AWS サーバーレスセキュリティご紹介

2021/04/21

アマゾン ウェブ サービス ジャパン株式会社
シニアパートナーソリューションアーキテクト
大場 崇令

Who am I ?

@takaohba

- 大場 崇令（オオバ タカノリ）
  - Partner Solutions Architect @Amazon Web Services Japan K.K. (Joined 2015/12)
- Background
  - AWS テクニカルトレーナー@AWSJ K.K.
  - Web サービスのインフラエンジニア
  - 国内クラウドベンダーにてテクニカルサポート
- 好きな AWS サービス
  - AWS Well-Architected Tool
  - AWS Systems Manager
  - AWS Service Catalog

Well-Architected Lead

aws

ご説明の内容

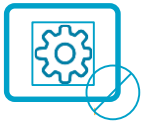
- サーバーレス概要
- サーバーレス 責任共有モデル
- Well-Architected サーバーレスアプリケーションレンズ
- OWASP Serverless Top 10
- コンピュートレイヤーのセキュリティ
- データレイヤーのセキュリティ
- システム管理レイヤーのセキュリティ
- デプロイメントのセキュリティ（CI/CD）

ご説明のスコープ

- 本セッションで取り扱うこと
  - AWS API Gateway, AWS Lambda, AWS DynamoDB, AWS Simple Storage Service などの主要サービスで構成されるサーバーレスアーキテクチャーに関するセキュリティ

- 本セッションで取り扱わないこと
  - コンテナ・アーキテクチャー、
    コンテナ関連ソリューションのセキュリティ
  - AWS Step Functions, AWS Batch などのサービスの詳細

サーバレスとは？



インフラ不要、管理不要

自動スケーリング

利用した分だけ課金

高可用性と安全性

サーバーレスアプリケーションを必要とするビジネス要件

企業における DX （デジタルトラスフォーメーション）を支える
アプリケーションを実行する基盤として短時間で迅速にリリースが出来る
サーバーレスアーキテクチャーは最適

短期間で利用者の需要に対応して変化していく、スマホアプリや、 IoT ソリューションなどがユースケース（RDS Proxyの登場で ACID 特性のあるような従来型の要件にも対応可能になってきている）

