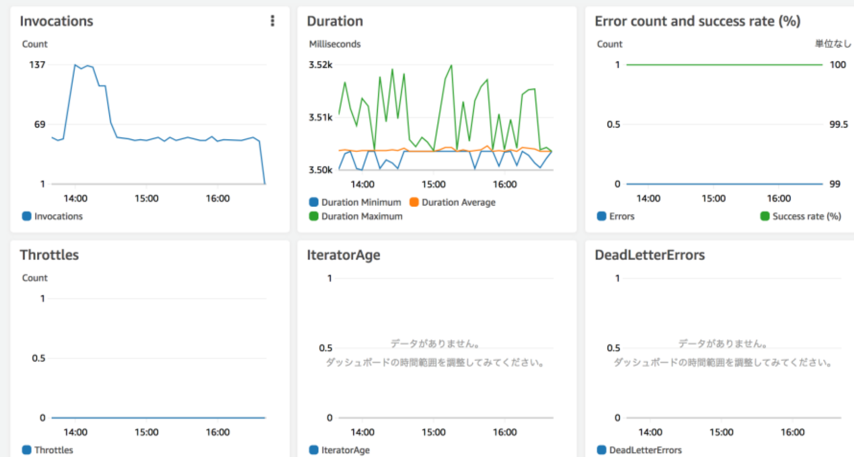
AWS Lambdaが提供するのはあくまでもRuntime APIであり、Runtimeそのものは自身で管理もしくはパートナー管理となる
- ランタイム起因のトラブルに対するサポートは得られない
- パートナーが提供しているものはパートナーからのサポート提供
- 管理対象が増えることをよく考慮すること

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

モニタリング

AWS CloudWatchを用いたMetricsの監視

- Invocations
- Errors
- Duration
- Throttle
- etc…



AWS X-Rayを用いたトラブルシュート

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

aws

メトリクスとその意味

項目	単位	意味
Invocations	count	Lambdaの実行回数。課金対象のカウントと等しくなる。 **≠同時実行数**
Duration	ms	Lambdaの実行時間を計測。（課金は100msで切り上げられる) ※コールドスタートのLambda起動までの時間は含まない
Errors	count	Lambdaが正常終了しなかった回数を計測したもの。スロットリングされたものと内部エラーによるものはカウントされない。
Throttles	count	Throttleの発生回数 アカウントにおける**Lambdaの同時実行数超過が発生した数** ＝足りない数なので制限緩和申請する際はこの値をベースに
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。
ConcurrentExecutions	count	特定の時点における特定の関数の同時実行数
UnreservedConcurrentExecutions	count	同時実行制限を指定していない関数の同時実行数の合計

