aws DEV DAY
ONLINE JAPAN

20-22.10.2020

DEV DAY

@_kensh

In Partnership with intel

aws DEV DAY
ONLINE JAPAN

20.10.2020

A - 1

今日から始める、サーバーレス Well-Architected Framework

Kensuke Shimokawa

Snr. Serverless Specialist Solutions Architect
Amazon Web Services Japan K.K

@_kensh

In Partnership with intel

自己紹介

Name

Kensuke Shimokawa

Company

Amazon Web Services Japan K.K.

Role

Serverless Specialist Solutions Architect

@_kensh

In Partnership with intel

Related breakouts

10/20 （火曜日）

B-4: 分散システムにおけるSagaパターンのAWS StepFunctionsによる実装

14:15-14:45 | Track B

10/20 （火曜日）

B-5 : Deep Dive on DevOps for Serverless Applications

14:55-15:25 | Track B

10/20 （火曜日）

C-7 :AWS AppSync Advanced Design Pattern

16:15-16:45 | Track C

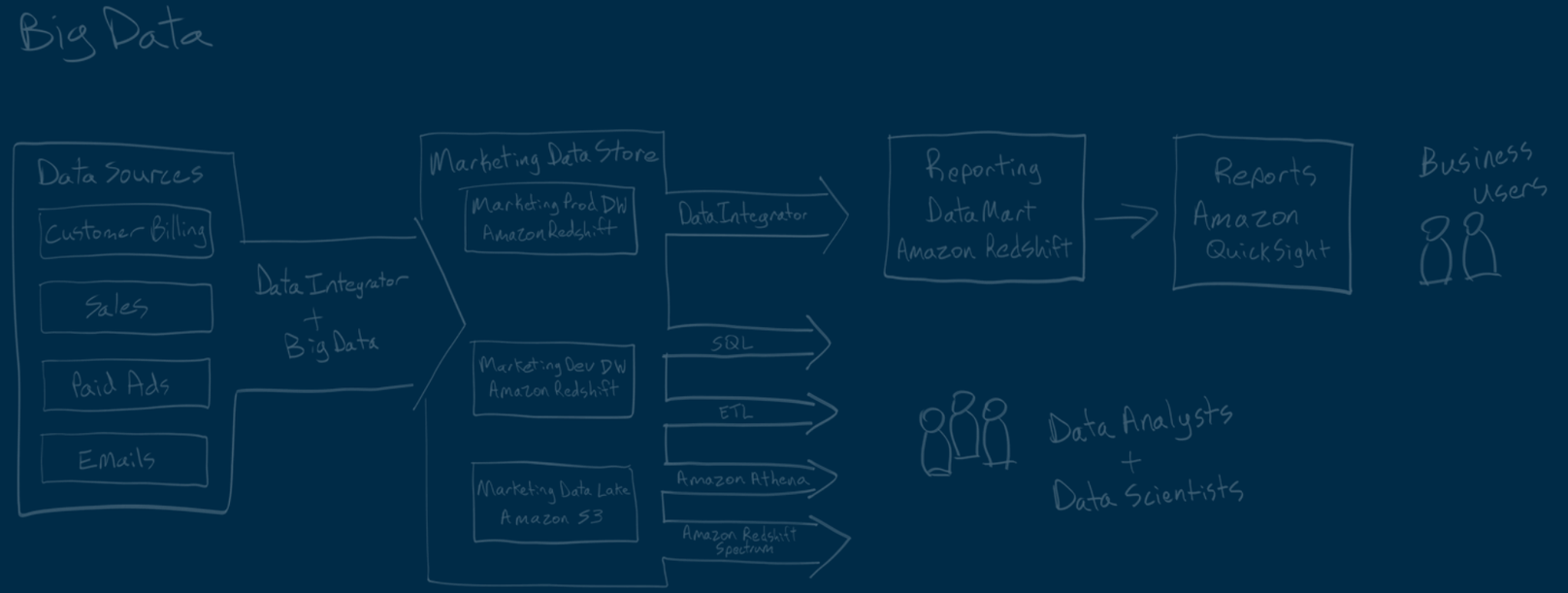
In Partnership with intel

Agenda

- Serverless Application Lens について

- 運用の優秀性
- セキュリティ
- 信頼性
- パフォーマンス効率
- コストの最適化

- まとめ



In Partnership with intel

aws DEV DAY
ONLINE JAPAN

Serverless Application Lens

In Partnership with intel

AWS Well-Architected Framework(W-A)とは?

2015年 AWS re:Invent にて発表

幅広い AWS サービス活用を支援する Well-Architected White Paper の発表

Werner の登壇により 2 日目のキーノートが始まると、例えば Amazon SES のインバウンド、Amazon Kinesis Stream, AWS CloudTrail インテグレーションなどを始め、過去数年を見ても 500 を超えるサービスアップデートがあったことを紹介しました。

また、前日の初日のキーノートで発表されたサービス、Amazon QuickSight、 Amazon Snowball、Amazon Kinesis Firehose, AWS Config Rules、Amazon Inspector、AWS Database Migration Service、AWS Schema Conversion Tool といった、セキュリティアップデートからマイグレーションのアップデートまで幅広い領域でのリリースを改めて紹介しました。

ここで、この日最初の発表となる Well-Architected White Paper が発表されました。これは AWS のソリューションアーキテクトが今までお客様と培ったクラウドアーキテクティングのノウハウをホワイトペーパー化されたもので、クラウドを使うことで自由にシステムを構築できるドキュメントです。

AWS Well-Architected Frameworkホワイトペーパー

設計原則と(質問と回答形式)のベストプラクティス集

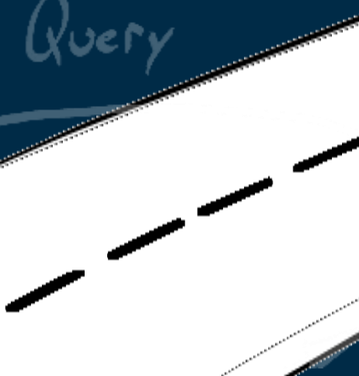
運用の優秀性

セキュリティ

信頼性

パフォーマンス効率

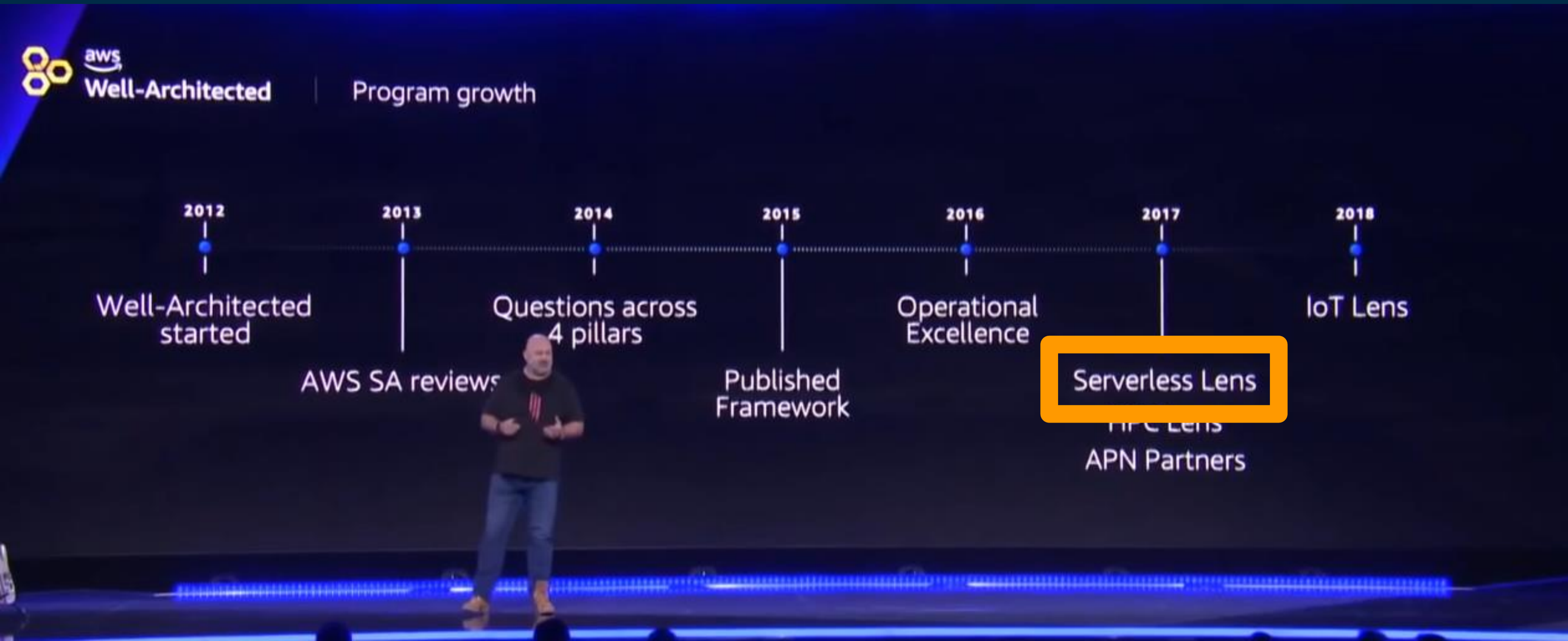
コストの最適化



In Partnership with intel

毎年アップデート

Serverless Lens : 2017年 AWS re:Invent にて発表



Kindle版も無料でダウンロードできます

洋書 › Computers & Technology › Networking

Serverless Applications Lens: AWS Well-Architected Framework (AWS Whitepaper) (English Edition) Kindle版

AWS Whitepapers (著) | 形式: Kindle版

★★★★☆ 24個の評価

› その他 の形式およびエディションを表示する

Kindle版 (電子書籍)
￥0

今すぐお読みいただけます: 無料アプリ

December 2019

This document describes the Serverless Applications Lens for the AWS Well-Architected Framework. The document covers common serverless applications scenarios and identifies key elements to ensure your workloads are architected according to best practices. This documentation is offered for free here as a Kindle book, or you can read it in PDF format at https://aws.amazon.com/whitepapers/.

販売： Amazon Services International, Inc.
紙の本の長さ: 91ページ　タイプセッティングの改善: 有効　Page Flip: 有効
言語: 英語

Kindle 価格: ￥0

注文を確定する

上のボタンを押すとKindleストア利用規約に同意したものとみなされます。

Kindle Cloud Reader

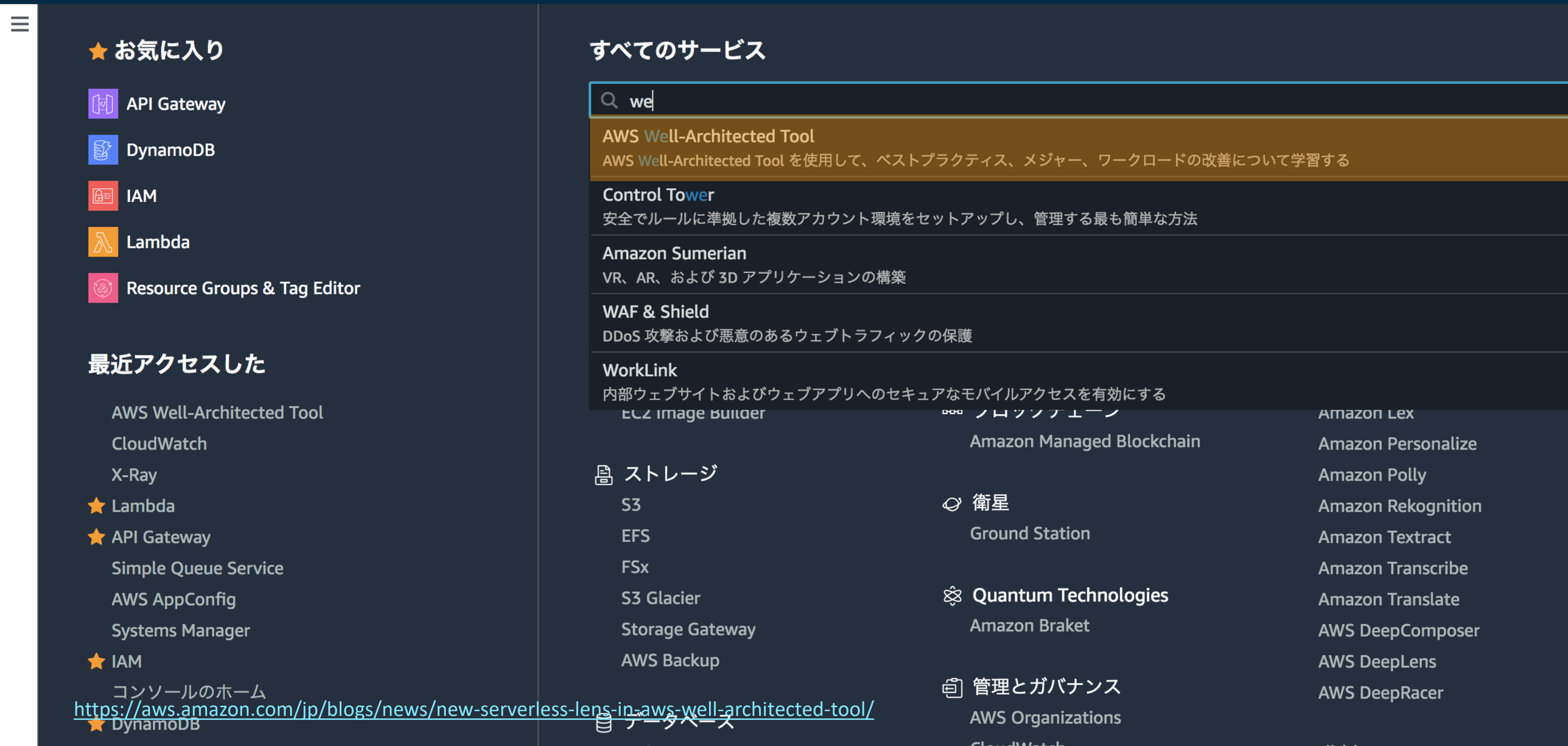
ほしい物リストに追加する

プロモーションコードまたはギフトカードを入力してください

シェアする

In Partnership with intel

AWS Well-Architected Tool



https://aws.amazon.com/jp/blogs/news/new-serverless-lens-in-aws-well-architected-tool/

AWS Serverless Application Lens

Serverless Lens

アーキテクチャ設計にリンクを追加

OPS 1. サーバーレスアプリケーションの状態をどのように評価しますか? 情報

メトリクス、分散トレース、ログ記録の評価を行うと、業務上および運用上の課題に関する知見が得られ、カスタマーエクスペリエンス向上のために最適化すべきサービスについて理解できます。

質問はこのワークロードには該当しません 情報

以下から選択します

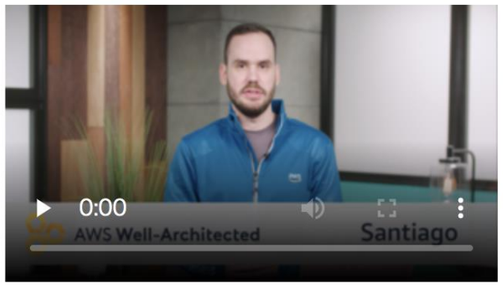
- [ ] 面倒な設定なしで与えられるメトリクスを理解し、分析し、アラートを出す 情報
- [x] 分散トレースを使用し、コードにはコンテキストが追加されている 情報
- [x] 構造化および一元化されたログ記録 情報
- [x] アプリケーション、ビジネス、オペレーションメトリクスの使用 情報
- [ ] いずれも該当しません 情報

メモ - *オプション*

この質問の改善は実行中です。

2084 文字残っています

便利なリソース



Amazon CloudWatch Metrics and Dimensions
AWS Personal Health Dashboard
Amazon CloudWatch Automated Dashboard
AWS Serverless Monitoring Partners
re:Invent 2019 - Production-grade full-stack apps with AWS Amplify

面倒な設定なしで与えられるメトリクスを理解し、分析し、アラートを出す

各マネージドサービスからは、面倒な設定なしでメトリクスが発行されます。比較の基盤として、またパフォーマンスが十分でないコンポーネントとパフォーマンスが過剰なコンポーネントを特定するための基盤として、マネージドサービスごとに主要なメトリクスを定めます。機能エラー、QD (queue depth)、マシンの実行不全状態、反応時間といった項目が主要メトリクスの一例です。

分散トレースを使用し、コードにはコンテキストが追加されている

ステータス、相関識別子、業績といった情報や、ワークロード全体のトランザクションフローを判断するための情報を出すよう、アプリケーションコードを計装します。

運用の優秀性

aws DEV DAY
ONLINE JAPAN

運用の優秀性 OPS1

In Partnership with intel

運用の優秀性

OPS 1. サーバーレスアプリケーションの状態をどのように評価しますか?

✔	面倒な設定なしで与えられるメトリクスを理解し、分析し、アラートを出す
✔	分散トレースを使用し、コードにはコンテキストが追加されている
✔	構造化および一元化されたログ記録
✔	アプリケーション、ビジネス、オペレーションメトリクスの使用

運用の優秀性

OPS 1. サーバーレスアプリケーションの状態をどのように評価しますか?

✔	面倒な設定なしで与えられるメトリクスを理解し、分析し、アラートを出す
✔	分散トレースを使用し、コードにはコンテキストが追加されている
✔	構造化および一元化されたログ記録
✔	アプリケーション、ビジネス、オペレーションメトリクスの使用

Lambda Powertools

AWS Lambda Powertools

https://awslabs.github.io/aws-lambda-powertools-python/

AWS Lambda Powertools Python

Homepage

COLLAPSE ALL

CORE UTILITIES

Tracer

Logger

Metrics

UTILITIES

Middleware factory

Parameters

SQS Batch Processing

Typing

Validation

Event Source Data Classes

Search AWS Lambda Powertools Python /

Homepage

AWS Lambda Powertools Python

A suite of utilities for AWS Lambda functions to ease adopting best practices such as tracing, structured logging, custom metrics, and more.

Looking for a quick run through of the core utilities?

Check out this detailed blog post with a practical example.