セキュリティーに関しても、ツボを押さえて効率良く、変化に耐えうる設計が必要

AWS のサーバーレス関連サービス

コンピュート

AWS Lambda

AWS Fargate

データストア

Amazon Simple Storage Service

Amazon Aurora

Amazon DynamoDB

インテグレーション

Amazon API Gateway

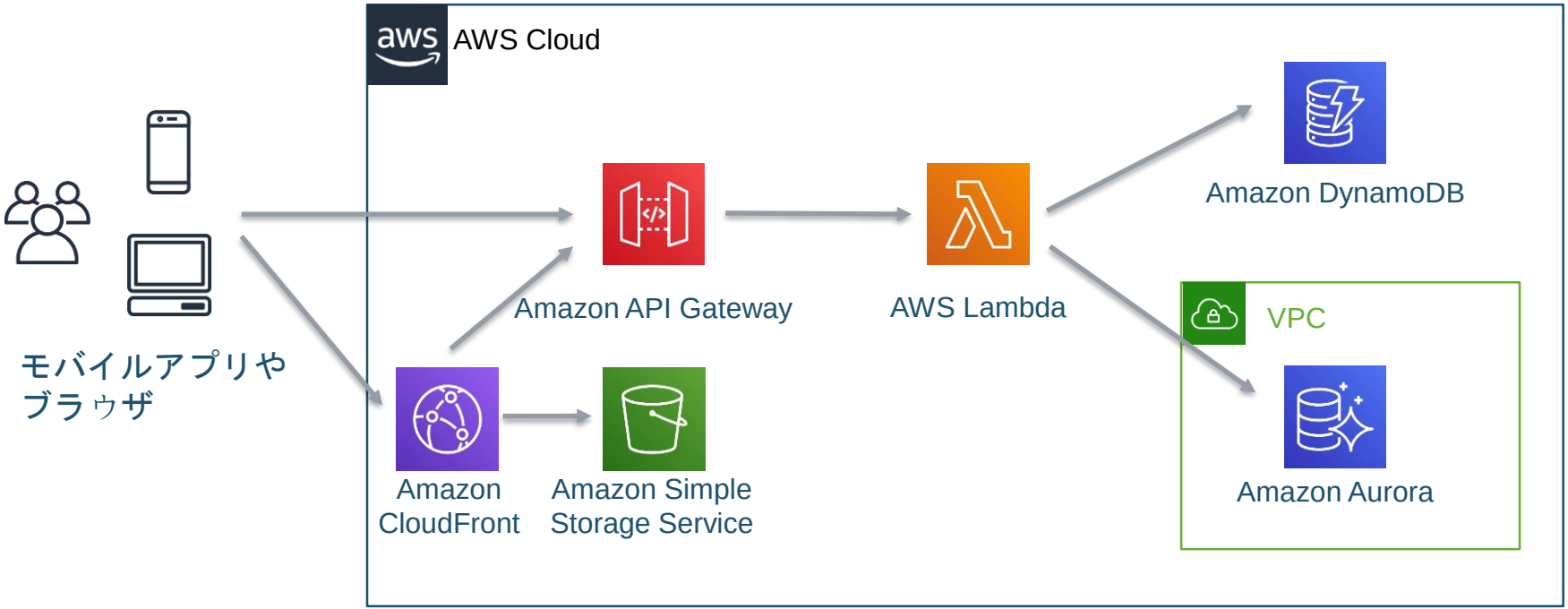
Amazon Simple Queue Service

Amazon Simple Notification Service

AWS Step Functions

AWS AppSync

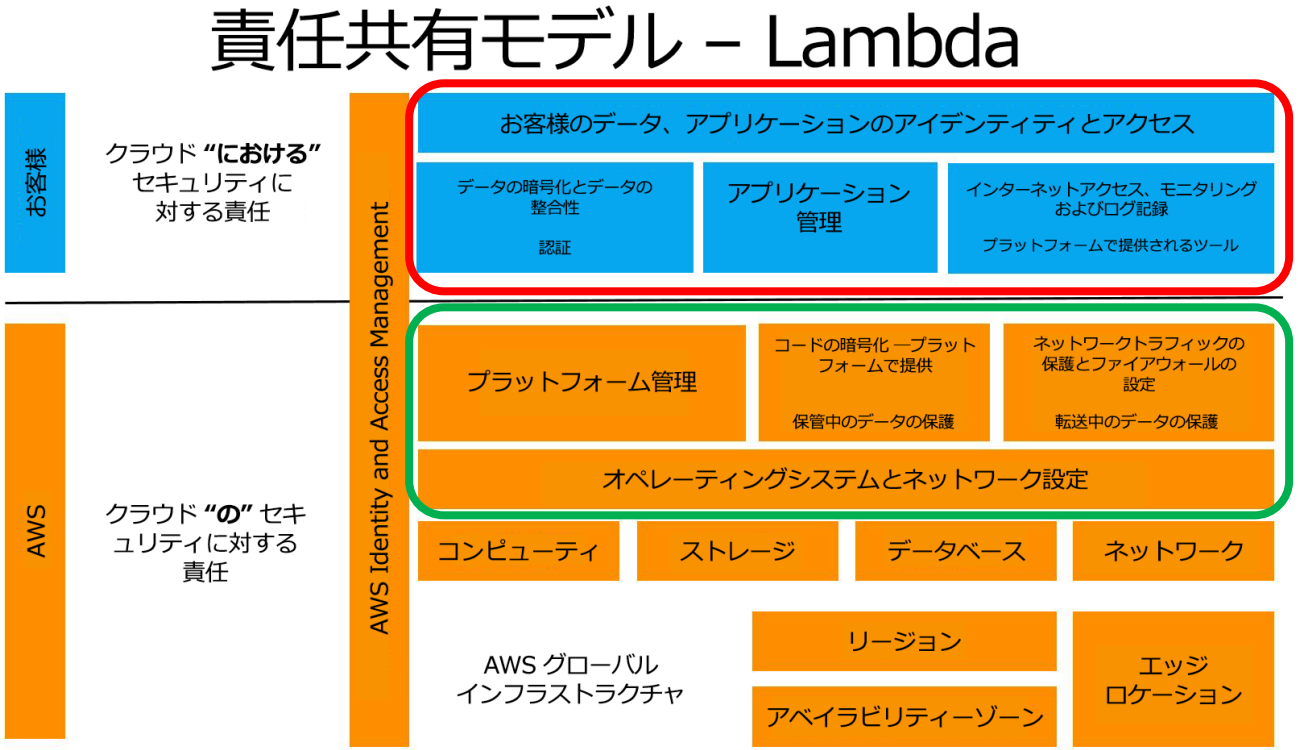
サーバーレスのシンプルなアーキテクチャー例



AWS 責任共有モデル（基本）

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

AWS Lambda の責任共有モデル



お客様はこの部分に注力

サーバーレスにすることでこの部分を AWS に任せられる

セキュリティに対するアプローチ

Well-Architected セキュリティの柱の原則

レイヤー、コンポーネント

	ID管理	トレーサビリティ	複層防御	自動化	データ保護	人手排除	インシデント対応
エッジ							
コンピュート							
データストア							
システム管理							
デプロイメント（CI/CD）							

抜けが無いように面で見て行く必要がある

aws

Well-Architected
サーバーレスアプリケーションレンズ

- サーバーレスアプリケーションのワークロードに着目したデザイン、デプロイ、アーキテクティングのベストプラクティス
- セキュリティについて、5つのベストプラクティスを定義

1. アイデンティティとアクセス管理（IAM）
2. 発見的統制
3. インフラストラクチャ保護
4. データ保護
5. インシデント対応（サーバーレスでは特に記述なし、一般的なWell-Architectedのガイドに従う）

リンク https://docs.aws.amazon.com/ja_jp/wellarchitected/latest/serverless-applications-lens/welcome.html?did=wp_card&trk=wp_card

Well-Architected Tool

マネージメントコンソール上で Well-Architected のレビューを実行可能

アイデンティティとアクセス管理

- SEC1: サーバレスAPIに対するアクセス制御
  - API は攻撃者のターゲットになる。攻撃者から保護するためのベストプラクティスとして、認証、認可を適切に実施する
  - AWS IAM 認可、 Amazon Cognito ユーザープール、Lambda オーソライザー、リソースポリシーを活用する（後半で説明）

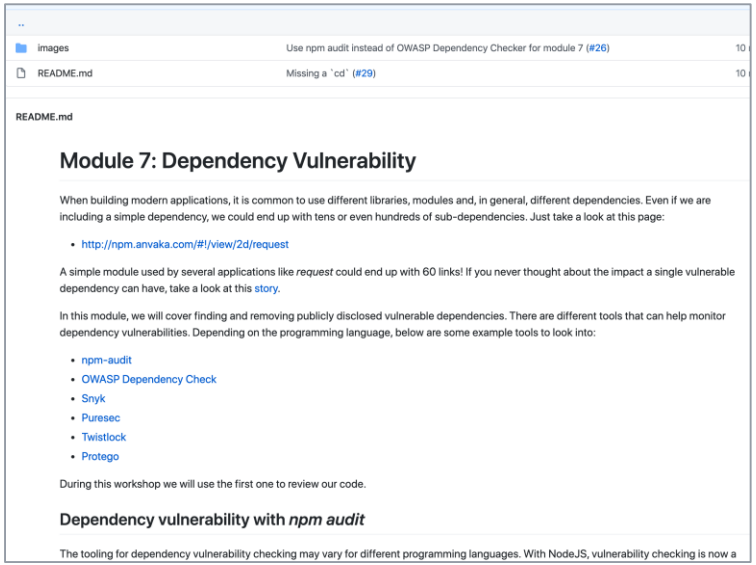
- SEC2 : サーバーレスアプリケーションのセキュリティ境界の管理
  - Lambda 関数に対して最小権限を与えるようにする（Lambda → AWS の他のサービスへのアクセス権限を最小化）
  - 関数を出来るだけシンプルにする（攻撃断面の最小化）
  - 1つの IAM ロールを複数の Lambda 関数で共有しない

発見的統制

- コンプライアンスやフォレンジックのためにログ管理を実施する

- ログの記録の他に、依存関係を構成しているアプリケーションに脆弱性がないか把握出来ることが重要。攻撃者は、プログラミング言語の種類に関係なく、既知の脆弱性を突いてくる

- アプリケーションレイヤーの脆弱性スキャンのためには、CI/CD パイプラインの中で OWASP 依存性チェックのようなオープンソースのソリューションや商用のソリューションが利用出来る



参考 : Serverless Security Workshop -> Dependency Vulnerability Check https://github.com/aws-samples/aws-serverless-security-workshop/tree/master/docs/07-dependency-vulnerability

