[AWS Black Belt Online Seminar]
形で考えるサーバーレス設計
サーバーレス ユースケースパターン解説
ソリューションカットシリーズ

福井 厚
シニアソリューションアーキテクト
アマゾンウェブサービスジャパン株式会社
2020/11/18

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

過去資料
https://amzn.to/JPArchive

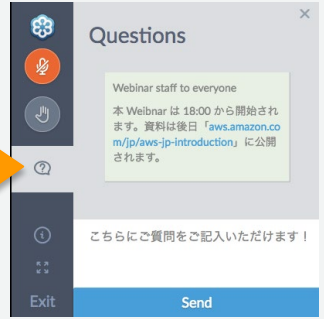
AWS Black Belt Online Seminar とは

「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2020年11月18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には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.

自己紹介

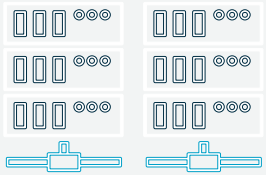
- **名前**
  - 福井 厚（ふくい あつし）fatsushi@
- **所属**
  - アマゾン ウェブ サービス ジャパン株式会社
  - 技術統括本部レディネスソリューション本部
  - シニアソリューションアーキテクト
サーバーレス スペシャリスト
- **関心領域**
  - ソフトウェア アーキテクチャ、オブジェクト指向設計、アジャイル開発
- **好きなAWSサービス**
  - サーバーレステクノロジー全般、 AWS Code シリーズ、AWS Amplify

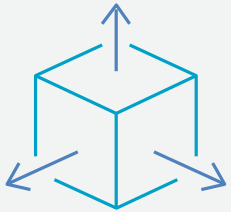
本日のアジェンダ

- サーバーレスのおさらい
- サーバーレスユースケースパターン解説
- まとめ

サーバーレスのおさらい

サーバーレスによる代表的な効果





サーバー管理が不要 （準備、OS保守 etc）	柔軟なスケーリング （拡張/縮退）	十二分に考慮された 高可用性	アイドル時の リソース確保が不要

- ユーザーの責任領域を小さくしそこだけに注力
  - 高生産性
  - マネージド 業務注力
  - 変更容易性
  - エンジニア 意識改革
- 実際の処理負荷に応じて自動で拡張/縮退
  - スケーラビリティ（機会損失防止）
  - マネージド 自動リソース管理
  - リアルタイム（付加価値/機能差別化）
  - コスト最適化

aws

サーバーレスによるお客様の効果例

5x 従来より生産性が向上
アプリ展開を加速化

1/6 安定した定常稼働により
運用の労力を大幅に短縮

1/3 コード量の減少（従来比）
= 生産性向上、保守改善

1人 運用を 1 人で楽に実施
機能改善に注力可能

2ヶ月 スケール、冗長化などの
考慮不要で短期実装可能

9:1 “開発:保守/運用” の作業
比率が 1:9 から大きく改善

1-2日 簡易な機能追加は短期で
実装・デプロイ可能

1/10 アイドル時間のリソースが
解放され、利用費が最適化

aws

では、サーバーレスにどこから手をつける?
そのために何を理解する必要がある?

- 関連するサービスの特性を理解してから設計をはじめる
  → 王道ながら、多くの知識/知見が必要

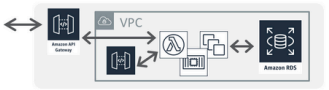
- 実績あるユースケースパターンから選択する
  → やりたいこと（ユースケース）から
  設計の形を考え始めることができる

サーバーレスユースケース パターン解説

代表的な適用シーン/ユースケースパターン

動的 Web / モバイルバックエンド
関連資料 | Tutorial
参考 - 動的 Web / モバイル

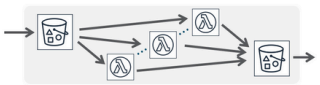
インタラクティブモバイル
関連リンク | 関連動画 / 資料



業務系 API / グループ企業間 API
関連記事 | 関連事例

配信系・インタラクティブ API
関連リンク

画像処理 / シンプルなデータ加工
関連リンク | Tutorial | 関連事例



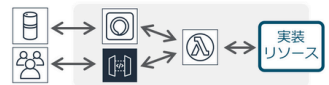
分散並列処理
関連事例1 | 関連事例2 | RefArch

イベント駆動の業務処理連携
SQS 連携機能 | 設定ガイド

アプリフロー処理
Tutorial（Workflow / エラー処理）

流入データの連続処理
関連資料 | Tutorial | RefArch

IoT バックエンド
関連資料 | 関連事例 | RefArch



チャットボット / Alexa スキル
Alexa スキル開発 | RefArch

データ変更トリガー処理
活用例 | Tutorial

ログデータ収集処理
関連資料 | Tutorial

データレイクからのデータ加工
関連リンク | DB Loader
or より包括的なソリューション

機械学習/ETLデータパイプライン
関連記事 | 関連事例

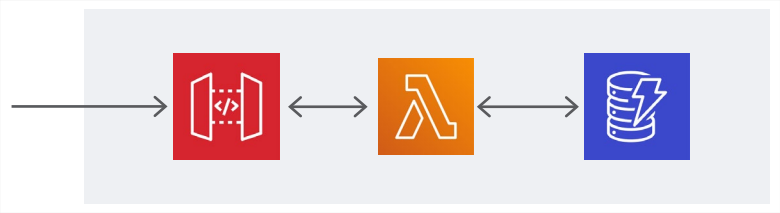
スケジュール・ジョブ/CRON
関連ドキュメント

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

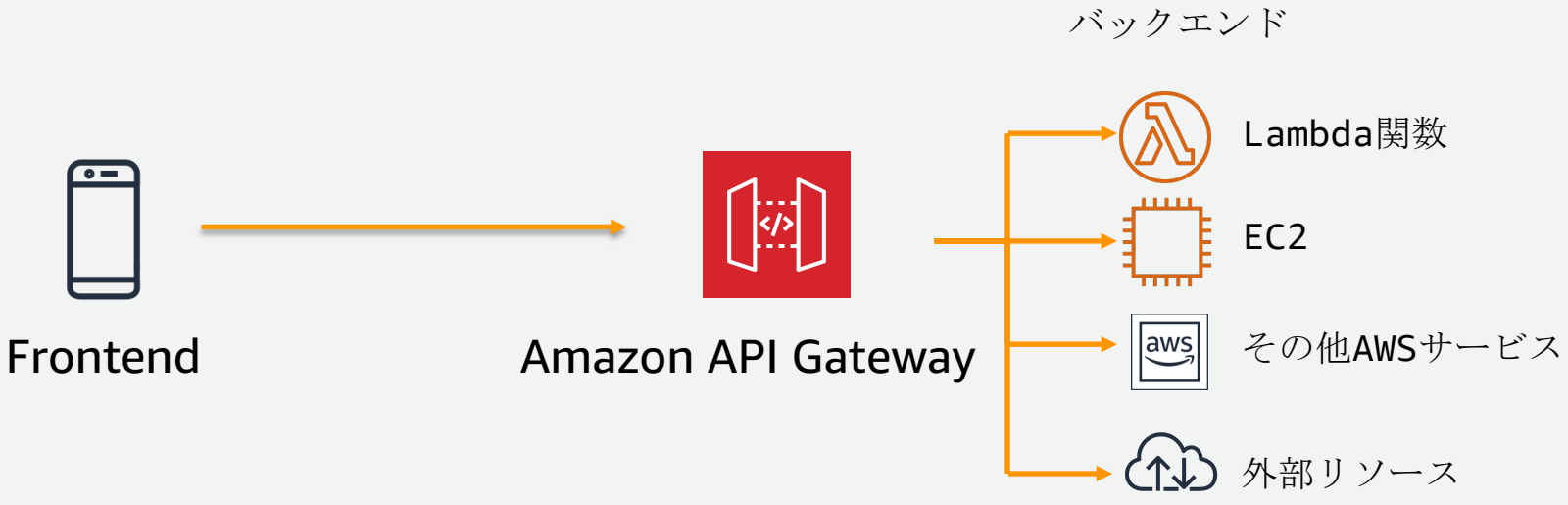
ユースケースパターン①：モバイル、API 関連

動的Web/モバイルバックエンド

検討TOP1

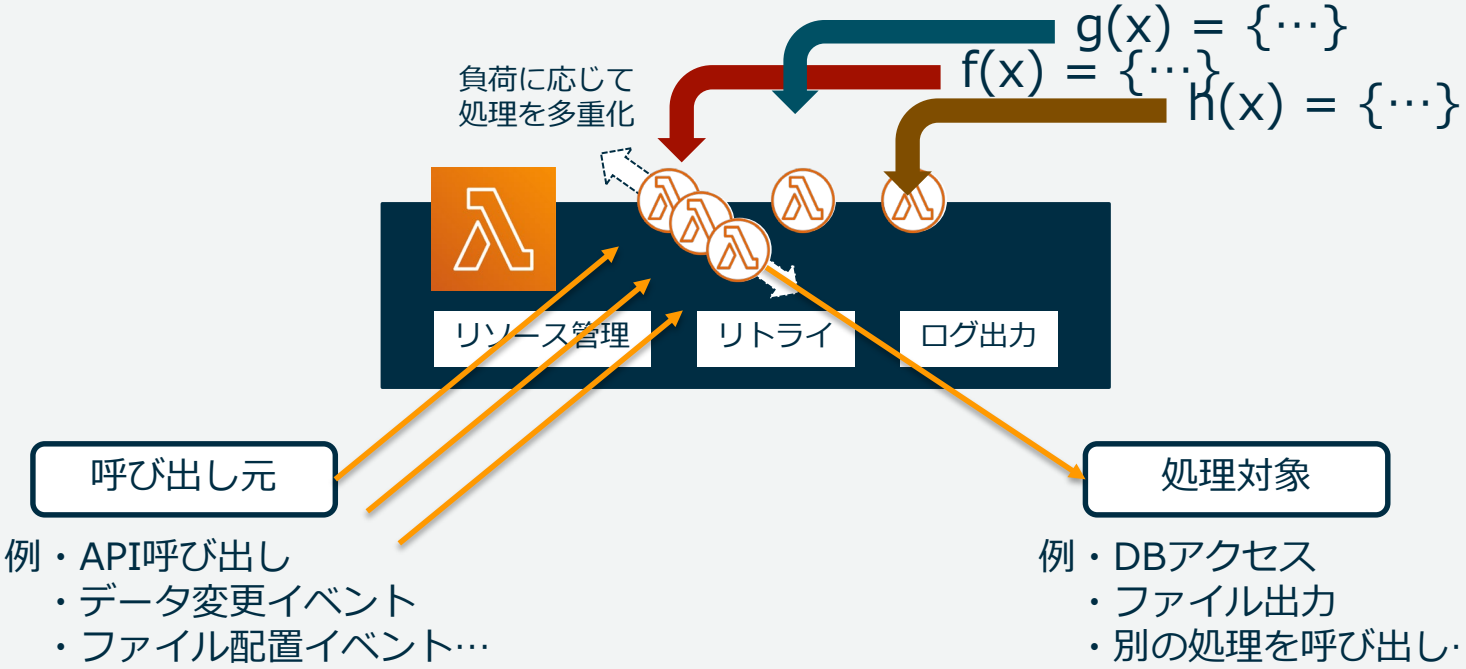
アーキテクチャ図	
利用サービス	• Amazon API Gateway • AWS Lambda • Amazon DynamoDB
ユースケース	• API公開の典型的なサーバーレス実装の形 • リクエスト/レスポンス型向け（同期モデル） • REST APIを経由してDBの情報を同期的に参照/更新する • SPAやモバイルアプリで多用されるパターン
設計ポイント	• レガシーなWebサーバー/APIサーバーの置き換えにも適用可能 • API Gatewayの統合タイムアウトは30秒、超えそうな場合は非同期呼び出しを検討 • Cognitoとの連携など、認証、認可について考慮する

Amazon API Gateway

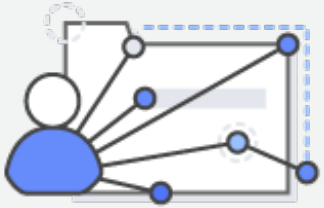


- REST APIとWeb Socketsをサポート
- APIの設定、デプロイ
- 認証、認可、アクセス制御
- 流量制御と保護（スロットリング）
- キャッシング

