



このコンテンツは公開から3年以上経過しており内容が古い可能性があります  
最新情報については[サービス別資料](#)もしくはサービスのドキュメントをご確認ください

# [AWS Black Belt Online Seminar]

## 実践的サーバーレスセキュリティ プラクティス

ソリューションカットシリーズ

Solutions Architect 鈴木 哲詩

2019/08/13

AWS 公式 Webinar

<https://amzn.to/JPWebinar>



過去資料

<https://amzn.to/JPArchive>



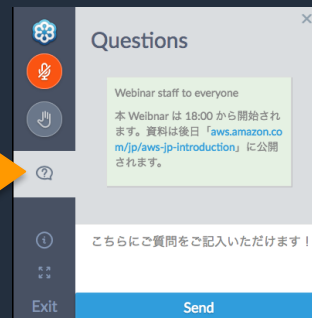
# AWS Black Belt Online Seminar とは

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

## 質問を投げることができます！

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

- ① 吹き出しをクリック
- ② 質問を入力
- ③ Sendをクリック



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

# 内容についての注意点

- 本資料では2019年8月13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おります。日本居住者のお客様が東京リージョンを使用する場合、別途消費税をご請求させていただきます。
- AWS does not offer binding price quotes. AWS pricing is publicly available and is subject to change in accordance with the AWS Customer Agreement available at <http://aws.amazon.com/agreement/>. Any pricing information included in this document is provided only as an estimate of usage charges for AWS services based on certain information that you have provided. Monthly charges will be based on your actual use of AWS services, and may vary from the estimates provided.

# 本日のアジェンダ

- サーバーレスとは
- AWS のセキュリティ概要
- サーバーレスアプリケーションにおけるセキュリティ

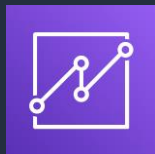
# サーバーレスとは

# サーバーレスとは

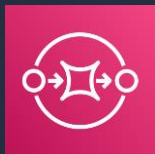
## 定義

利用者によるサーバーのプロビジョニングやメンテナンス、耐障害性の確保が不要なサービス

## 代表的なサーバーレスなAWSサービス



Amazon QuickSight



Amazon Simple Queue Service



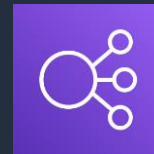
AWS Lambda



Amazon Personalize



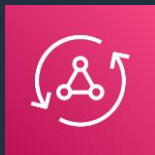
AWS Systems Manager



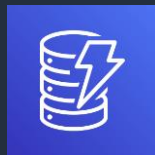
Elastic Load Balancing



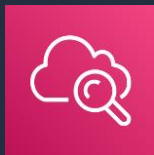
Amazon Kinesis



AWS AppSync



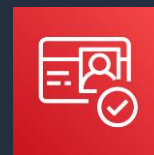
Amazon DynamoDB



Amazon CloudWatch



Amazon API Gateway



Amazon Cognito

# サーバレスの主な特徴

## サーバーの管理が不要

- 事前に調達が必要。
- 必要な時にプロビジョニング可能
- インフラの保守・運用をAWSが実施

## 価値に対する支払い

- 利用したリソースや処理時間等をもとにした従量課金制
- 初期投資不要

## 柔軟なスケーリング

利用状況に応じてスケールアウト・スケールインを自動的に実施

## 自動化された高可用性

- 冗長化構成等の可用性、耐障害性を高める仕組みが組み込み済み

<https://aws.amazon.com/jp/serverless/>

# サーバーレスを採用する理由

## AWS Lambdaのお客様事例

- スケーラビリティ・可用性の確保

「常のトラフィックの最大 30 倍のスパイクを確実に処理できるようになりました」

- コスト削減

「画像処理に要する時間が、数時間からわずか 10 秒あまりに短縮され、インフラストラクチャと運用のコストが削減されました」

<https://aws.amazon.com/jp/lambda/resources/customer-case-studies/>



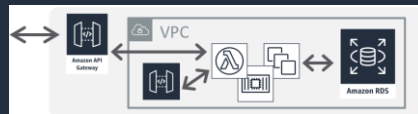
# 代表的な適用シーン/ユースケースと実装形



動的 Web / モバイルバックエンド



インタラクティブモバイル



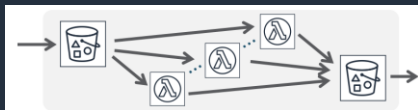
業務系 API / グループ企業間 API



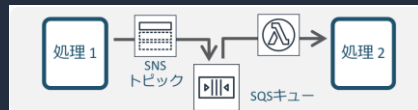
配信系・インタラクティブ API



画像処理 / シンプルなデータ加工



分散並列処理



イベント駆動の業務処理連携



アプリフロー処理



流入データの連続処理



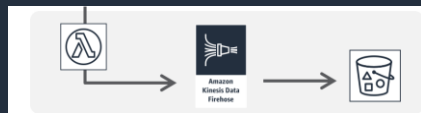
IoT バックエンド



チャットボット / Alexa スキル



データ変更トリガー処理



ログデータ収集処理



データレイクからのデータ加工



機械学習/ETLデータパイプライン



スケジュール・ジョブ/CRON

<https://aws.amazon.com/jp/serverless/patterns/serverless-pattern/>

# AWS のセキュリティ概要

# お客様の関心事

- AWS はセキュリティやコンプライアンスに対してどのような取り組みをしているのか？
- AWS クラウドを利用することでセキュリティやコンプライアンス対応はどのように変わるのか？
- AWS を利用したときのお客様がすべき責任範囲は何か？

# AWSにおけるセキュリティの位置づけ

- セキュリティは、AWSにおいて最優先されるべき事項
- セキュリティ/コンプライアンスへの大規模な投資
- セキュリティ/コンプライアンスは継続的な投資対象
- 専門部隊を設置

AWS クラウドセキュリティ

クラウドの力を借りたセキュリティでデータを保護します。

クラウドセキュリティ 導入テスト セキュリティ速報 リソース コンプライアンス パートナー

クラウドセキュリティはAWSの最優先事項です。AWSのお客様は、セキュリティを最も重視し、AWSのクラウドセキュリティとネットワークアーキテクチャを利用できます。

AWS クラウドの利点は、お客様が安全な環境を維持しながらその環境をスケーリングし、進化させることを可能にすることです。実際に使用したサービスに対してのみ料金が発生するため、事前の投資なしに必要なセキュリティを獲得でき、オンプレミス環境よりもコストを削減できます。

AWS re:Inforce 2019 のハイライト

<https://aws.amazon.com/jp/security/>

AWS コンプライアンスプログラム

グローバル

CSA cloud security alliance	ISO 9001	ISO 27001	ISO 27017	ISO 27018
CSA クラウドセキュリティアライアンスの統制	ISO 9001 規格 グローバル品質標準	ISO 27001 規格 セキュリティ管理による統制	ISO 27017 規格 クラウド固有の統制	ISO 27018 規格 個人データの保護
PCI DSS レベル 1 支払いカード規格	AICPA SOC	AICPA SOC	AICPA SOC	
PCI DSS レベル 1 支払いカード規格	SOC 1 監査統制レポート	SOC 2 セキュリティ、アベイラビリティ、& 機密保持レポート	SOC 3 全般的統制レポート	

<https://aws.amazon.com/jp/compliance/programs/>

# AWS責任共有モデル

## 定義

お客様と AWS の間で  
コンプライアンス、セキュリティ、IT 統制の責任を共有する

### お客様

クラウド内のセキュリティ  
に対する責任

SECURITY 'IN' THE CLOUD

利用者のコンテンツとアプリケーションのセキュリティに関連して、  
利用者が実装し、運用するセキュリティ対策

### AWS

クラウドのセキュリティ  
に対する責任

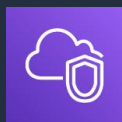
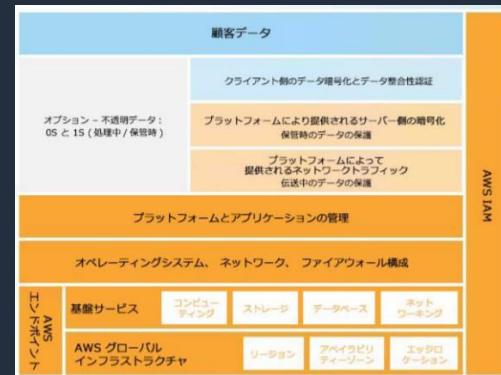
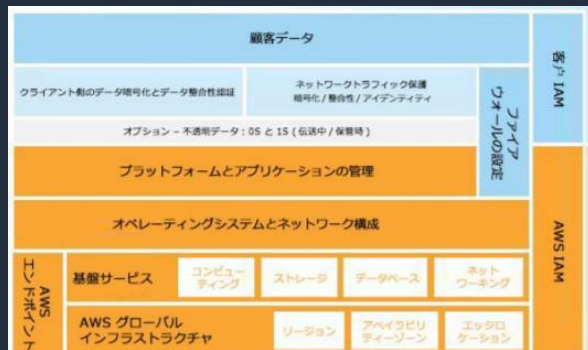
SECURITY 'OF' THE CLOUD

クラウド事業者（AWS）が実装し、運用する  
セキュリティ対策

# AWS責任共有モデル

## 概要

AWSのサービスごとに責任の範囲は変わる



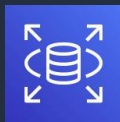
Amazon VPC



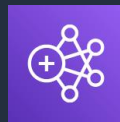
Amazon EBS



Amazon EC2



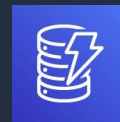
Amazon RDS



Amazon EMR



Amazon S3



Amazon DynamoDB

[https://d1.awsstatic.com/whitepapers/ja\\_JP/Security/AWS\\_Security\\_Best\\_Practices.pdf](https://d1.awsstatic.com/whitepapers/ja_JP/Security/AWS_Security_Best_Practices.pdf)

# AWS Lambdaの責任共有モデル

## 顧客データ、アプリケーション 認可 と 認証

データ暗号化と  
データ整合性・認証

アプリケーションの管理

インターネットアクセス、  
監視、ロギング  
  
(プラットフォームによって  
提供されるツール)

プラットフォームの管理

(プラットフォームによっ  
て提供される)  
コードの暗号化  
(保管時のデータ保護)

ネットワーク  
トラフィック保護  
(暗号化 / 整合性 / アイデンティ  
ティ (伝送中のデータ保護))