インフラストラクチャ保護

- サーバーレスアプリケーションが VPC やオンプレミスにある他のコンポーネントと通信する場合、ネットワークの境界を考慮することが重要

- Lambda 関数は VPC 内のリソースにアクセスするように設定可能。このようなケースの通信を AWS Well-Architected フレームワークで説明しているようにコントロールする必要がある。外部への通信をコンプライアンスのためにフィルターする必要がある場合には、従来型のアーキテクチャーと同様に プロキシー などの活用を検討する

- サービス間の通信には、静的なキーではなく、AWS IAM の一時認証情報のような動的な認証情報を利用する。 API Gateway と AWS AppSync は IAM 認可をサポートしており、AWS サービスとの間の通信を制御出来る

データ保護 1

- API Gateway のアクセスログ（CloudWatch Logs）を有効化する場合には、必要な項目のみを選択する。ログには機微情報が含まれている可能性がある。サーバーレスアプリケーションの中で渡される機微データは暗号化することが推奨される

- API Gateway と App Sync は全ての通信でTLSを使用する。HTTPプロトコルのペイロードは暗号化されるが、リクエストしているPathやURLの一部であるクエリーストリングは暗号化されていない可能性があり、標準出力にそのまま出力すると、CloudWatch Logs に機微情報が記録されてしまう可能性があるので注意が必要

https://docs.aws.amazon.com/ja_jp/apigateway/latest/developerguide/set-up-logging.html

データ保護 2

- SEC3 想定されるワークロードを踏まえて
アプリケーションセキュリティ実装する

ポイント

- 入力データのサニタイズ
- 通信の暗号化
- シークレットの管理
- OWASPのアプリケーションセキュリティガイドラインや
AWS Security Bulletins の確認とテストの実施

AWS Security Bulletins （セキュリティ速報）
https://aws.amazon.com/jp/security/security-bulletins/?card-body.sort-by=item.additionalFields.bulletinDateSort&card-body.sort-order=desc

OWASP Serverless Top 10

OWASP が定義する サーバーレス Webアプリケーションの問題 Top 10

A1:2017 Injection （SQL/NoSQL インジェクションのような攻撃）
A2:2017 Broken Authentication（サーバーレスにおけるマイクロコンポーネント化されたコンピューティングモデルの認証の穴を狙う攻撃）
A3:2017 Sensitive Data Exposure（認証情報、アクセスキーの漏洩によるデータ侵害）
A4:2017 XML External Entities (XXE)（XML DTDの仕組みを悪用してサーバー上のリソースにアクセスする手法）
A5:2017 Broken Access Control（アクセス制御の穴を使って、アクセス制御をバイパスする）
A6:2017 Security Misconfiguration
A7:2017 Cross-Site Scripting (XSS)
A8:2017 Insecure Deserialization (安全でないシリアライズされたオブジェクトの復元処理）
A9:2017 Using Components with Known Vulnerabilities
A10:2017 Insufficient Logging and Monitoring

出典：https://owasp.org/www-project-serverless-top-10/

侵入テスト

- 侵入テストが許可されているサービス
  - Amazon CloudFront, Amazon API Gateway, Amazon Lambda 関数および Lambda Edge 関数
  - 参照 https://aws.amazon.com/jp/security/penetration-testing/
- ネットワーク負荷テスト、DDoS シミュレーションテスト
  - 参照 https://aws.amazon.com/jp/ec2/testing/ https://aws.amazon.com/jp/security/ddos-simulation-testing/

AWS のサーバーレス関連サービス

コンピュート

AWS Lambda

AWS Fargate

データストア

Amazon Simple Storage Service

Amazon Aurora

Amazon DynamoDB

インテグレーション

Amazon API Gateway

Amazon Simple Queue Service

Amazon Simple Notification Service

AWS Step Functions

AWS AppSync

aws

コンピュート・レイヤーのセキュリティ概要

入力値検証

AWS WAF：
XSS攻撃ルール
SQLインジェクションルール

API Gateway:
リクエスト検証

Lambda関数：
入力値のサニタイゼーション

依存性に起因する脆弱性

Lambda 関数
依存関係の最小化

Lambda レイヤーの利用

シークレットの管理

AWS Secrets Manager

AWS Systems Manager

パラメータ
ストア

Lambda 環境変数の暗号化

AWS WAF による Amazon API Gateway の保護

※ AWS WAF は REST API の場合のみ利用可能

https://docs.aws.amazon.com/ja_jp/apigateway/latest/developerguide/apigateway-control-access-aws-waf.html

AWS WAFが提供する機能

悪意のあるリクエストのブロック

- SQLインジェクション
- クロスサイトスクリプト
- IPブラックリスト

カスタムルールに基づいたWebトラフィックのフィルタ

- レートベースのルール
- IP Match & Geo-IP filters
- 正規表現パターン、文字列
- サイズ制限
- アクション：許可/拒否
- マネージドルールも利用可能