モニタリングにおける注意事項

AWS Lambdaが保証しているのは呼び出されれば最低一回は実行すること
つまり、呼び出し側が呼び出さない、複数回呼び出すことは発生し得る

この事象はAWS Lambdaでは監視/検知ができないので全体の仕組みで検討する必要がある

対処例

- アプリケーション側での冪等性の確保
- 例えばS3であれば イベント発火用バケットと処理済みバケットを作成することで、イベント発火漏れを検知するなど

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

ストリーム型での監視事項

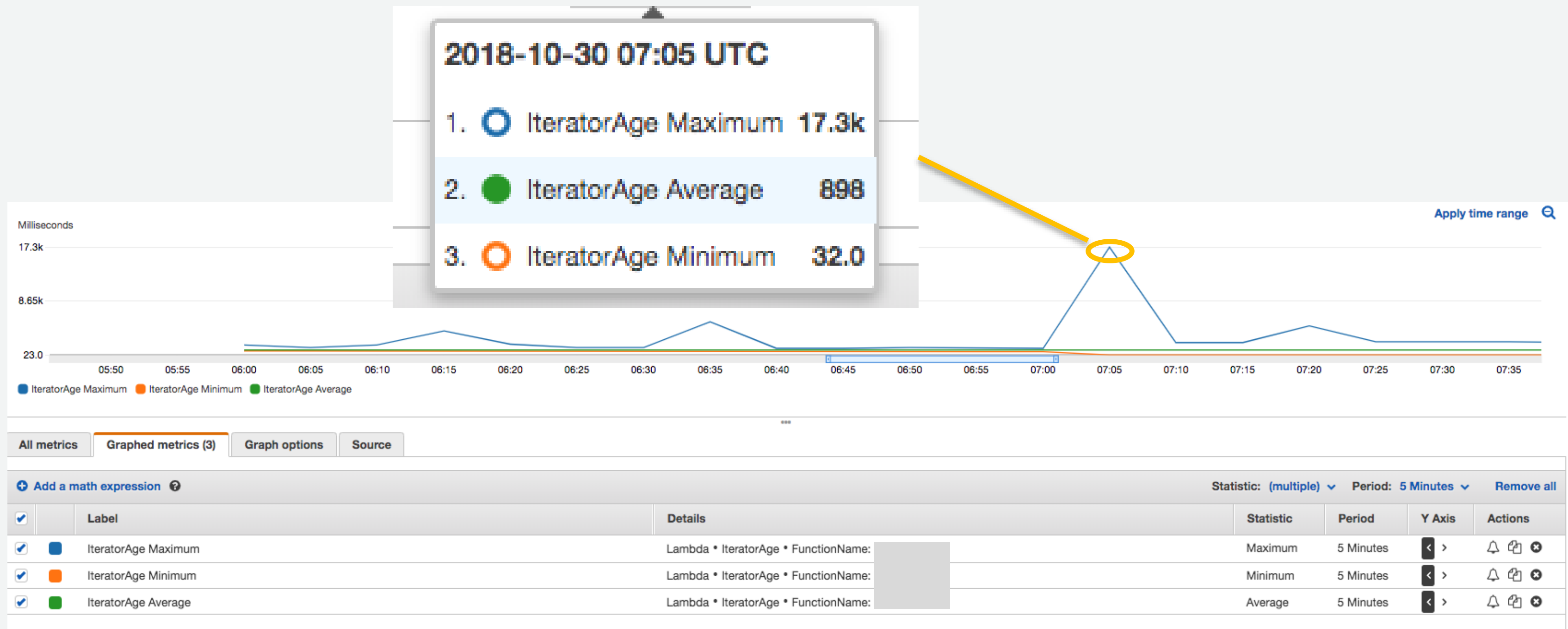
0.3秒間隔でdataをkinesis streamsへ送信し、途中でshard分割を実行したLambdaのモニタ画面例

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

aws

先程のIteratorAgeをCloudwatch metricsでみると

意味を間違うと大違いになるので、表示の意味を理解することが重要



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

ストリームベースでの監視/運用ポイント

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により実行時間の短縮を考える
- 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（shard数＝同時実行数）
  - 東京リージョンの場合、デフォルトでは200shardまで（制限緩和可能）

Error metricsが上がった場合は、処理がロックされている可能性がある。

- Lambdaのプログラム改修
- kinesis streamsの読み込みをlatest/timestampベースで再設定
  - データを読まずに捨てることになるので注意

Streamの場合、Errorを挙げないように設計/テストをすることが重要

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

aws

SQS型起動のLambdaでのmetrics

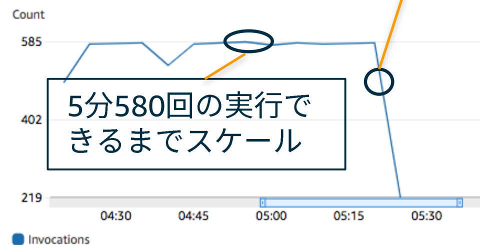
0.5秒間隔でメッセージを送信した場合。metricsは標準の5min

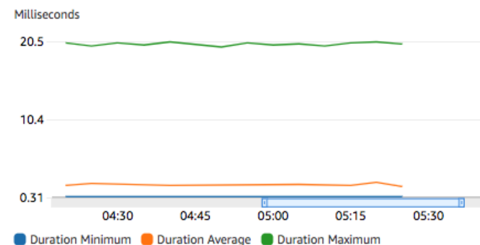
CloudWatch metrics at a glance

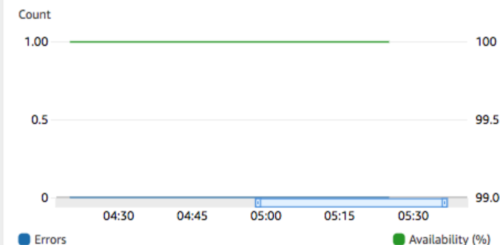
メッセージの送信を止めたことで並行起動のLambdaが減少

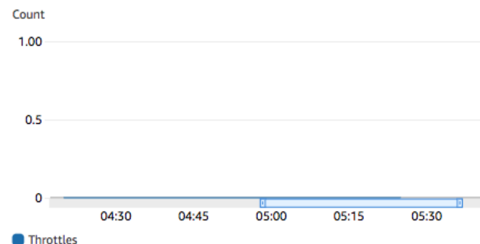
View logs in CloudWatch　View traces in X-Ray