AWS IAM

## オペレーティングシステム、ネットワーク、ファイアウォール構成

AWS  
インフラポイン  
ト

基盤サービス

コンピュ  
ー  
ティ  
ン  
グ

ストレージ

データベース

ネット  
ワー  
キ  
ン  
グ

AWS グローバル  
インフラストラクチャ

リージョン

アベイラビ  
リ  
ティ  
ー  
ゾ  
ン

エッジ  
ロケーション



AWSの  
お客様が  
管理  
  
(クラウド内の  
セキュリティ  
に対する責任)



アマゾン  
ウェブ  
サービスが  
管理  
  
(クラウドの  
セキュリティ  
に対する責任)

# お客様の関心事

- AWS はセキュリティやコンプライアンスに対してどのような取り組みをしているのか？

セキュリティは最優先事項であり継続的に投資を行っている

- AWS クラウドを利用することでセキュリティやコンプライアンス対応はどのように変わるのか？

AWS クラウドを利用することでお客様自身がすべきワークロードの一部を AWS が実施し負荷が軽減される

- AWS を利用したときのお客様がすべき責任範囲は何か？

AWS 責任共有モデルに基づきご利用いただくAWSサービスに応じて果たすべき責任の範囲が変わる



# サーバーレスアプリケーションに おけるセキュリティ

# サーバーレスなサービス利用時のお客様の関心事

## アクセス制御

- ・ リソースの構成（作成・変更）権限の制御
- ・ リソース内のデータへのアクセス権限の制御

## データ保護

- ・ 通信の暗号化の有無
- ・ 保存データの暗号化の有無

## インフラ保護

- ・ DDoS 対策/ファイアウォール設置
- ・ 脆弱性診断

## ロギング・ モニタリング

- ・ サービスのログの利用
- ・ アプリ部分のログの利用
- ・ 監視対象項目の選定

<https://aws.amazon.com/jp/compliance/shared-responsibility-model/>

# サーバーレスなサービス利用時のお客様の関心事

## アクセス制御

- ・ リソースの構成（作成・変更）権限の制御
- ・ リソース内のデータへのアクセス権限の制御

## データ保護

- ・ 通信の暗号化の有無
- ・ 保存データの暗号化の有無

## インフラ保護

- ・ DDoS 対策/ファイアウォール設置
- ・ 脆弱性診断

## ロギング・ モニタリング

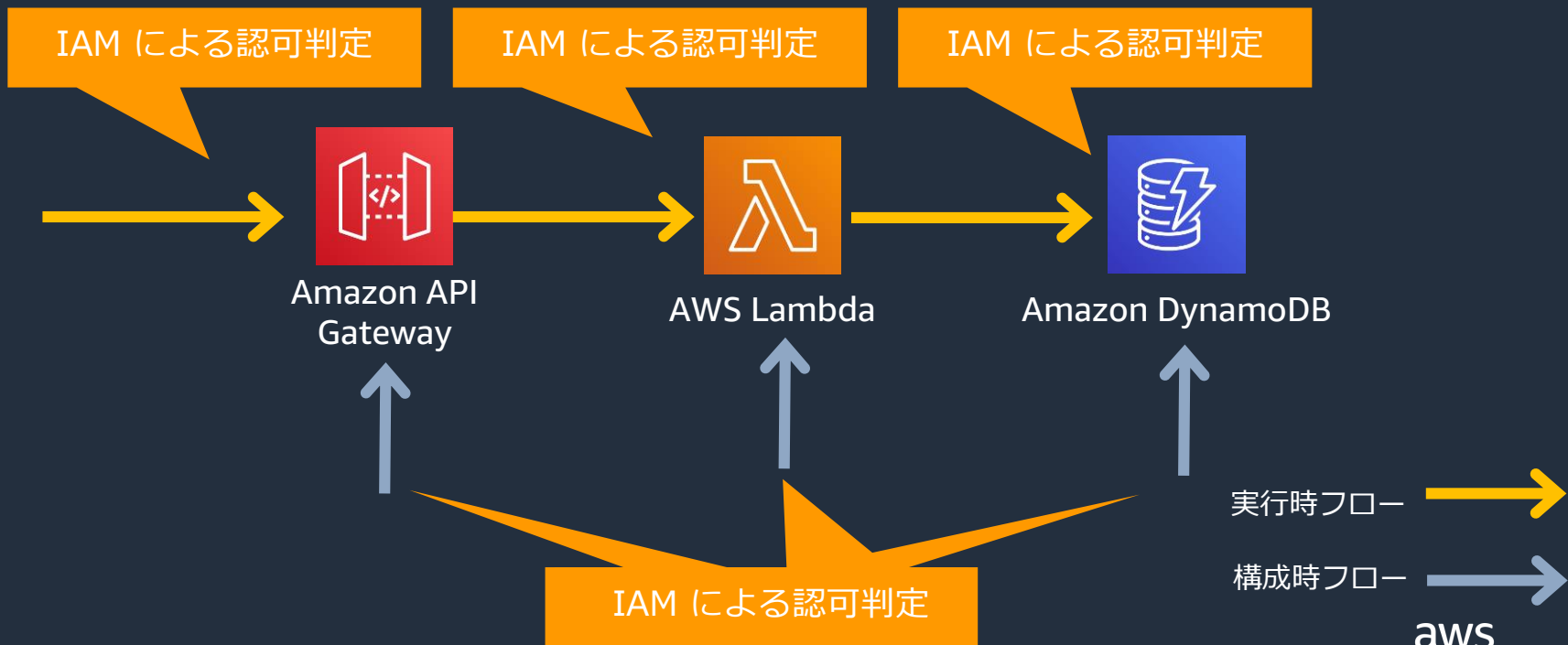
- ・ サービスのログの利用
- ・ アプリ部分のログの利用
- ・ 監視対象項目の選定

<https://aws.amazon.com/jp/compliance/shared-responsibility-model/>

# AWS サービスのアクセス制御

## 要約

AWS API エンドポイントへのアクセスは IAM による認可が必要  
サーバーレスだから変わるものではない



# Amazon DynamoDB のデータアクセス制御 概要

## 要約

Amazon DynamoDB はテーブルに格納された項目や属性に対するきめ細やかなアクセス制御をサポート

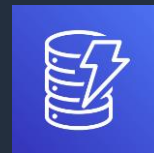


AWS Lambda

```
def get_user(event, context):  
    user_id = event['pathParameters']['id']  
    response = table.get_item(  
        Key={'id': user_id})  
    item = response.get('Item')  
    return {  
        'statusCode': 200,  
        'body': json.dumps(item)}
```

関数実装

GetItem



Amazon DynamoDB

```
TableName: User  
AttributeDefinitions:  
  - AttributeName: id  
    AttributeType: S  
KeySchema:  
  - AttributeName: id  
    KeyType: HASH
```

テーブル定義

# Amazon DynamoDB のデータアクセス制御 概要

## 要約

Amazon DynamoDB はテーブルに格納された項目や属性に対するきめ細やかなアクセス制御をサポート



AWS Lambda

GetItem

Lambda 関数の IAM ロールに許容するアクションを定義

- `dynamodb:*` は好ましくない
- 極力必要なアクションのみを定義

```
- Effect: Allow
Action:
  - dynamodb:GetItem
Resource:
  - Fn::Sub:
    arn:aws:dynamodb:${AWS::Region}:${AWS::AccountId}:table/User
```

IAM ロール

```
TableName: User
AttributeDefinitions:
  - AttributeName: id
    AttributeType: S
KeySchema:
  - AttributeName: id
    KeyType: HASH
```

テーブル定義

# Amazon DynamoDB のデータアクセス制御 概要

## 要約

Amazon DynamoDB はテーブルに格納された項目や属性に対するきめ細やかなアクセス制御をサポート

アクションを許容する対象のリソースを指定

- `arn:aws:dynamodb:*:*:*` は好ましくない
- 明示的に限定

```
- Effect: Allow
Action:
  - dynamodb:GetItem
Resource:
  - Fn::Sub:
    arn:aws:dynamodb:${AWS::Region}:${AWS::AccountId}:table/User
```

IAM ロール

GetItem

Lambda 関数の IAM ロールに許容するアクションを定義

- `dynamodb:*` は好ましくない
- 極力必要なアクションのみを定義

```
TableName: User
AttributeDefinitions:
  - AttributeName: id
    AttributeType: S
KeySchema:
  - AttributeName: id
    KeyType: HASH
```

テーブル定義

# Amazon DynamoDB のデータアクセス制御 概要

## 要約

Amazon DynamoDB はテーブルに格納された項目や属性に対するきめ細やかなアクセス制御をサポート

```
- Effect: Allow
Action:
  - dynamodb:GetItem
Resource:
  - Fn::Sub:
arn:aws:dynamodb:${AWS::Region}:${AWS::AccountId}:table/User
```

### Condition:

ForAllValues:StringEquals:

dynamodb:LeadingKeys:

- \${www.amazon.com:user\_id}

dynamodb:Attributes:

- id
- name
- email

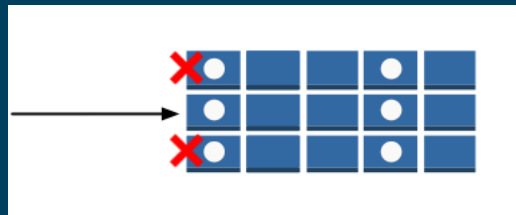
StringEqualsIfExists:

dynamodb:Select: SPECIFIC\_ATTRIBUTES

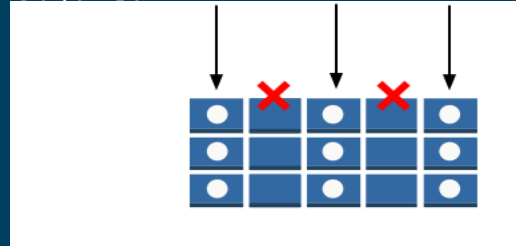
IAM ロール

条件付きのアクセス制御が可能

- 操作可能なアイテムの制限



- 操作可能なアトリビュートの制限



[Using IAM Policy Conditions for Fine-Grained Access Control](#)