モニタリングとチューニング

- Amazon CloudWatch
- メトリクス/アラーム
- サンプルログ
- フルログ
- カウントアクションモード（検知モード）

AWS Managed Rules for AWS WAF (AMR)とは

AWS WAFに組み込み可能なビルトインルールセット

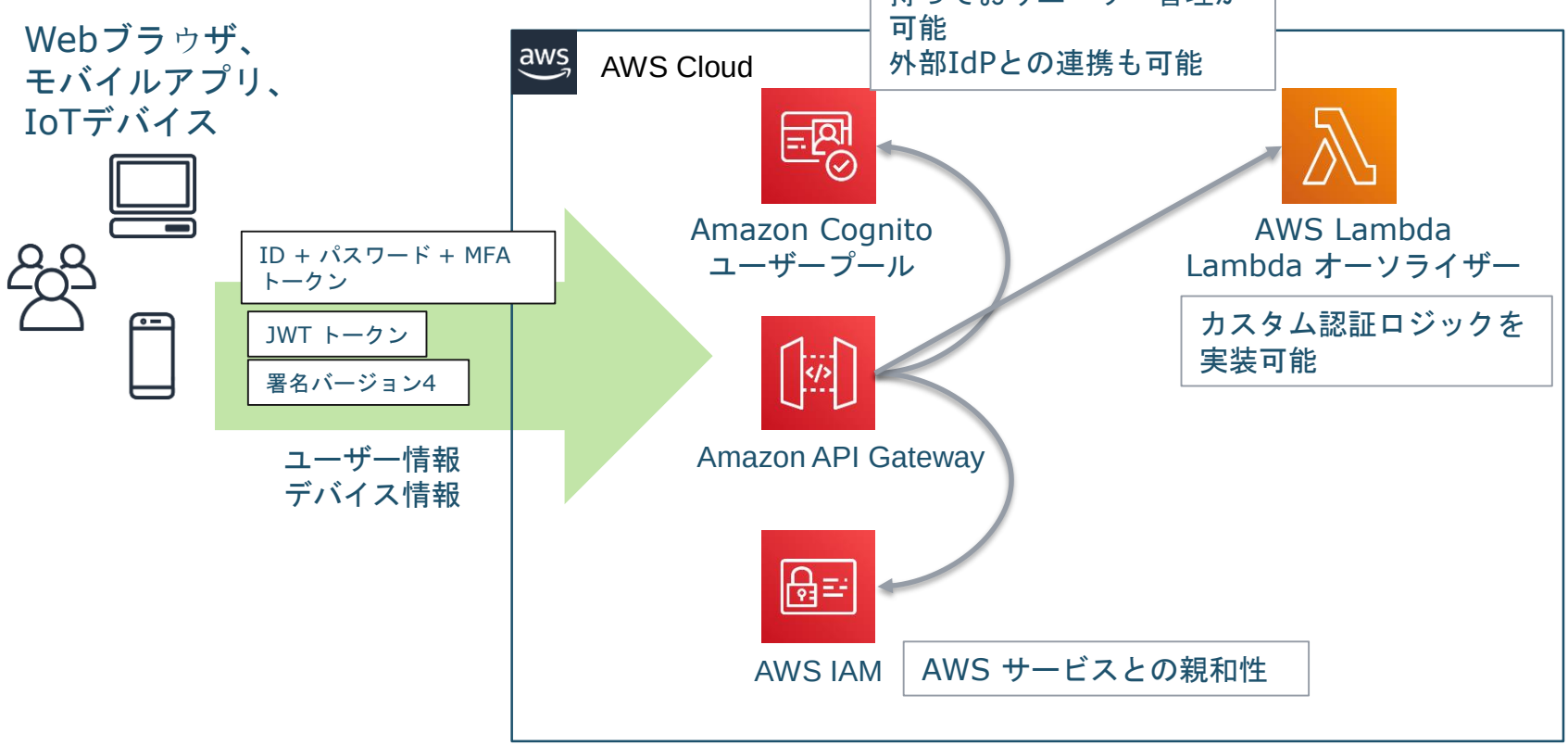
- AWS Threat Research Team (TRT)が作成及びメンテナンスを実施
- OWASP Top 10 をメインに対応を実施

追加費用は不要

- WAF Capacity Unit (WCU/後述)は消費するがそれ以外の追加費用は不要

Category	Ruleset	Description
CRS	Core Ruleset	Based on OWASP Top 10
EXR	Admin Protection	Blocks common administrative access
EXR	SQL DB	Predefined SQL injection detection
EXR	Linux	Linux based path traversal attempts
EXR	Known Bad Inputs	Well known bad request indicators
EXR	PHP	PHP specific exploits
EXR	WordPress	WordPress specific exploits
EXR	Posix	Posix based path traversal attempts
EXR	Windows	Windows based path traversal attempts
IP List	AWS IP Reputation List	Blocks IP that is known to have bot activities
IP List	Anonymous IP list	VPN, proxies, Tor nodes, and hosting providers.

API Gateway の認証認可



認証・認可方式の特徴とユースケース

方式	Cognito ユーザープール	AWS IAM 認証、署名バージョン4	Lambda オーソライザー（カスタムオーソライザー）	JWT オーソライザー
ユースケース	• クライアントAPIベースのアプリ • スマホアプリからのユーザー認証 • ユーザー管理機能	AWS リソースをIAMユーザーでアクセスする アプリから埋め込んだ認証情報でアクセス	認証方式、アイデンティティをAWS以外のサービスで認証したい場合 ロジックベースで認証が出来る場合	外部認証サービスが発行したJWTトークンからID情報を読み出して認可する
認証	○	○	△（外部に委ねる場合は認証は実行しない）	X （発行元が正しいか確認のみ可能）
認可	後続のリソースあるいは、IDプール連携で認可する	IAM ポリシーで可能	○	○（ルールベース）
コスト	Cognito の利用料金	無償	Lambda の利用料金	無償
API サポート	REST	REST, HTTP	REST, HTTP	HTTP

Amazon API Gateway リクエスト検証

- リクエストの内容を検証して、条件を満たさない場合にバックエンドにリクエストを渡さない設定が可能（ブロックするかパススルーするか選択可能だが、セキュリティ的にはブロックが望ましい）

- 機械的にリクエストを送信してくるような攻撃をブロックする

- 検証の条件
  - URI、クエリーストリング、リクエストヘッダーについて指定したものが含まれており空白でない
  - リクエストペイロードが、メソッドで設定されているJSONスキーマやリクエストモデルに従っている

aws

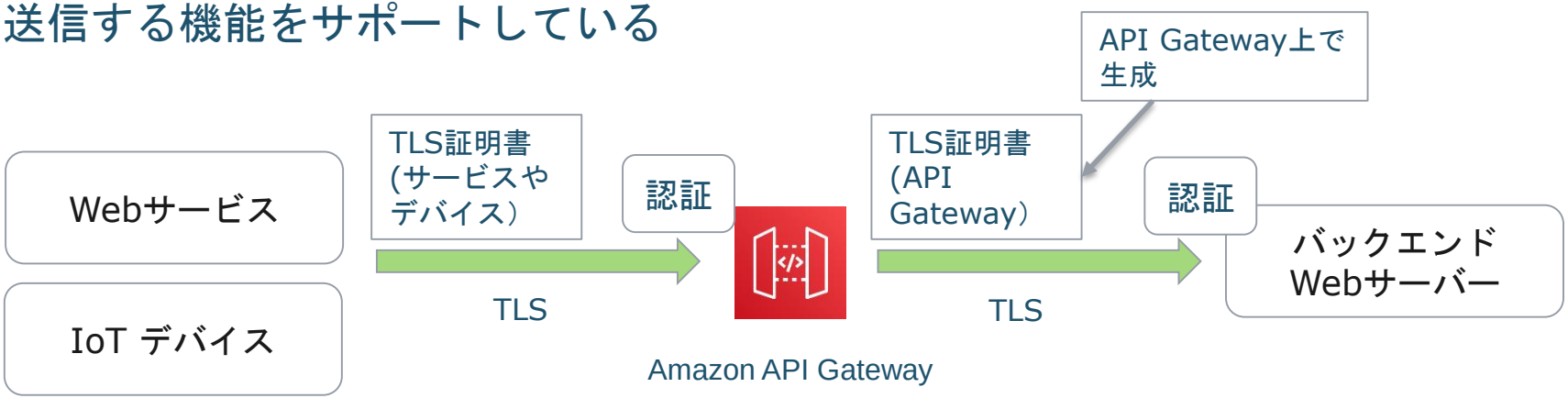
API Gateway における CORS サポート

- Cross-Origin Resource Sharing（CORS）は Web ブラウザで実行されるスクリプトからのクロスオリジン HTTP リクエストを制限する Web ブラウザのセキュリティ機能