Install

Powertools is available in PyPi. You can use your favourite dependency management tool to install it

- poetry: `poetry add aws-lambda-powertools`
- pip: `pip install aws-lambda-powertools`

Homepage

Install

Features

Environment variables

Debug mode

Tenets

Edit on GitHub

対応しているランタイム

python™

aws-samples / aws-lambda-powertools-python

Java™

aws-samples / aws-lambda-powertools-java

In Partnership with intel

構造化ログ

```python
from aws_lambda_powertools import Logger

logger = Logger(service="payment")

@logger.inject_lambda_context
def handler(event, context):
    logger.info("Aha")
```

デコレータで関数Context情報も出力可能

Result

```
{
  ...
  "lambda_function_name":..,
  "lambda_function_memory_size": 128,
  "lambda_function_arn": ”...",
  "lambda_request_id": "52fd..",
  "message": ”Aha",
  "cold_start": true
}
```

Lambda Powertools

コールドスタートログのみを抽出 (CloudWatch Logs Insights)

CloudWatch > CloudWatch Logs > Logs Insights

Switch to the original interface.

Select log group(s)

5m 30m 1h 3h 12h Custom

Clear /aws/lambda/logger-sample-MetricsFunction-1GL0J7DWBVZ97

```
1  fields @timestamp, @message
2  | filter cold_start=1
3  | sort @timestamp desc
4
```

Run query Save History

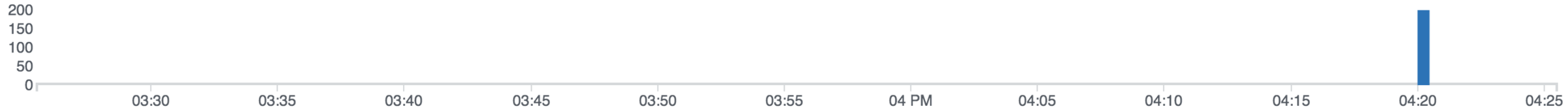
Logs Visualization

Export results Add to dashboard

Showing 200 of 200 records matched

6,678 records (2.2 MB) scanned in 3.1s @ 2,149 records/s (733.5 kB/s)

Hide histogram



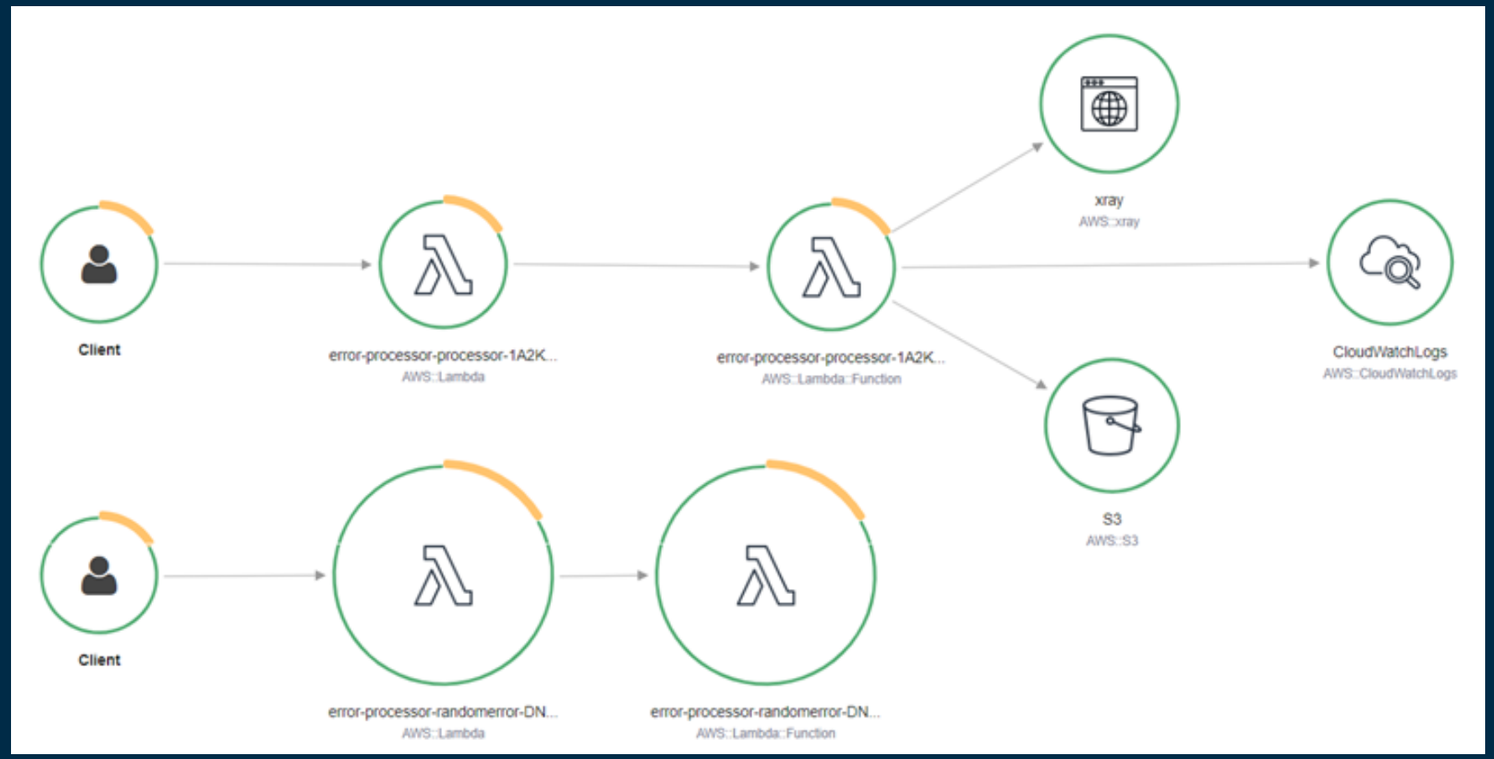
#	@timestamp	@message
▶ 1	2020-10-07T16:20:26.…	{"level": "INFO", "location": "lambda_handler:51", "message": {"operation": "collect_payment", "charge_id": "this is charged"}, "timestamp": "2020…
▶ 2	2020-10-07T16:20:26.…	{"level": "ERROR", "location": "lambda_handler:60", "message": "Received an exception", "timestamp": "2020-10-07 07:20:26,517", "service": "paymen…
▶ 3	2020-10-07T16:20:26.…	{"level": "INFO", "location": "lambda_handler:48", "message": "Collecting payment", "timestamp": "2020-10-07 07:20:26,516", "service": "payment", …

運用の優秀性

OPS 1. サーバーレスアプリケーションの状態をどのように評価しますか?

✔	面倒な設定なしで与えられるメトリクスを理解し、分析し、アラートを出す
✔	分散トレースを使用し、コードにはコンテキストが追加されている
✔	構造化および一元化されたログ記録
✔	アプリケーション、ビジネス、オペレーションメトリクスの使用

分散トレースを使用 (AWS X-Ray)



X-Ray を使用すると、フロントエンド API からバックエンドのストレージとデータベースまで、アプリケーション内のリソースを横断する際にリクエストをトレースできます

In Partnership with intel

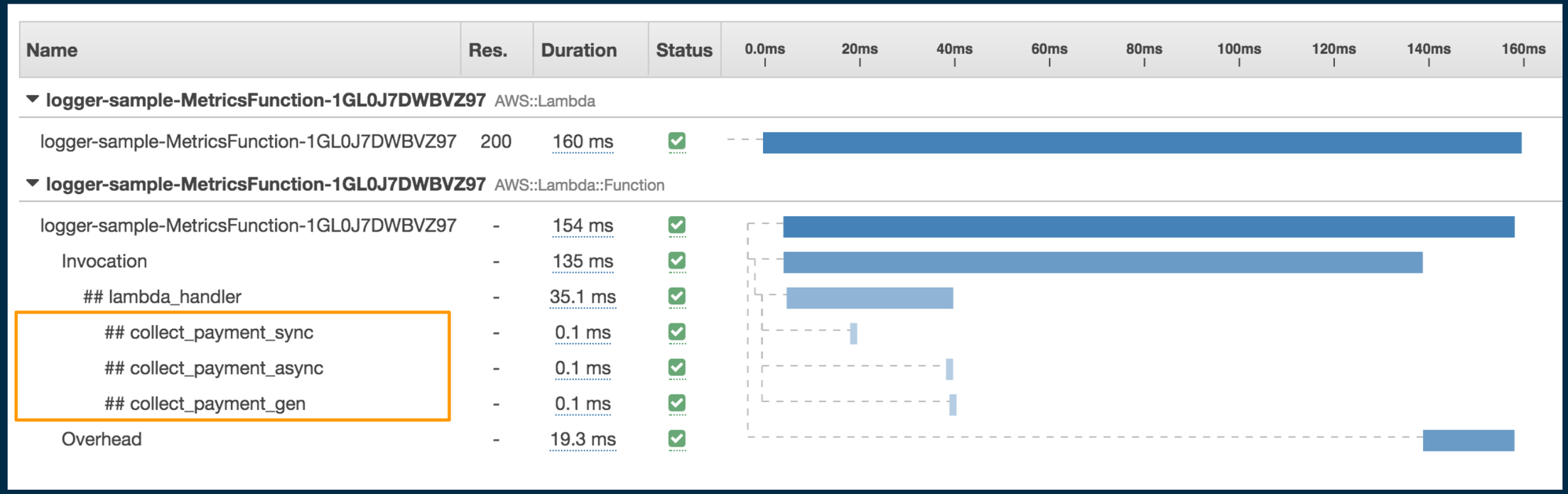
トレース / サービスマップ

```python
from aws_lambda_powertools import Tracer
# POWERTOOLS_SERVICE_NAME defined
tracer = Tracer()

Add Debug Configuration
@tracer.capture_lambda_handler
def lambda_handler(event, context):
    tracer.put_annotation(key="PaymentStatus", value="SUCCESS")
    tracer.put_metadata(key="payment_response", value="some payment response")

    collect_payment_sync(110)

Add Debug Configuration
@tracer.capture_method
def collect_payment_sync(id):
    ret = "this is my collect_payment_sync logic." # logic
    tracer.put_annotation("PAYMENT_STATUS", "SUCCESS") # custom annotation
    return ret
```

デコレータでサブセグメントも定義

トレース / サービスマップ

Result



関数セグメントを拡張して、そのサブセグメントを表示

運用の優秀性

OPS 1. サーバーレスアプリケーションの状態をどのように評価しますか?

✔	面倒な設定なしで与えられるメトリクスを理解し、分析し、アラートを出す
✔	分散トレースを使用し、コードにはコンテキストが追加されている
✔	構造化および一元化されたログ記録
✔	アプリケーション、ビジネス、オペレーションメトリクスの使用

Embedded Metric Format でのメトリクスの記録

```python
from aws_lambda_powertools import Metrics
from aws_lambda_powertools.metrics import MetricUnit

metrics = Metrics(namespace="e-commerce-app", service="shopping-cart")

@metrics.log_metrics(capture_cold_start_metric=True)
def handler(event, context):
    metrics.add_metric(name="CartCheckedOut", unit=MetricUnit.Count, value=1)
    metrics.add_metadata(key="cart_details", value=cart_state)
```

Embedded Metric Format（JSON）でCloudWatch Logsにログデータを送るとカスタムメトリクスが発行されグラフ化される

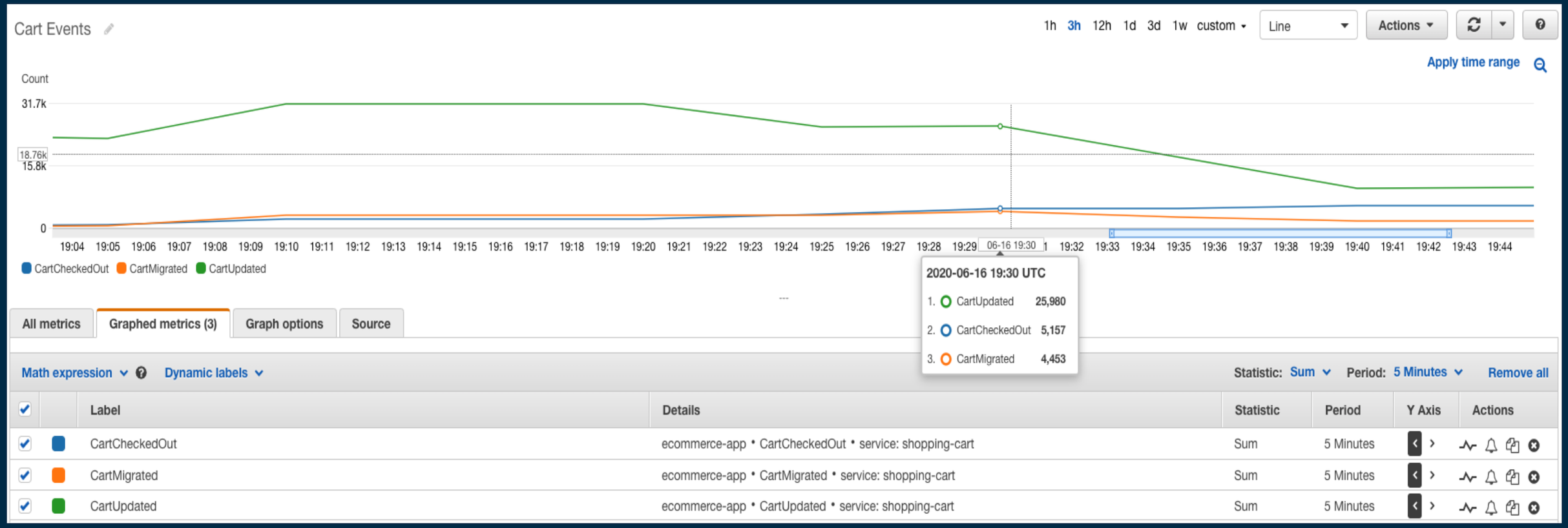
```json
{
    "_aws": {
        "Timestamp": 1594658871,
        "CloudWatchMetrics": [{
                "Namespace": "e-commerce-app",
                "Dimensions": [
                    ["service", "function_name"]
                ],
                "Metrics": [
                    {
                        "Name": "CartCheckedOut",
                        "Unit": "Count"
                    },
                    {
                        "Name": "ColdStart",
                        "Unit": "Count"
                    }
                ]
        }]
    },
    "CartCheckedOut": 1.0,
    "service": "booking",
    "cart_details": {...}
}
```

Embedded Metric Format

カスタムメトリックの出力

Result

ビジネスKPIを可視化！



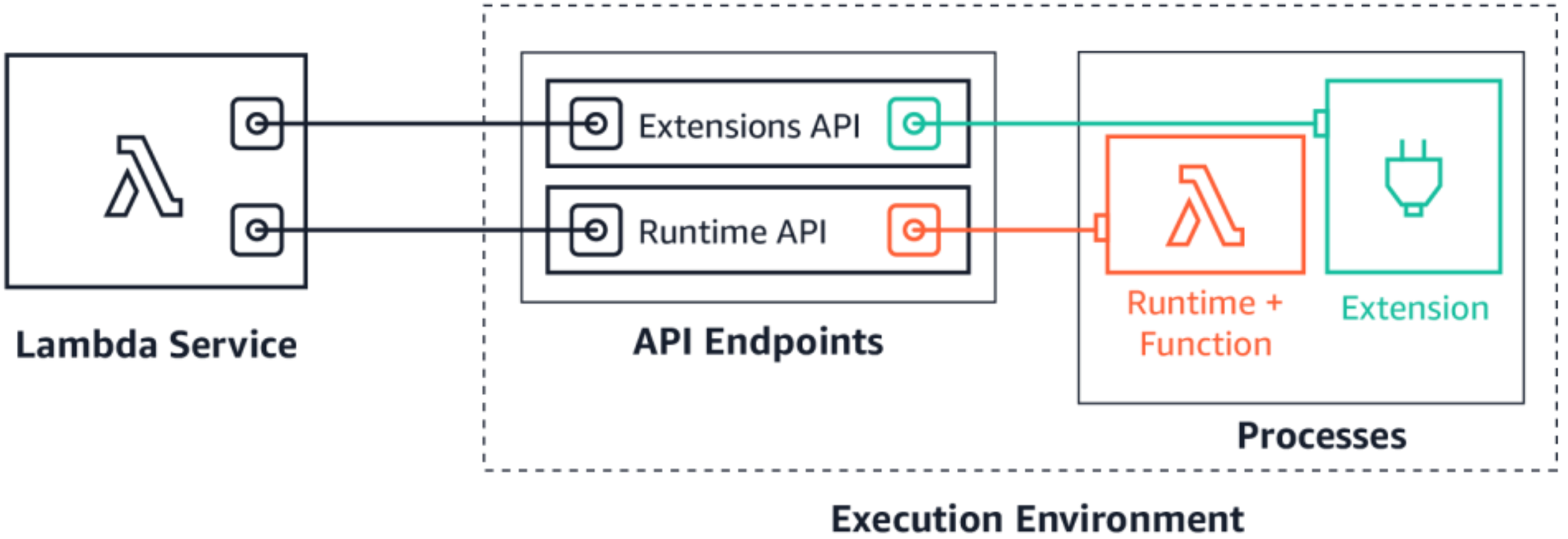
In Partnership with intel

運用の優秀性

OPS 1. サーバーレスアプリケーションの状態をどのように評価しますか?