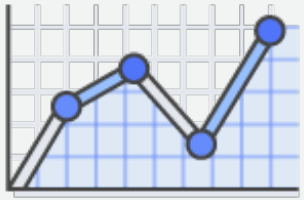
AWS Lambda



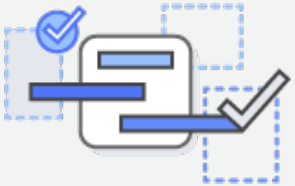
Amazon DynamoDB



容量制限のない、完全マネージド型のNoSQL
データベース



ハイスケーラブル、3箇所のレプリケーション
による高可用性

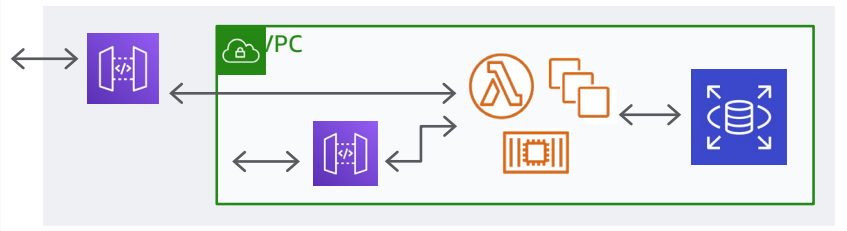


10 ミリ秒未満のレイテンシと
On-Demand ModeとProvisioned Mode

aws

業務系 API グループ企業間API

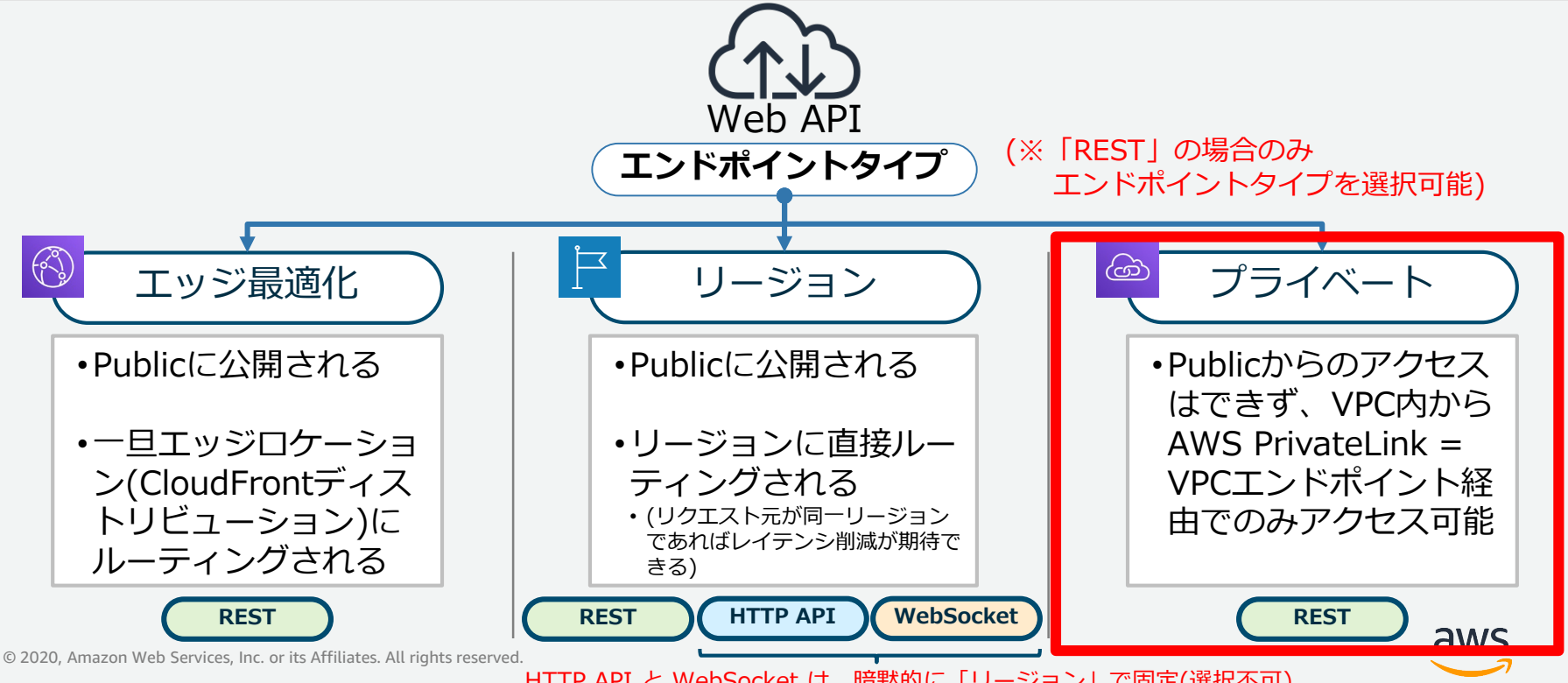
検討TOP2

アーキテクチャ図	
利用サービス	• Amazon API Gateway • AWS Lambda • Amazon RDS*
ユースケース	• 内部データの公開ルートをAPI 化
設計ポイント	• API GatewayのVPCエンドポイント対応によりプライベートなAPIを作成可能 • API Gatewayのキャッシュ、スロットリング機能により社内バックエンドのリソースを過度な負荷から保護 • AWS LambdaのHyperplane ENIによるVPCリソースへのアクセス • RDBへのリクエストがバーストするユースケースにおいては、Amazon RDS Proxyの利用も検討

* Amazon Relational Database Service (RDS)

API設定 - エンドポイントタイプ

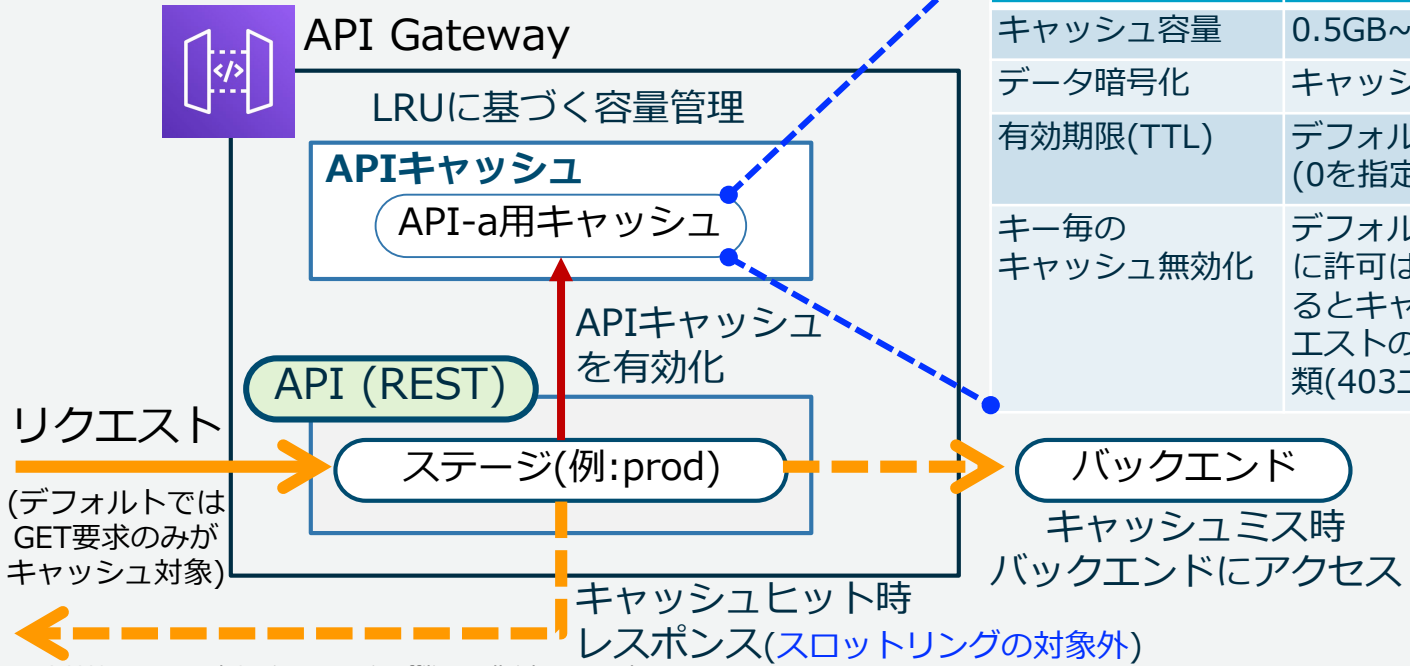
REST の 場合、3種類のエンドポイントタイプから1つを選択
(クライアントから見たアクセス先エンドポイントとしての性質を決定)



API設定 - キャッシュ

REST

REST では APIの ステージ毎 に キャッシュ を定義し、
バックエンドへのトラフィック削減 と 低レイテンシ の実現に利用可能



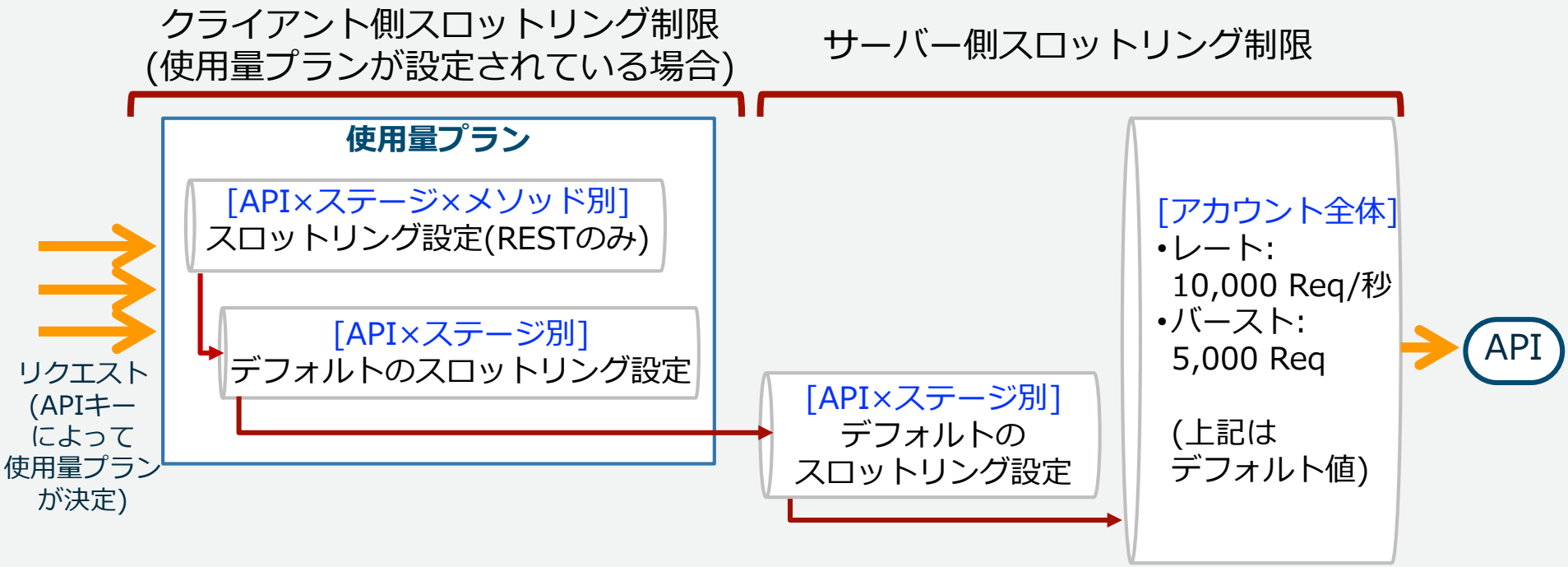
設定項目	説明
キャッシュ容量	0.5GB~237GBでの8種類から選択
データ暗号化	キャッシュデータの暗号化有無を指定
有効期限(TTL)	デフォルト300、最長3600(単位:秒) (0を指定するとキャッシュ無効)
キー毎の キャッシュ無効化	デフォルトではキャッシュ無効化指示に許可は不要の設定(false)。trueにするとキャッシュ無効化を指示するリクエストの扱いを、予め用意された3種類(403エラー/警告/無視)から選択

aws

REST/WebSocket の スロットリング

REST WebSocket

REST/WebSocketでスロットリングは 「サーバー側のスロットリング制限」と「クライアント側のスロットリング制限」によって行われる



※スロットリングのみであれば、
HTTP APIプロトコルの機能「スロットリング」設定で対応可能

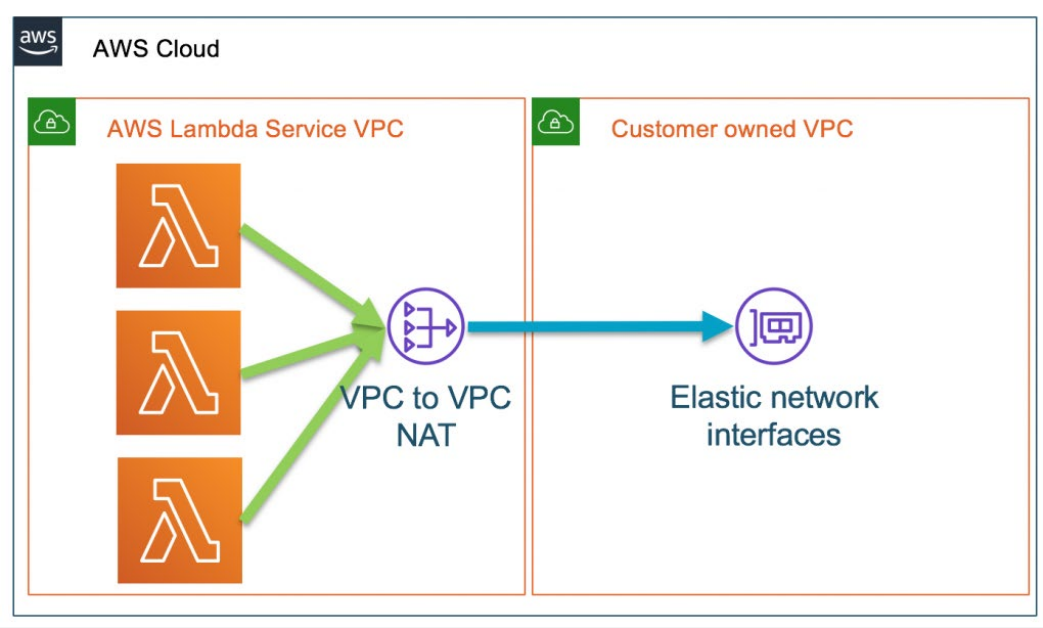
aws

AWS LambdaのVPCリソースアクセス

- Blog記事 **[発表] Lambda 関数が VPC 環境で改善されます**
  https://aws.amazon.com/jp/blogs/news/announcing-improved-vpc-networking-for-aws-lambda-functions/