- API Gateway の REST API がクロスオリジンの HTTP リクエストを受け取る必要がある場合に、 CORS サポートを有効にする必要がある
  - 有効にすると、API Gateway の返す HTTP レスポンスヘッダに Access-Control-Allow-Origin : <許可されたURL> が追加される

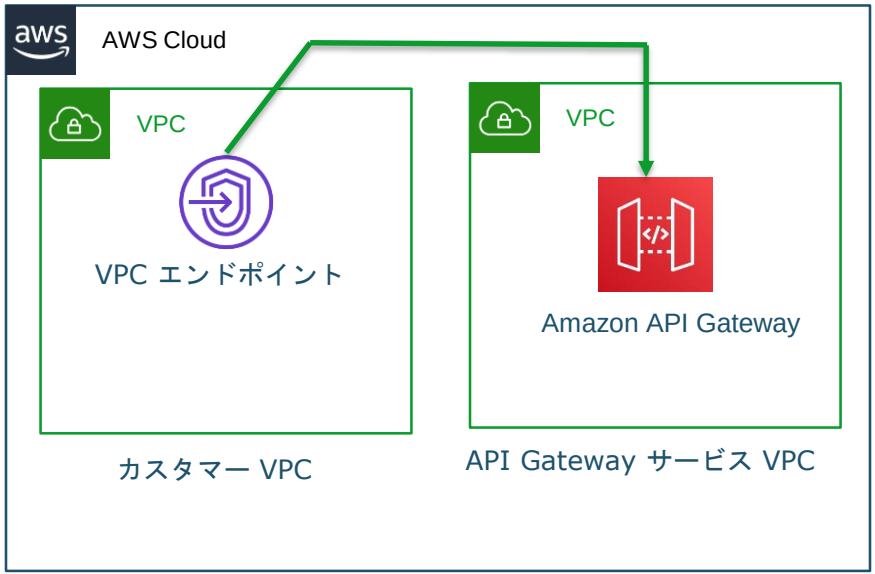
API Gateway における クライアント証明書サポート

API Gateway はリクエスト受信時の TLS証明書による認証と、
他のサービスにリクエストを送る時に、API Gateway から TLS証明書を
送信する機能をサポートしている



API Gateway プライベート API

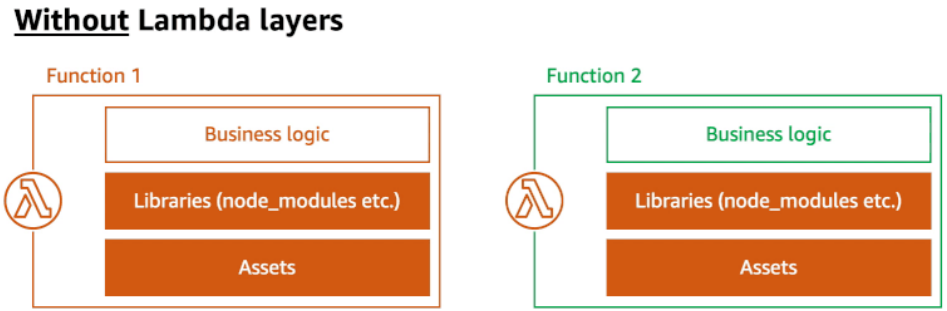
- API Gateway にインターフェース VPC エンドポイントを作成して Amazon VPC からのみアクセス出来るようにする
- VPC から API Gateway の通信は Amazon ネットワークを通り、インターネットには出ない



https://docs.aws.amazon.com/ja_jp/apigateway/latest/developerguide/apigateway-private-apis.html

Lambda レイヤー

- Lambda レイヤーは容易にコードの共有を可能にする機能。レイヤーとしてアップロードしているコードは全ての関数から参照可能
- 境界を提供、開発者はビジネスロジックの開発に迅速に取り組める
- エコシステムによる安全な共有機能をビルトインで提供



Lambda 関数ベストプラクティス

- マイクロVMの再利用を最適化するための
コールドスタートに備える
  - 必要なパッケージサイズを最小化する
  - VPC サポートのためのENIは
    コールドスタート中にアタッチされる
  - AWSクライアントやデータベースクライアントを
    Lambdaハンドラのスコープ外でインスタンス化する
- 不要であれば、/tmpからの読み込み、
書き込みを避ける
- カスタムランタイムによる複雑性を避ける
ためにAWSサポートのランタイムを使用する

コールドスタート
時に実行される

```
import sys
import logging
import rds_config
import pymysql

rds_host = "rds-instance"
db_name = rds_config.db_name
try:
        conn = pymysql.connect(
except:
        logger.error("ERROR:
def handler(event, context):
        with conn.cursor() as cur:
```