✔	面倒な設定なしで与えられるメトリクスを理解し、分析し、アラートを出す
✔	分散トレースを使用し、コードにはコンテキストが追加されている
✔	構造化および一元化されたログ記録
✔	アプリケーション、ビジネス、オペレーションメトリクスの使用

AWS Lambda Extensions – In preview



Lambda Extensionsは、Lambdaをお気に入りの監視、セキュリティツールおよびガバナンスツールと簡単に統合する新しい方法です

Lambda Insights

Lambda Insights Performance monitoring

CloudWatch > Lambda Insights > Performance monitoring

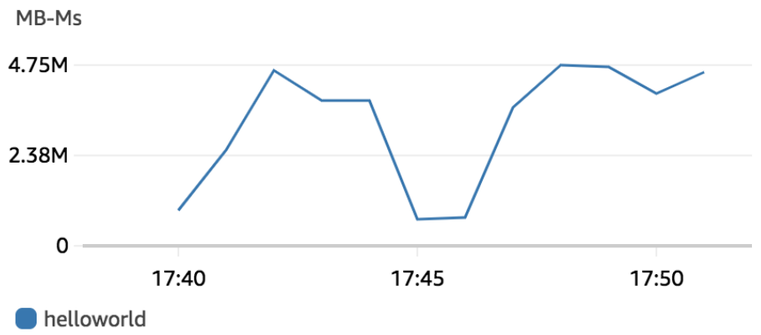
Performance monitoring

1h 3h 12h 1d 3d 1w Custom (15m) Add to dashboard

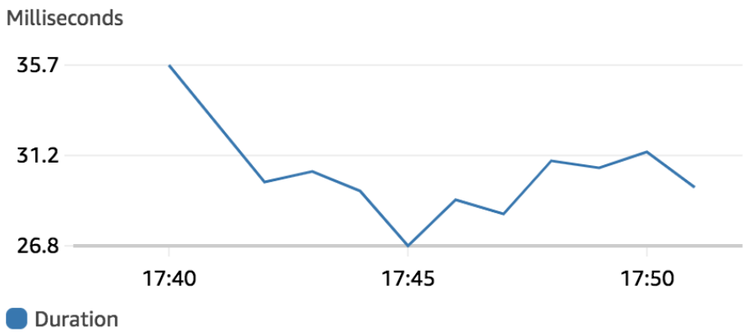
アラーム状態 0 データ不足 1 OK 0

Filter metrics by function name

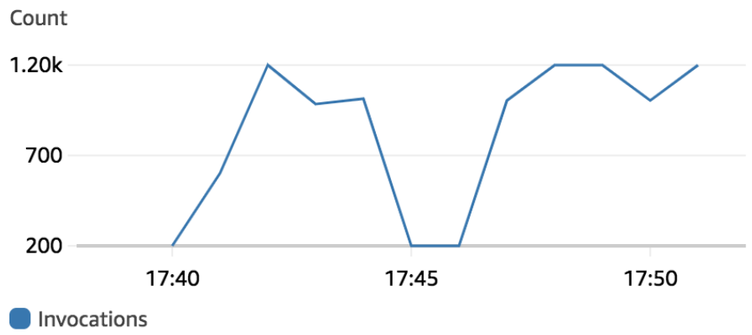
Function Cost



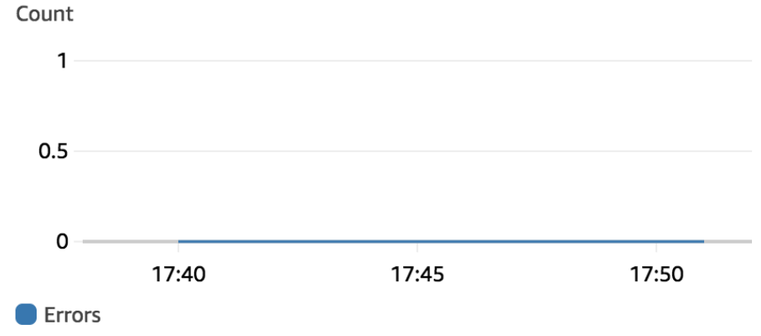
Duration



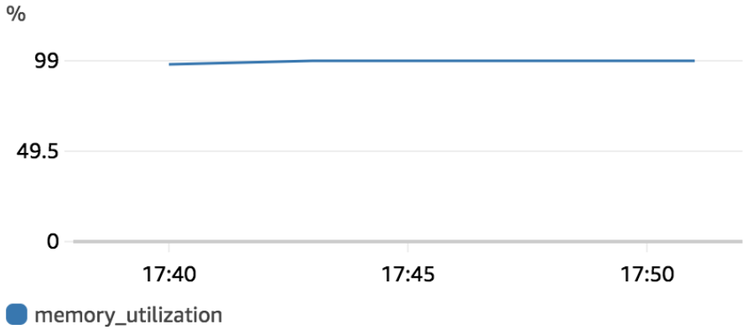
Invocations



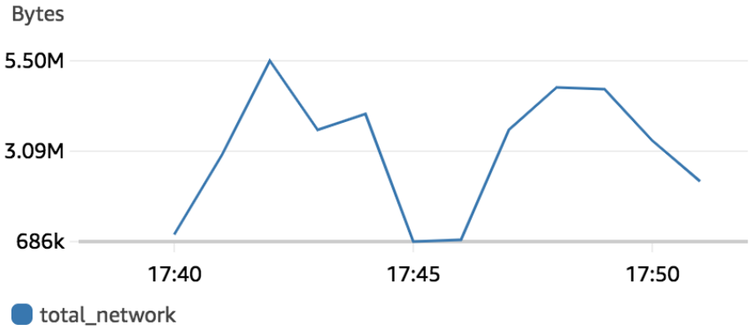
Errors



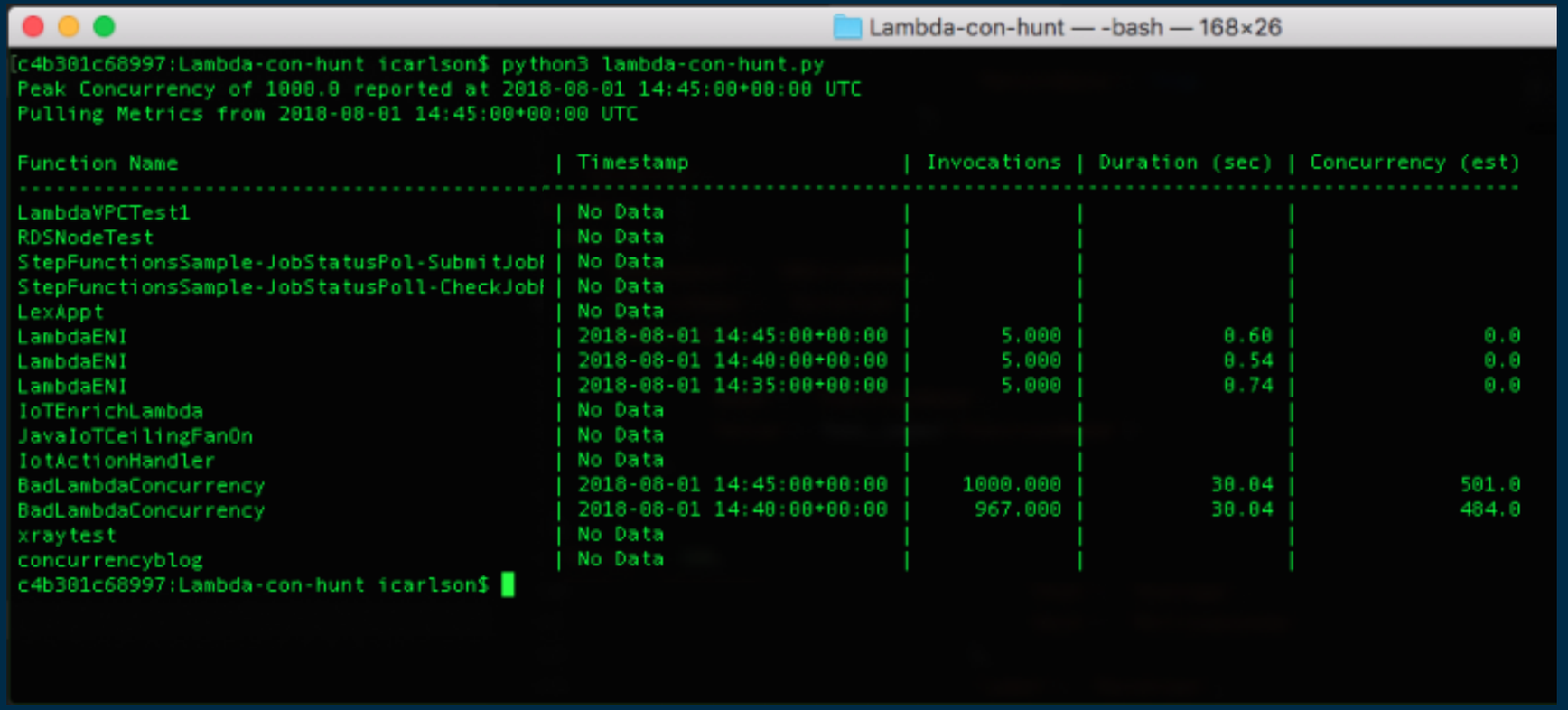
Memory Usage (Max)



Network Usage



Lambda Concurrency Hunt でスパイクを発見



過去7日間のMetricsに対して同時実行数が最も高い期間を見つけ、そのSpikeの前の6分間の情報を出力する。

- 関数の呼び出し回数
- 平均処理時間
- 同時実行数

※ Lambdaの1 分間隔Metricsにおいて、15 日間は1分の分解能を持つ。以降も使用可能だが、5 分に集約された分解能となる
https://aws.amazon.com/jp/cloudwatch/faqs/

$ curl https://raw.githubusercontent.com/aws-samples/aws-lambda-concurrency-hunt/master/lambda-con-hunt.py -o lambda-con-hunt.py
$ python3 lambda-con-hunt.py

In Partnership with intel

aws DEV DAY
ONLINE JAPAN

運用の優秀性 OPS2

In Partnership with intel

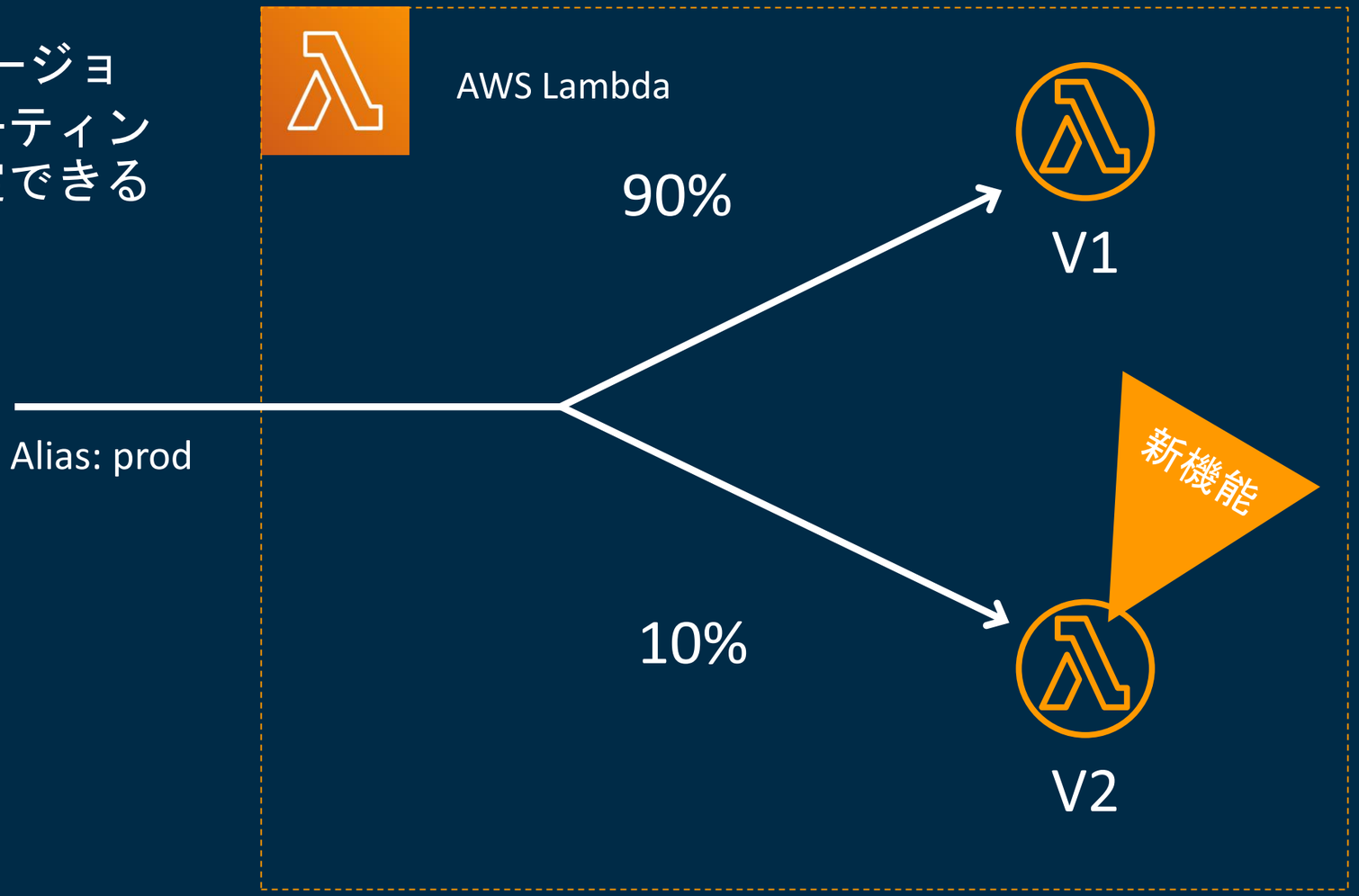
運用の優秀性

OPS 2. アプリケーションのライフサイクル管理について教えて下さい

✔	別々の環境に分離されたコードおよびステージとしてインフラストラクチャを使用する
✔	一時的な環境を使用し、新機能の原型を作る
✔	ロールアウトデプロイメカニズムを使用する
✔	構成管理を使用する
✔	関数ランタイムの廃止ポリシーを確認する

Lambdaのトラフィックシフト機能

Lambda エイリアスに 2 つの関数バージョンを指定して、各バージョンにルーティングされるトラフィックの割合を設定できる



In Partnership with intel

Lambdaのトラフィックシフト機能

AWS SAM を使用してLambda関数を更新する場合、CodeDeploy が組み込まれているため、Lambda関数を線形にデプロイ可能

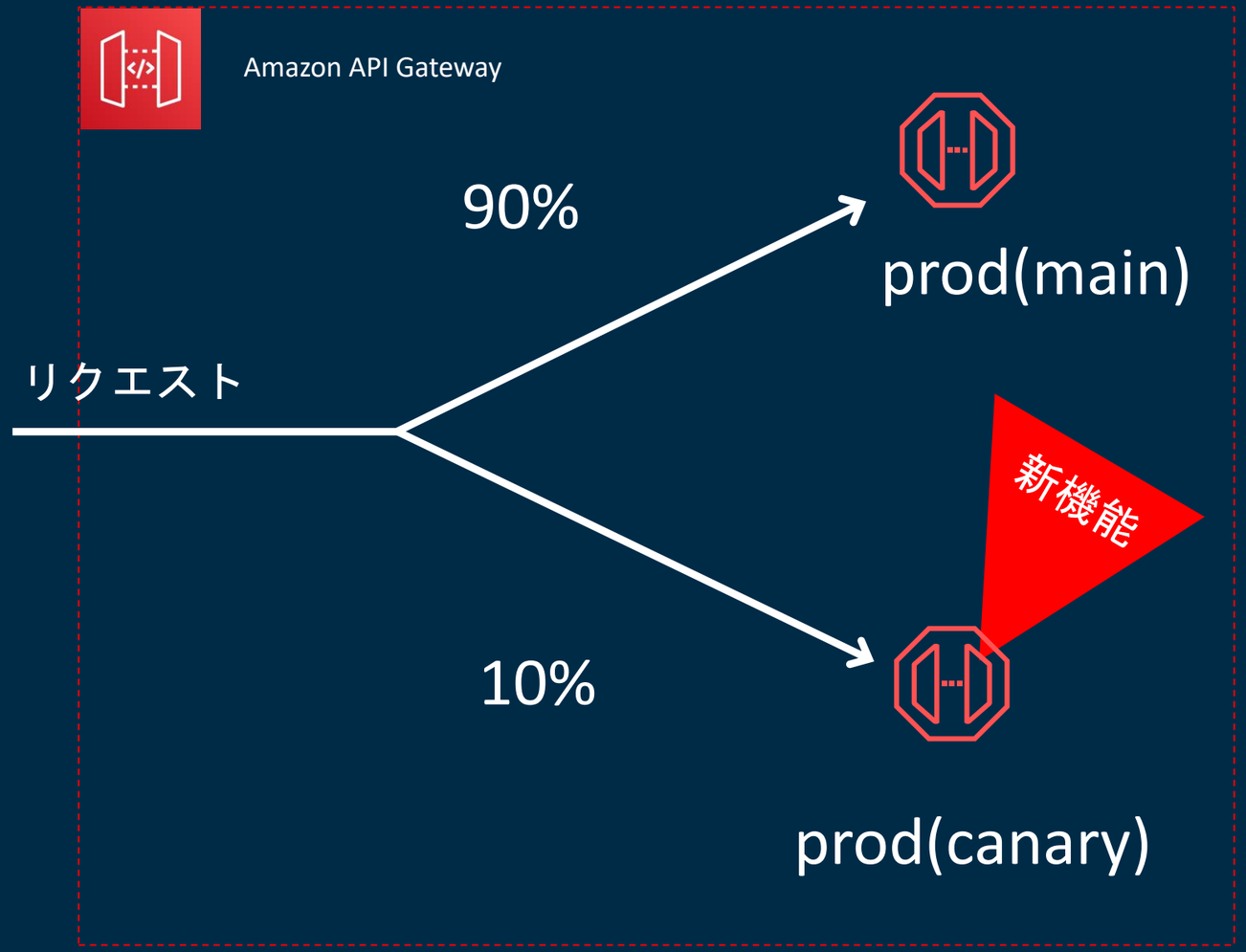
```
HelloFunction:
  Type: AWS::Serverless::Function
  Properties:
    Runtime: python3.8
    Handler: index.handler
    AutoPublishAlias: !Ref Stage
    DeploymentPreference:
      Enabled: true
      Type: Linear10PercentEvery1Minute
```

https://docs.aws.amazon.com/ja_jp/serverless-application-model/latest/developerguide/automating-updates-to-serverless-apps.html