- ネットワークインターフェイスの作成はLambda関数の作成時

- アカウント内のセキュリティグループ、サブネット単位にENIを共有



Amazon RDS Proxy

Amazon RDS 向けの高可用性フルマネージド型データベースプロキシ
アプリケーションのスケーラビリティやデータベース障害に対する回復力と安全性の向上を実現

データベース接続をプールおよび共有する事でアプリケーションのスケーリングを改善

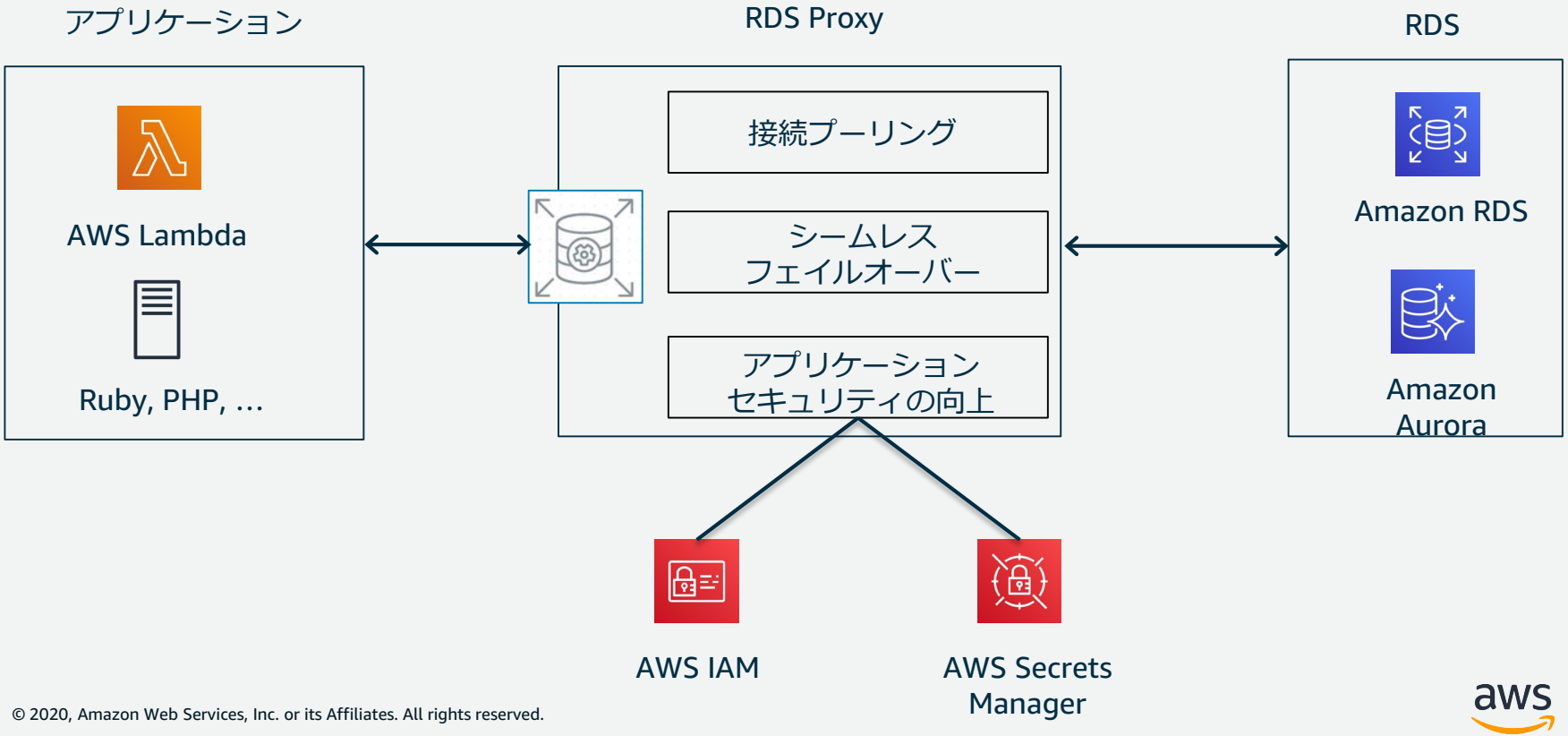
アプリケーションの可用性を高め、データベースのフェイルオーバー時間を短縮

データベースアクセス制御で、アプリケーションデータのセキュリティを管理

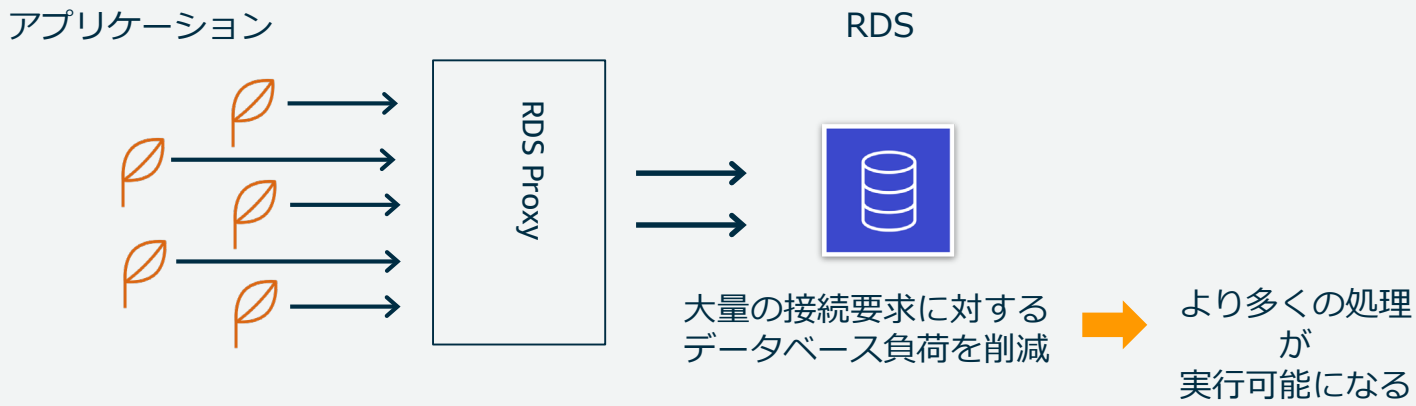
フルマネージドデータベースプロキシ、データベースとの完全な互換性

aws

Amazon RDS Proxy 全体像



接続プーリング



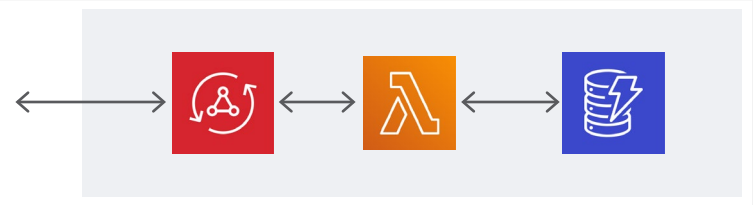
接続プーリング

- 接続の開閉に伴うデータベースの負荷 (TLS/SSL のハンドシェイク、認証、ネゴシエーション機能などのCPU負荷など) を削減

接続の多重化

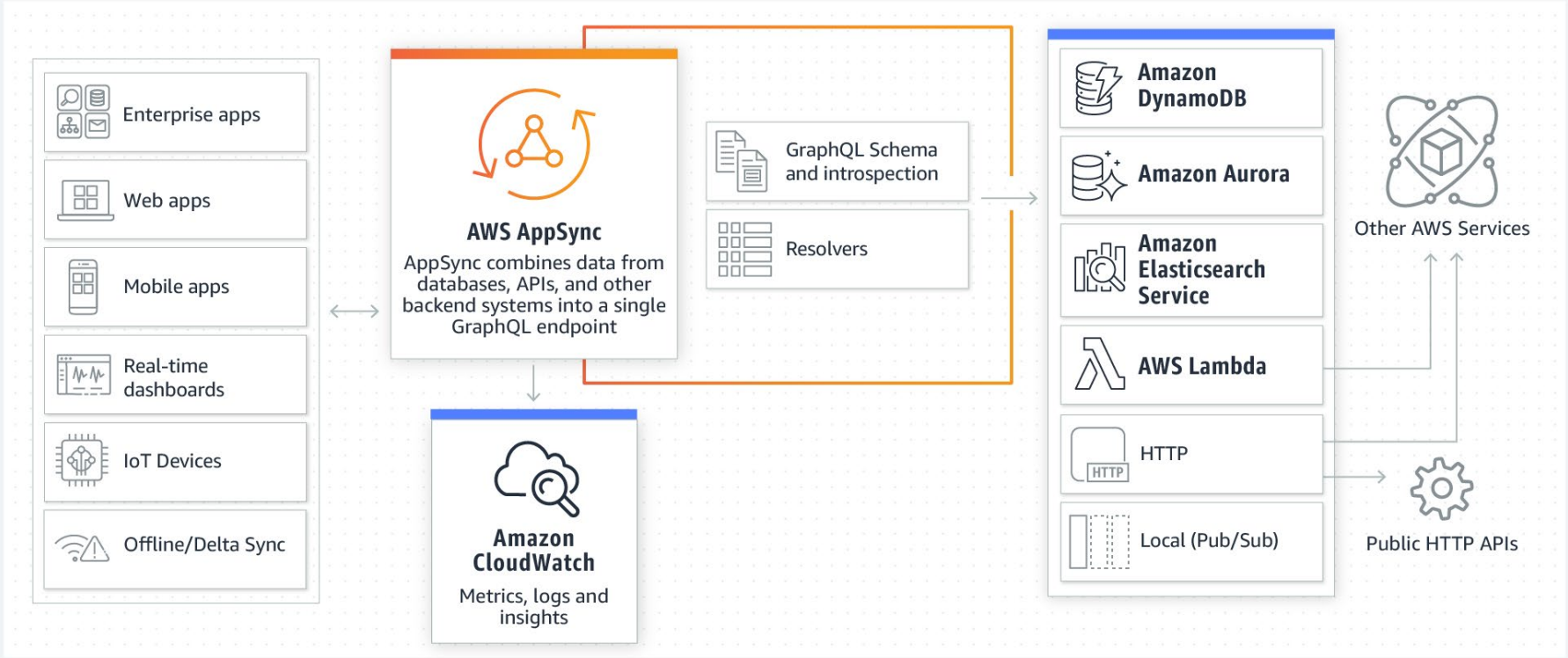
- 接続の再利用により、データベース接続に必要なコンピューティングリソース (主にメモリ) を削減
- max_connections エラーの発生頻度の抑制。

リアルタイムモバイル モバイル オフライン処理

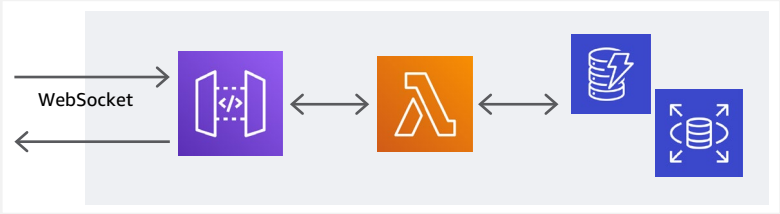
アーキテクチャ図	
利用サービス	• AWS AppSync • AWS Lambda • Amazon DynamoDB
ユースケース	• リアルタイム通信要件や非接続状態（オフライン）要件があるモバイル向け
設計ポイント	• AWS AppSyncはリアルタイム機能とオフライン機能を備えたフルマネージドGraphQLサービス • フロントエンドにはAWS Amplifyを利用可能 • GraphQLのモデリングによってAPIを設計* • データソースとしてAWS Lambda、Amazon DynamoDB、Amazon Aurora Serverless、Amazon ElasticSearch Service、HTTPエンドポイントをサポート

* https://docs.aws.amazon.com/ja_jp/appsync/latest/devguide/designing-a-graphql-api.html