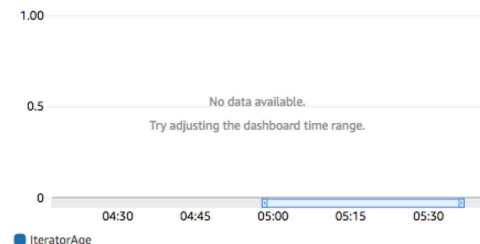
1h 3h 12h 1d 3d 1w custom

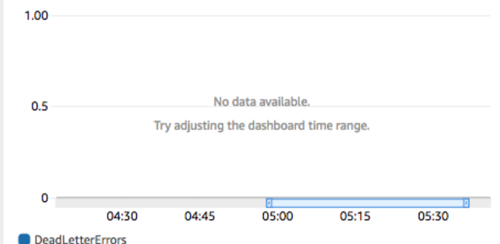












© 2(

LambdaのSQS起動における注意事項

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キューのポーリングを開始、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、同時実行数の上限までスケールする。

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がある

- 他のLambda関数の起動を妨げる可能性がある
- 大規模に使用する際には対象のLambda関数に対して同時実行数の設定を検討する

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

モニタリングで気をつけるポイント

Throttleを観測した場合

- どのLambda関数が大量に使っているかを特定
  - CloudWatchのAll across accountでアカウント全体での使用状況と個別Lambdaの使用状況を確認
- 同時実行数の制限緩和申請を行う

Error metricsの増加

- 権限設定漏れによって実行がすべてエラーになっているパターン
- SQS/ストリームであり得るパターン、バッチサイズとプログラム処理時間とLambdaのタイマ値の関係を確認
  - 1処理時間 x バッチサイズ > タイマ値 となるこれも処理が進まないパターンになる