API Gatewayのカナリアリリース

ステージに対して「Canary」を定義すると、APIの「デプロイ」操作でそのステージを指定した際に一旦Canary に対してのみデプロイされるようになる(メインの内容は変わらない)

- Canaryに対して以下の操作が可能：
  - 昇格
    - Canaryのデプロイ内容をメインのステージに反映
  - 削除
    - Canaryを削除して、メインステージのみの構成に戻す



In Partnership with intel

API Gatewayのカナリアリリース

AWS SAM を使用してAPI Gateway REST APIを更新デプロイする場合は、 カナリアの設定を組込めるため、ステージに安全にデプロイ可能

```
HelloWorldApi:
  Type: AWS::Serverless::Api
  Properties:
    StageName: !Ref StageName
    CanarySetting:
      PercentTraffic: 10
      UseStageCache: false
```

https://docs.aws.amazon.com/ja_jp/apigateway/latest/developerguide/canary-release.html

運用の優秀性

OPS 2. アプリケーションのライフサイクル管理について教えて下さい

✔	別々の環境に分離されたコードおよびステージとしてインフラストラクチャを使用する
✔	一時的な環境を使用し、新機能の原型を作る
✔	ロールアウトデプロイメカニズムを使用する
✔	構成管理を使用する
✔	関数ランタイムの廃止ポリシーを確認する

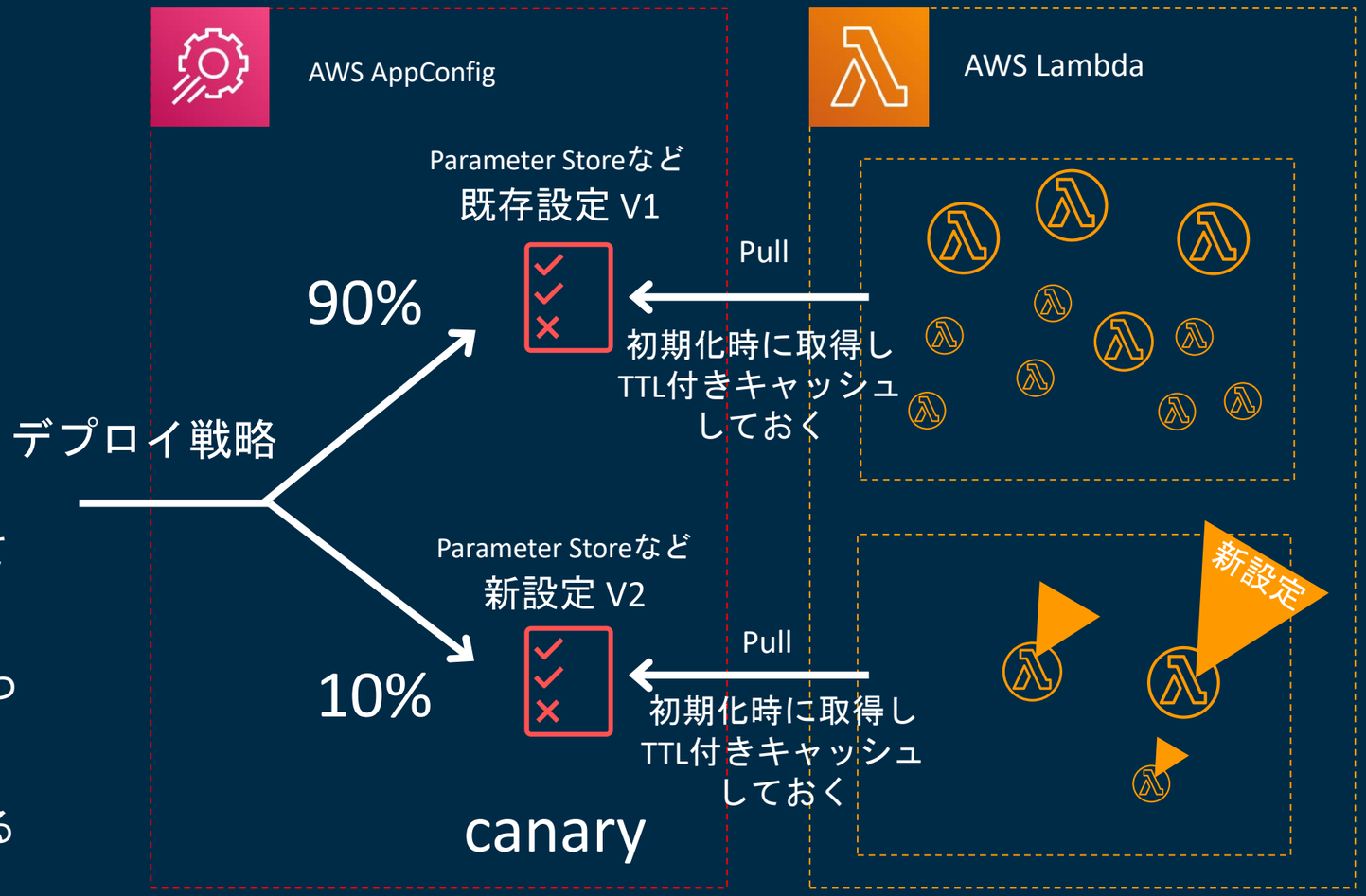
AWS AppConfig によるロジックと設定の分離

- ベータ環境や本番環境などの論理的なデプロイグループを管理可能

- 論理環境ごとに Amazon CloudWatch アラームを設定し AppConfig によってモニタリングすることが可能で、デプロイ中にアラームが発生するとロールバックをトリガー (設定値のValidationも可能)

- アプリケーションからは AppConfig の GetConfiguration APIでパラメータを取得できるため、関数インスタンスごとにユニークなClientIDを振っておけば自動的に戦略に沿ったカナリアが実現

- 設定用と関数本体用にCICDパイプラインを分離し2つ用意する
  - ここでの設定とはLambda サービスの設定ではなく、アプリケーション内のロジックにおける設定のこと （例. フィーチャーフラグ on/off）



https://aws.amazon.com/jp/blogs/news/safe-deployment-of-application-configuration-settings-with-aws-appconfig/

In Partnership with intel


セキュリティ

aws DEV DAY
ONLINE JAPAN

セキュリティ SEC1

In Partnership with intel

セキュリティ

SEC 1. サーバーレス API へのアクセスをどのようにコントロールしますか?

✔	適切なエンドポイントタイプとメカニズムを使用して、API へのアクセスを保護する
✔	認証および承認のメカニズムを使用する
✔	ID のメタデータに基づいてアクセスの範囲を設定する

Amazon API Gatewayのエンドポイントタイプ

- REST APIの場合３種類のエンドポイントタイプを指定可能
- これらのタイプによってクライアントからのアクセス特性を決定する

インターネット公開

インターネット非公開

エッジ最適化

- エッジロケーションにルーティング(Amazon CloudFrontディストリビューション)

リージョン

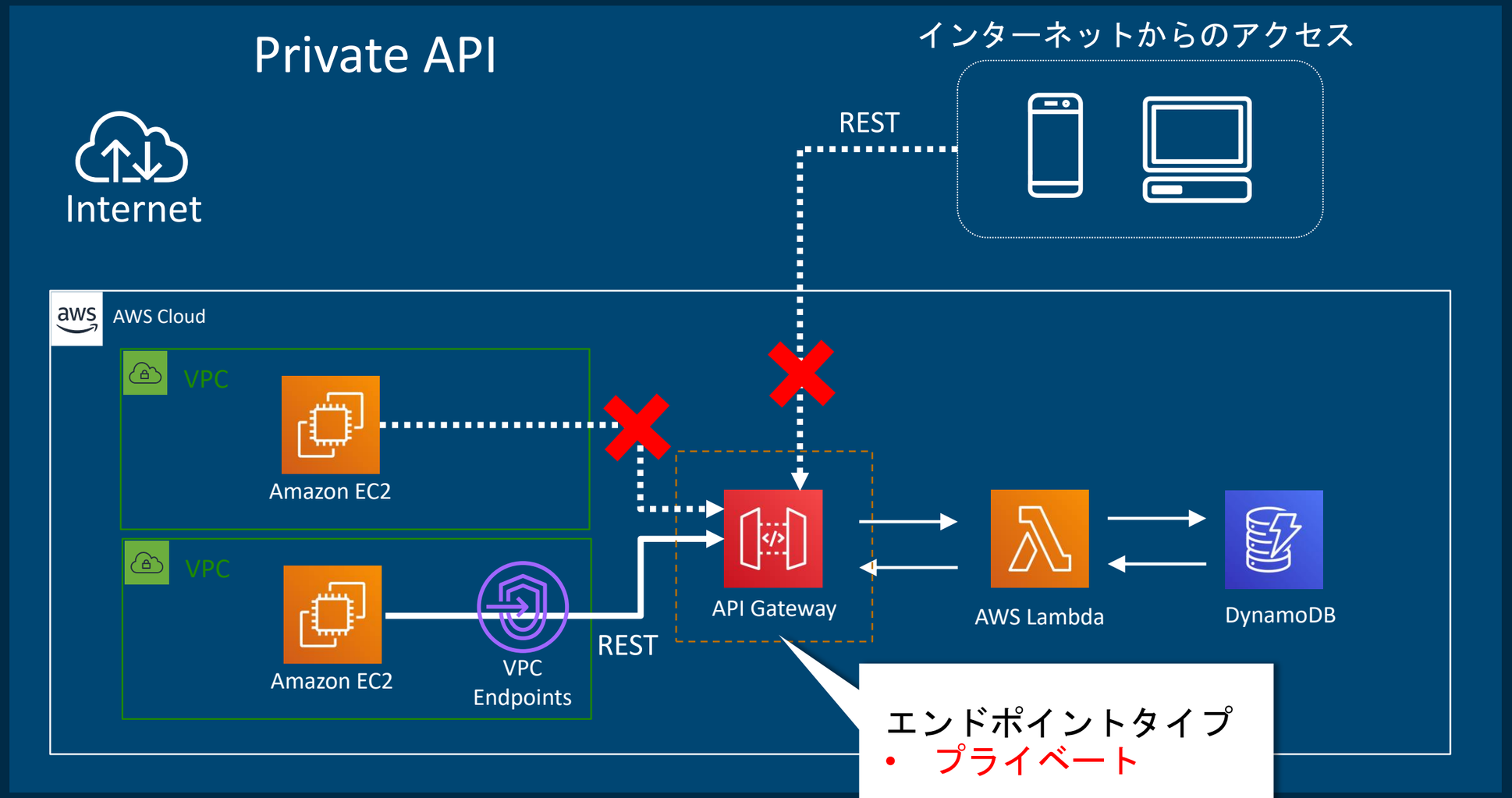
- リージョンに直接ルーティング
- 同一リージョンの場合レイテンシ削減

プライベート

- インターネットからのアクセスはできず、VPC内からVPCエンドポイント(PrivateLink)経由でのみアクセス可能

プライベートAPI

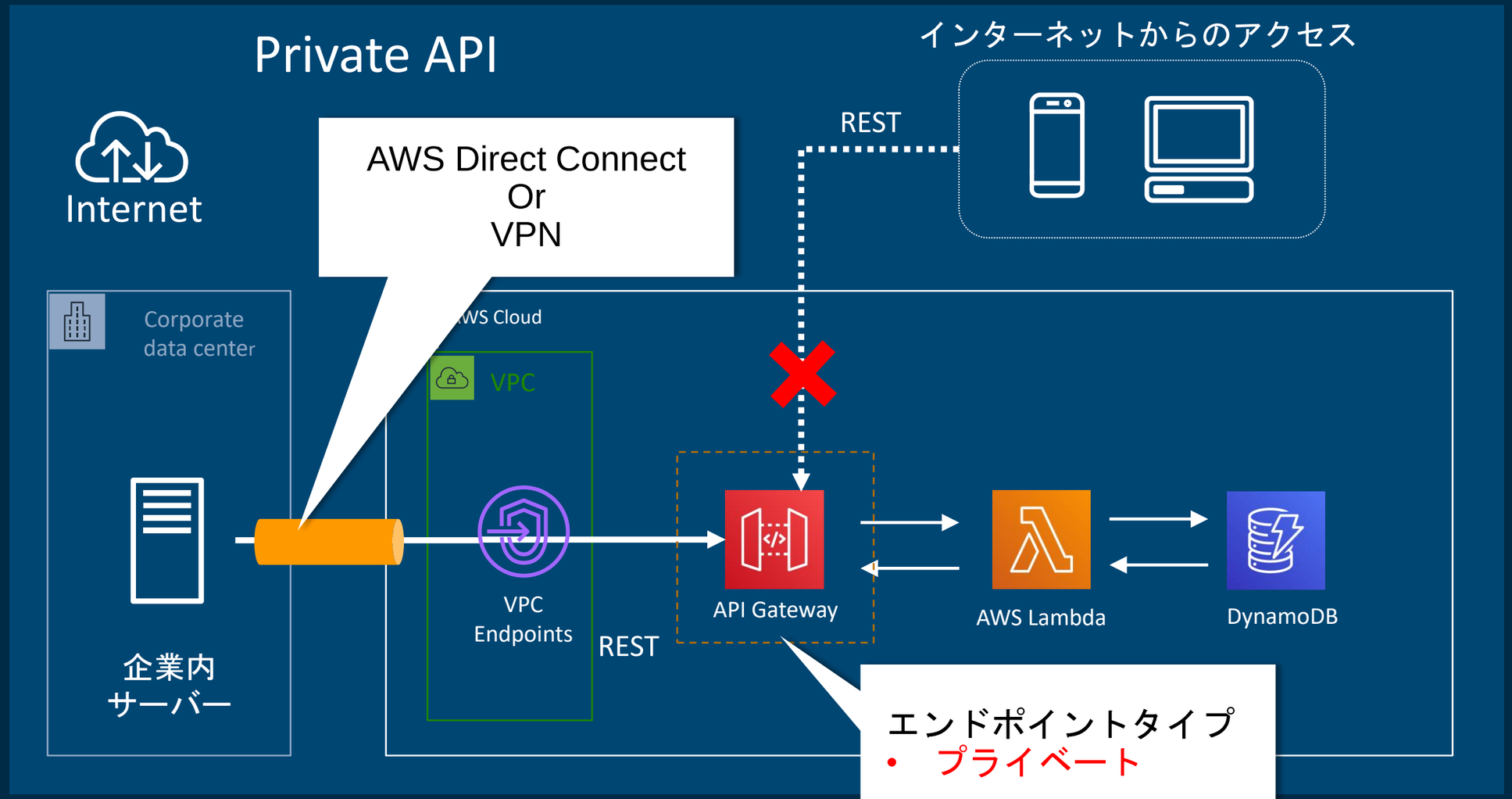
- VPC内に、VPCエンドポイントを作成
- API Gatewayのエンドポイントタイプをプライベートに設定しておく必要があります
- インターネットからのアクセスは不可
- VPCエンドポイントを作成していない他のVPCからのアクセスは不可



In Partnership with intel

オンプレミスからのアクセス（DX or VPN）

- あらかじめ、オンプレミスからAWS VPC への接続を確立しておく。(AWS Direct ConnectまたはVPN)
- インターネットからのアクセスは不可
- Route 53 Resolverを使用して名前解決可能



In Partnership with intel

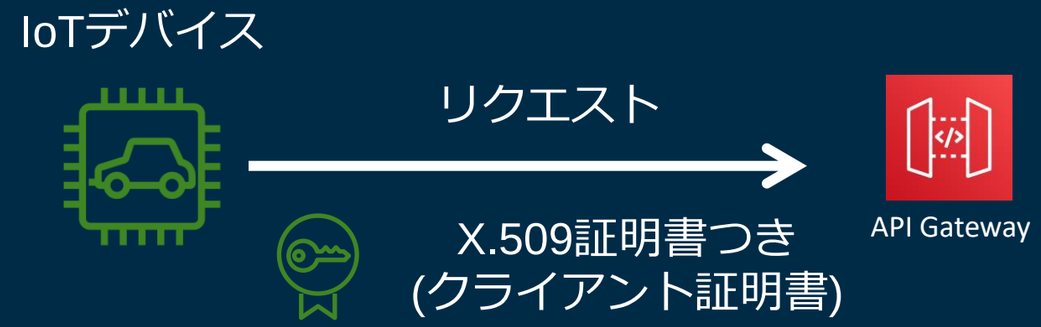
セキュリティ

SEC 1. サーバーレス API へのアクセスをどのようにコントロールしますか?