AWS AppSync

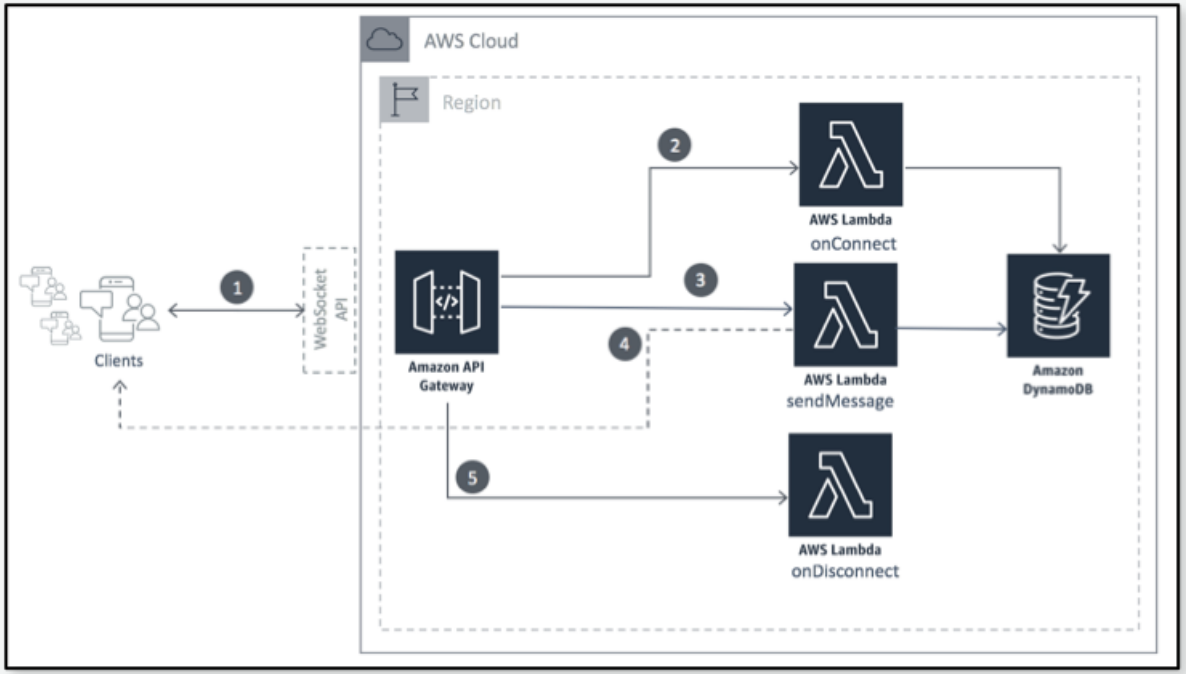


インタラクティブAPI データ配信API

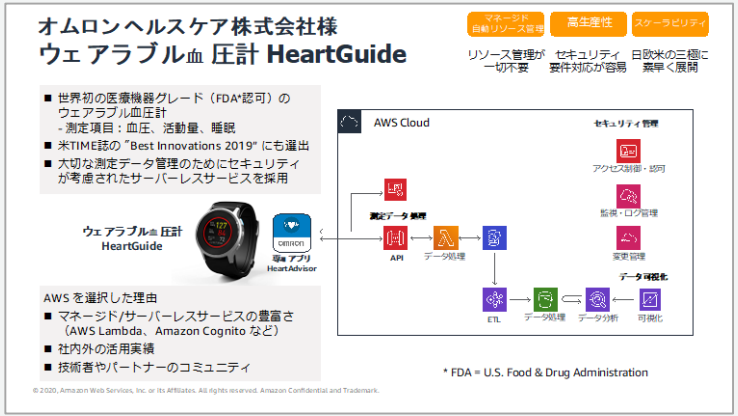
アーキテクチャ図	
利用サービス	• Amazon API Gateway • AWS Lambda • Amazon RDS • Amazon DynamoDB
ユースケース	• WebSockets でリアルタイムに情報を配信 • クラウド側のデータ変更をPush配信可能
設計ポイント	• Amazon API GatewayがWebSocketsに対応、AWS LambdaでWebSocketsを利用した送受信のコードを実装可能 • 接続先のグループ情報管理はDynamoDBなどを利用 • Amazon RDSを利用する場合は、Amazon RDS Proxyも検討

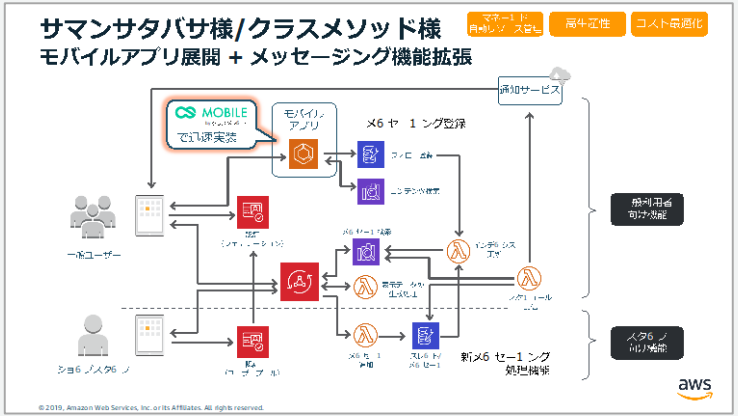
Amazon API Gateway WebSocketsサポート

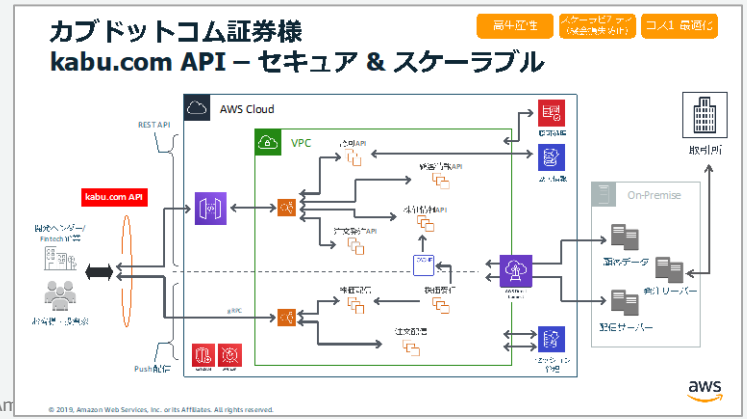
- Blog記事 [発表]Amazon API GatewayでWebsocketが利用可能
  https://aws.amazon.com/jp/blogs/news/announcing-websocket-apis-in-amazon-api-gateway/

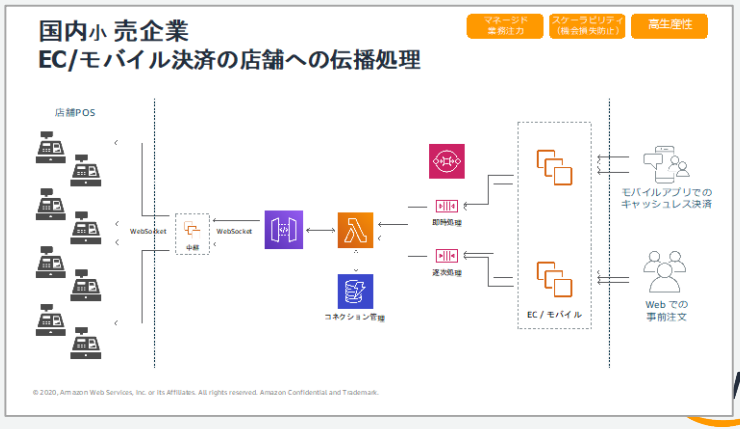


ユースケースパターン: モバイル、API 関連 – 活用例





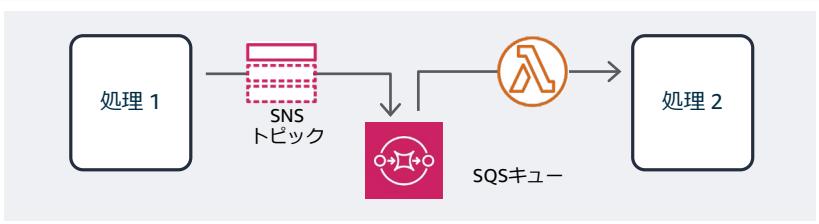




ユースケースパターン②：データ加工、連携処理

イベント駆動の業務処理連携

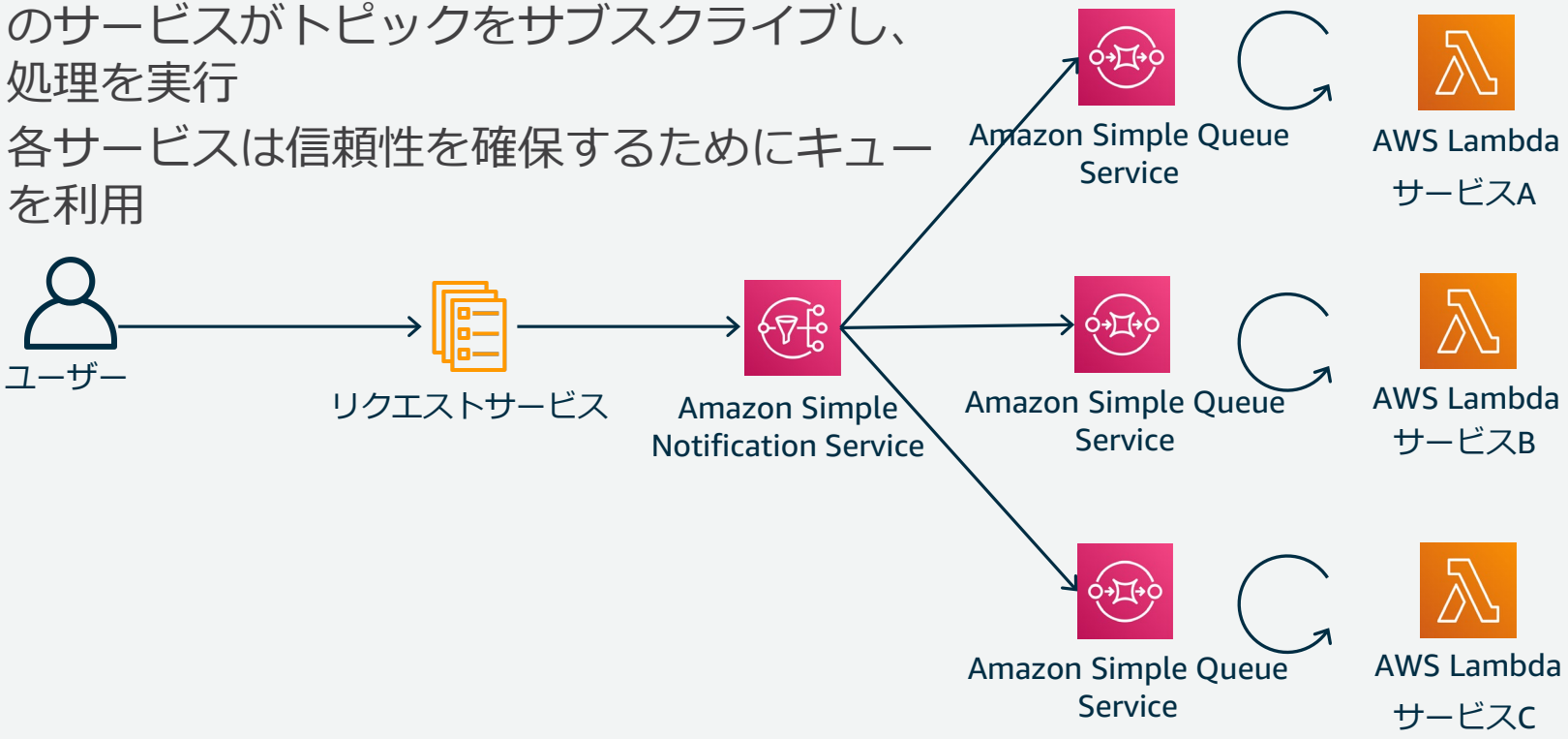
検討TOP3

アーキテクチャ図	
利用サービス	• Amazon SNS * • Amazon SQS ** • AWS Lambda
ユースケース	• 次の処理のためのタスクをキュー（またはS3）にPushし、非同期で連携
設計ポイント	• SNSトピックは業務のイベントとして設計、キューに保存することで信頼性を確保し、Lambdaで処理を実行 • ひとつのトピックに対して複数の処理を個別に実行することも可能（ファンアウトパターン） • FIFO SNSとFIFO SQSの組み合わせも可能

* Amazon Simple Notification Service (SNS)　　** Amazon Simple Queue Service (SQS)

参考：ファンアウトパターン

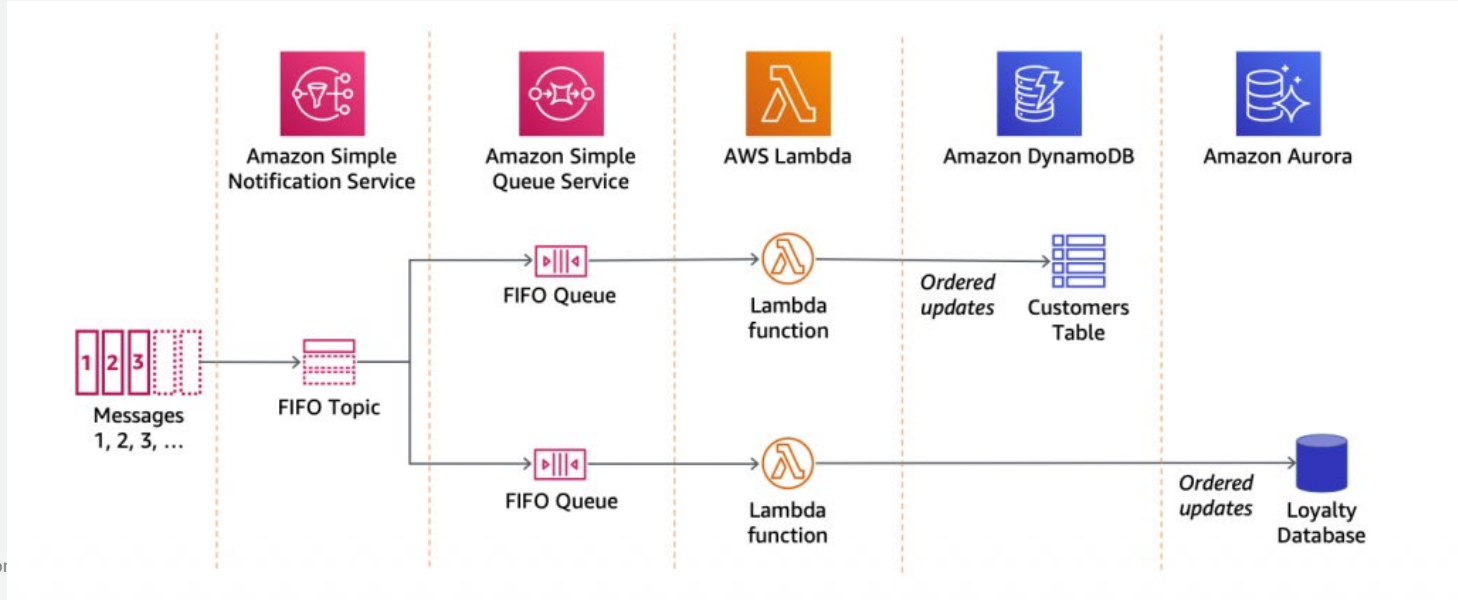
- あるイベント（トピック）に関心がある複数のサービスがトピックをサブスクライブし、処理を実行
- 各サービスは信頼性を確保するためにキューを利用