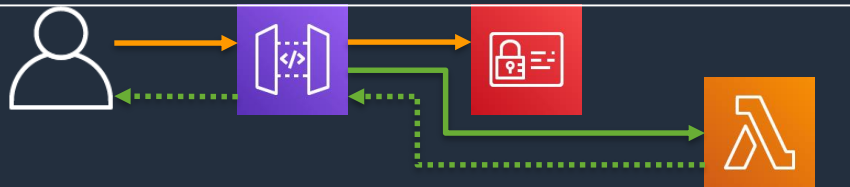


# Amazon API Gateway のアクセス制御

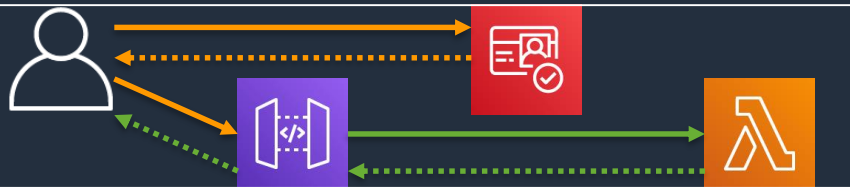
## 要約

Amazon API Gateway と連携するアクセス制御は3種類

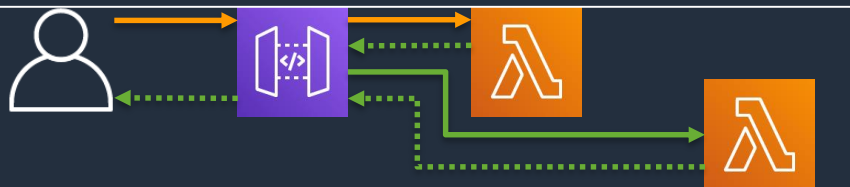
IAM アクセス権限



Amazon Cognito  
UserPool



Lambda  
Authorizer

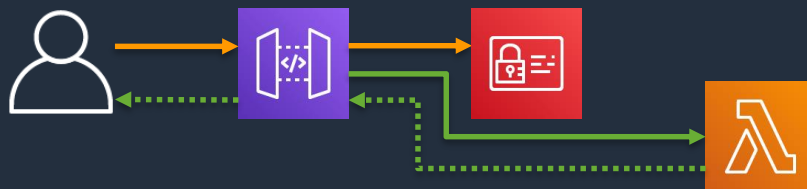


# Amazon API Gateway のアクセス制御 – IAM アクセス権限

## 要約

IAM の持つアクセス権限によってコントロール

- IAM Credential を sigv4 を作成
  - Access key
  - Secret Access key
- sigv4 を含めてリクエスト
- 権限を検証

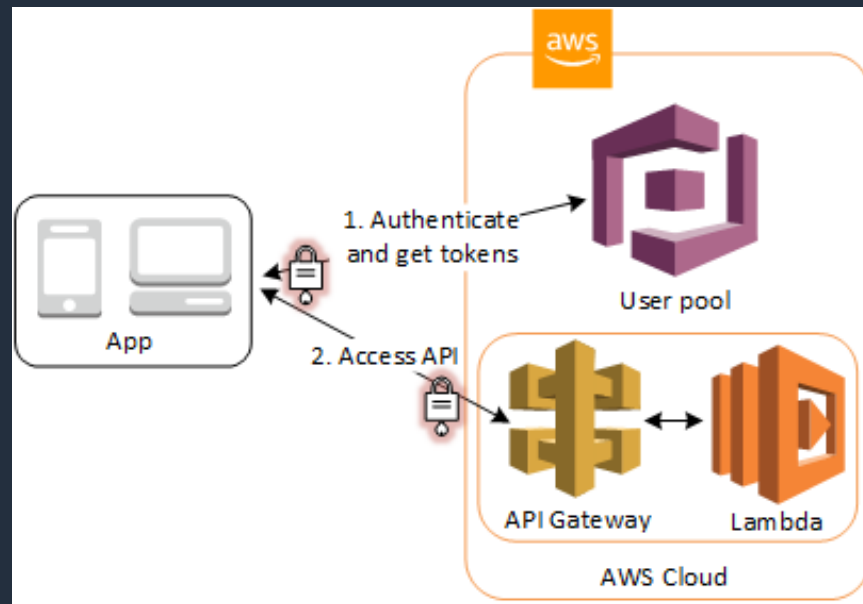


# Amazon API Gateway のアクセス制御 – Cognito UserPool

## 要約

Cognito UserPool がユーザーの認証情報を管理

- ユーザーをサインアップ
- ユーザーをサインイン
- ユーザーの ID トークンを取得
- トークンをリクエストヘッダに含めてリクエスト



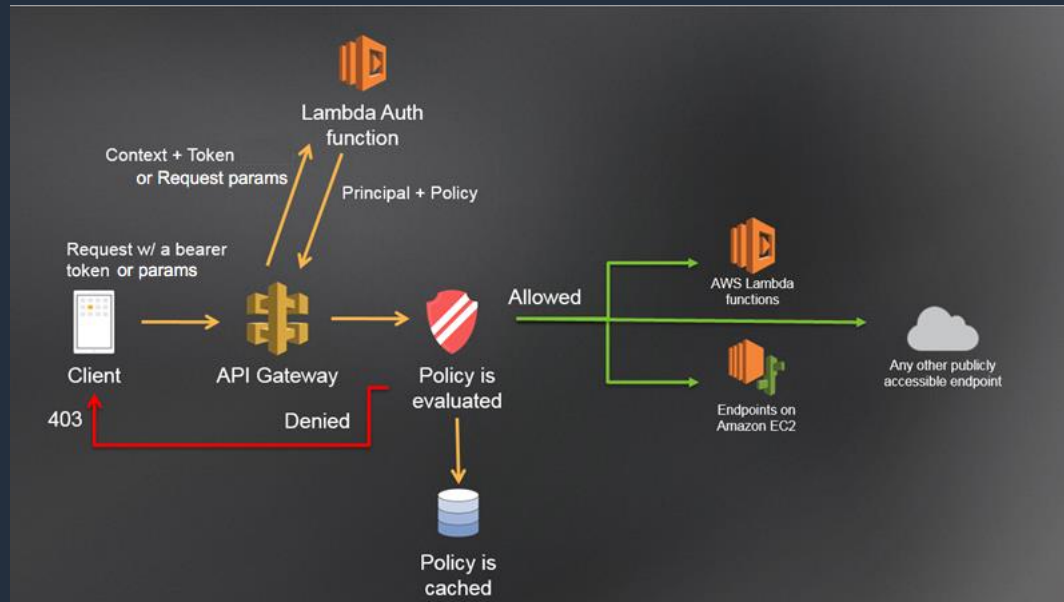
<https://docs.aws.amazon.com/cognito/latest/developerguide/user-pool-accessing-resources-api-gateway-and-lambda.html>

# Amazon API Gateway のアクセス制御 – Lambda Authorizer

## 要約

AWS Lambda がポリシーを返してそれを基にアクセス制御が行われる

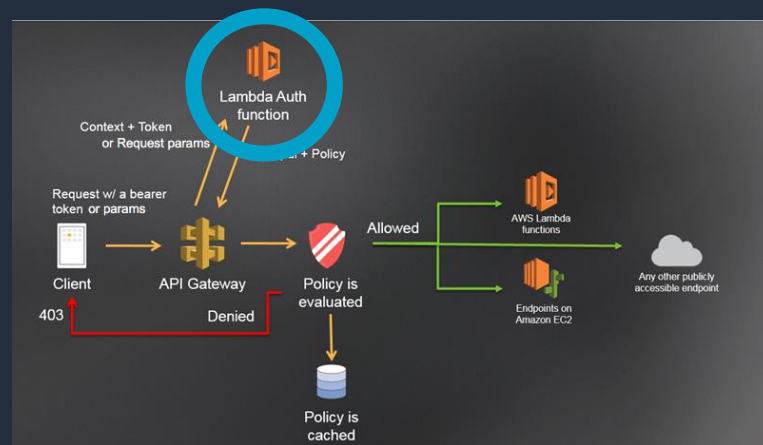
- クライアントから認証トークンやパラメータを受け取る
- Authorizer 関数をキック
- Authorizer 関数内で認証認可ロジックを評価
- API Gateway が返却されたポリシーを評価



[https://docs.aws.amazon.com/ja\\_jp/apigateway/latest/developerguide/apigateway-use-lambda-authorizer.html](https://docs.aws.amazon.com/ja_jp/apigateway/latest/developerguide/apigateway-use-lambda-authorizer.html)

# Lambda Authorizer - 実装例

```
def authorize(event, context):  
    user_id = event['pathParameters']['id']  
    token = event['headers']['Authorization']  
    effect = 'Allow' if is_authorized(user_id, token) else 'Deny'  
  
    return {  
        'principalId': user_id,  
        'policyDocument': {  
            'Version': '2012-10-17',  
            'Statement': [{  
                'Action': 'execute-api:Invoke',  
                'Effect': effect,  
                'Resource': event['methodArn'],  
            }]  
        }  
    }
```



# Lambda Authorizer - 実装例

```
def authorize(event, context):  
    user_id = event['pathParameters']['id']  
    token = event['headers']['Authorization']  
    effect = 'Allow' if is_authorized(user_id, token) else 'Deny'
```

- *event* と *context* を受け取るのはお約束どおり

```
return {  
    'principalId': user_id,  
    'policyDocument': {  
        'Version': '2012-10-17',  
        'Statement': [{  
            'Action': 'execute-api:Invoke',  
            'Effect': effect,  
            'Resource': event['methodArn'],  
        }]  
    }  
}
```

- 返却すべきオブジェクト構成
  - プリンシパル ID
  - ロジック内で生成された IAM ポリシー

# Lambda Authorizer - 実装例

```
def authorize(event, context):  
    user_id = event['pathParameters']['id']  
    token = event['headers']['Authorization']  
    effect = 'Allow' if is_authorized(user_id,  
  
    return {  
        'principalId': user_id,  
        'policyDocument': {  
            'Version': '2012-10-17',  
            'Statement': [{  
                'Action': 'execute-api:Invoke',  
                'Effect': effect,  
                'Resource': event['methodArn'],  
            }]  
        }  
    }
```

• *event* は Amazon API Gateway からのペイロードを含む

• すなわちエンドポイントにマッピングされている関数と同じ内容が取得出来る

# Lambda Authorizer - 実装例

```
def authorize(event, context):
    user_id = event['pathParameters']['id']
    token = event['headers']['Authorization']
    effect = 'Allow' if is_authorized(user_id, token) else 'Deny'

    return {
        'principalId': user_id,
        'policyDocument': {
            'Version': '2012-10-17',
            'Statement': [{
                'Action': 'execute-api:Invoke',
                'Effect': effect,
                'Resource': event['methodArn'],
            }]
        }
    }
```