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

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

aws

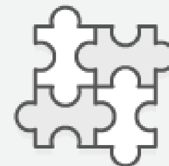
AWS X-Rayを利用したトラブルシューティング

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

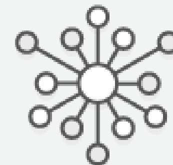


AWSとの連携

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

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

AWS X-Ray コンポーネント

User

Request

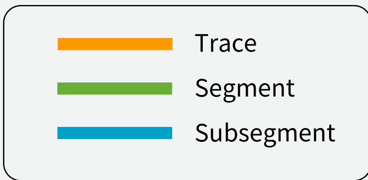
Response

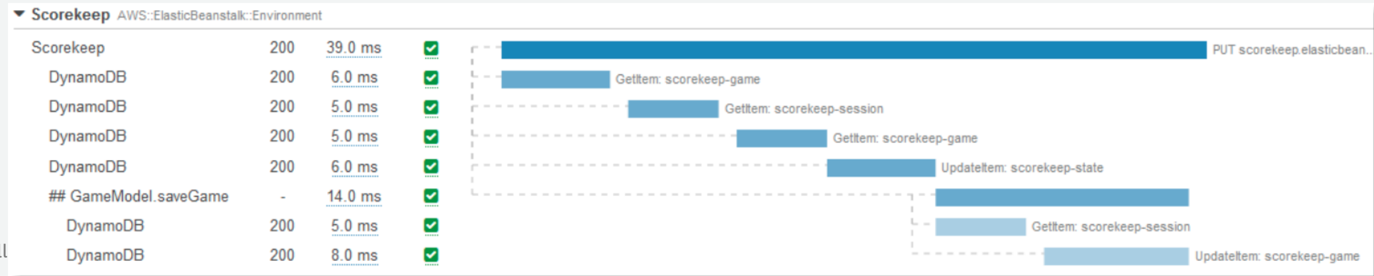
Amazon API Gateway

AWS Lambda

Amazon SQS queue

Amazon DynamoDB





© 2019, Amazon Web Services, Inc. or its Affiliates. All

AWS LambdaでX-Rayを利用するためには

デプロイパッケージにX-Ray SDKを追加し、アクティブトレースをONにする。
実行ロールへの権限追加も必要、Managed policyが用意されている

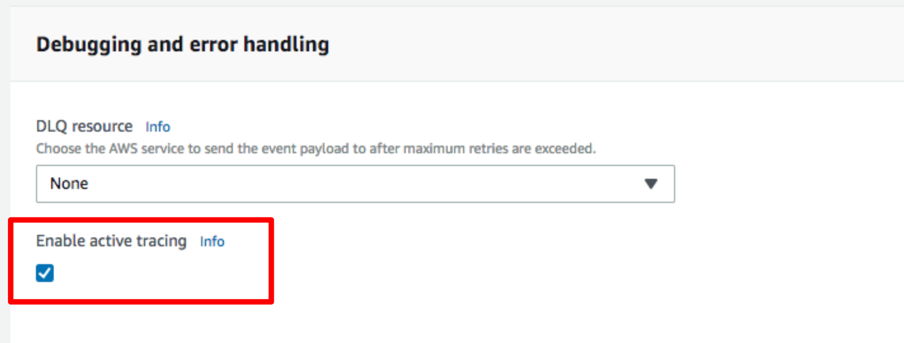
Pythonの場合
- Python 2.7, Python3.6以降

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

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

実装例

```
import boto3
import json
import requests
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
```

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

実装例

```
import boto3
import json
import requests
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
```

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

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

実装例

```
import boto3
import json
import requests
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

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

実装例

```
import boto3
import json
import requests
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
```

処理の中でセグメントブロックを定義することも可能

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

aws

実装例

```
import boto3
import json
import requests
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
```

処理をとじる

© 2019, 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
色で分類し、割合を円グラフに

- 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

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

aws

X-Rayデーモン

Lambda関数をトレースする場合、X-Rayデーモンが自動的に実行され、トレースデータが収集されてX-Rayに送信される

トレース時は、X-Ray デーモンによって最大 16 MB または関数のメモリ割り当ての 3 パーセントを消費

Lambda は関数のメモリ制限を超えないよう、X-Rayデーモンを終了しようとする

- Lambda 関数に128 MB が割り当てられている場合、X-Ray デーモンには 16 MB が割り当てられ、Lambda 関数には112 MBのメモリ割り当てとなる
- 関数が 112 MB を超える場合、メモリ不足エラーがスローされないように X-Ray デーモンは終了される

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

aws

AWS CloudTrailによるログ記録

AWS CloudTrailを用いて、AWS Lambdaのユーザー、ロール、または AWS サービスによって実行されたアクションを記録可能

- AWS Lambda の API コールをイベントとしてキャプチャ
- 各ログエントリには、誰がリクエストを生成したかに関する情報が含まれる

以下のアクションをイベントとして CloudTrailログファイルに記録

AddPermission
CreateEventSourceMapping
CreateFunction
DeleteEventSourceMapping
DeleteFunction
GetEventSourceMapping
GetFunction
GetFunctionConfiguration
GetPolicy
ListEventSourceMappings
ListFunctions
RemovePermission
UpdateEventSourceMapping
UpdateFunctionCode
UpdateFunctionConfiguration

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

デバッグ