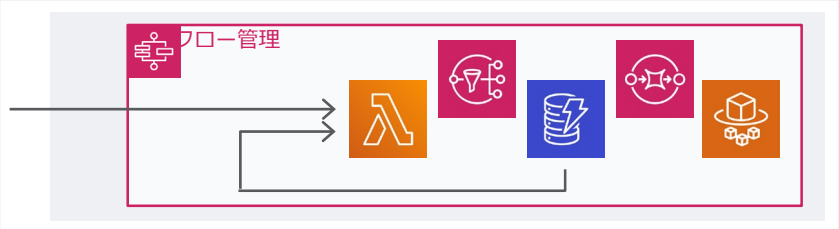
Amazon SNSがFIFOをサポート

New！

- 順序性が大事なビジネスロジックの処理に、Amazon SQSのFIFOキューと組み合わせて実行可能
- こちらのBlogもご参照ください。
  https://aws.amazon.com/jp/blogs/news/introducing-amazon-sns-fifo-first-in-first-out-pub-sub-messaging/



アプリケーションフロー処理

アーキテクチャ図	
利用サービス	• AWS Step Functions • AWS Lambda • Amazon SNS • Amazon DynamoDB • Amazon SQS • AWS Fargate
ユースケース	• 一連の処理フローを可視化、エラー処理のフロー管理としても利用可能
設計ポイント	• AWS Step Functionsによってリトライや例外処理を宣言的に設定することが可能 • AWSサービスとの統合が可能で、Step FunctionsからStep Functionsへの呼び出しや、他の処理の完了を待機してから再開することも可能

AWS Step Functions

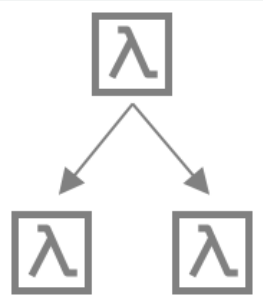
AWSのフルマネージドなステートマシン

- 弾力性のあるワークフローオートメーション
- 組み込みのエラーハンドリング
- AWSサービスとの強力な統合
- 独自のサービスとの統合サポート
- 実行履歴の監査とビジュアルモニタリング

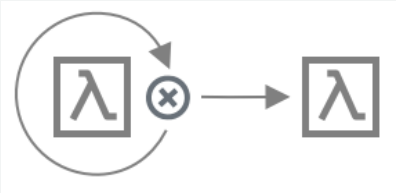
AWS Step Functionsのユースケース

1. 機能オーケストレーション

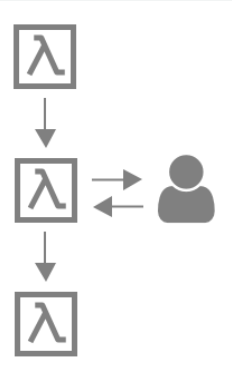
2. 分岐



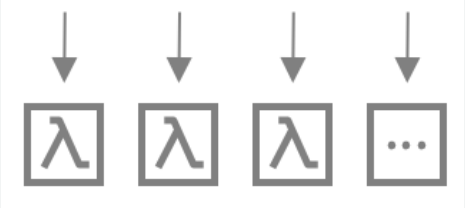
3. エラー処理



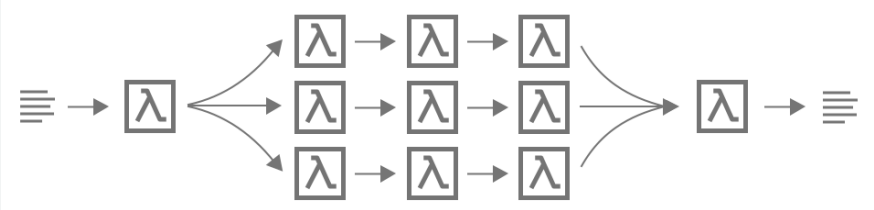
4. ループ中の人間



5. 並列処理



6. 動的並列処理

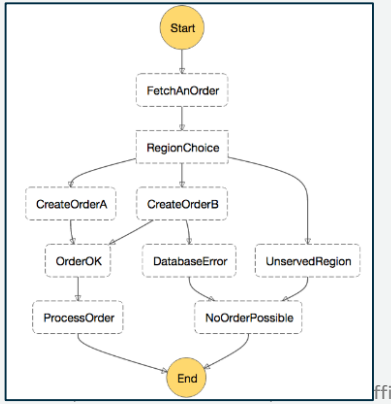


AWS Step Functions

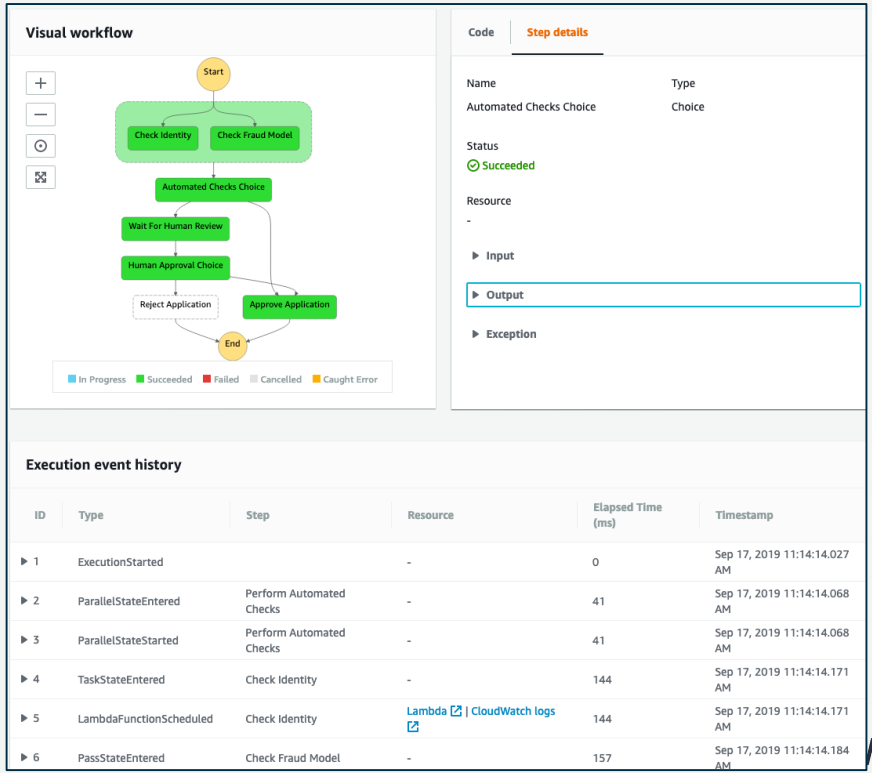
JSONで定義(Amazon States Language)

```
{
    "Comment": "Manage opening an account",
    "StartAt": "Perform Automated Checks",
    "States": {
        "Perform Automated Checks": {
            "Type": "Parallel",
            "Branches": [{
                    "StartAt": "Check Identity",
                    "States": {
                        "Check Identity": {
                            "Type": "Task",
                            "Parameters": {
```

コンソールで視覚化



実行結果をモニタリング



標準ワークフローとExpressワークフロー

最大期間	標準ワークフロー	Expressワークフロー
最大期間	1年	5分
実行開始レート	2,000/秒以上	100,000/秒以上
状態遷移レート	1アカウントあたり4,000/秒以上	ほぼ無制限
料金*	状態移行ごと。状態遷移は実行ステップが完了するごとにカウント	実行回数、実行時間、およびメモリ消費量
実行セマンティクス	Exactly-once	At-least-once

詳細は以下のURLを参照ください。
https://docs.aws.amazon.com/ja_jp/step-functions/latest/dg/concepts-standard-vs-express.html

＊料金はこちらを参照 http://aws.amazon.com/step-functions/pricing

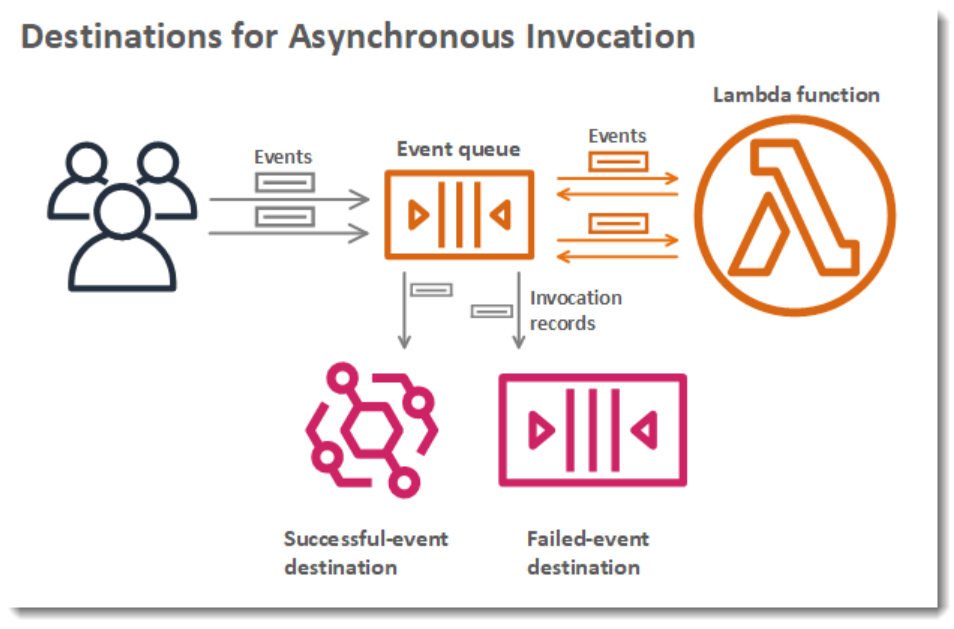
画像処理 シンプルなデータ加工

アーキテクチャ図	
利用サービス	• Amazon S3 * • AWS Lambda
ユースケース	• データ投入をきっかけにファイル情報を引き渡して処理を起動
設計ポイント	• Amazon S3からAWS Lambdaを非同期に呼び出し • 関数からエラーが返された場合は最大 2 回再試行 • スロットルエラー（429）およびシステムエラー（500番台）の場合、Lambdaはイベントをキューに返し、最大 6 時間関数を再実行。再試行回数、イベントの最大有効期間は設定可能。 • Amazon S3のイベント発行はAt Least Onceなので発火漏れはないが重複して呼び出された場合の処理を考慮する

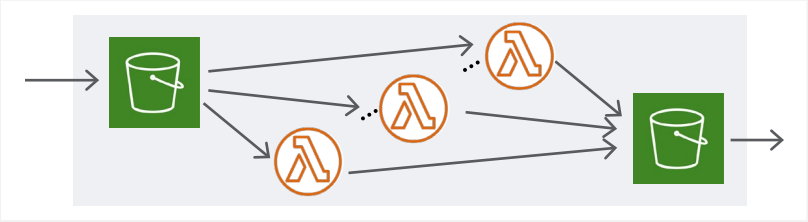
* Amazon Simple Storage Service (S3)

AWS Lambda 非同期呼び出しと送信先の設定

- Amazon S3やAmazon SNSなど、AWSのいくつかのサービスでは、関数を非同期的に呼び出しイベントを処理
- 設定可能な項目
  - 関数エラー時は最大２回再試行
  - スロットルエラー（429）、システムエラー（500番代）の場合はイベントをキューに返し最大６時間再試行
  - 成功時、失敗時の送信先指定
    - Amazon SQS
    - Amazon SNS
    - AWS Lambda
    - Amazon EventBridge

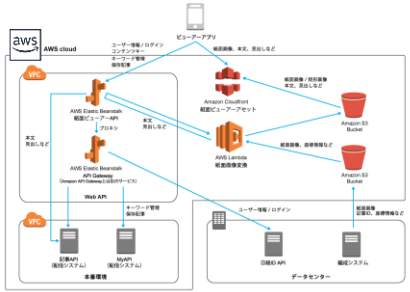


短時間処理の並列実行

アーキテクチャ図	
利用サービス	• Amazon S3 • AWS Lambda 
ユースケース	• 並列度の高い処理を同時実行させて完了時間を短縮化
設計ポイント	• S3のイベント通知機能でLambda関数が非同期に呼び出されるが、at least onceなので重複して呼び出される可能性を考慮 • バージョンを有効にしていないオブジェクトに対して同時に書き込みを行うと単一のイベントのみ送信される可能性がある。個別のイベントを発生させたい場合はバージョンを有効にする • Lambda関数で書き込むバケットがその関数をトリガーするバケットの場合、処理がループする可能性があるので注意