✔	適切なエンドポイントタイプとメカニズムを使用して、API へのアクセスを保護する
✔	認証および承認のメカニズムを使用する
✔	ID のメタデータに基づいてアクセスの範囲を設定する

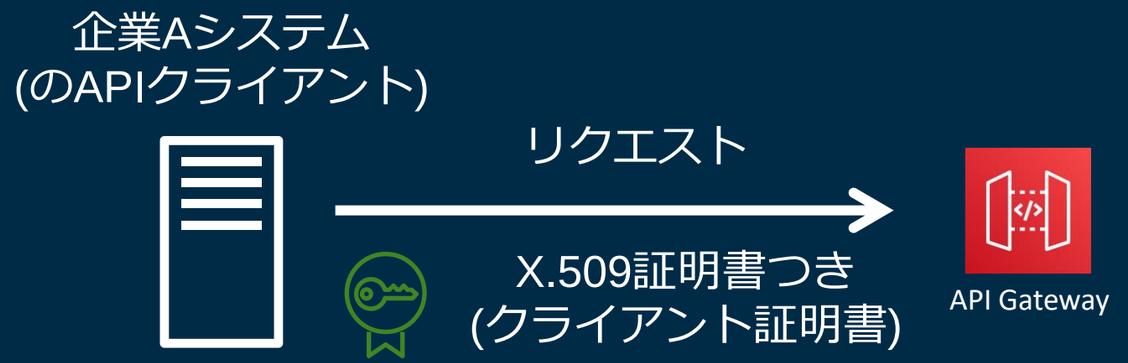
API GatewayのMutual TLS (mTLS) 認証

ユースケース例：IoTデバイス⇔API Gateway通信



事前にクライアント証明書を導入したデバイスからのアクセスのみに制限したい

ユースケース例：B2B API通信



事前にビジネス上合意した企業システムからのアクセスのみに制限したい

In Partnership with intel

aws DEV DAY
ONLINE JAPAN

セキュリティ SEC2

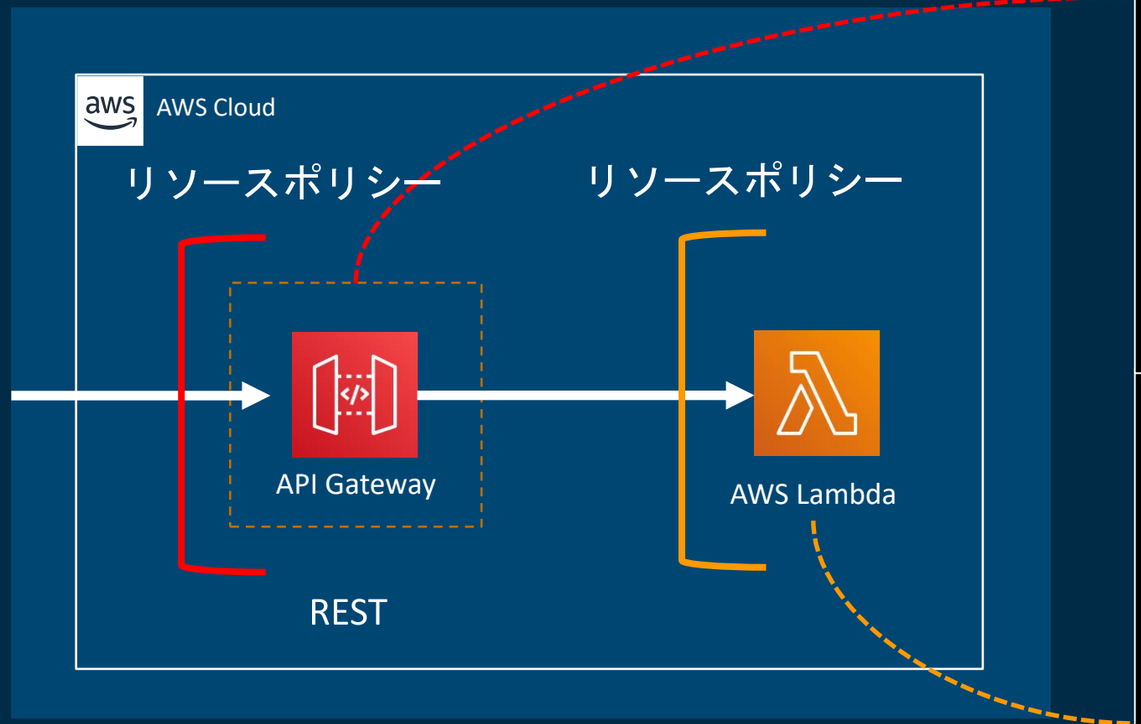
In Partnership with intel

セキュリティ

SEC 2. サーバーレスアプリケーションのセキュリティ境界をどのように管理していますか?

✔	リソースポリシーの評価と定義
✔	リソースとコンポーネント間で一時的な認証情報を使用する
✔	すべてのレイヤーでネットワークトラフィックを制御する
✔	コンパクトなデザイン、単一用途の関数

リソースポリシーの評価と定義




```
{
 "Version": "2012-10-17",
 "Statement": [{
  "Effect": "Allow",
  "Principal": "*",
  "Action": "execute-api:Invoke",
  "Resource": "execute-api:/*/*/*"
 },{
  "Effect": "Deny",
  "Principal": "*",
  "Action": "execute-api:Invoke",
  "Resource": "execute-api:/*/*/*",
  "Condition": {
   "NotIpAddress": {"aws:SourceIp": ["sourceIpOrCIDRBlock”]}
  }
 }]
}
```

```
{
 "Version": "2012-10-17",
 "Id": "default",
 "Statement": [{
  "Sid": ”123456c7-a198-4b2d-b5fc-57fb5fefa0e8",
  "Effect": "Allow",
  "Principal": {"Service": "apigateway.amazonaws.com"},
  "Action": "lambda:InvokeFunction",
  "Resource": ”LambdaFunctionARN",
  "Condition": {
   "ArnLike": {
    "AWS:SourceArn": ”sourceServiceARN"
   }
  }
 }]
}
```

セキュリティ

SEC 2. サーバーレスアプリケーションのセキュリティ境界をどのように管理していますか?

✔	リソースポリシーの評価と定義
✔	リソースとコンポーネント間で一時的な認証情報を使用する
✔	すべてのレイヤーでネットワークトラフィックを制御する
✔	コンパクトなデザイン、単一用途の関数

VPC Lambdaについて

VPCリソースへのアクセスに、Lambda関数をVPC接続設定する必要がある

VPC Lambdaはデフォルトでインターネットアクセスがない

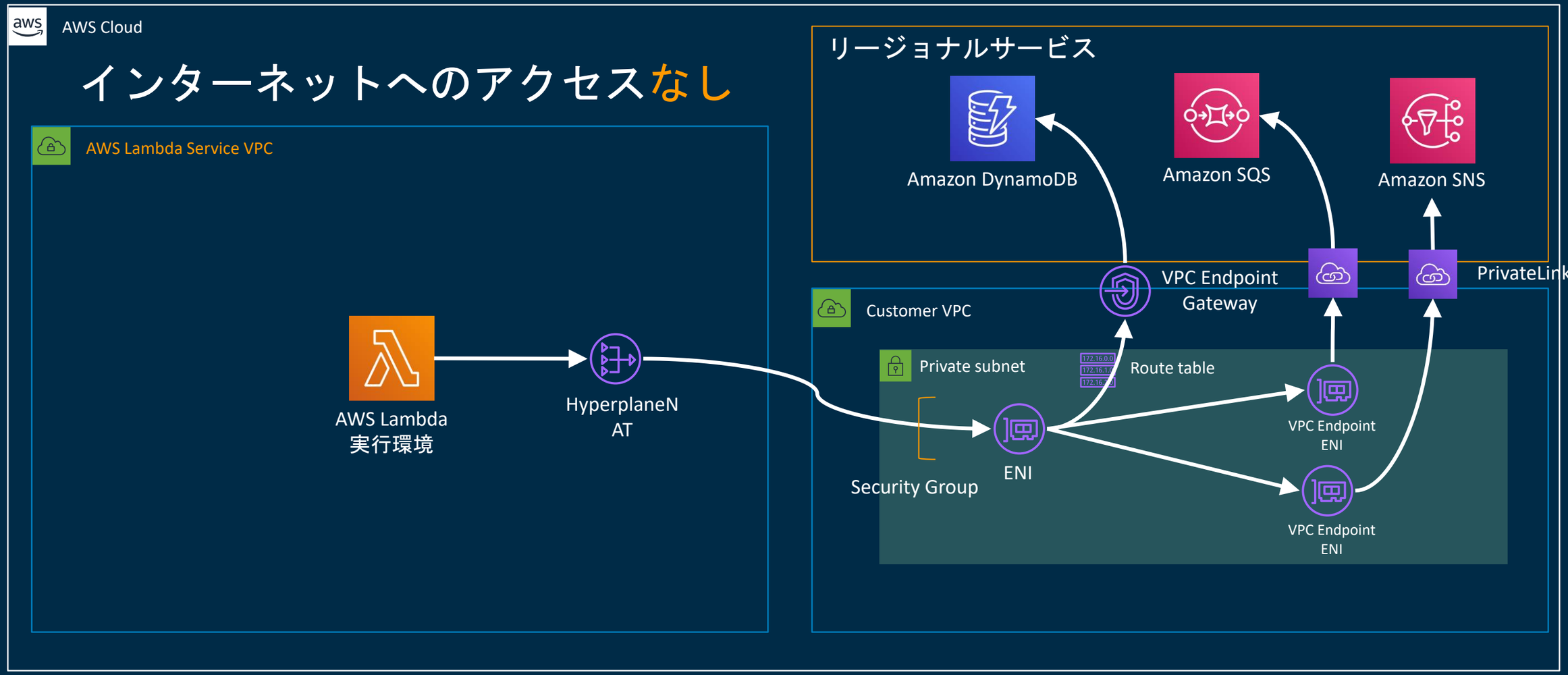
- インターネットアクセスするためには
  - Public SubnetにNAT Gatewayを設置
  - Internet GatewayをVPCに設置
  - Route Tableでインターネットへの経路を追加

今回、着目するのはこの性質

VPC Lambdaはデフォルトでリージョナルサービスへのアクセスがない

- リージョナルサービスへアクセスするには (以下のいずれかが必要)
  - インターネットへの経路を確保
  - サービスごとのVPC Endpointを利用

ネットワークトラフィックを制御する



In Partnership with intel

VPC Lambda利用方法の発想を変えてみる

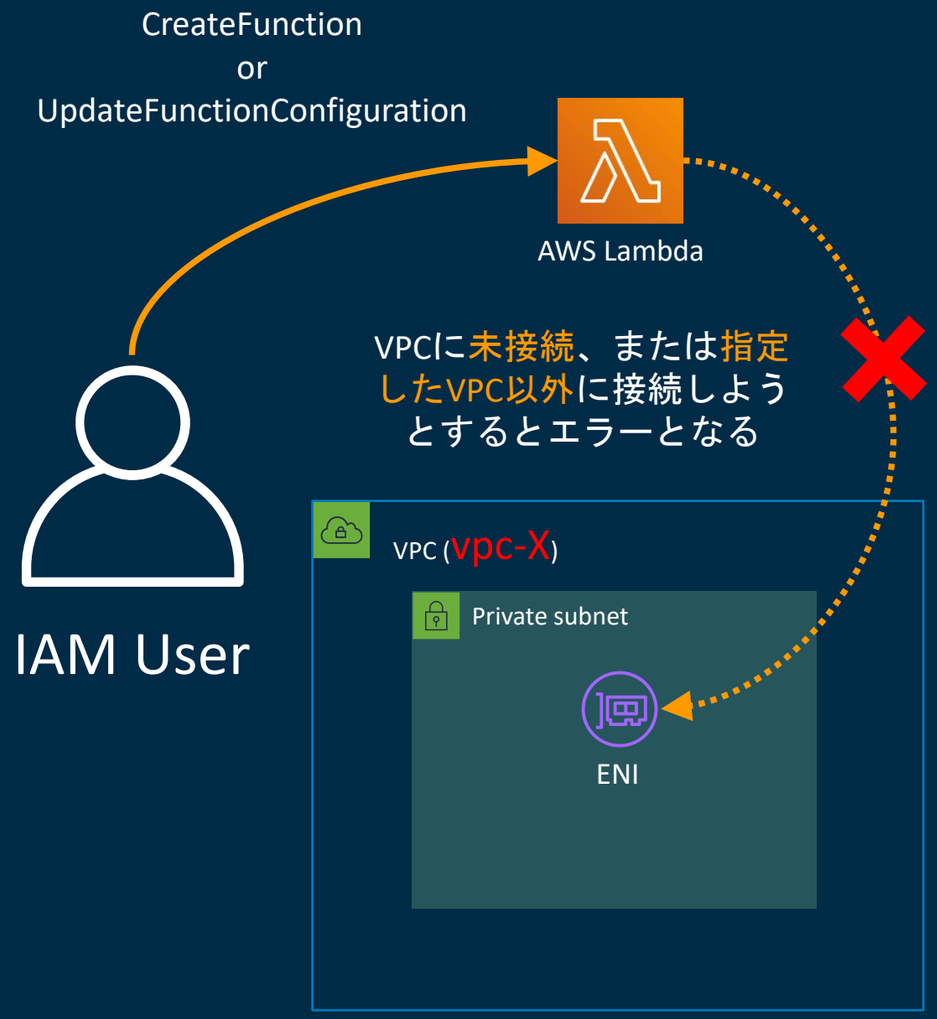
VPC リソースにアクセスするから、VPC Lambdaとして設定する

VPC リソースにアクセスしなくても、VPC Lambdaとして設定する

VPC LambdaとVPCエンドポイントの利用でインターネットから分離

VPC接続を強制し、他VPCにも接続不可とする制御

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Sid": "Stmt159186333251",
   "Action": [
    "lambda:CreateFunction",
    "lambda:UpdateFunctionConfiguration"
   ],
   "Effect": "Deny",
   "Resource": "*",
   "Condition": {
    "ForAllValues:StringNotEquals": {
     "lambda:VpcIds": [
      "vpc-0eebf3d0fe63a2db1"
     ]
    }
   }
  }
 ]
}
```



In Partnership with intel

aws DEV DAY ONLINE JAPAN

セキュリティ SEC3

In Partnership with intel

セキュリティ

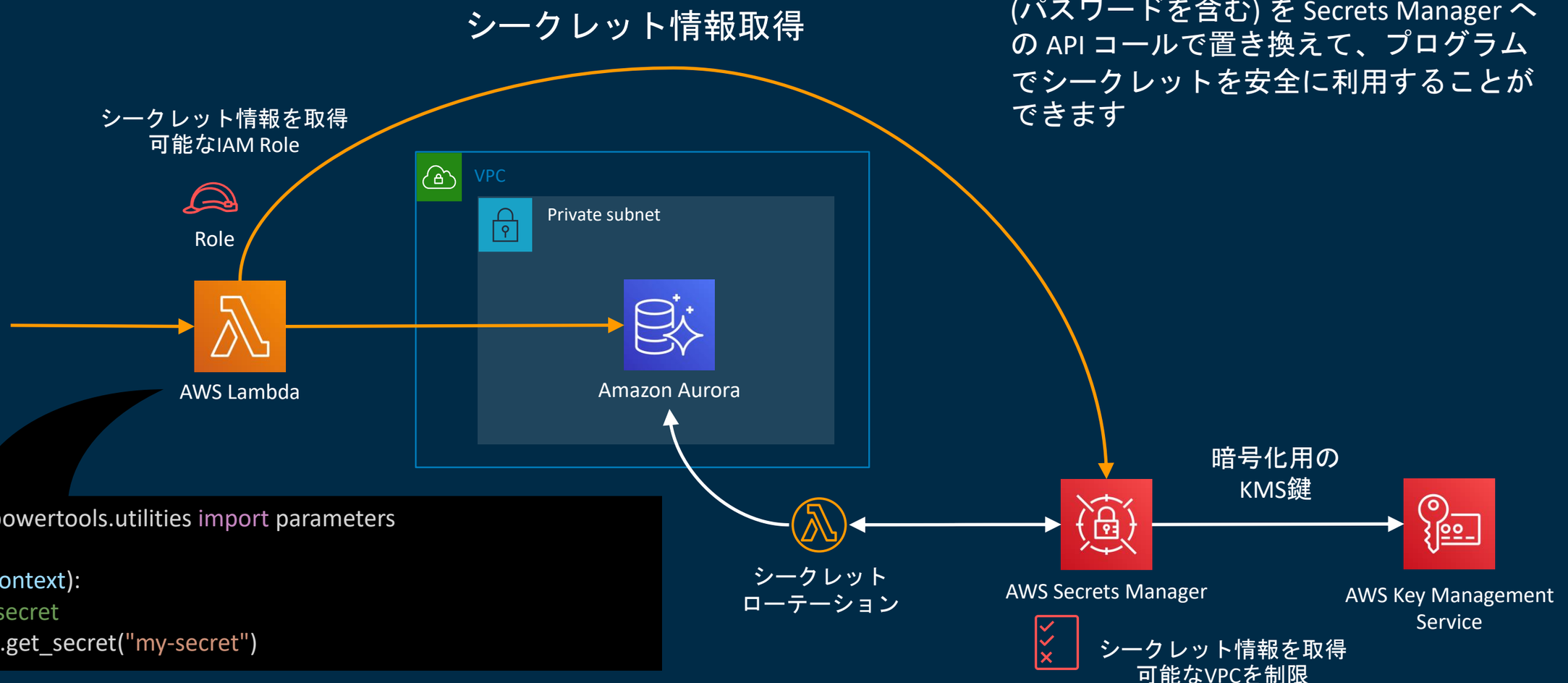
SEC 3. ワークロードにどのようにアプリケーションセキュリティを実装しますか?

✔	セキュリティ意識向上ドキュメントを頻繁に確認する
✔	コードで使用される機密情報を安全に保管する
✔	ランタイム保護を実装して、悪意のあるコードの実行を防止する
✔	ワークロードコードの依存関係/ライブラリを自動的に確認する
✔	インバウンドイベントを検証する

In Partnership with intel

コードで使用される機密情報を安全に保管する

コード内のハードコードされた認証情報(パスワードを含む) を Secrets Manager への API コールで置き換えて、プログラムでシークレットを安全に利用することができます