- 認証認可ロジックに *user\_id*, *token* を渡す
- その結果によって *Allow* か *Deny* を決める

たとえば

- 決定された可否を *Effect* に



# Lambda Authorizer - 実装例

Important!!

```
def authorize(event, context):
    user_id = event['pathParameters']['id']
    token = event['headers']['Authorization']
    effect = 'Allow' if is_authorized(user_id, token) else 'Deny'

    return {
        'principalId': user_id,
        'policyDocument': {
            'Version': '2012-10-17',
            'Statement': [{
                'Action': 'execute-api:Invoke',
                'Effect': effect,
                'Resource': event['methodArn'],
            }]
        }
    }
```

- アクションにはワイルドカードを指定することも可能
- しかしながら明示的に指定することが重要

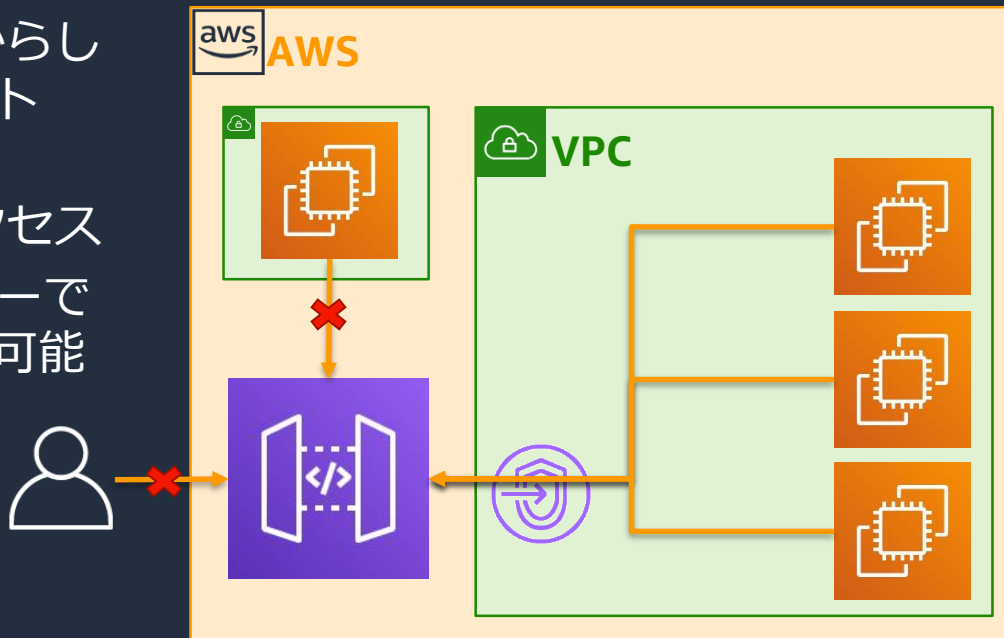
- リソースも同様に明示的に指定する
- `event['methodArn']` はまさにリクエストをしようとしているエンドポイントパスと HTTP メソッドが明示された識別子

# Amazon API Gateway のアクセス制御 – Private API

## 要約

VPC エンドポイント経由でのみリクエストを受け付けるプライベートな API

- Amazon Virtual Private Cloud からしかアクセスできないプライベート API
- VPC エンドポイント経由でアクセス
- API Gateway のリソースポリシーで柔軟なアクセスコントロールが可能
- TLS 1.2 のみをサポート