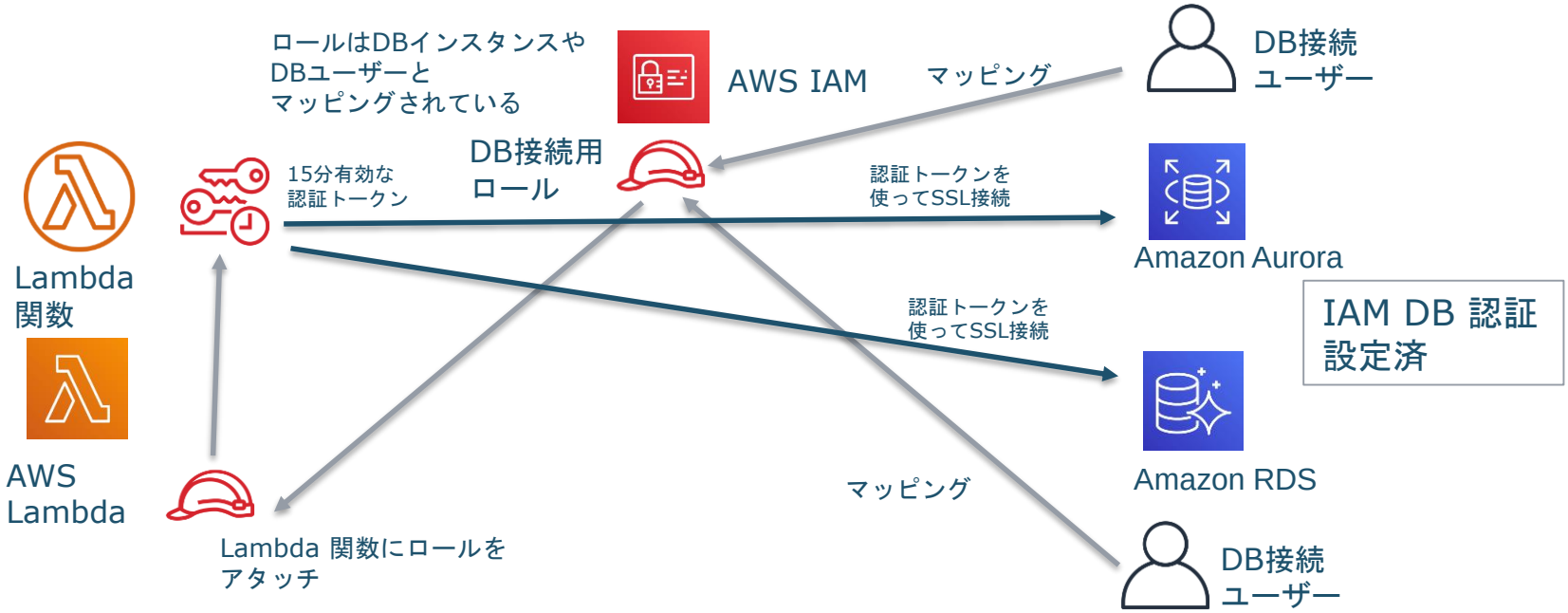
毎回の実行時に実
行される

シークレットをソースコードに埋め込まない

- データベースやサービスに接続するための ID + パスワード 等がシークレット
- シークレットはソースコードに埋め込まない。Lambda 環境変数で暗号化して格納するか、AWS Secrets Manager や AWS Systems Manager パラメータストアを利用する
- Amazon RDS , RDS Proxy や Aurora では IAM 認証でシークレット無しで接続することも可能

	Lambda 環境変数	AWS Secrets Manager	AWS SystemsManager パラメータストア
シークレットの暗号化	○	○	○
KMS による鍵管理	○	○	○
シークレットの ローテーション機能		○	
容易なコードからの呼び出し	◎	○	○
コスト	Lambda に含まれる	保管するシークレットの数とAPIの数で決まる	無料

IAM DB 認証 を使った Amazon RDS, Amazon Aurora への接続



Amazon Aurora :
https://docs.aws.amazon.com/ja_jp/AmazonRDS/latest/AuroraUserGuide/UsingWithRDS.IAMDBAuth.html

Amazon RDS : https://aws.amazon.com/jp/premiumsupport/knowledge-center/users-connect-rds-iam/
https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/UsingWithRDS.IAMDBAuth.html

AWS Secrets Manager によるシークレットの管理

IAM ロール、認証情報と Secrets Manager の組み合わせで、
アプリケーションは信頼性と安全性および認証情報のローテーション対応機能を利用可能

AWS のサーバーレス関連サービス

コンピュート

AWS Lambda

AWS Fargate

データストア

Amazon Simple Storage Service

Amazon Aurora

Amazon DynamoDB

インテグレーション

Amazon API Gateway

Amazon Simple Queue Service

Amazon Simple Notification Service

AWS Step Functions

AWS AppSync

aws

データレイヤーのセキュリティ概要

データ検出

Amazon Macie

データフロー追跡

AWS X-Ray

データ トークナイゼーション

- 自社開発
- AWS マーケットプレース

保管時暗号化

AWS KMS

- サーバーサイド暗号化
- クライアントサイト暗号化

伝送時暗号化

Amazon API Gateway

- HTTPS 接続設定

AWS Certificate Manager

- カスタムドメイン用のSSL証明書管理

データバックアップ/レプリケーション

Amazon S3

- バージョニング
- MFA デリート
- クロスリージョンレプリケーション

Amazon DynamoDB

- オンデマンドバックアップ
- ポイントインタイムリストア
- ストリーム

Amazon RDS

- 自動バックアップ

DynamoDB の暗号化

- DynamoDB に保存される全てのユーザーデータは、AWS KMS を利用して暗号化される

- 鍵管理には AWS 所有CMK、AWSマネージドCMK、カスタマーマネージドCMKを選択可能

- これは DynamoDB の保管時の暗号化であり、DynamoDB から取り出されたデータの通信時の暗号化はTLSを使って別途暗号化する必要がある

アイデンティとアクセス管理レイヤーのセキュリティ

ユーザーやサービスの認証と認可

Amazon API Gateway

4種類の認証、認可

- IAM
- Lambda カスタムオーソライザー
- Amazon Cognito ユーザープールオーソライザー
- JWT オーソライザー

Amazon Cognito

- マネージドユーザーディレクトリ、外部IdPとのフェデレーション
- 標準のJWTトークン、AWS 認証情報

サービス間のアクセス制御

AWS Identity and Access Management

- IAM プリンシパルの確認
- 最小権限の実装
- アクセス許可境界の強制

サーバーレス環境における
ネットワーク・セキュリティ

- Lambda, API Gateway などのサーバレスのサービスのネットワーク
セグメンテーション（境界防御）は AWS が実施

- Lambda のアクセスには、必ず AWS の認証情報が必要（アクセスキーもしくは、
STSによる一時認証情報）

- API Gateway はフロントに近い位置に配置されるので、Amazon Cognito などの
認証サービスとの組み合わせ、 AWS WAF のような攻撃検知、不正なアドレスからの
リクエスト拒否などのサービスを組み合わせることが推奨される

- 広範囲で大規模なアクセスを想定する場合には、Amazon CloudFrontや
AWS Shield Advancedによる大規模DDoS攻撃緩和策が推奨される