```
exports.handler =
  function(event, context)  {
    console.log('Received event:');
    var bucket =
      event.Records[0].s3.bucket.name;
    var key =
      event.Records[0].s3.object.key;
    console.log(‘Bucket: ‘+bucket);
    console.log(‘Key:    ‘+key);
    …
```

実行ログがCloudWatch Logsに出力される

- 各Lambda関数ごとのロググループ

実行開始/終了と消費したリソースに関するデフォルトのログエントリ

- メモリ使用量（Max Memory Used）
- 実行時間（Duration）
- 課金対象時間（Billed Duration）

カスタムログエントリの追加も可能

- 関数でconsole.logなどを用いて出力
- 各言語のログ出力ステートメントも利用可能

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

aws

スケジュール実行

特定時刻または繰り返しによる関数実行をサポート

- 現状は最短で1分インターバル

設定はイベントソースから

- 「CloudWatch Events – Schedule」を選択
- Cron形式の指定もサポート

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

aws サービス リソースグループ sample バージニア北部 サポート

alb-sample-function

スロットリング 限定条件 ▼ アクション ▼ sample ▼ テスト 保存

トリガーの設定

ルール

既存のルールを選択するか、新しいルールを作成します。

新規ルールの作成 ▼

選択または新規ルールを作成

ルール名*

ルールを一意に識別するための名前を入力します。

sample

ルールの説明

オプションでルールの説明を指定します。

ルールタイプ

イベントパターンに基づいて、または自動化されたスケジュールに基づいて、ターゲットをトリガーします。

- イベントパターン
- スケジュール式

スケジュール式*

cron または rate 式を使用して、自動化されたスケジュールに基づいてターゲットを自己トリガーします。cron 式は、協定世界時 (UTC) です。

rate(1 minute)

例: rate(1 day)、cron(0 17 ? * MON-FRI *)

フィードバック 日本語 © 2008 - 2019, Amazon Web Services, Inc. またはその関連会社。無断転用禁止。 プライバシーポリシー 利用規約

スケジュール指定方法

rate(Value Unit)

- インターバル実行する場合の指定方法
- Value： 正の整数を指定
- Unit：分、時、日を指定

例

- 5分ごとに実行 => rate(5 minites)
- 1時間ごとに実行 => rate(1 hour)
- 7日ごとに実行 => rate(7 days)

Valueが複数の場合はUnitも複数形にすること

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

aws

スケジュール指定方法

cron(Minutes Hours Day-of-month Month Day-of-week Year)

- 全フィールドが必須
- タイムゾーンはUTCのみ
- ワイルドカードも利用可

例

- 毎日午前 10:00 実行 => cron(0 10 * * ? *)
- 毎月曜～金曜の午後 06:00に実行 => cron(0 18 ? * MON-FRI *)
- 毎月最初の日の午前 8:00に実行 => cron(0 8 1 * ? *)
- 月曜～金曜の 10 分ごとに実行 => cron(0/10 * ? * MON-FRI *)
- 月曜～金曜の午前 8:00 ～ 午後 5:55の間5分ごとに実行
=> cron(0/5 8-17 ? * MON-FRI *)

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

Cron形式で利用可能なワイルドカード

文字	定義	例
/	増分を指定します	minutes フィールドの0/15は、15分ごとに実行が発生するように指定します。
L	"最後" を指定します	Day-of-month フィールドに使用された場合、その月の末日が指定されます。Day-of-week フィールドに使用された場合、週の最後の曜日 (土曜日) が指定されます。
W	平日を指定します	日付とともに使用した場合（5/W など）、その月の 5 日に最も近い平日が指定されます。5 日が土曜日の場合、実行は金曜日に発生します。5 日が日曜日の場合、実行は月曜日に発生します。
#	その月の n 番目の日を指定します	3#2 は、月の第 2 火曜日を意味します（火曜日は週 7 日の 3 番目の曜日です）。
*	すべての値を指定します	Day-of-month フィールドで使用した場合、月のすべて日を意味します。
?	値を指定しません	指定した別の値とともに使用されます。たとえば、特定の日付を指定したが、その日が何曜日であってもかまわない場合です。
-	範囲を指定します	10-12 は 10、11、および 12 を意味します
,	追加の値を指定します	SUN, MON, TUE は、日曜日、月曜日、および火曜日を意味します
/	増分を指定します	5/10 は、5、15、25、35 などを意味します

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

アプリケーションの管理

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

aws

アプリケーションの管理

詳細は次回（5/15）のPart3で！

管理する対象はLambda関数、イベントソース、その他のリソース（ダウンストリームなど）

管理するためのツールセット

- AWS Serverless Application Repository
- 数クリックでアカウントにデプロイできる Lambda アプリケーションのリポジトリ

AWS CloudFormation

- AWSリソースの定義とプロビジョニング自動化

AWS Serverless Application Model (AWS SAM)

- サーバーレスアプリケーションの定義とパッケージング・デプロイ
- AWS CloudFormationテンプレート言語の拡張
- よりシンプルにサーバーレスアプリケーションを定義可能
- 通常のCloudFomationテンプレートの記述とも混在可能することもできる

AWS CLI および AWS SAM CLI

- コマンドラインツール
- AWS CLI は、デプロイパッケージのアップロードやテンプレートの更新などのタスクを簡素化するコマンドをサポート
- AWS SAM CLI はテンプレートの検証やローカルテストを含む追加機能を提供

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

aws

Any questions?

Photo credit: rolands.lakis via VisualHunt.com / CC BY

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

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