# Amazon API Gateway のアクセス制御 – Private API

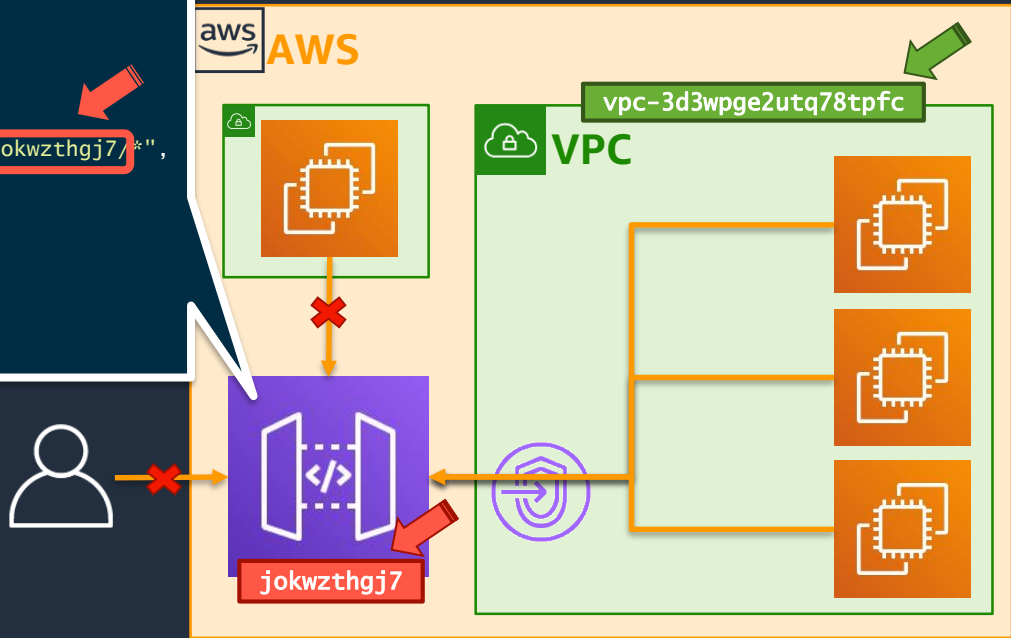
## 要約

VPC エンドポイント経由でのみリクエストを受け付けるプライベートな API

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Principal": "*",
      "Action": "execute-api:Invoke",
      "Resource": "arn:aws:execute-api:{region}:{accountId}:jokwzthgj7/*",
      "Condition": {
        "StringNotEquals": {
          "aws:sourceVpc": "vpc-3d3wpge2utq78tpfc"
        }
      }
    }
  ]
}
```

リソースポリシー

API Gateway のリソースポリシーで  
アクセス可能な VPC を限定出来る

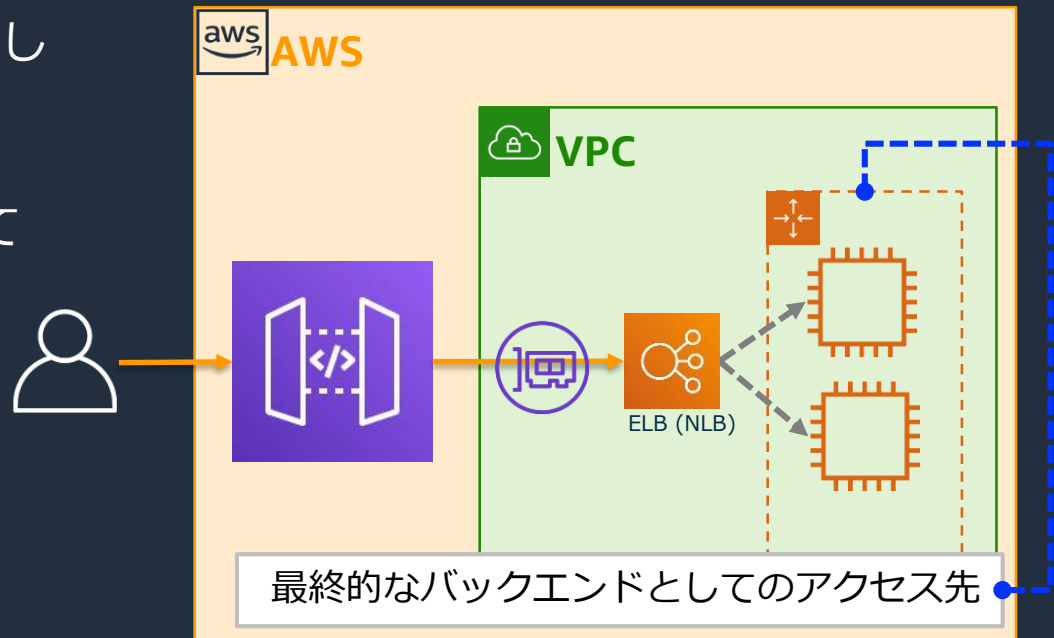


# Amazon API Gateway のアクセス制御 – VPC リンク

## 要約

Amazon API Gateway から IGW のない VPC  
あるいは Private Subnet にリンク可能

- API Gateway のバックエンドとして VPC 内のリソースを扱える
- VPC リンクと Elastic Load Balancing (NLB) を組み合わせて連携



# サーバレスなサービス利用時のお客様の関心事

## アクセス制御

- ・ リソースの構成（作成・変更）権限の制御
- ・ リソース内のデータへのアクセス権限の制御

## データ保護

- ・ 通信の暗号化の有無
- ・ 保存データの暗号化の有無

## インフラ保護

- ・ DDoS対策/ファイアウォール設置
- ・ 脆弱性診断

## ロギング・ モニタリング

- ・ サービスのログの利用
- ・ アプリ部分のログの利用
- ・ 監視対象項目の選定

<https://aws.amazon.com/jp/compliance/shared-responsibility-model/>

# 保存データの暗号化

## 要約

AWS Key Management Service と統合されている場合  
AWS KMS の Customer Master Key を利用した保存時暗号化が可能


### Amazon DynamoDB

- デフォルトでは [AWS 所有の CMK](#) を使用
- AWS アカウントにある DynamoDB (*aws/dynamodb*) 用の [AWS 管理 CMK](#) も使用可能
- [お客様管理の CMK](#) は使用不可

### Amazon S3

- デフォルトでは暗号化されない
- (SSE-KMS を選択の場合)
- AWS アカウントにある S3 (*aws/s3*) 用の [AWS 管理 CMK](#) が使用可能
  - [お客様管理の CMK](#) も使用可能

### AWS Lambda 環境変数

- デフォルトでは AWS アカウントにある Lambda (*aws/lambda*) 用の [AWS 管理 CMK](#) を使用
- [お客様管理の CMK](#) も使用可能
- マネジメントコンソール上で暗号化ヘルパーを使う場合は [お客様管理の CMK](#) を指定 

# 保存データの暗号化

## 要約

AWS Key Management Service と統合されている場合  
AWS KMS の Customer Master Key を利用した保存時暗号化が可能

### Amazon DynamoDB

- デフォルトでは [AWS 所有の CMK](#) を使用
- AWS アカウントにある DynamoDB (*aws/dynamodb*) 用の [AWS 管理 CMK](#) も使用可能
- [お客様管理の CMK](#) は使用不可

### Amazon S3

- デフォルトでは暗号化されない
- (SSE-KMS を選択の場合)
- AWS アカウントにある S3 (*aws/s3*) 用の [AWS 管理 CMK](#) が使用可能
  - [お客様管理の CMK](#) も使用可能

### AWS Lambda 環境変数

- デフォルトでは AWS アカウントにある Lambda (*aws/lambda*) 用の [AWS 管理 CMK](#) を使用
- [お客様管理の CMK](#) も使用可能
- マネジメントコンソール上で暗号化ヘルパーを使う場合は [お客様管理の CMK](#) を指定

# 保存データの暗号化

## 要約

AWS Key Management Service と統合されている場合  
AWS KMS の Custom CMK と AWS 管理 CMK の違い

が可能

## Amazon DynamoDB

- デフォルトでは [AWS 所有の CMK](#) を使用
- AWS アカウントにある DynamoDB (`aws/dynamodb`) 用の [AWS 管理 CMK](#) も使用可能
- [お客様管理の CMK](#) は使用不可

- [AWS 所有 CMK](#)
  - お客様の AWS アカウントからは見えない
  - 複数の AWS アカウントで使用される
- [AWS 管理 CMK](#)
  - お客様のアカウント専用
  - 統合サービスがお客様の代わりに管理

環境変数

は AWS  
ある  
(`lambda`)

(SSE-KMS を選択の場合)

- AWS アカウントにある S3 (`aws/s3`) 用の [AWS 管理 CMK](#) が使用可能
- [お客様管理の CMK](#) も使用可能

用の [AWS 管理 CMK](#) を使用

- [お客様管理の CMK](#) も使用可能
- マネジメントコンソール上で暗号化ヘルパーを使う場合は [お客様管理の CMK](#) を指定





# 保存データの暗号化

## 要約

AWS Key Management Service と統合されている場合  
AWS KMS の Custom CMK と AWS 管理 CMK の違い

## Amazon DynamoDB

- デフォルトでは [AWS 所有の CMK](#) を使用
- AWS アカウントにある DynamoDB (`aws/dynamodb`) 用の [AWS 管理 CMK](#) も使用可能
- [お客様管理の CMK](#) は使用不可

- **AWS 所有 CMK**
  - お客様の AWS アカウントからは見えない
  - 複数の AWS アカウントで使用される
- **AWS 管理 CMK**
  - お客様のアカウント専用
  - 統合サービスがお客様の代わりに管理

いずれかの要件にマッチする場合は **AWS 管理 CMK** を使う

- CMK のポリシーを参照する必要がある
- AWS CloudTrail で AWS KMS への DynamoDB テーブルの暗号化と復号化を監査したい

# 保存データの暗号化

## 要約

AWS Key Management Service と統合されている場合  
AWS KMS の Customer Master Key を利用した保存時暗号化が可能


### Amazon DynamoDB

- デフォルトでは [AWS 所有の CMK](#) を使用
- AWS アカウントにある DynamoDB (*aws/dynamodb*) 用の [AWS 管理 CMK](#) も使用可能
- [お客様管理の CMK](#) は使用不可

### Amazon S3

- デフォルトでは暗号化されない
- (SSE-KMS を選択の場合)
- AWS アカウントにある S3 (*aws/s3*) 用の [AWS 管理 CMK](#) が使用可能
  - [お客様管理の CMK](#) も使用可能

### AWS Lambda 環境変数

- デフォルトでは AWS アカウントにある Lambda (*aws/lambda*) 用の [AWS 管理 CMK](#) を使用
- [お客様管理の CMK](#) も使用可能
- マネジメントコンソール上で暗号化ヘルパーを使う場合は [お客様管理の CMK](#) を指定 

# 保存データの暗号化

## 要約

AWS Key Management Service と統合されている場合  
AWS KMS の Customer Master Key を利用した保存時暗号化が可能


### Amazon DynamoDB

- デフォルトでは [AWS 所有の CMK](#) を使用
- AWS アカウントにある DynamoDB (*aws/dynamodb*) 用の [AWS 管理 CMK](#) も使用可能
- [お客様管理の CMK](#) は使用不可

### Amazon S3

- デフォルトでは暗号化されない
- (SSE-KMS を選択の場合)
- AWS アカウントにある S3 (*aws/s3*) 用の [AWS 管理 CMK](#) が使用可能
  - [お客様管理の CMK](#) も使用可能

### AWS Lambda 環境変数

- デフォルトでは AWS アカウントにある Lambda (*aws/lambda*) 用の [AWS 管理 CMK](#) を使用
- [お客様管理の CMK](#) も使用可能
- マネジメントコンソール上で暗号化ヘルパーを使う場合は [お客様管理の CMK](#) を指定 

# 保存データの暗号化

## 要約

AWS Key Management Service と統合されている場合  
AWS KMS の Customer Master Key を利用した保存時暗号化が可能

### 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。これにより、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

FOO	BAR	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:ap-northeast-1:123456789012:key:12345678-9012-3456-7890-123456789012

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

arn:aws:kms:ap-northeast-1:123456789012:key:12345678-9012-3456-7890-123456789012

## AWS Lambda 環境変数

- デフォルトでは AWS アカウントにある Lambda (`aws/lambda`) 用の [AWS 管理 CMK](#) を使用
- [お客様管理の CMK](#) も使用可能
- マネジメントコンソール上で暗号化ヘルパーを使う場合は [お客様管理の CMK](#) を指定 

# 保存データの暗号化

## 要約

AWS Key Management Service と統合されている場合  
AWS KMS の Customer Master Key を利用した保存時暗号化が可能

### 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](#)

FOO	.....	復号	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:ap-northeast-1:123456789012:key:12345678-9012-3456-7890-123456789012


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

arn:aws:kms:ap-northeast-1:123456789012:key:12345678-9012-3456-7890-123456789012

## AWS Lambda 環境変数

- デフォルトでは AWS アカウントにある Lambda (`aws/lambda`) 用の [AWS 管理 CMK](#) を使用
- [お客様管理の CMK](#) も使用可能
- マネジメントコンソール上で暗号化ヘルパーを使う場合は [お客様管理の CMK](#) を指定 

# 保存データの暗号化

## 要約

## AWS Key Management Service と統合されている場合

使用しようとしている CMK の キーポリシー でマネジメントコンソールにログインしているユーザーにアクションを許可していないとエラーとなる

- 暗号化ボタンを押したが `kms:Encrypt` が許可されていなかった場合

[Using Key Policies in AWS KMS](#)

### ▼ 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:ap-northeast-1

⚠ AWS KMS 呼び出しが失敗した理由: User: arn:aws:iam: is not authorized to perform `kms:Encrypt` in resource: arn:aws:kms:ap-northeast-1

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

arn:aws:kms:ap-northeast-1

### キーポリシー

編集

```
37     },
38     {
39         "Sid": "Allow use of the key",
40         "Effect": "Allow",
41         "Principal": {
42             "AWS": "arn:aws:iam:::user/arn:"
43         },
44         "Action": [
45             "kms:Decrypt",
46             "kms:ReEncrypt*",
47             "kms:GenerateDataKey*",
48             "kms:DescribeKey"
49         ],
50         "Resource": "*"
51     },
52 }
```

用可能

- マネジメントコンソール上で暗号化ヘルパーを使う場合は [お客様管理の CMK](#) を指定 [aws](#)

# 保存データの暗号化

## 要約

AWS Key Management Service と統合されている場合  
AWS KMS の Customer Master Key を利用した保存時暗号化

復号化

### 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](#)

FOO	.....	復号	コード	削除
キー	値	暗号化	コ	

```
import boto3
import os

from base64 import b64decode

ENCRYPTED = os.environ['FOO']
# Decrypt code should run once and variables stored outside of the function
# handler so that these are decrypted once per container
DECRYPTED = boto3.client('kms').decrypt(CiphertextBlob=b64decode(ENCRYPTED))['Plaintext']

def lambda_handler(event, context):
    # TODO handle the event here
    pass
```

## AWS Lambda 環境変数

- 「コード」ボタンを押下すると復号化実装のサンプルが表示される
- 関数の言語設定ごとに
- 実際に関数内で動作させるためには適切な権限設定が必要

# 保存データの暗号化

復号化

## 要約

AWS Key Management Service と統合されている場合  
AWS KMS の Customer Master Key を利用した保存時暗号化

### 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

Lambda 関数の IAM ロールに `kms:Decrypt` を許可するポリシーが必要

```
import boto3
import os

from base64 import b64decode

ENCRYPTED = os.environ['FOO']
# Decrypt code should run once and variables stored outside of the function
# handler so that these are decrypted once per container
DECRYPTED = boto3.client('kms').decrypt(CiphertextBlob=b64decode(ENCRYPTED))['Plaintext']

def lambda_handler(event, context):
    # TODO handle the event here
    pass
```

## AWS Lambda 環境変数

- 「コード」ボタンを押下すると復号化実装のサンプルが表示される
- 関数の言語設定ごとに
- 実際に関数内で動作させるためには適切な権限設定が必要



# 保存データの暗号化

復号化

## 要約

AWS Key Management Service と統合されている場合  
AWS KMS の Customer Master Key を利用した保存時暗号化

CMK のキーポリシーにもこのロールに `kms:Decrypt` を許可するポリシーが必要

[Using Key Policies in AWS KMS](#)

環境変数を、関数コードからアクセスして取得することができます。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

Lambda 関数の IAM ロールに `kms:Decrypt` を許可するポリシーが必要

```
import boto3
import os

from base64 import b64decode

ENCRYPTED = os.environ['FOO']
# Decrypt code should run once and variables stored outside of the function
# handler so that these are decrypted once per container
DECRYPTED = boto3.client('kms').decrypt(CiphertextBlob=b64decode(ENCRYPTED))['Plaintext']

def lambda_handler(event, context):
    # TODO handle the event here
    pass
```

## キーポリシー

編集