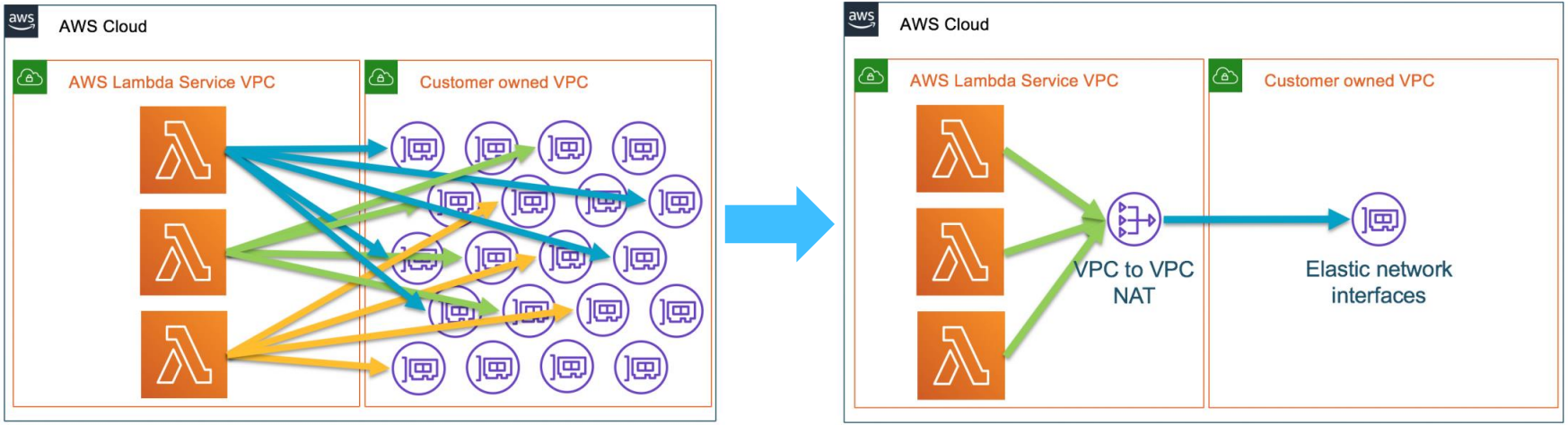
- 各サービス間の通信にSSL/TLSを使用する

ネットワークアクセス制御 : VPC エンドポイントの活用

VPC エンドポイントを利用することで、インターネットを経由しない
アクセスが可能になり、特定の VPC のみにアクセスを制限することが可能

- API Gateway
  - インターフェース VPC エンドポイント経由でプライベートAPI を提供可能
- Lambda
  - インターフェース VPC エンドポイント経由でアクセス可能
- DynamoDB、S3
  - VPC エンドポイント経由でアクセス可能

Lambda と VPC の新しい接続形態



従来より Lambda は VPC 内リソースに ENI を使ってアクセス可能だったがコールドスタートのたびに新しいENIが作成され、コールドスタートの時間がかかったり、ENI用のIPアドレスが消費される問題があった。
新しいモデルでは、ネットワーク・インターフェースは Lambda 実行環境で共有されるため、起動時間への影響や、IPアドレスの消費が無くなる

https://aws.amazon.com/jp/blogs/news/announcing-improved-vpc-networking-for-aws-lambda-functions/

システム管理レイヤーのセキュリティ

ロギングとトレース

Amazon API Gateway

- アクセスログ
- 実行ログ

AWS Lambda

- CloudWatch Logs

AWS X-Ray

メトリクス

Amazon API Gateway

- CloudWatch メトリクス
- CloudWatchメトリクス詳細モニタリング

AWS Lambda

- CloudWatch メトリクス
- CloudWatch メトリクス詳細モニタリング
- CloudWatch Logs からのメトリクス
- サードパーティーツール
  - IOPipe, Datadlog等

コンプライアンスチェック

AWS Config

Amazon CloudWatch Events

AWS Budgets

API Gateway, Lambda のロギング、監査

- API Gateway
  - CloudWatch Logs に実行ログとアクセスログを出力可能
  - アクセスログは、CLF, JSON, XML ,CSV から形式を選択可能

- Lambda
  - Lambda 関数の呼び出しは、CloudWatch Logs に記録される
  - CloudWatch Logsへの標準の出力形式ではなく、必要な情報のみをアクセスログ形式で出力したい場合には、Lambda 関数の中でログを書き出すロジックを実装する必要がある

DynamoDB のログ、監査

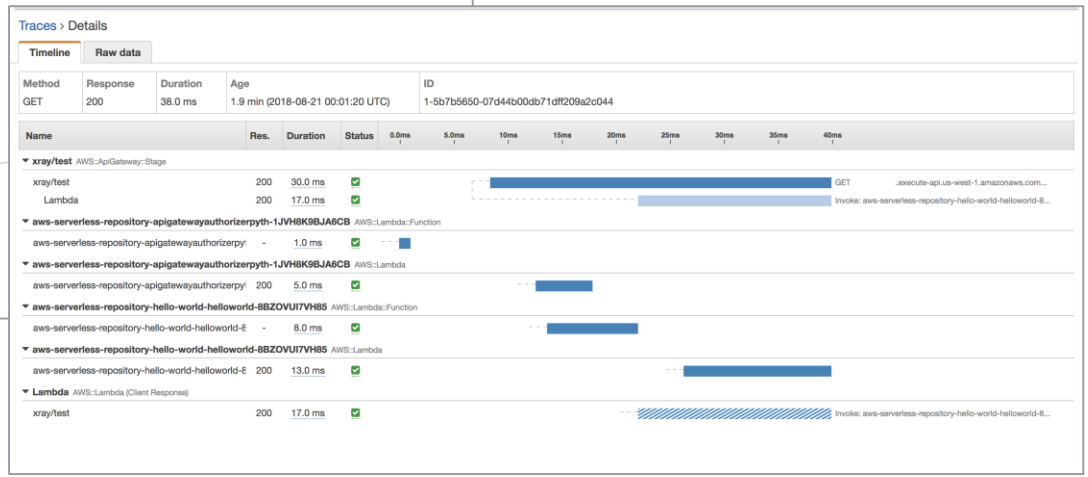
- CloudTrail
  - DynamoDB の管理系のテーブル操作が記録される
- DynamoDB のアイテムレベルの変更（作成、更新、削除）
  - DynamoDB ストリームからイベントを出力することが可能
  - 参照については、DynamoDB ストリームにも含まれないので、
    アプリケーション側でログを出力する仕組みをコードに組み込む必要がある