ユースケースパターン: データ加工、連携処理 – 活用例

日本経済新聞社様
日経電子版 紙面ビューア



豊富なコンテンツを提供する
インターネットメデ6 ア

マネージド 自動リソース管理

トラフルはほぼゼロ。負荷が高くなれば瞬間的に数カ台分の処理に拡張。

マネージド 業務注力

運用の手間が大幅に削減。時間と手間を新たなサービス開発にあてられる。

コスト最適化

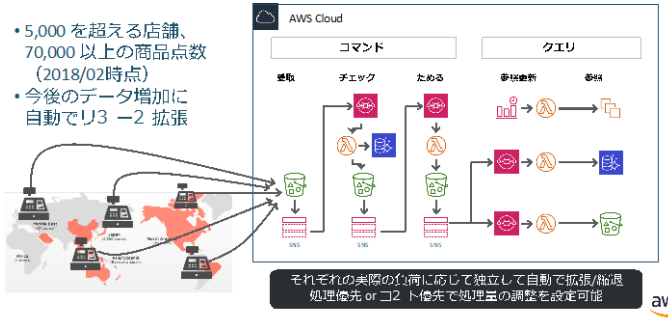
従来型の構成での運用と比較して1/10のコスト削減。

aws

ダイソー様
サーバーレスによるPOSデータ処理

マネージド 業務注力 | マネージド 自動リソース管理 | 変更容易性

- 5,000 を超える店舗、70,000 以上の商品点数（2018/02時点）
- 今後のデータ増加に自動でリ3 ー2 拡張

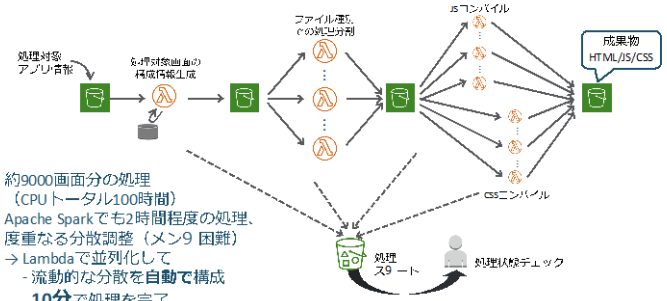


それぞれの実際の負荷に応じて独立して自動で拡張/縮退
処理優先 or コ2 ト優先で処理玉の調整を設定可能

aws

ワークスアプリケーションズ様
アプリ画面生成処理の並列化

スケーラビリティ | マネージド 自動リソース管理 | コスト最適化

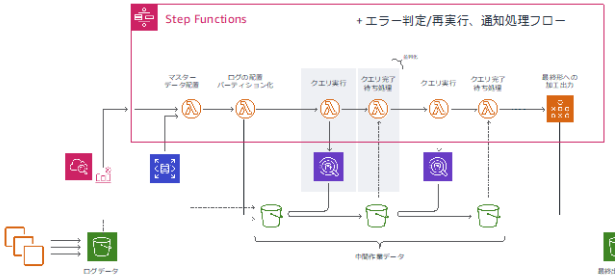


- 約9000画面分の処理（CPUトータル100時間）
- Apache Sparkでも2時間程度の処理、度重なる分散調整（メン9 困難）
  → Lambdaで並列化して
  - 流動的な分散を**自動で**構成
  - **10分**で処理を完了

aws

Fringe81様
広告配信ログデータの処理フロー

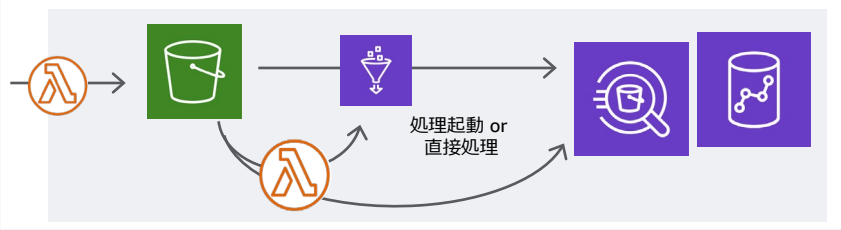
スケーラビリティ（処理の並列/多重化） | マネージド リソース自動管理 | コスト最適化



ユースケースパターン③：バックエンドデータ処理

データレイク周りのデータ加工

検討TOP4

アーキテクチャ図	
利用サービス	• AWS Lambda • Amazon S3 • AWS Glue • Amazon Athena • Amazon Redshift
ユースケース	• データレイクのデータの加工処理やDBへのデータローディング
設計ポイント	• AWS Glueはフルマネージドな ETL(extract, transform and load) サービスでデータの分類、クリーニング、加工を容易に実現 • メタデータリポジトリのAWS Glueデータカタログ、PythonまたはScalaコードを自動生成するETLエンジン、ジョブのモニタリング、再試行を処理するスケジューラで構成 • Amazon Athenaは標準SQLを使用してS3のデータの直接クエリするインタラクティブなサービス。AWS Glueデータカタログと統合することでS3の永続的なメタデータストアを提供

ETLサービス – AWS Glue

AWS Glue

サーバーレス

スケジューラーと
ワークフロー

コードに集中

データソースの
メタデータ管理

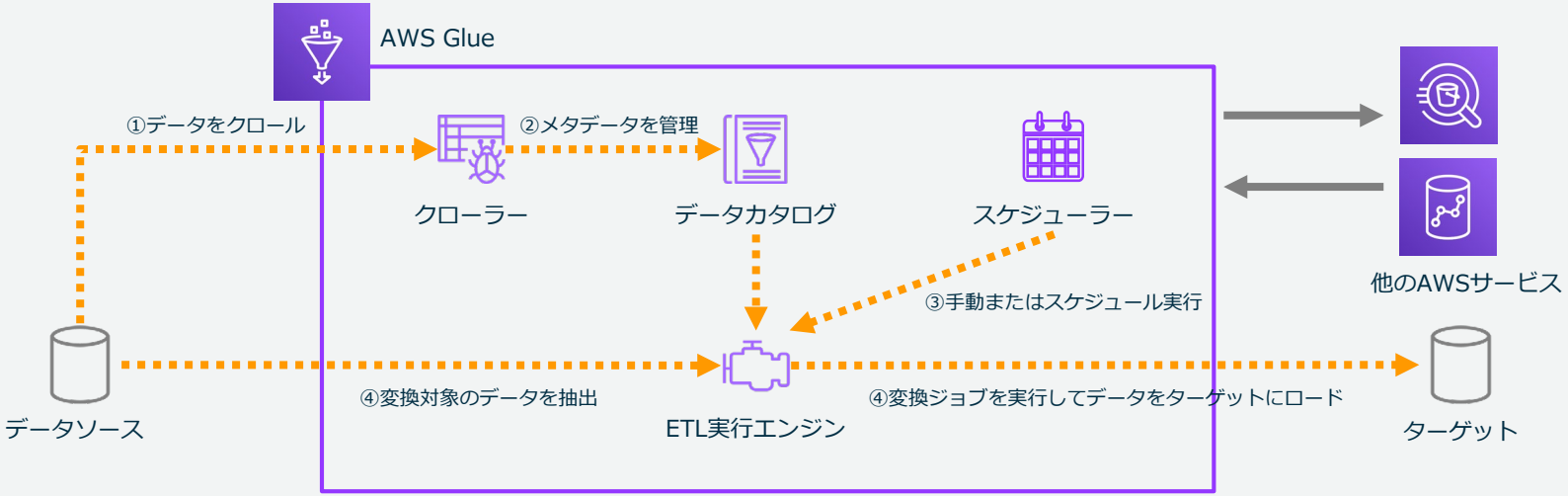
VPC内からのアクセス

他のAWSサービスと
容易に連携

セキュア

Notebookでの開発

AWS Glueの全体像



概要

①クローラーにてデータソースのメタデータをクロールして、データカタログに登録・更新

②データカタログにてメタデータを管理

③スケジューラーにてジョブの実行タイミングを定義

④データソースからデータを抽出し、ETL実行エンジンにてジョブをサーバーレスで実行
(ジョブはSpark(PySpark、Scala)またはPython Shellを選択)

参考：AWS Glue DataBrew

New！

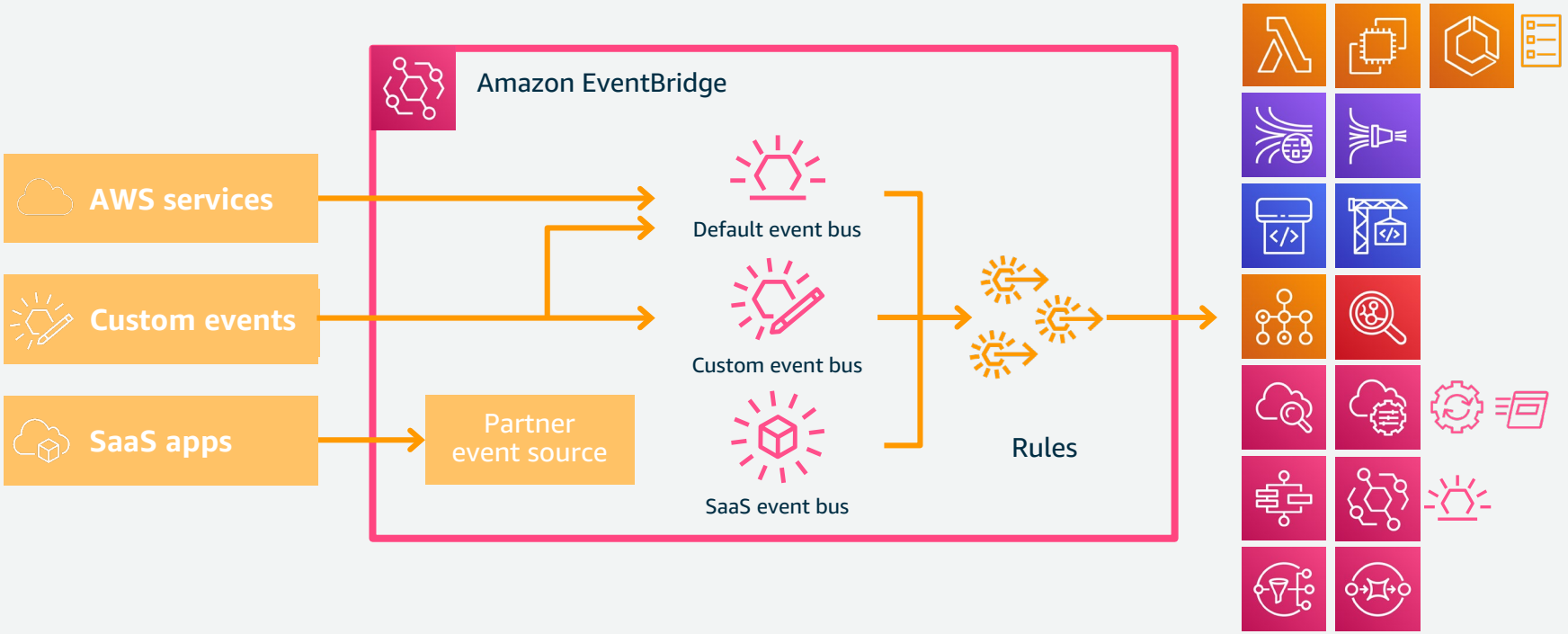
- データのクリーニング、正規化をコーディングなしで実現するビジュアルデータプレパレーションツール
- 異常値フィルタ、フォーマット変換など250以上の組み込みトランスフォーメーション
- サーバーレスで直感的なインターフェイス

https://docs.aws.amazon.com/ja_jp/databrew/latest/dg/what-is.html

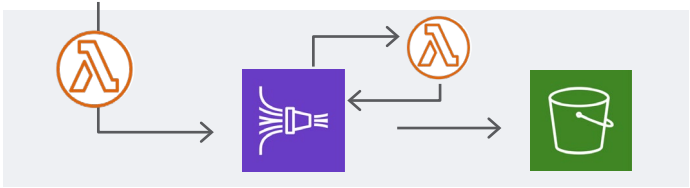
スケジュール・ジョブ/CRON SaaS イベント連携

アーキテクチャ図	
利用サービス	• Amazon EventBridge • Amazon CloudWatch • AWS Lambda
ユースケース	• 一定時間ごとのジョブやアラートなどのシグナル、SaaS イベントから処理を起動
設計ポイント	• Amazon EventBridgeはサーバーレスのイベントバスサービス • SaaS、カスタムアプリ、AWSサービスをソースとしてイベントを発火、AWS Lambdaなどを実行するイベント駆動型アプリケーションの実装に最適 • 以前CloudWatch Eventと呼ばれていた機能はEventBridgeに統合 • イベントターゲットとしてLambda関数など18のAWSサービスと4つのビルドインAPI Call、クロスアカウントイベントをサポート