```
1 {
2   "Version": "2012-10-17",
3   "Id": "key-consolepolicy-3",
4   "Statement": [
5     {
6       "Sid": "Enable IAM User Permissions",
7       "Effect": "Allow",
8       "Principal": {
9         "AWS": "arn:aws:iam::[REDACTED]:role/service-role/[REDACTED]"
10      },
11      "Action": "kms:Decrypt",
12      "Resource": "*"
13    },
14    {
15      "Sid": "Allow access for Key Administrators",
```

# 保存データの暗号化

## 要約

要件に応じてクライアントサイド暗号化も可能

### Amazon DynamoDB

- 相互互換のあるクライアント実装
- e.g.) Java クライアントでデータを暗号化して Python クライアントでデータを復号化出来る
- KMS による暗号化にも対応

[What Is the Amazon DynamoDB Encryption Client?](#)

### Amazon S3

- AWS KMS に保存されているマスターキーを使用
- もしくはアプリケーション内に保存するマスターキーを使用

[Protecting Data Using Client-Side Encryption](#)

### AWS Lambda 環境変数

- 専用クライアントの提供はない
- AWS KMS API を使って事前に暗号化

[Programming the AWS KMS API](#)

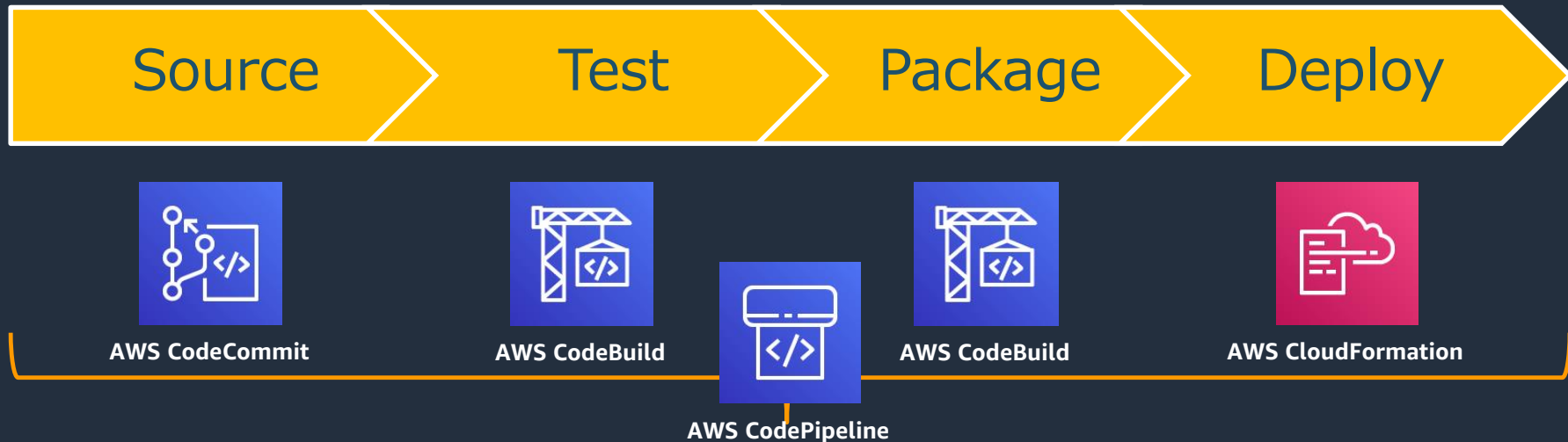
# セキュアなアプリケーション運用のための自動化を検討

## 要約

作業者に割り当てる IAM ユーザーのポリシー管理を徹底

- 手動オペレーションの危険性を知る
  - 開発者全員がマネジメントコンソールで Lambda のコードを書きますか？
  - 開発者全員がマネジメントコンソールで設定を管理しますか？
- 権限を適切に統制する
  - マネジメントコンソールのアクセス権限コントロール
  - AWS CLI のための IAM ユーザーに適切なポリシーを適用
- 適切に自動化して人の手を介在させない
  - CI/CD パイプラインを設ける

# CI/CD 構成例



- ロジック実装
- テスト実装
- ブランチ管理
  - ローカル
  - リモート
- プルリクエスト
- ピアレビュー

- ユニットテスト
- Lint チェック
- 脆弱性チェック

- デプロイパッケージ作成
- S3 にアップロード

- AWS CloudFormation
  - ChangeSet 作成
  - ChangeSet 実行



# サーバレスなサービス利用時のお客様の関心事

## アクセス制御

- ・ リソースの構成（作成・変更）権限の制御
- ・ リソース内のデータへのアクセス権限の制御

## データ保護

- ・ 通信の暗号化の有無
- ・ 保存データの暗号化の有無

## インフラ保護

- ・ DDoS対策/ファイアウォール設置
- ・ 脆弱性診断

## ロギング・ モニタリング

- ・ サービスのログの利用
- ・ アプリ部分のログの利用
- ・ 監視対象項目の選定

<https://aws.amazon.com/jp/compliance/shared-responsibility-model/>

# 脆弱性テストと侵入テスト

## 事前申請が不要なサービス

- Amazon EC2 インスタンス、NAT ゲートウェイ、Elastic Load Balancer
- Amazon RDS
- Amazon CloudFront
- Amazon Aurora
- Amazon API Gateway
- AWS Lambda 関数および Lambda Edge 関数
- Amazon Lightsail リソース
- Amazon Elastic Beanstalk 環境

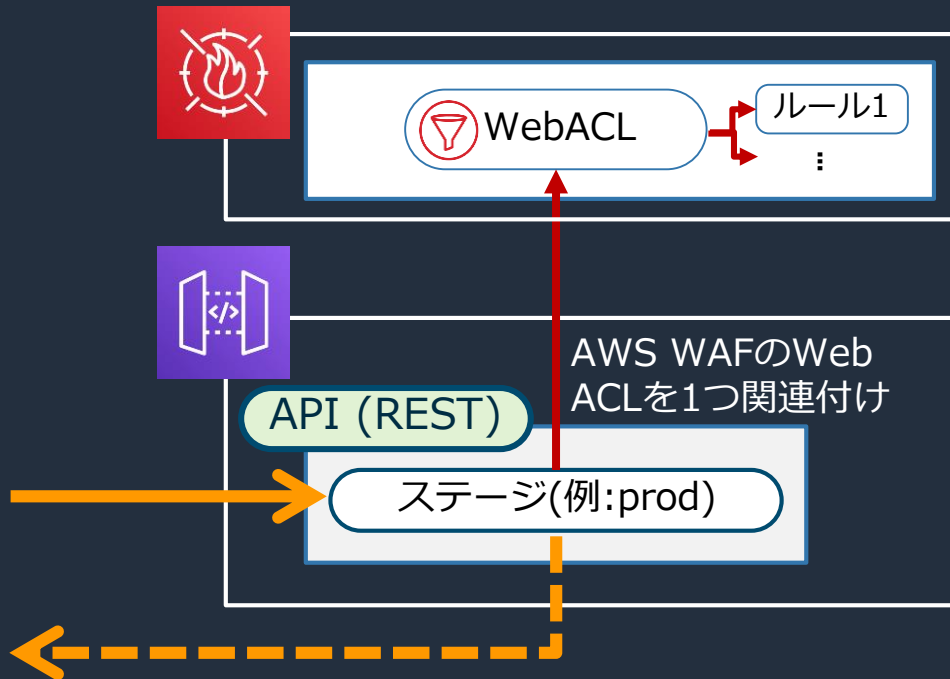
## 禁止アクティビティ

- Amazon Route 53 ホストゾーン経由の DNS ゾーンウォーキング
- サービス妨害 (DoS)、分散サービス妨害 (DDoS)、シミュレートされた DoS、シミュレートされた DDoS
- ポートフラッディング
- プロトコルフラッディング
- リクエストフラッディング (ログインリクエストフラッディング、API リクエストフラッディング)

# AWS WAFで Amazon API Gateway の REST API を保護

## 要約

REST APIでは APIステージに AWS WAF の WebACL を指定可能



- AWS WAFのWeb API保護を利用
  - SQLインジェクションやXSSなどの攻撃からAPIを保護
- アクセスポリシーや認証など、その他のアクセス制御よりも前にWebACLによって評価される
- APIステージ設定UI または WAF WebACLの新規作成画面でAPI GatewayのAPI/ステージを指定

# サーバレスなサービス利用時のお客様の関心事

## アクセス制御

- ・ リソースの構成（作成・変更）権限の制御
- ・ リソース内のデータへのアクセス権限の制御

## データ保護

- ・ 通信の暗号化の有無
- ・ 保存データの暗号化の有無

## インフラ保護

- ・ DDoS対策/ファイアウォール設置
- ・ 脆弱性診断

## ロギング・ モニタリング

- ・ サービスのログの利用
- ・ アプリ部分のログの利用
- ・ 監視対象項目の選定