```
from aws_lambda_powertools.utilities import parameters

def handler(event, context):
  # Retrieve a single secret
  value = parameters.get_secret("my-secret")
```


In Partnership with intel

セキュリティ

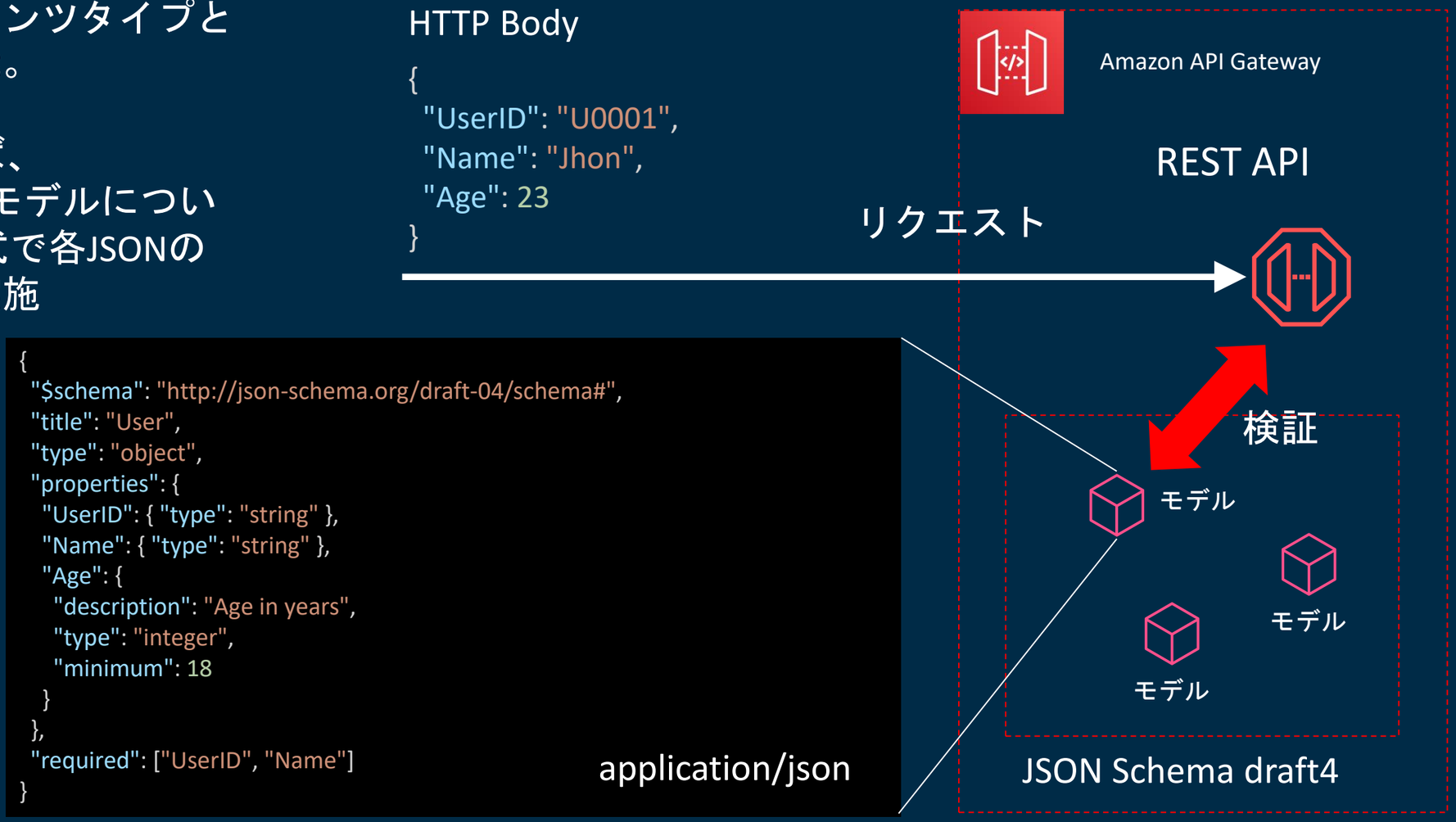
SEC 3. ワークロードにどのようにアプリケーションセキュリティを実装しますか?

✔	セキュリティ意識向上ドキュメントを頻繁に確認する
✔	コードで使用される機密情報を安全に保管する
✔	ランタイム保護を実装して、悪意のあるコードの実行を防止する
✔	ワークロードコードの依存関係/ライブラリを自動的に確認する
✔	インバウンドイベントを検証する

In Partnership with intel

API Gateway REST API のリクエスト検証

- リクエストボディはコンテンツタイプとモデルのセットで検証可能。
- JSON形式のボディであれば、application/jsonを設定し、モデルについてはJSON Schema draft4形式で各JSONのプロパティに対して検証実施



In Partnership with intel

信頼性

aws DEV DAY
ONLINE JAPAN

信頼性 REL1

In Partnership with intel

信頼性

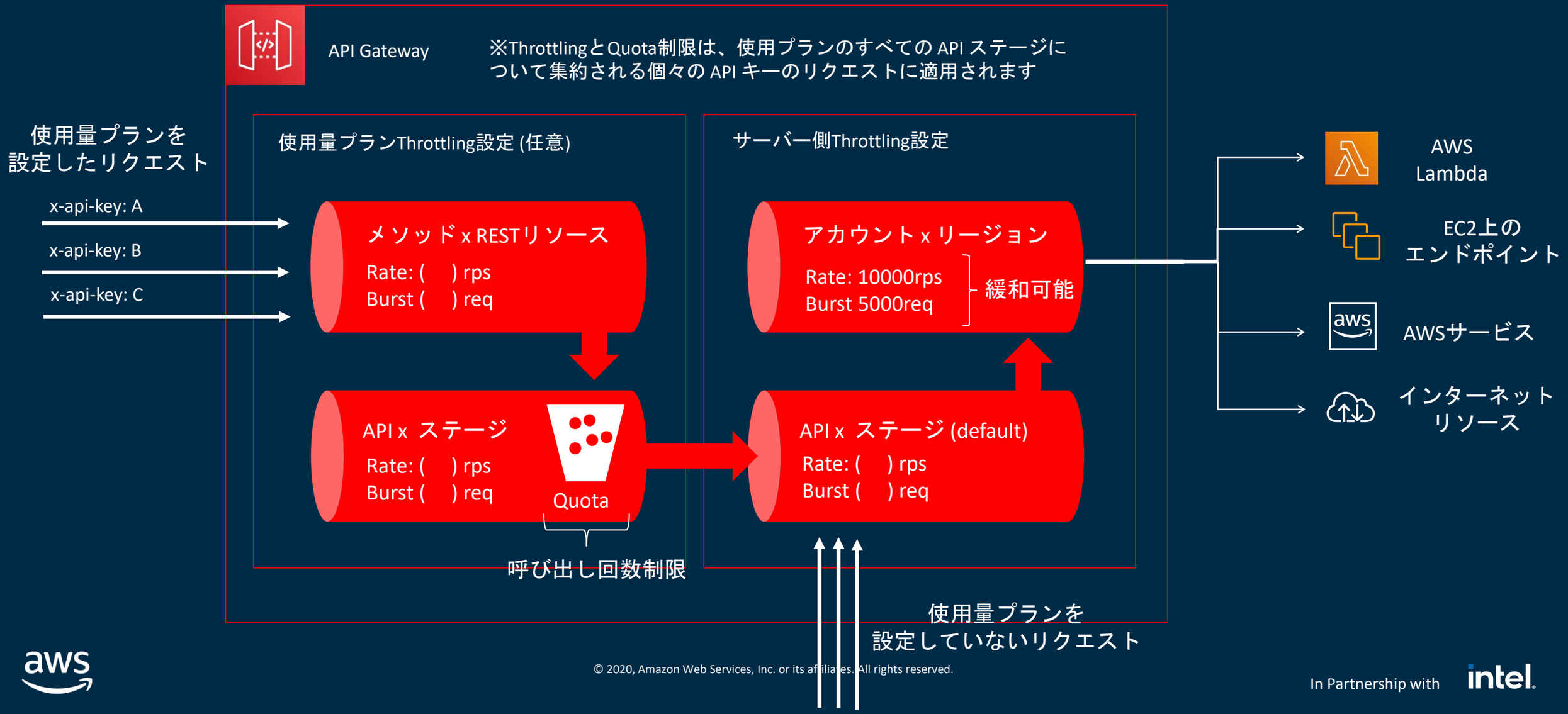
REL 1. インバウンドリクエスト率はどのように調整していますか?

✔	スロットリングを使用してインバウンドリクエスト率を制御する
✔	API クォータを使用、分析、適用する
✔	スケーラブルではないリソースを保護するメカニズムを利用する

API Gatewayによるスロットリング制御

- 閾値を超えるとHTTPステータスコード429(Too Many Requests)エラー
  - リトライ実装はAPIクライアント開発者の責務
- Rate/Burstによるスロットリング設定が可能
- 使用量プランを使うと
  - Quota(呼び出し回数制限)の設定が可能
    - (日/週/月あたりのリクエスト回数)

API Gatewayによるスロットリング制御



In Partnership with intel

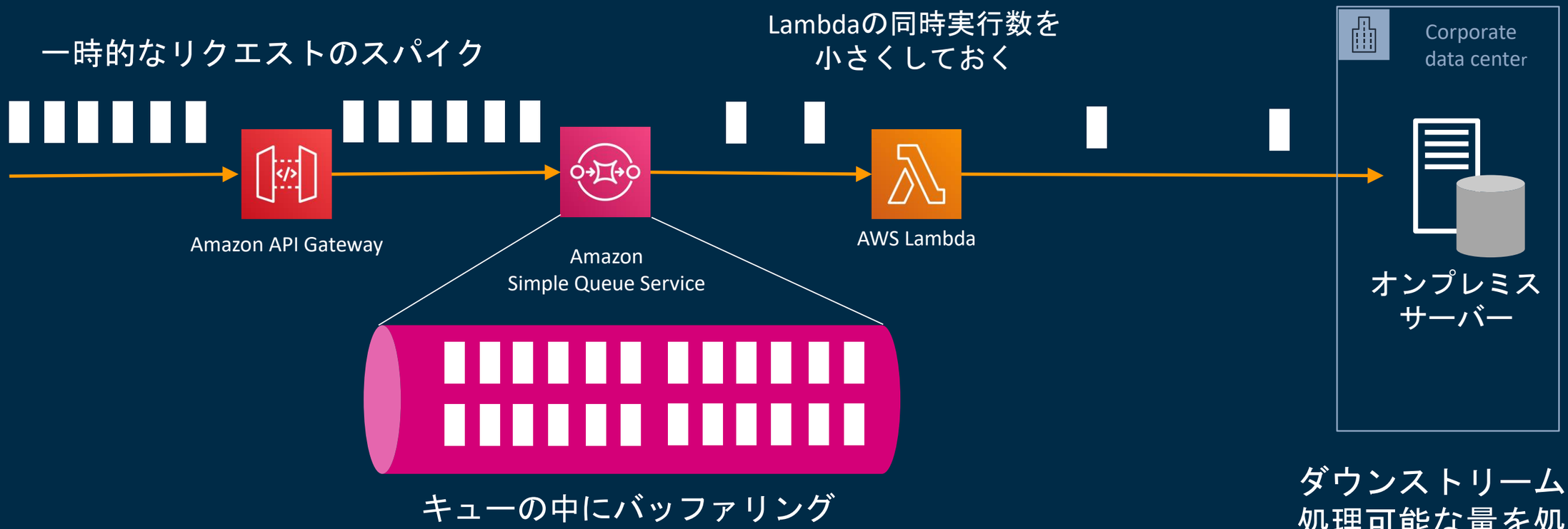
信頼性

REL 1. インバウンドリクエスト率はどのように調整していますか?

✔	スロットリングを使用してインバウンドリクエスト率を制御する
✔	API クォータを使用、分析、適用する
✔	スケーラブルではないリソースを保護するメカニズムを利用する

スケーラブルではないリソースを保護

オンプレミスサーバーや永続層などサーバーレス に対応するスケーラビリティが見込まれないダウンストリームへのアクセスについては、直接行うのではなく、キューやストリーム等のバッファリングによって受け取るトランザクションの数を抑制することでコンポーネントスループットを調整



In Partnership with intel

aws DEV DAY
ONLINE JAPAN

信頼性 REL2

In Partnership with intel

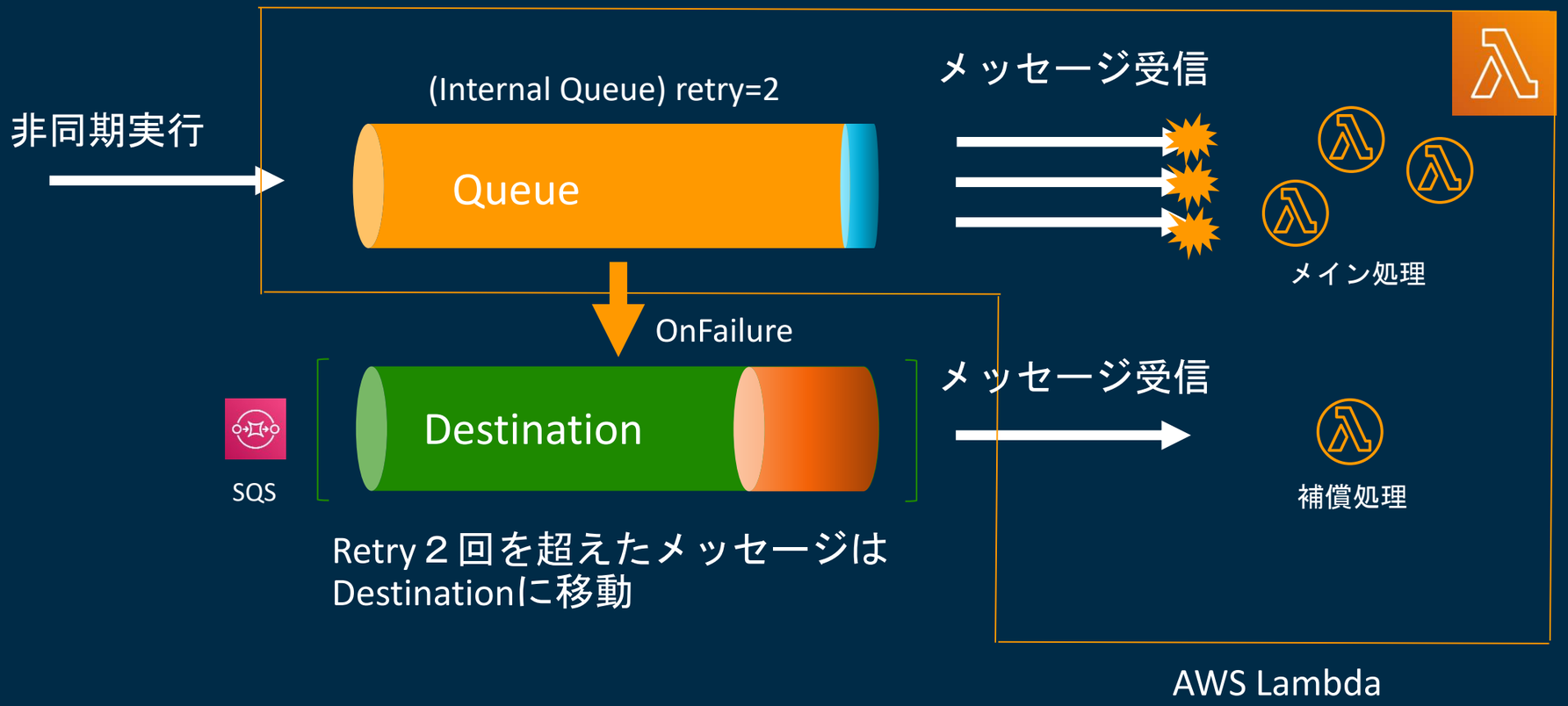
信頼性

REL 2. サーバーレスアプリケーションには回復力をどのように組み込んでいますか?

✔	トランザクション障害、部分的な障害、断続的な障害に対処する
✔	重複した不要なイベントに対処する
✔	長時間実行のトランザクションを調整する
✔	バーストレートにおけるパターンのスケーリングを検討する

Lambda非同期実行によるエラー対処

- Lambdaの非同期実行は、Lambdaサービス内にキューを持つ
  - 非同期実行のLambdaにはDestinationを設定可能、エラーデータを退避しておく



In Partnership with intel

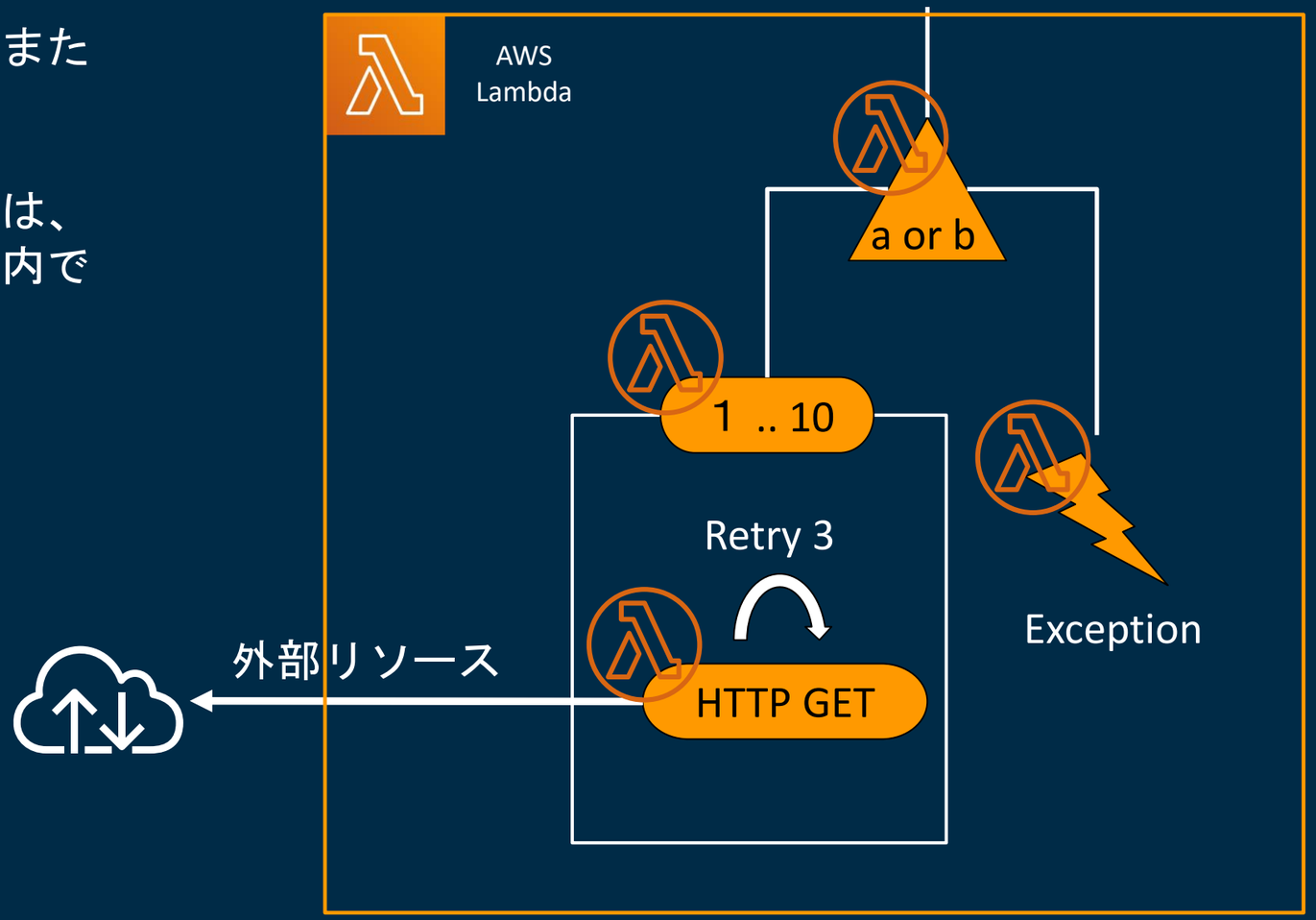
信頼性

REL 2. サーバーレスアプリケーションには回復力をどのように組み込んでいますか?

✔	トランザクション障害、部分的な障害、断続的な障害に対処する
✔	重複した不要なイベントに対処する
✔	長時間実行のトランザクションを調整する
✔	バーストレートにおけるパターンのスケーリングを検討する

Lambda関数内でワークフローを独自実装

- 長時間実行のトランザクションは複数の関数にまたがって処理されます
- 複数の同期的な依存性の強い呼び出しチェーンは、単一コンポーネントのアプリケーションコード内で処理せず、ステートマシンの利用を検討

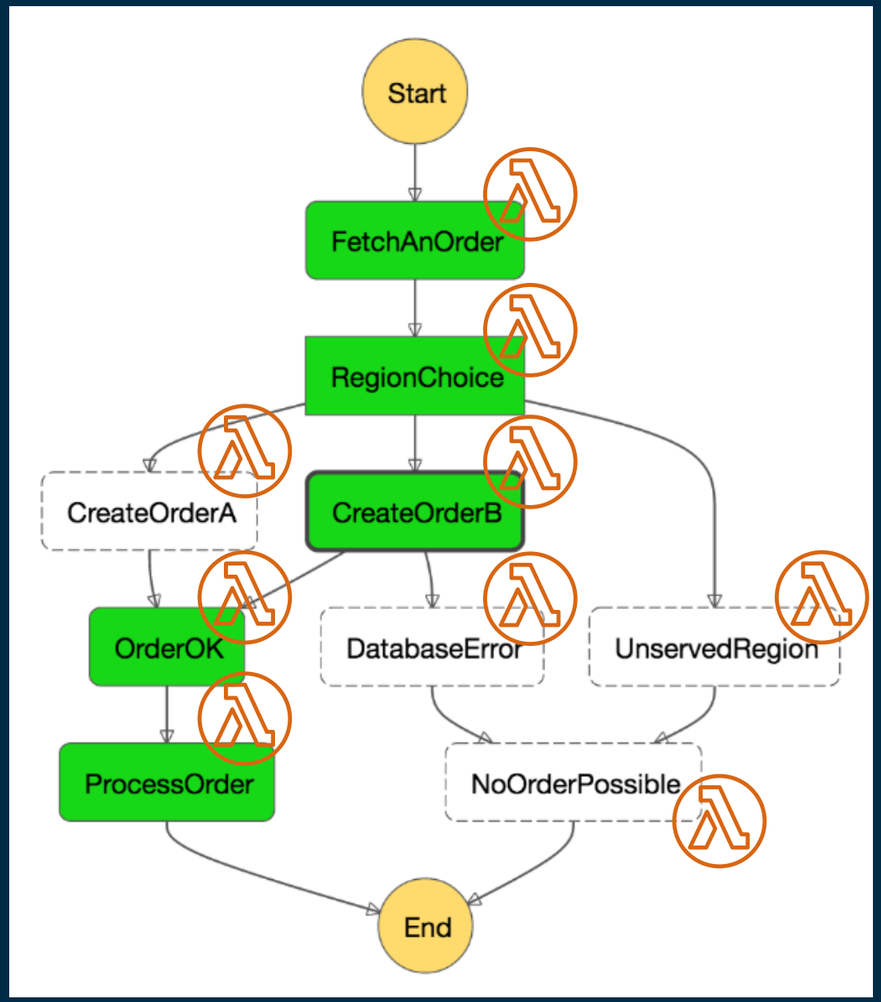


Lambda関数からLambda関数を呼び出すパターン

In Partnership with intel

ワークフローをStepFunctionsで実装

- ステートマシンを使用して分散トランザクションを可視化し、オーケストレーションロジックからビジネスロジックを分離
- AWS Step Functions を使用すると、ステートマシンを介して複数の AWS のサービスをサーバーレスワークフローとして、統合可能
- Step Functions では、宣言型言語を使用して、再試行、バックオフレート、最大試行、間隔、タイムアウトをステートマシンのすべてのステップに個別に設定可能



In Partnership with intel


パフォーマンス効率

aws DEV DAY
ONLINE JAPAN

パフォーマンス効率 PERF1

In Partnership with intel

パフォーマンス効率

PERF 1. どのようにサーバーレスアプリケーションのパフォーマンスを最適化しますか?

✔	最適なキャパシティー単位を測定、評価、選択する
✔	関数の起動時間を測定して最適化する
✔	非同期およびストリームベースの関数呼び出しによる同時実行性を活用する
✔	アクセスパターンを最適化し、必要に応じてキャッシュを適用する
✔	可能な場合は関数を介してマネージドサービスと直接統合する

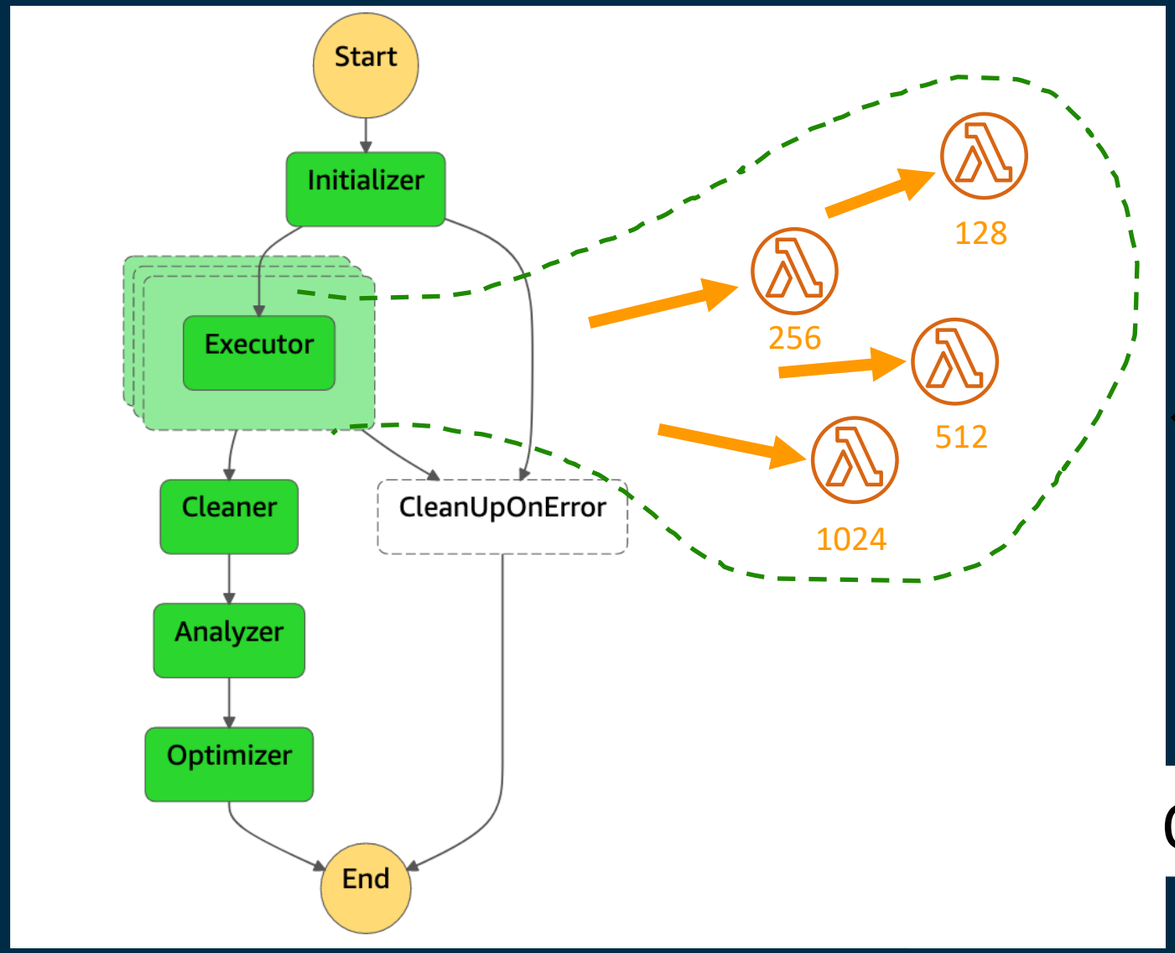
最適なキャパシティーを選択する

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

適切なメモリ量を割り当てるのが課題

In Partnership with intel

AWS Lambda Power Tuning によるチューニング

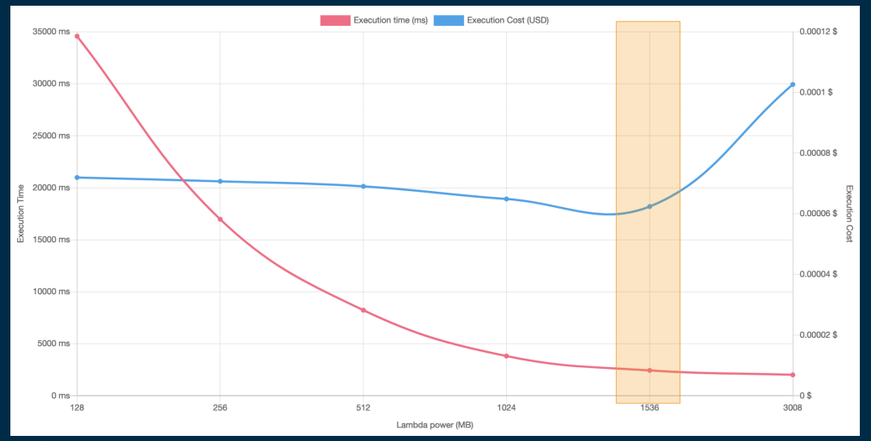


AWS Lambda Power Tuningは、AWS Step Functionsを利用したステートマシンであり、Lambda関数のコストやパフォーマンスを最適化するのに役立ちます。

INPUT