Amazon EventBridge アーキテクチャ

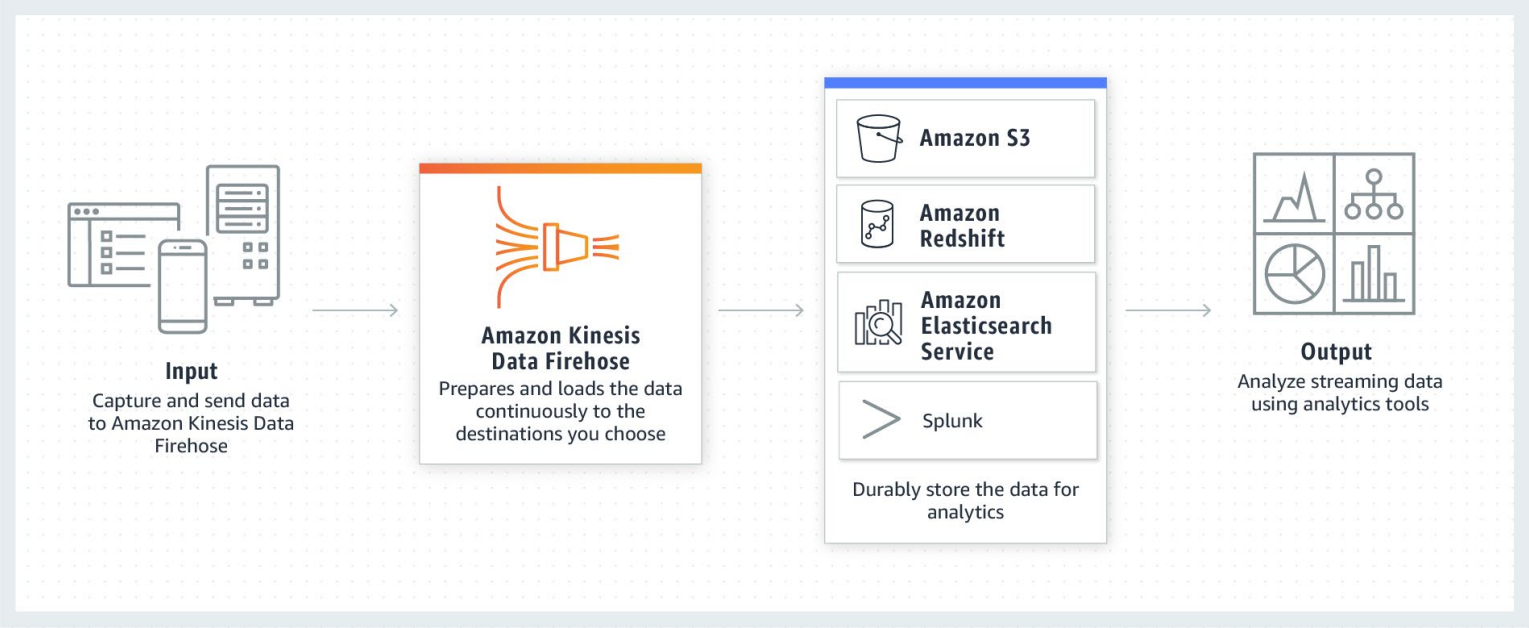


ログデータ収集処理

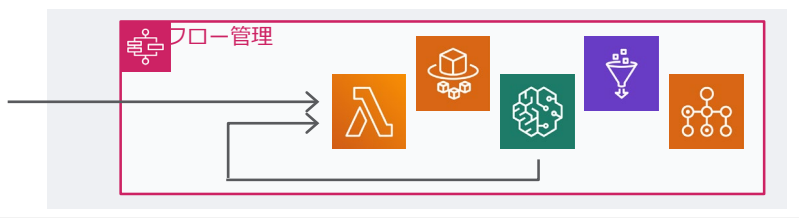
アーキテクチャ図	
利用サービス	• AWS Lambda • Amazon Kinesis Data Firehose • Amazon S3
ユースケース	• ログイベントを受信し、必要に応じてデータ加工しながら S3 へ格納
設計ポイント	• Kinesis Data FirehoseはLambda関数を実行し入力データを変換し送信することが可能（Lambdaの呼び出し時間は最大5分） • Kinesis Data Firehose data transformationを有効にすると3MBまでデータをバッファリングすることが可能で、バッファリングされた個々のバッチでLambda関数を同期モードで呼び出し、変換結果をバッファリングして送信

Amazon Kinesis Data Firehose

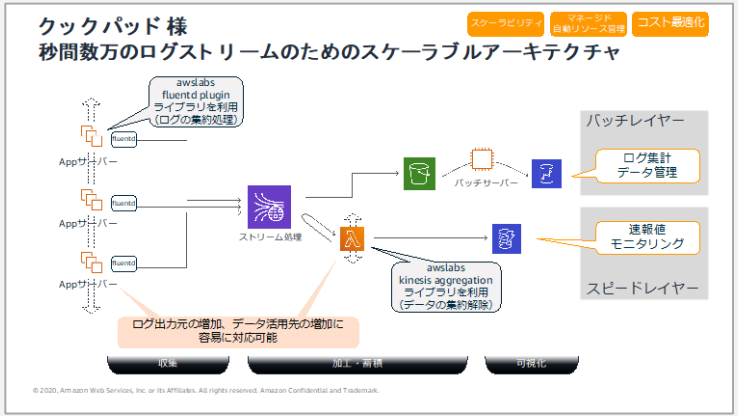
- Amazon S3、Amazon Redshift、Amazon Elasticsearch Serviceと統合
- 汎用HTTPエンドポイントやSplunkなどのサービスプロバイダーに直接データを配信可能

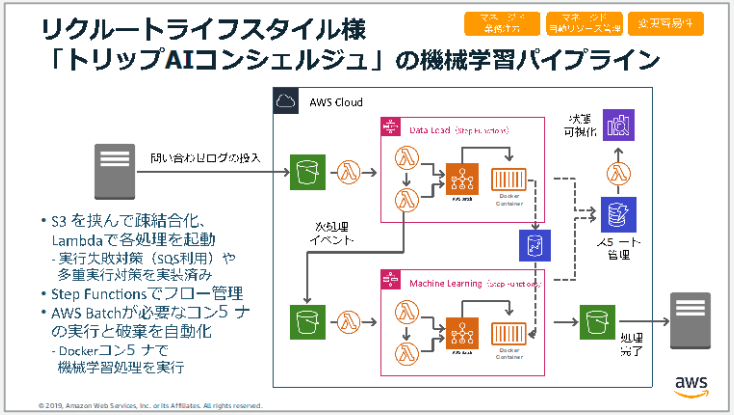


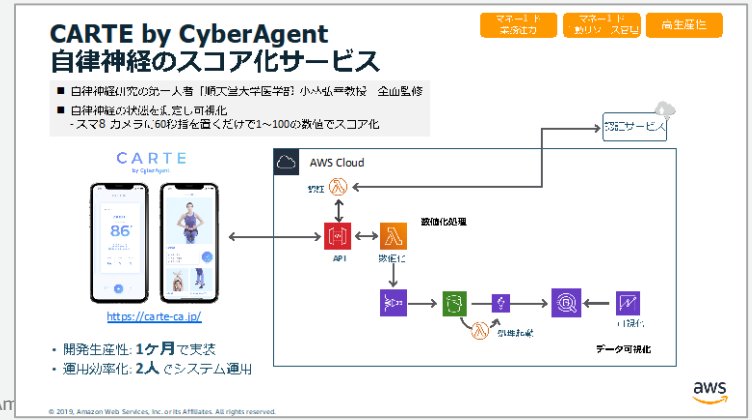
機械学習/ETLデータパイプライン

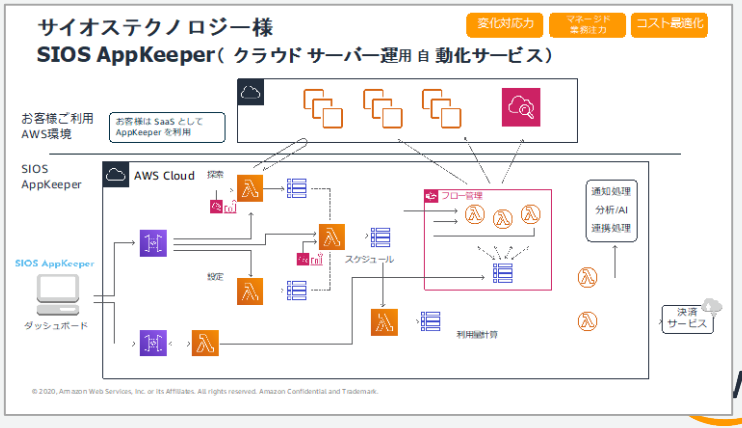
アーキテクチャ図	
利用サービス	• AWS Step Functions • AWS Lambda • AWS Fargate • Amazon SageMaker • AWS Glue • AWS Batch
ユースケース	• 一連のデータ加工や集計処理、学習処理、後処理をフローで管理
設計ポイント	• AWS StepFunctionsを利用することで異なるビジネスデータの処理を組み合わせて新たなデータを生成することが可能 • フルサーバーレスで構築することで運用コストを大幅に削減 • Amazon SageMakerと連携し機械学習モデルの構築（トレーニング環境の構築、モデルのトレーニング、推論の構築、モデルの作成）とデプロイ（エンドポイントの作成、更新）を自動化

ユースケースパターン: バックエンドデータ処理 – 活用例



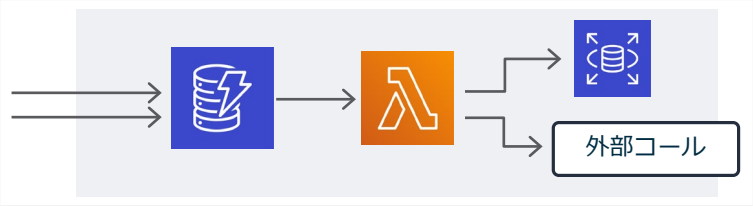




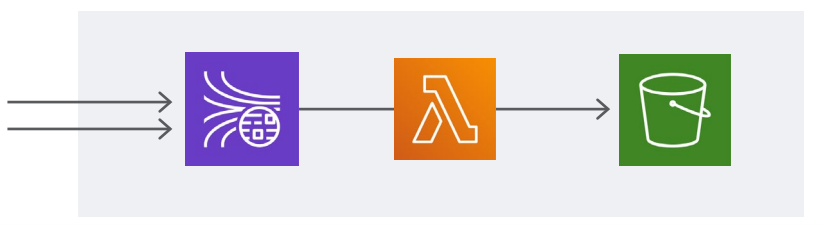


ユースケースパターン④：データイベント処理

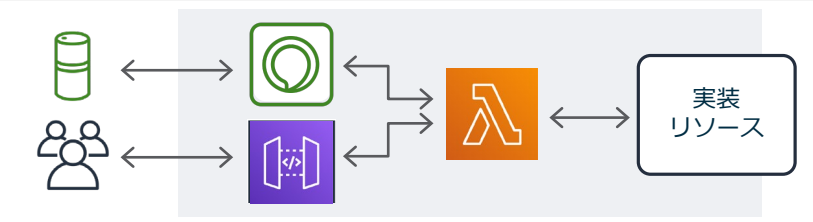
データ変更トリガー（変更に起因する処理の実行）

アーキテクチャ図	
利用サービス	• Amazon DynamoDB • AWS Lambda • Amazon RDS
ユースケース	• DynamoDBに実行されたデータ変更処理に反応したイベント処理
設計ポイント	• AWS Lambdaはストリームのシャードを秒間4回のレートでポーリングし、新しいストリームのレコードを検出するとLambda関数を同期的に呼び出して結果を待機。 • Lambdaが一度に処理するレコード数をバッチサイズ（最大1,000）で指定、指定したレコード数が溜まるまで最大5分待機 • エラーによるリトライの繰り返し（ポイズンメッセージ）を防ぐため、スプリットしてリトライ、リトライ回数、古くなったレコードの廃棄を設定可能。廃棄されたイベントはSQSかSNSに送信可能 • 並列化係数により同一シャードで複数バッチを実行（最大10個）

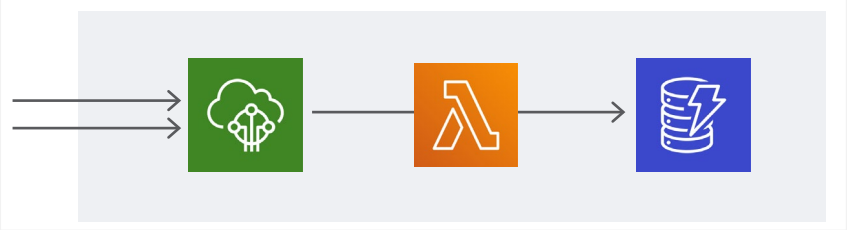
流入データの連続処理

アーキテクチャ図	
利用サービス	• Amazon Kinesis Data Streams • AWS Lambda • Amazon S3
ユースケース	• Kinesis に流入するデータを定期的に受信してデータ加工を施して格納
設計ポイント	• 連続して送信されるストリームデータを取りこぼすことなく処理することが可能 • 送信されるデータ量に応じてKinesis Data Streamsのシャード数を指定（APIでの設定も可能） • ストリーミングデータを変換してAmazon S3やAmazon RedShift、Amazon Elasticsearch Serviceなどに保存するユースケースであればAmazon Kinesis Data Firehoseの利用も検討

チャットボット / Alexa スキル

アーキテクチャ図	
利用サービス	• Amazon Alexa • Amazon API Gateway • AWS Lambda
ユースケース	• テキスト入力やAlexaからの音声入力に反応したイベント処理
設計ポイント	• Alexa Skill Kitによるスキルの作成については以下のURLを参照 https://developer.amazon.com/ja-JP/docs/alexa/ask-overviews/build-skills-with-the-alexa-skills-kit.html • カスタムスキルのビルド手順 https://developer.amazon.com/ja-JP/docs/alexa/custom-skills/steps-to-build-a-custom-skill.html • サンプルとテンプレート https://github.com/alexa

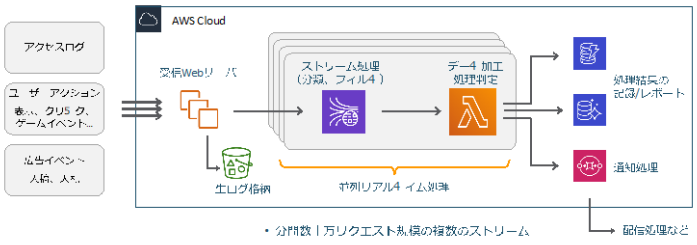
IoTバックエンド

アーキテクチャ図	
利用サービス	• AWS IoT Core • AWS Lambda • Amazon DynamoDB
ユースケース	• IoT機器からのデータ（IoTイベント）に反応したイベント処理
設計ポイント	• AWS IoT Coreを利用することでデバイスのクライアント証明書を用いたセキュアで双方向の通信をMQTTで実行可能 • IoT Rules engineによって様々なAWSサービスと連携が可能 • 詳細は下記のドキュメントを参照 「What is AWS IoT?」 https://docs.aws.amazon.com/iot/latest/developerguide/what-is-aws-iot.html