<https://aws.amazon.com/jp/compliance/shared-responsibility-model/>



# AWS Lambda ログ

- AWS Lambda のログは Amazon CloudWatch Logs に記録される
- アプリケーション内部でロガーライブラリなどによって出力されるログも同様
- Lambda 自身が書き込むログは極めてシンプルな情報のみ

時間 (UTC +09:00)	メッセージ
2019-08-13	
04:39:25	いまのところ古いイベントはありません。 <a href="#">再試行</a> 。 START RequestId: bdfd2ae0-3a20-4165-86a1-403edb4a1603 Version: \$LATEST
	START RequestId: bdfd2ae0-3a20-4165-86a1-403edb4a1603 Version: \$LATEST
04:39:25	Something has been happened: RuntimeError Traceback (most recent call last): File "/va Something has been happened: RuntimeError Traceback (most recent call last): File "/var/task/lambda_function.py", line 14, in lambda_handler raise RuntimeError('Something has been happened') RuntimeError: Something has been happened
04:39:25	END RequestId: bdfd2ae0-3a20-4165-86a1-403edb4a1603
	END RequestId: bdfd2ae0-3a20-4165-86a1-403edb4a1603
04:39:25	REPORT RequestId: bdfd2ae0-3a20-4165-86a1-403edb4a1603 Duration: 1.36 ms Billed REPORT RequestId: bdfd2ae0-3a20-4165-86a1-403edb4a1603 Duration: 1.36 ms Billed Duration: 100 ms Memory Size: 128 MB Max Memory Used: 67 MB

# AWS Lambda ログ

## 要約

ログを Amazon CloudWatch Logs に書き込むためには適切なポリシーを割り当てる必要がある

- PolicyDocument:

Version: 2012-10-17

Statement:

- Effect: Allow

Action:

- logs:CreateLogGroup
- logs:CreateLogStream
- logs:PutLogEvents

Resource:

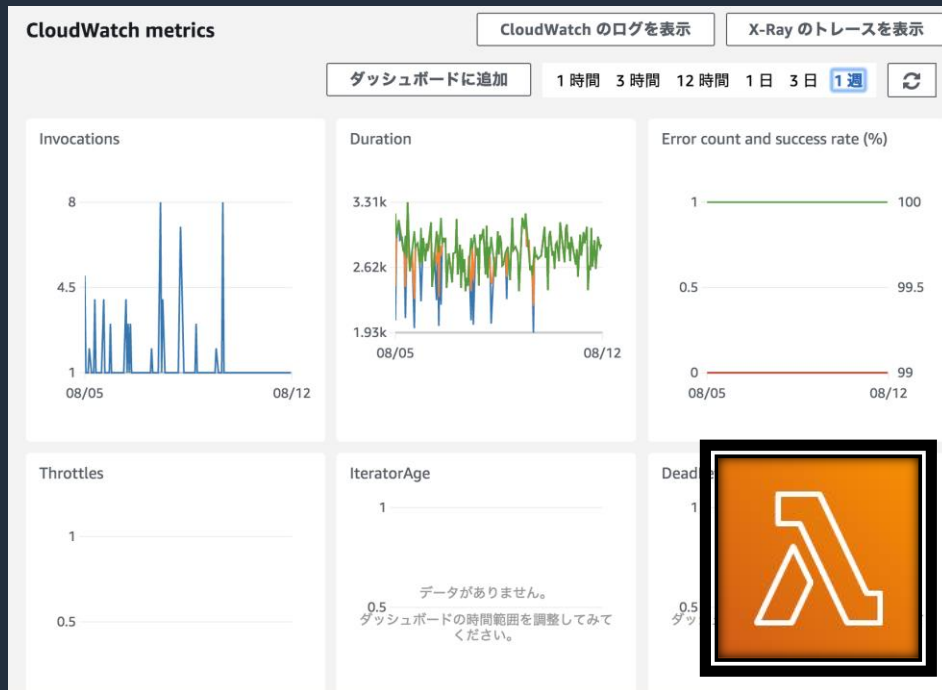
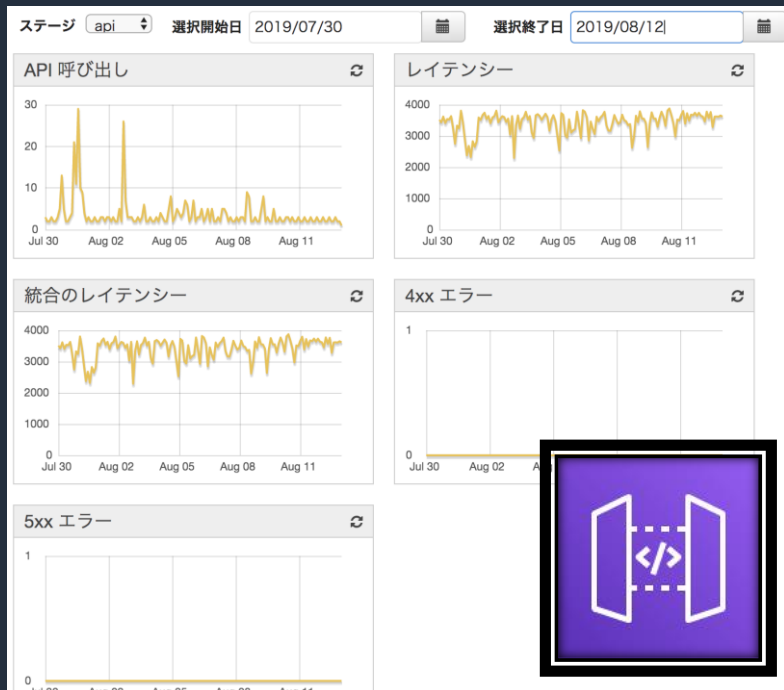
- arn:aws:logs:\*:\*:\*

- CloudWatch Logs に書き込む権限
- *AWSLambdaBaseExecutionRole* と同内容
- *Effect: Allow* なので許可されている

# Amazon API Gateway / AWS Lambda のメトリクス

## 要約

Amazon CloudWatch に統合された  
メトリクスを各ダッシュボードからも参照可能



# AWS X-Ray による分散トレーシングを有効化

## 要約

可視化されたプロセスフローでリクエストとレスポンスをトレーシング

- マネジメントコンソールで有効化



- SAM の各リソースの *Properties* 配下に追加
  - `AWS::Serverless::Api`
    - `TracingEnabled: true`
  - `AWS::Serverless::Function`
    - `Tracing: Active`

# AWS X-Ray による分散トレーシングを有効化

## 要約

可視化されたプロセスフローでリクエストとレスポンスをトレーシング

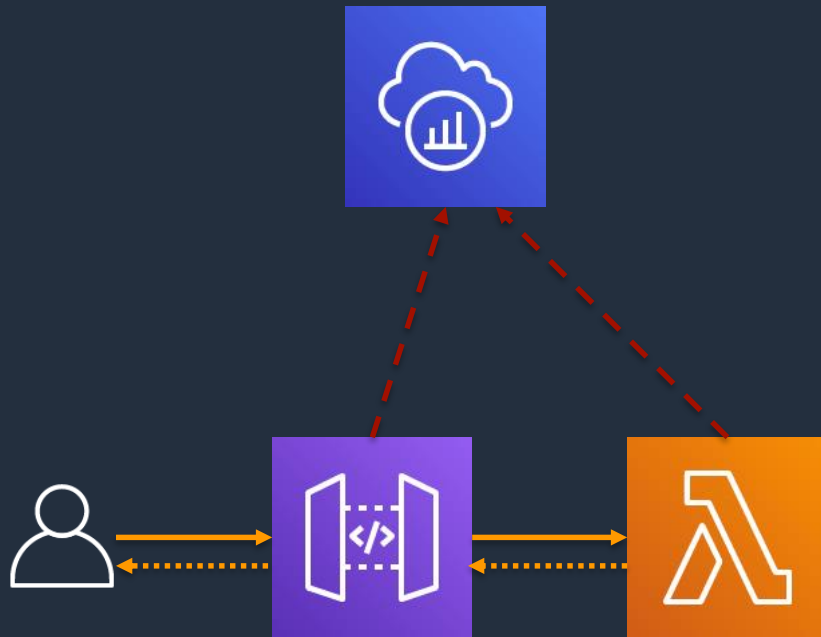
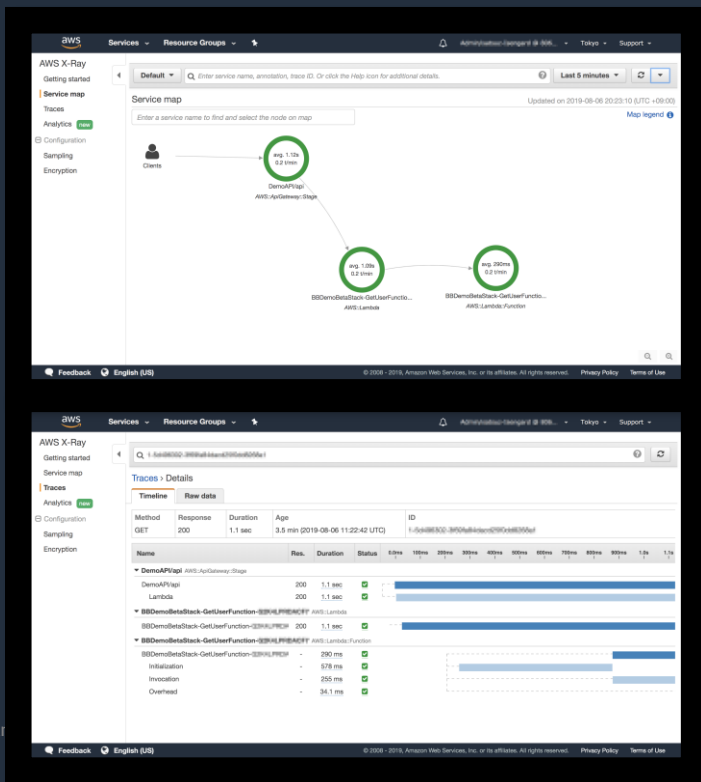
- Lambda 関数に関連付けられている IAM ロールに X-Ray のポリシーを追加
  - *xray:PutTraceSegments*
  - *xray:PutTelemetryRecords*
  - *xray:GetSamplingRules*
  - *xray:GetSamplingTargets*
  - *xray:GetSamplingStatisticSummaries*

<https://github.com/aws-labs/serverless-application-model/blob/master/versions/2016-10-31.md>

# AWS X-Ray による分散トレーシングを有効化

## 要約

可視化されたプロセスフローでリクエストとレスポンスをトレーシング



# AWS X-Ray によるトレーシング対象を追加

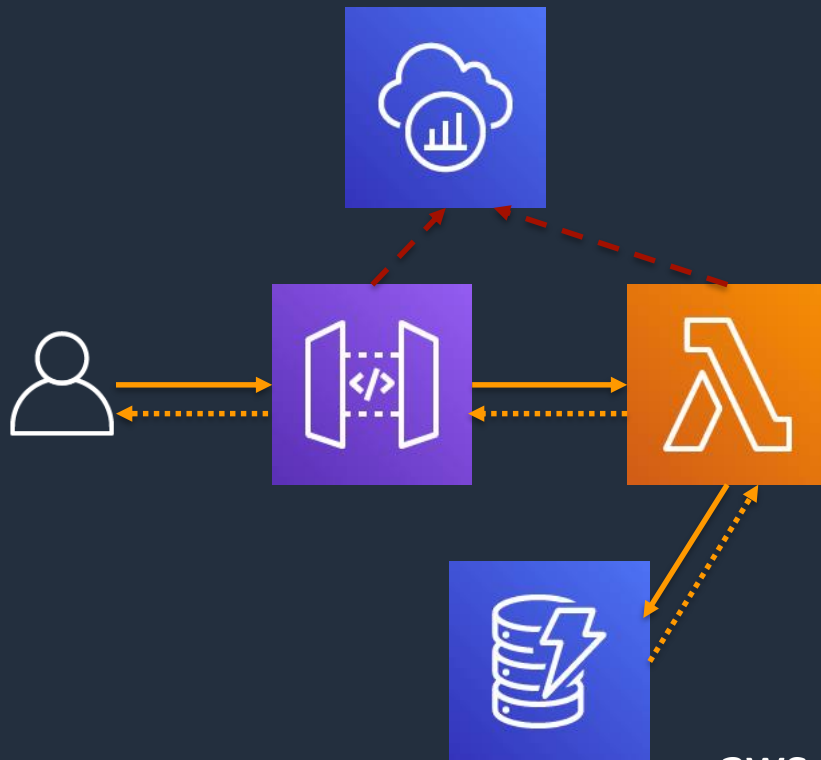
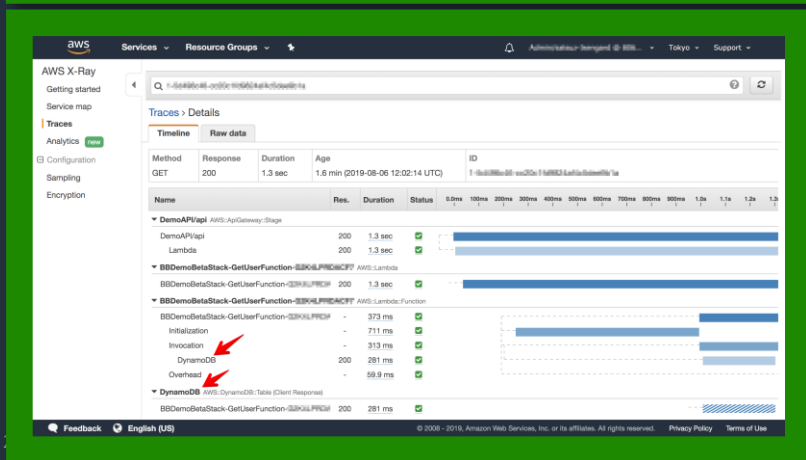
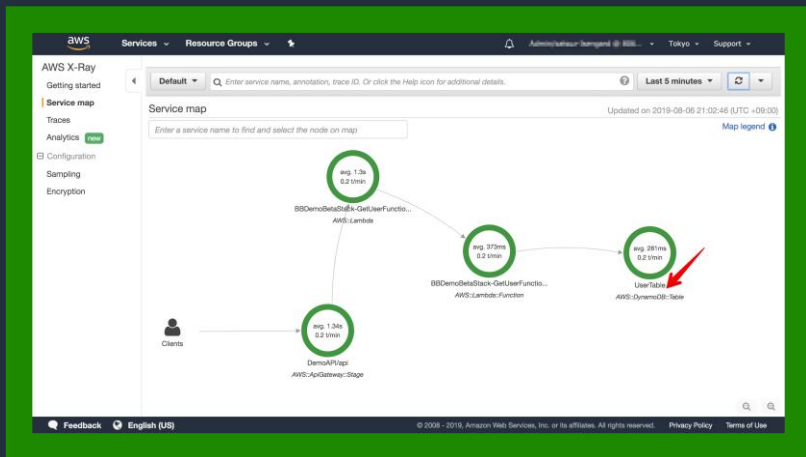
- Amazon DynamoDB へのアクセスもトレーシングの対象に追加したい場合
  - AWS X-Ray SDK for Python の場合
  - アプリケーションの実装に以下のいずれかの定義を追加