```
{
  "lambdaARN": "function-arn",
  "powerValues": [128, 256, 512, 1024],
  "num": 50,
  "payload": {}
}
```

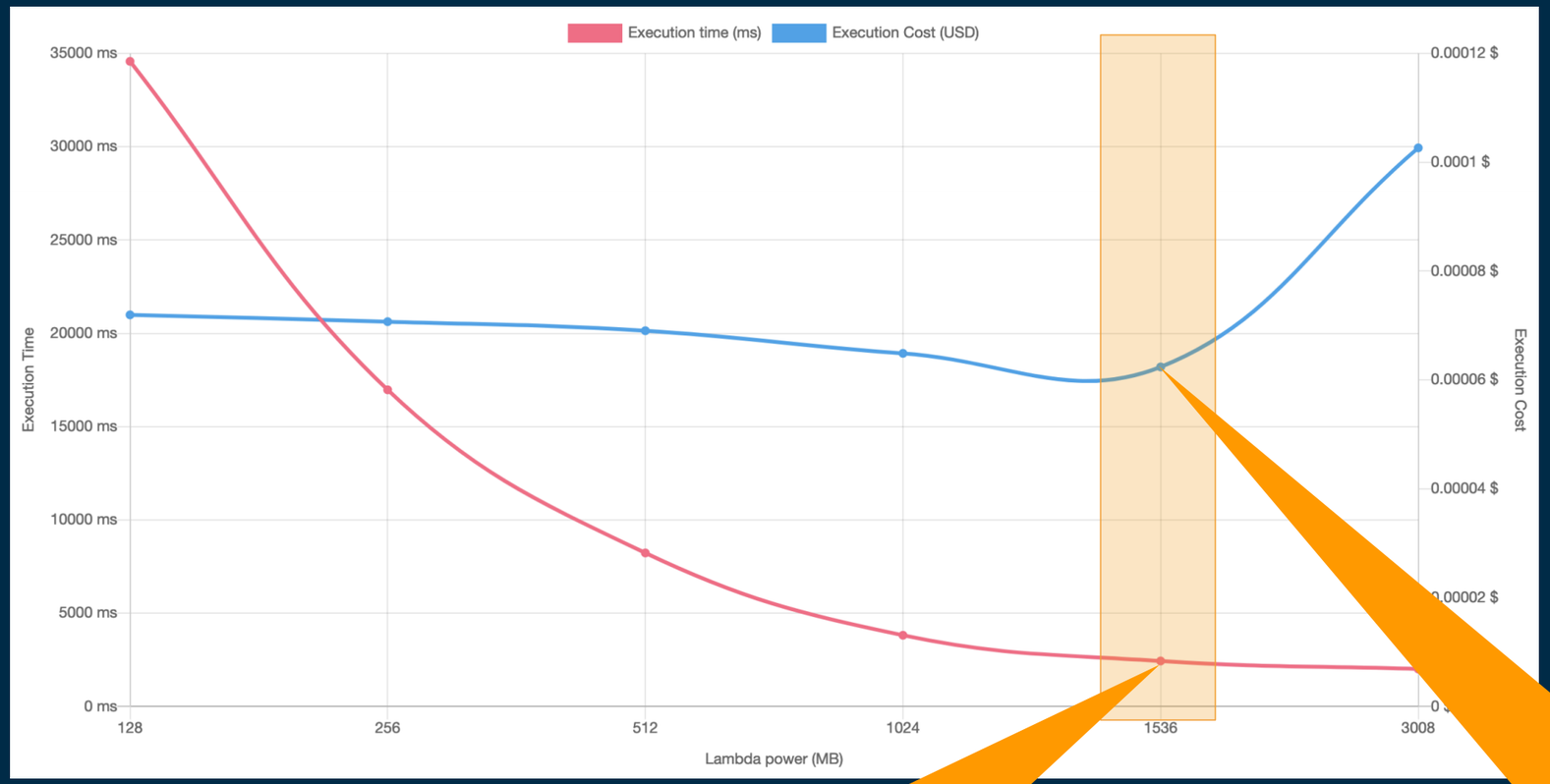
OUTPUT



https://github.com/alexcasalboni/aws-lambda-power-tuning

In Partnership with intel

CPUバウンドなワークロード




```
def lambda_handler(event, context):

  heavy_CPU_job()

return {
  'statusCode': 200,
  'body': json.dumps('CPU worked'),
}
```

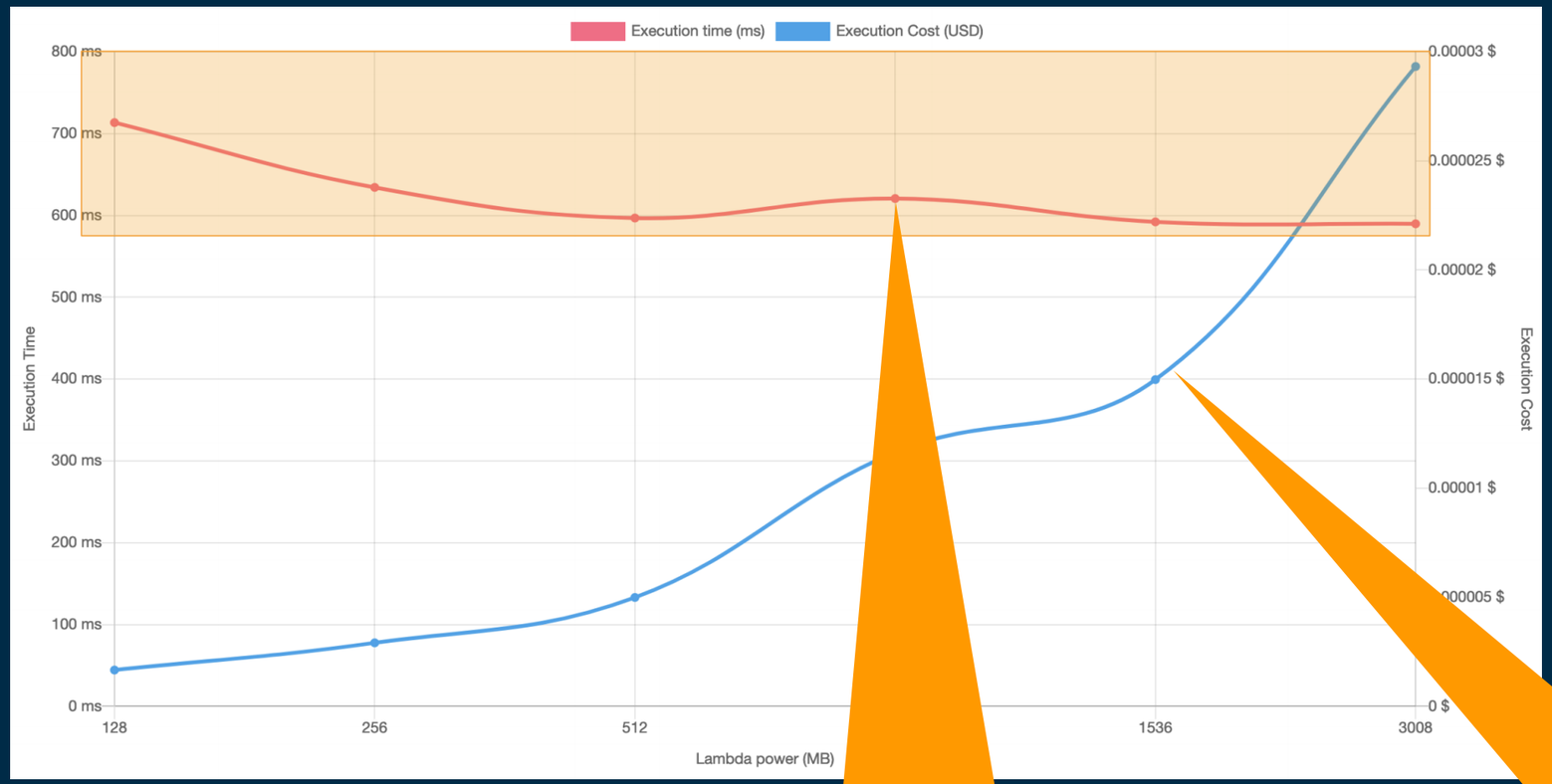
コスト = GB秒 x リクエスト件数

メモリが1536MBの時に
処理時間が最低付近になり
その後は減少しない

メモリが1536MBの時に
コストも底を打つ

In Partnership with intel

外部APIコールによる処理時間増大