https://aws.amazon.com/jp/blogs/news/applying-best-practices-for-securing-sensitive-data-in-amazon-dynamodb/

aws

API Gateway, Lambda と AWS X-Ray との連携

AWS X-Ray を用いることで、 APIを呼び出しの流れを追跡し、可視化することが可能



デプロイメントのセキュリティ

バージョンコントロール

CodeCommit

- 認証
- リソースベースポリシー
- IAM ポリシー

コードの品質

CodePipeline

- セキュリティコントロール
- サードパーティーツール
  - Protego
  - Snyk
  - Twistlock
  - PureSec

Lambda

- Lambda レイヤー

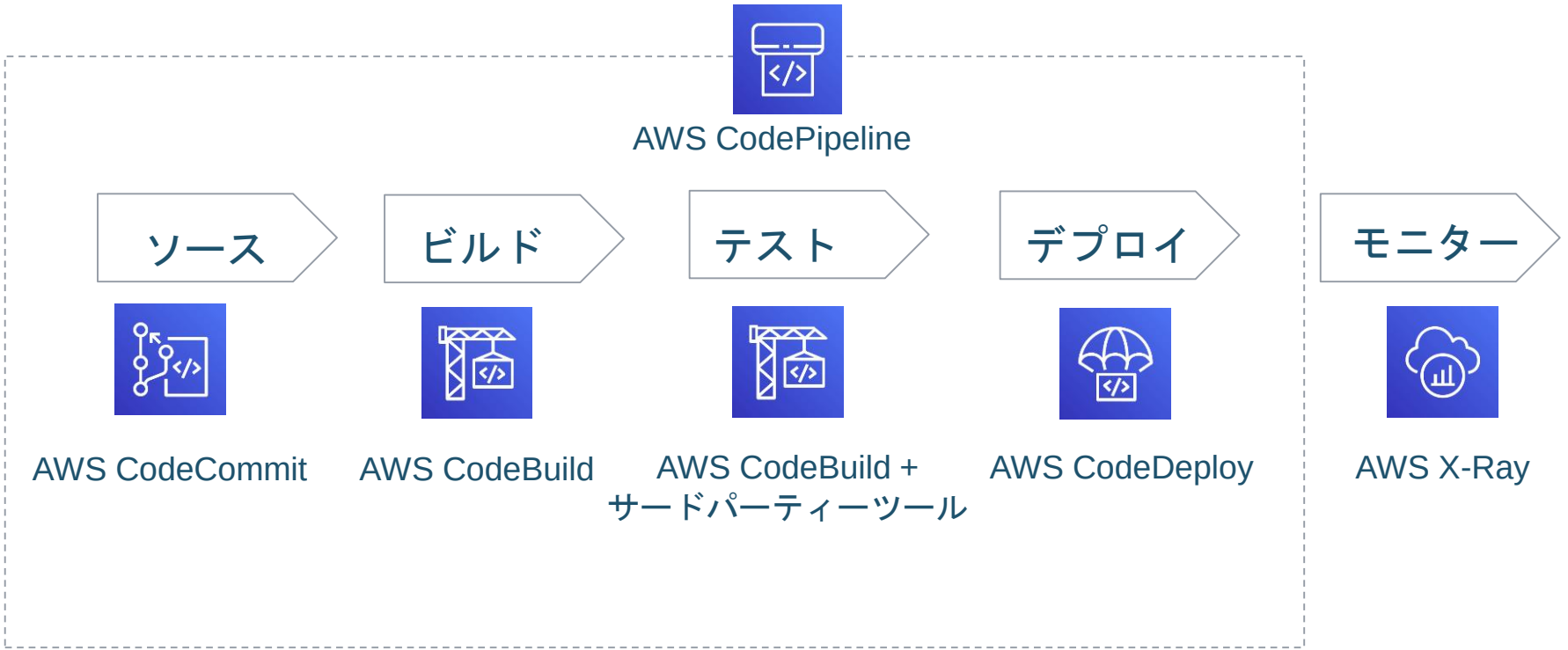
デプロイ・ストラテジー

CodeDeploy

- Blue-greenデプロイ
- カナリアリリース

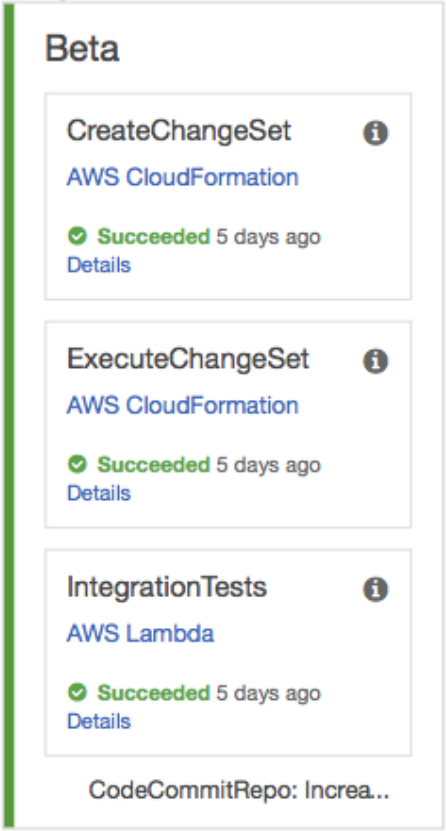
X-Ray

CI/CD をサポートする AWS サービス



CodePipeline 経由のデリバリ

1. ソースコードリポジトリにコードをコミット
2. CodeBuild でテストとパッケージングを実行
3. CodePipelineよりCloudFormation アクションを使ってAWS SAM テンプレートによるスタックを作成もしくは更新する
   オプション: 変更セットを利用する
4. ステージと環境にまたがって、アプリケーションをテストし、コントロール重篤度を向上する
   オプション: 手動承認を利用する



まとめ

サーバーレス・アーキテクチャーを採用することで、責任共有モデルのお客様責任部分を AWS 管理にすることが出来るので、セキュリティ対応の構築工数や運用工数を削減可能

サーバーレス・アーキテクチャーにおいては、アプリケーションレイヤー、特にIAM、データ保護、ログ管理、デプロイメント管理に着目して
セキュリティ設計、実装を行うことが重要

Well-Architected の基本原則を参照しながら、機能追加の速度が速いため、最新機能のチェックも忘れずに。コードのデザインや実装に関しては、OWASP のガイドもチェックする

ありがとうございました