```
from aws_xray_sdk.core import patch_all
patch_all()
```

```
from aws_xray_sdk.core import patch
patch(('botocore', 'boto3'))
```

[https://docs.aws.amazon.com/ja\\_jp/xray/latest/devguide/xray-sdk-python.html](https://docs.aws.amazon.com/ja_jp/xray/latest/devguide/xray-sdk-python.html)

# AWS X-Ray によるトレーシング対象を追加





# aws\_xray\_sdk.core.patch, aws\_sdk\_core.patch\_all の補足

- サポートされているライブラリをパッチ
- それらの呼び出し時にリクエストとレスポンスを AWS X-Ray に記録
- 特定のライブラリのみパッチしたい場合は *patch*
- サポートされているすべてのライブラリを一気にパッチしたい場合は *patch\_all*

## サポートされているライブラリ

- *botocore, boto3*
- *pynamodb*
- *aiobotocore, aioboto3*
- *requests, aiohttp*
- *httplib, http.client*
- *sqlite3*
- *mysql-connector-python*

```
from aws_xray_sdk.core import patch_all
patch_all()
```

```
from aws_xray_sdk.core import patch
patch(('botocore', 'boto3'))
```

[https://docs.aws.amazon.com/ja\\_jp/xray/latest/devguide/xray-sdk-python-patching.html](https://docs.aws.amazon.com/ja_jp/xray/latest/devguide/xray-sdk-python-patching.html)

# AWS CloudTrail

## 要約

AWS の API 操作の操作内容を記録 (サーバーレス固有ではない)

- AWS の操作履歴を **90日間** 保存
  - 管理イベントのみ
- API コールをキャプチャ
  - マネジメントコンソール
  - AWS CLI
  - AWS SDK
  - API 直接呼び出し
- AWS アカウント開設時に自動的に有効化

The screenshot displays the AWS CloudTrail console interface. At the top, there are filter options: '読み取り専用' (Read-only) set to 'false' and a '時間範囲' (Time range) selector. Below this is a table of events with columns for 'イベント時間' (Event time), 'ユーザー名' (User name), 'イベント名' (Event name), 'リソースタイプ' (Resource type), and 'リソース名' (Resource name). The table lists several events, including 'CreateTrail', 'PutBucketPolicy', 'StartLogging', 'PutEventSelectors', and 'CreateBucket'. A detailed view of a 'CreateTrail' event is shown below the table, including fields for 'AWS アクセスキー', 'AWS リージョン', 'エラーコード', 'イベント ID', 'イベント名', 'イベントソース', 'イベント時間', '読み取り専用', 'リクエスト ID', '発信元 IP アドレス', and 'ユーザー名'. At the bottom, there is a section for '参照リソース (3)' (Referenced resources) with a table listing 'CloudTrail Trail', 'CloudTrail Trail', and 'S3 Bucket' with their respective resource names and configuration links.

イベント時間	ユーザー名	イベント名	リソースタイプ	リソース名
2019-08-11, 08:13:01 PM	aws:iam/arn:aws:iam::123456789012:user/iam-user-1	CreateTrail	CloudTrail Trail および 1 項目	arn:aws:cloudtrail:us-east-1:123456789012:trail/trail-1
2019-08-11, 08:13:01 PM	aws:iam/arn:aws:iam::123456789012:user/iam-user-1	PutBucketPolicy	S3 Bucket	arn:aws:s3:::bucket-1
2019-08-11, 08:13:01 PM	aws:iam/arn:aws:iam::123456789012:user/iam-user-1	StartLogging	CloudTrail Trail	arn:aws:cloudtrail:us-east-1:123456789012:trail/trail-1
2019-08-11, 08:13:01 PM	aws:iam/arn:aws:iam::123456789012:user/iam-user-1	PutEventSelectors	CloudTrail Trail	arn:aws:cloudtrail:us-east-1:123456789012:trail/trail-1
2019-08-11, 08:13:00 PM	aws:iam/arn:aws:iam::123456789012:user/iam-user-1	CreateBucket	S3 Bucket	arn:aws:s3:::bucket-2

リソースタイプ	リソース名	Config のタイムライン
CloudTrail Trail	arn:aws:cloudtrail:us-east-1:123456789012:trail/trail-1	<⌂
CloudTrail Trail	arn:aws:cloudtrail:us-east-1:123456789012:trail/trail-1	<⌂
S3 Bucket	arn:aws:s3:::bucket-1	<⌂

# AWS CloudTrail

## 要約

## AWS の API 操作の操作内容を記録（サーバーレス固有ではない）

### 証跡の有効化

- S3 バケットにログを保存
- 対象リージョンを限定可能
- 管理イベント種別を限定可能
- データイベントを有効化可能
  - Amazon S3
  - AWS Lambda

### 有効化すると統合可能となる

- Amazon CloudWatch Logs
- Amazon Athena

The screenshot shows the AWS CloudTrail console interface. At the top, there's a 'Trail settings' section with a dropdown arrow and an edit icon. Below it, there's a paragraph of Japanese text explaining that trail information is stored in Amazon S3 buckets. There are three main sections: '管理イベント' (Management Events), '読み込み/書き込みイベント' (Data Events), and 'データイベント' (Data Events). The 'データイベント' section is currently selected, and the 'Lambda' tab is active. Below the tabs, there's a table with columns for '機能 (16 個を選択済み)' (Features) and 'リージョン' (Region). The table lists several Lambda functions with checkboxes in the first column and their respective regions in the second column.

機能 (16 個を選択済み)	リージョン
<input checked="" type="checkbox"/> 現在および将来のすべての関数を記録	すべてのリージョン
<input checked="" type="checkbox"/> BBDemoBetaStack-CustomAuthFunction-WDKB0NUSODN	アジアパシフィック (東京)
<input checked="" type="checkbox"/> BBDemoBetaStack-GetUserFunction-G2K4LPRDACF7	アジアパシフィック (東京)
<input checked="" type="checkbox"/> example-dev	米国西部 (オレゴン)
<input checked="" type="checkbox"/> formadata-dev	米国西部 (オレゴン)

# まとめ

# サーバーレスなサービス利用時のお客様の関心事

## アクセス制御

- ・ リソースの構成（作成・変更）権限の制御
- ・ リソース内のデータへのアクセス権限の制御

## データ保護

- ・ 通信の暗号化の有無
- ・ 保存データの暗号化の有無

## インフラ保護

- ・ DDoS 対策/ファイアウォール設置
- ・ 脆弱性診断

## ロギング・ モニタリング

- ・ サービスのログの利用
- ・ アプリ部分のログの利用
- ・ 監視対象項目の選定

<https://aws.amazon.com/jp/compliance/shared-responsibility-model/>

# Q&A

いただいたご質問については

AWS Japan Blog 「<https://aws.amazon.com/jp/blogs/news/>」にて  
後日掲載します。

# AWS の日本語資料の場所「AWS 資料」で検索



日本担当チームへお問い合わせ サポート 日本語 ▼ アカウント ▼

コンソールにサインイン

製品 ソリューション 料金 ドキュメント 学習 パートナー AWS Marketplace その他 🔍

## AWS クラウドサービス活用資料集トップ

アマゾン ウェブ サービス (AWS) は安全なクラウドサービスプラットフォームで、ビジネスのスケールと成長をサポートする処理能力、データベースストレージ、およびその他多種多様な機能を提供します。お客様は必要なサービスを選択し、必要な分だけご利用いただけます。それらを活用するために役立つ日本語資料、動画コンテンツを多数ご提供しております。(本サイトは主に、AWS Webinar で使用した資料およびオンデマンドセミナー情報を掲載しています。)

[AWS Webinar お申込 »](#)

[AWS 初心者向け »](#)

[業種・ソリューション別資料 »](#)

[サービス別資料 »](#)

<https://amzn.to/JPArchive>



# AWS Well-Architected 個別技術相談会

毎週“W-A個別技術相談会”を実施中

- AWSのソリューションアーキテクト(SA)に  
対策などを相談することも可能

- **申込みはイベント告知サイトから**

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

AWS イベント

で[検索]





# ご視聴ありがとうございました

AWS 公式 Webinar

<https://amzn.to/JPWebinar>



過去資料

<https://amzn.to/JPArchive>