```
def lambda_handler(event, context):

  external_API_call()

return {
  'statusCode': 200,
  'body': json.dumps('CPU worked'),
}
```

コスト = GB秒 x リクエスト件数

メモリが上がっても
処理時間に対する
貢献が少ない

メモリを上げると
コストも上がる

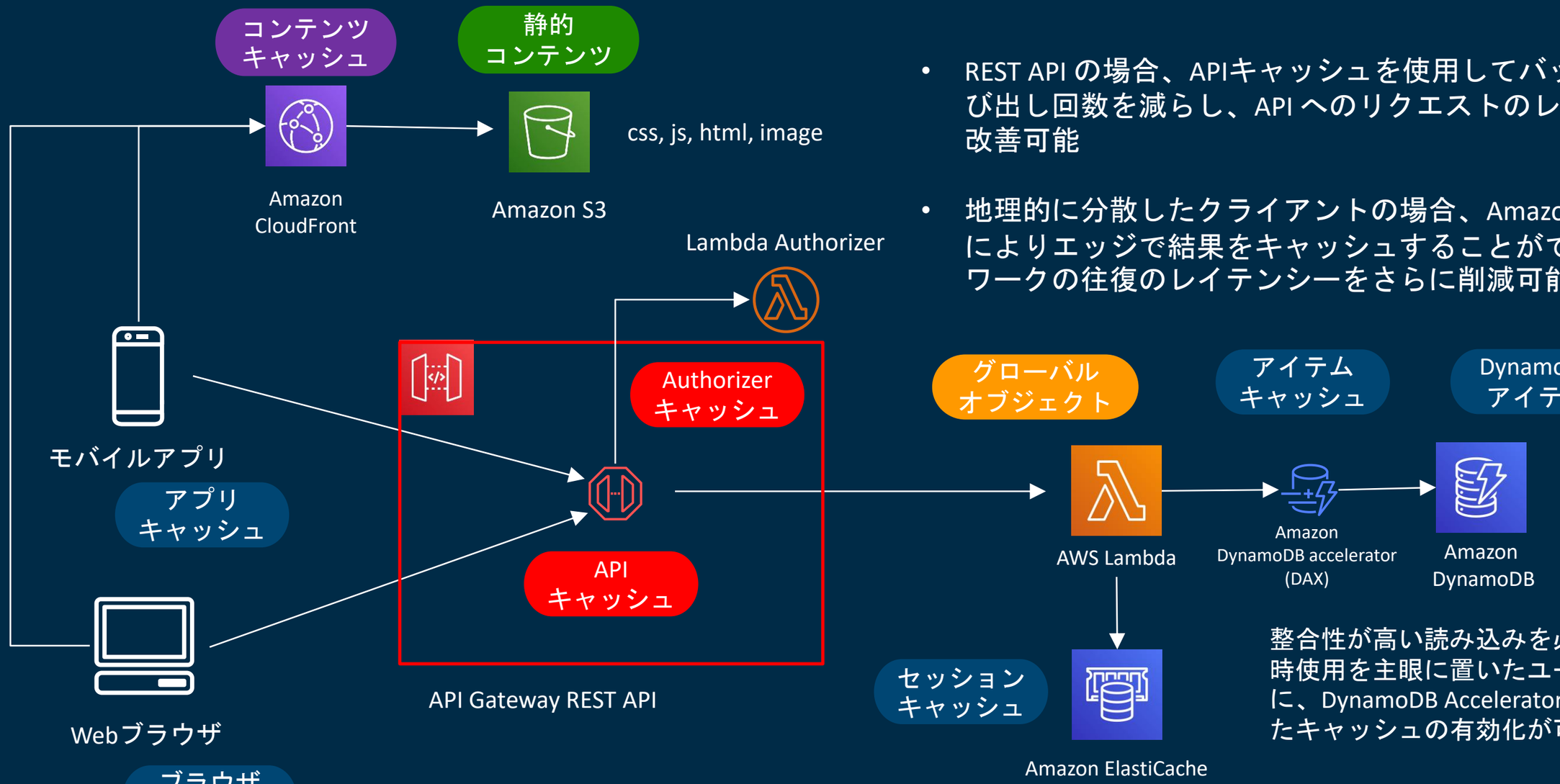
パフォーマンス効率

PERF 1. どのようにサーバーレスアプリケーションのパフォーマンスを最適化しますか?

✔	最適なキャパシティー単位を測定、評価、選択する
✔	関数の起動時間を測定して最適化する
✔	非同期およびストリームベースの関数呼び出しによる同時実行性を活用する
✔	アクセスパターンを最適化し、必要に応じてキャッシュを適用する
✔	可能な場合は関数を介してマネージドサービスと直接統合する

In Partnership with intel

必要に応じてキャッシュを適用する



- REST API の場合、APIキャッシュを使用してバックエンドの呼び出し回数を減らし、API へのリクエストのレイテンシーを改善可能
- 地理的に分散したクライアントの場合、Amazon CloudFront によりエッジで結果をキャッシュすることができ、ネットワークの往復のレイテンシーをさらに削減可能

整合性が高い読み込みを必要とせず、即時使用を主眼に置いたユースケース向けに、DynamoDB Accelerator (DAX) を使用したキャッシュの有効化が可能

In Partnership with intel


コスト最適化

aws DEV DAY
ONLINE JAPAN

コスト最適化 COST1

In Partnership with intel

コスト最適化

COST 1. どのようにサーバーレスアプリケーションのコストを最適化しますか?

✔	外部呼び出しと関数コードの初期化を最小限に抑える
✔	ログ出力とその保持を最適化する
✔	関数設定を最適化してコストを削減する
✔	コストを意識した使用パターンをコードで使用する

ログ出力とその保持を最適化する

```
Resources:
  HelloFunc:
    Type: AWS::Serverless::Function
    Properties:
      FunctionName: hello-func
      CodeUri: src
      Handler: app.handler
      Runtime: python3.8
      AutoPublishAlias: prod
      Timeout: 10
      MemorySize: 128

  HelloFuncLogGroup:
    Type: AWS::Logs::LogGroup
    Properties:
      LogGroupName: !Sub /aws/lambda/${HelloFunc}
      RetentionInDays: 7
```

- Lambda関数の定義と同時に AWS::Logs::LogGroup によってLog Groupを作成する
- RetentionInDays にログ保持期間を設定する

コスト最適化

COST 1. どのようにサーバーレスアプリケーションのコストを最適化しますか?

✔	外部呼び出しと関数コードの初期化を最小限に抑える
✔	ログ出力とその保持を最適化する
✔	関数設定を最適化してコストを削減する
✔	コストを意識した使用パターンをコードで使用する

Amazon API Gateway HTTP API の利用

HTTP API

リージョン: アジアパシフィック (東京)

API コール

リクエスト数 (月間)	価格 (100 万あたり)
最初の 3 億	1.29USD
3 億以上	1.18USD

*HTTP API の計量は、512 KB 単位とします。

REST API

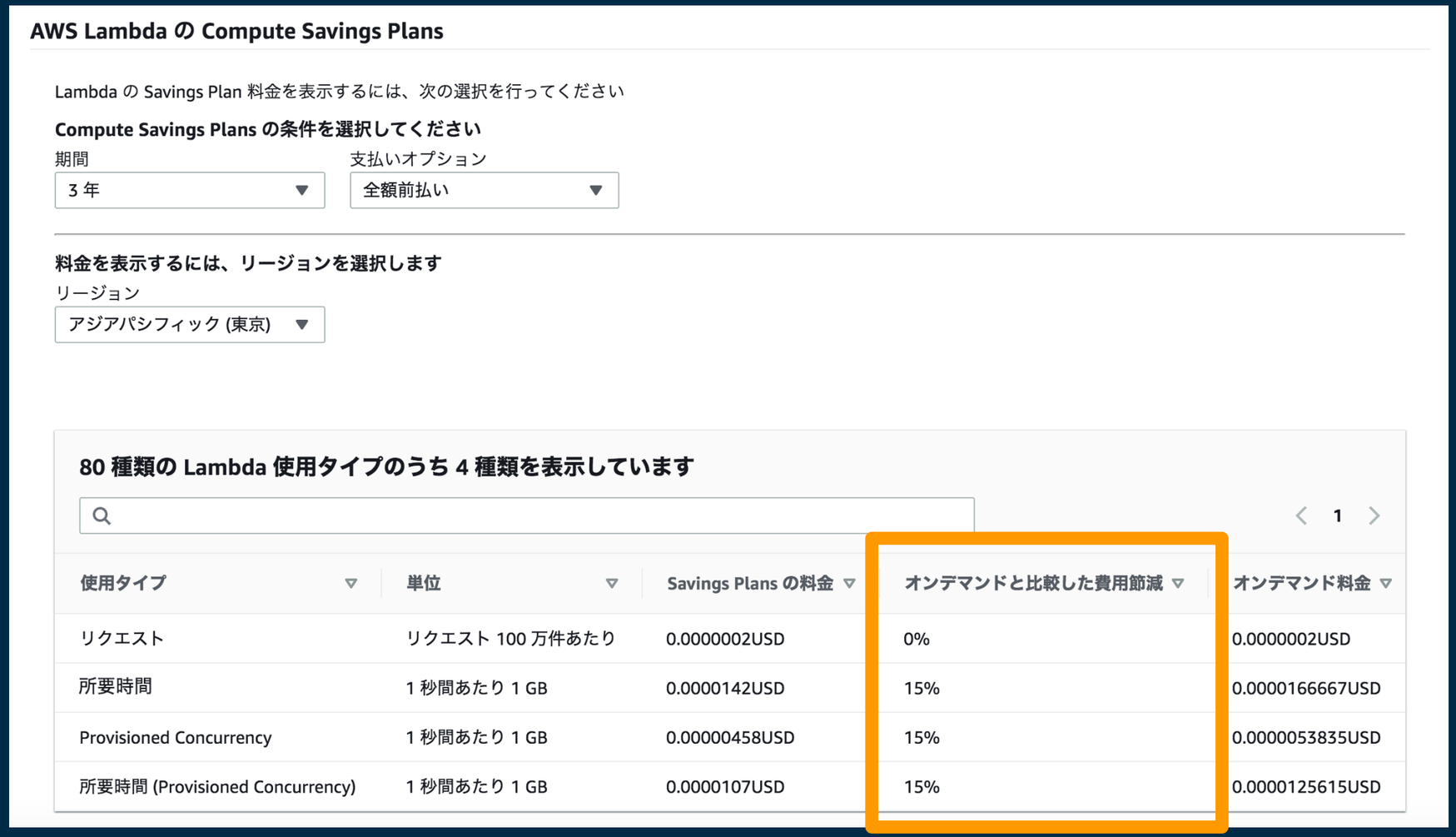
API コール

リクエスト数 (月間)	価格 (100 万あたり)
最初の 3 億 3,300 万	4.25USD
次の 6 億 6,700 万	3.53USD
次の 190 億	3.00USD
200 億以上	1.91USD

REST APIと比較し、多くのお客様でAPI呼び出し料金が安価に
(およそ70%低下も)

In Partnership with intel

AWS Lambda の Compute Savings Plans



AWS Lambda の Compute Savings Plans

Lambda の Savings Plan 料金を表示するには、次の選択を行ってください

Compute Savings Plans の条件を選択してください

期間: 3 年　　支払いオプション: 全額前払い

料金を表示するには、リージョンを選択します

リージョン: アジアパシフィック (東京)

80 種類の Lambda 使用タイプのうち 4 種類を表示しています

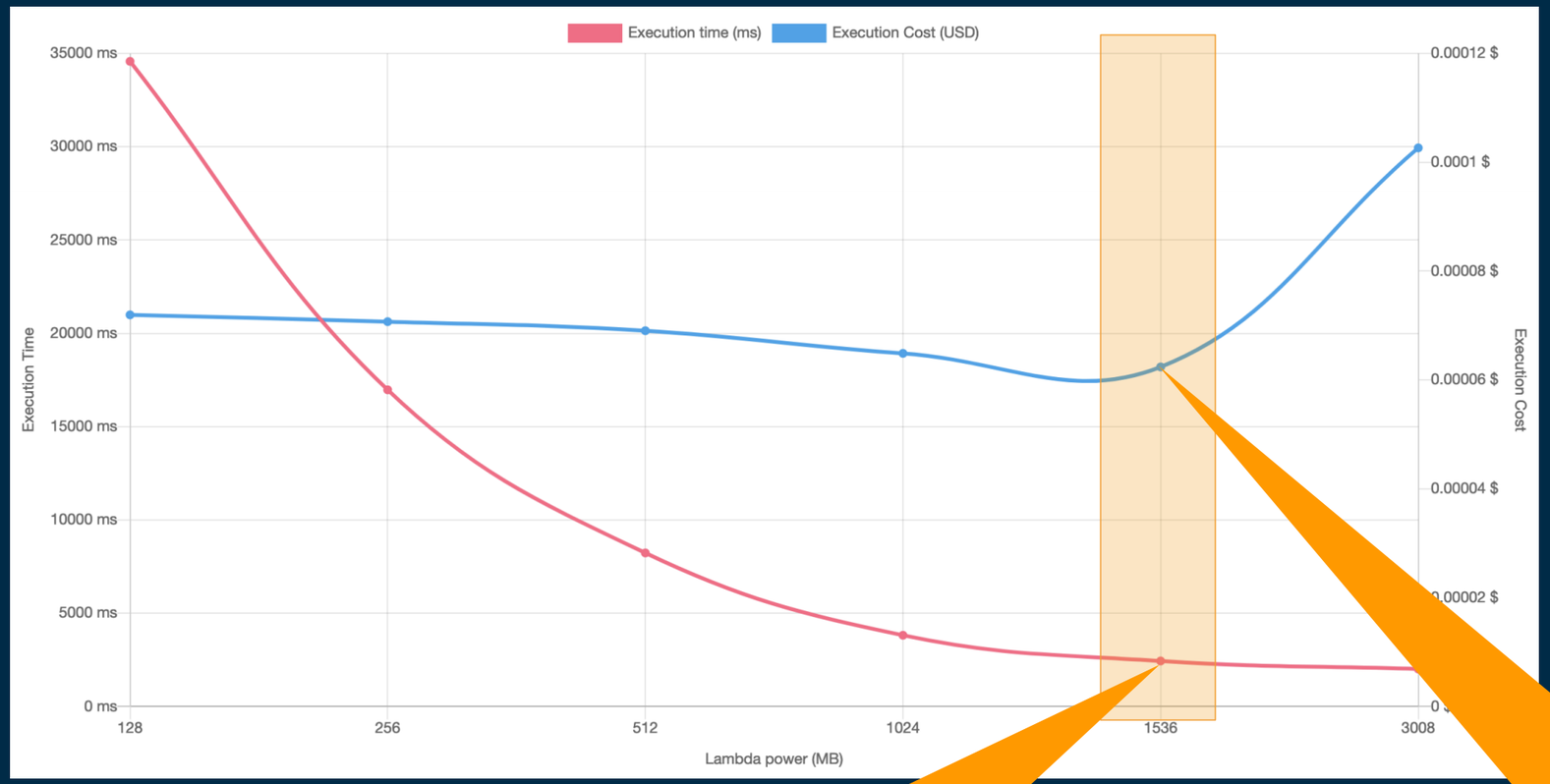
1

使用タイプ	単位	Savings Plans の料金	オンデマンドと比較した費用節減	オンデマンド料金
リクエスト	リクエスト 100 万件あたり	0.0000002USD	0%	0.0000002USD
所要時間	1 秒間あたり 1 GB	0.0000142USD	15%	0.0000166667USD
Provisioned Concurrency	1 秒間あたり 1 GB	0.00000458USD	15%	0.0000053835USD
所要時間 (Provisioned Concurrency)	1 秒間あたり 1 GB	0.0000107USD	15%	0.0000125615USD

- Compute Savings Plans は、1 年または 3 年の期間の使用量を契約すれば、EC2、Lambda、Fargate の低額の使用料金が提供される柔軟な料金モデルです
- Savings Plan にサインアップすると、契約量までの使用料金は割引された Savings Plans料金で課金されます

In Partnership with intel

CPUバウンドなワークロード




```
def lambda_handler(event, context):

  heavy_CPU_job()

return {
  'statusCode': 200,
  'body': json.dumps('CPU worked'),
}
```

コスト = GB秒 x リクエスト件数

メモリが1536MBの時に
処理時間が最低付近になり
その後は減少しない

メモリが1536MBの時に
コストも底を打つ

In Partnership with intel

aws DEV DAY
ONLINE JAPAN

まとめ

In Partnership with intel

まとめ

AWS Well-Architected Framework(W-A)とは?

設計原則と(質問と回答形式)のベストプラクティス集

In Partnership with intel

まとめ

AWS Well-Architected Framework(W-A)とは?

1度だけではなく
定期的な見直し(KAIZEN)が重要

- システム設計・運用のベストプラクティス集
- 情報に基づいた意思決定を行い、その意思決定が持つ影響を理解する(ためのツール)
- リスクを「把握できていること」が重要。リスクに対処するかは、ビジネス的な判断次第

In Partnership with intel

まとめ

AWSのSAに相談することも出来る

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]

In Partnership with intel

aws DEV DAY ONLINE JAPAN

Thank you!

Kensuke Shimokawa

@_kensh

In Partnership with intel