ユースケースパターン: データイベント処理 – 活用例

Zucks様
アドネットワークにおけるリアルタイムストリーム処理

リアルタイム　スケーラビリティ　変更容易性



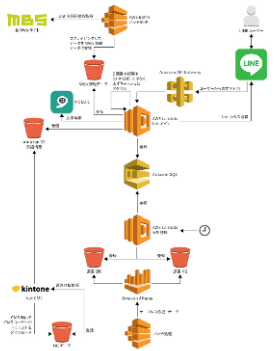
・分間数１万リクエスト規模の複数のストリーム

毎日放送様 オフィシャルLINE BOT
『おしゃべり らいよんチャン』

マネージドリソース自動管理　トランジション高速改革

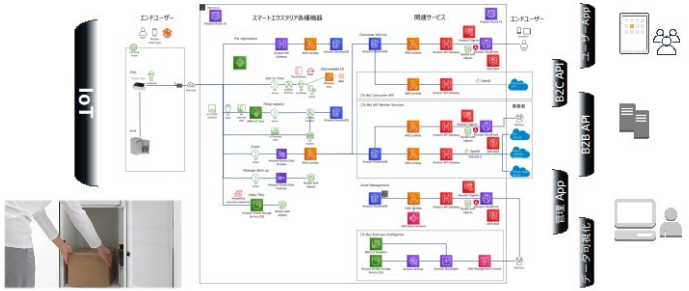
要件の特徴
- 時間帯によるピーク性が正確に想定できない
- テレビ番組と連動した際の突発的なアクセスの増加を事前に予測できない

サーバーレス「Lambda」の採用
- サーバーの運用管理と
  4 ャパシティ管理が不要
- 運用・保守の6 スト削減と
  アクセス負荷の変動に強い



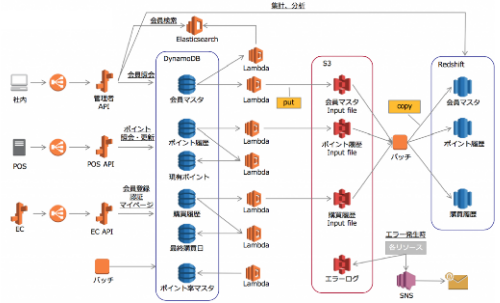
LIXIL 様
スマート 宅配ボスト サービス

変更容易性　マネージドリソース自動管理　コスト最適化



東急ハンズ様
サーバーレスによるポイント処理システム

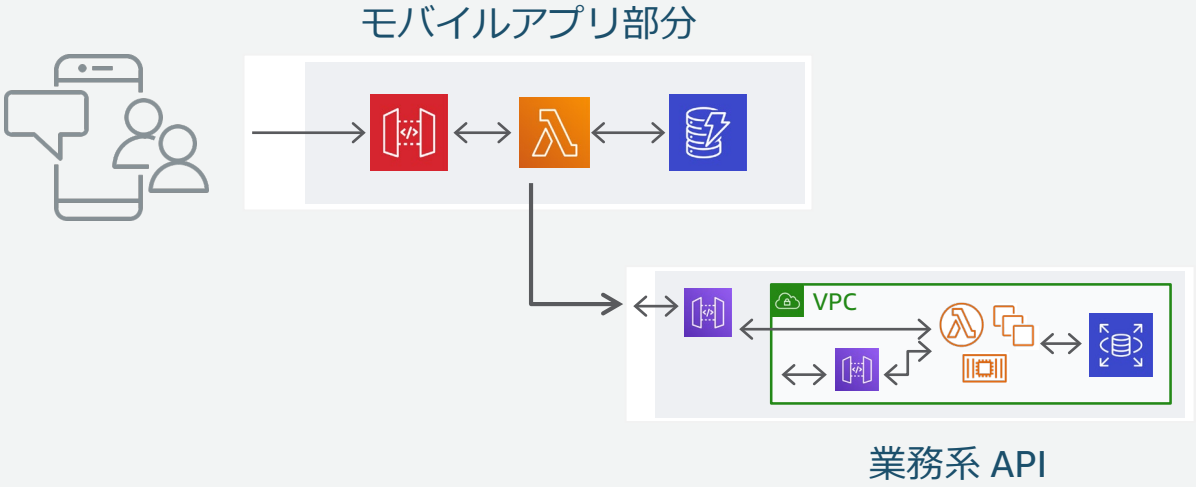
スケーラビリティ　マネージドリソース自動管理　コスト最適化



- 約500万の会員の
  3 イント管理
  - 高可用性
  - 繁忙時/バーゲン時の処理負荷への対応
  - 反対に、平常時のシステムリソースの無駄の排除
  - 運用の省力化

組み合わせ活用例

モバイル + API

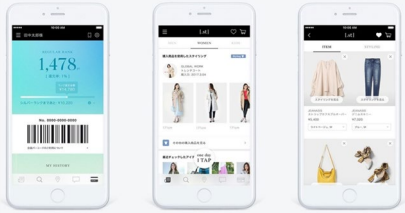


マネージド
自動リソース管理

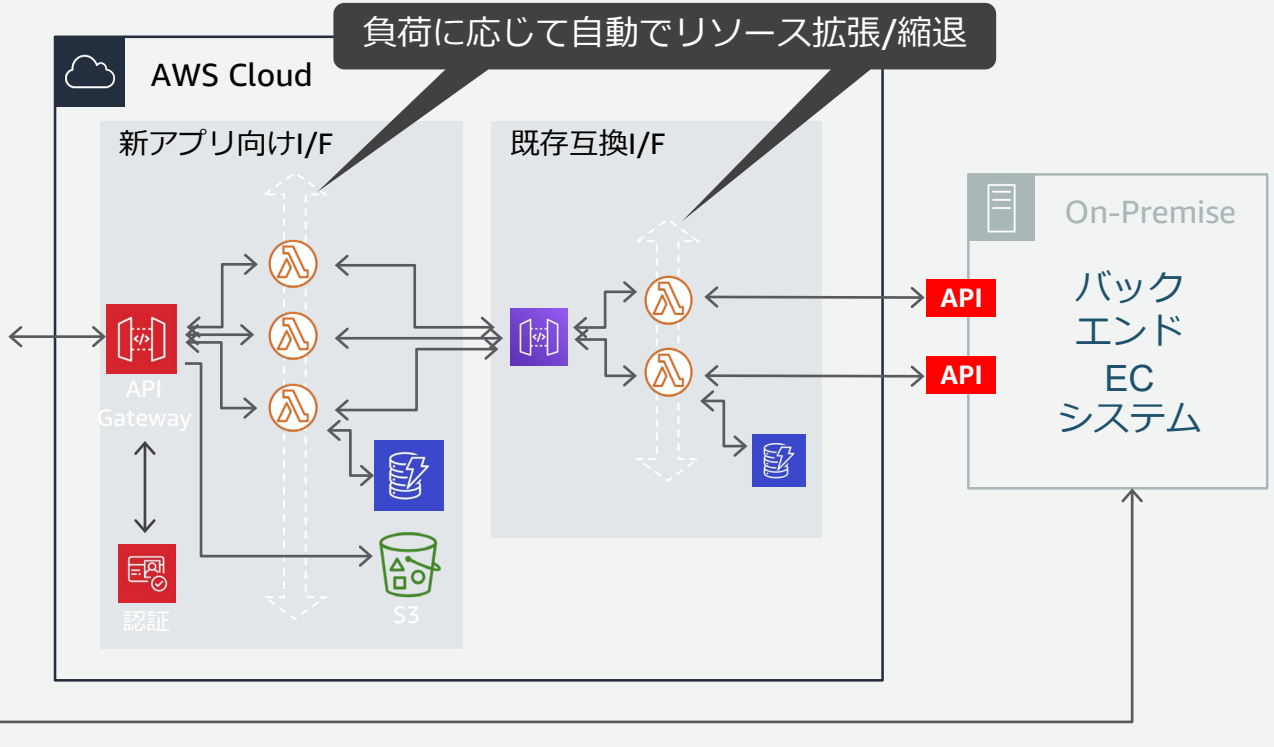
マネージド
業務注力

アダストリア様
サーバーレスによるモバイルバックエンド

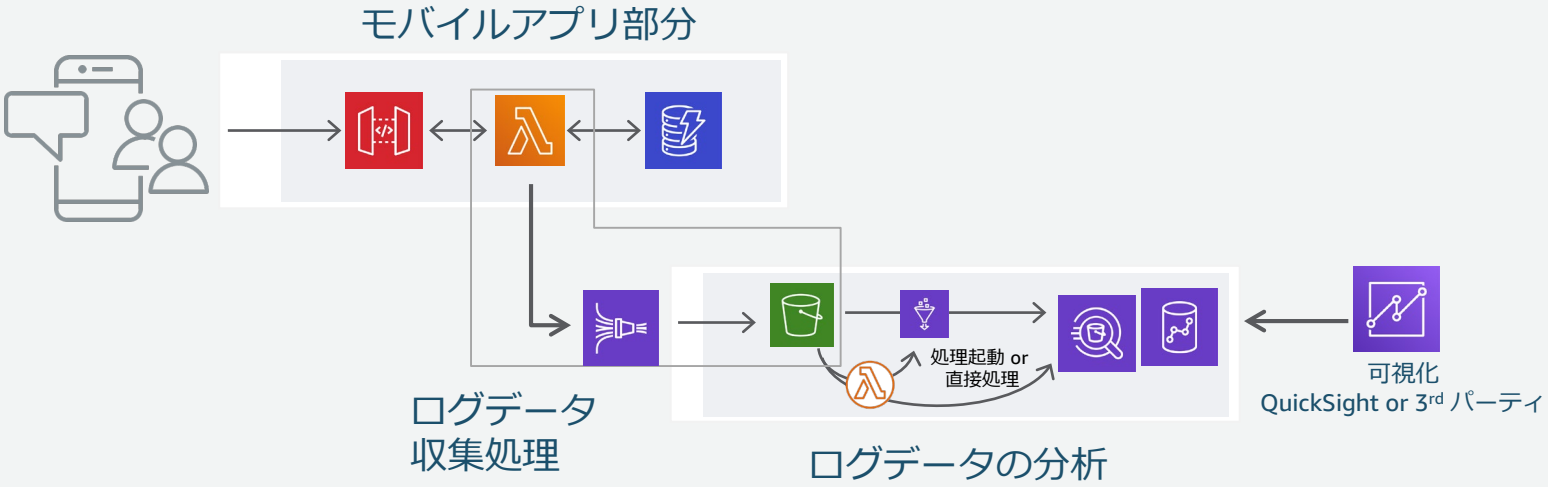
- モバイルアプリ「.st」



- PC Web用 ECサイト



ユーザー行動モニタリング型モバイルアプリの例



マネージド
業務注力

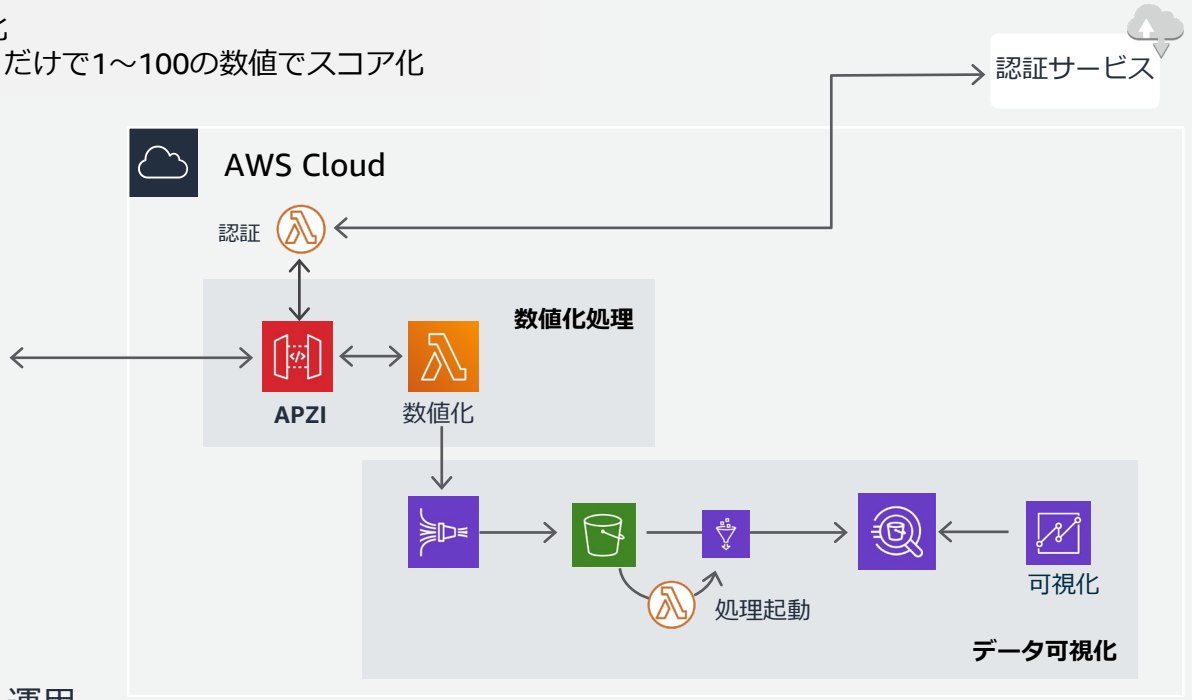
マネージド
自動リソース管理

高生産性

CARTE by CyberAgent
自律神経のスコア化サービス

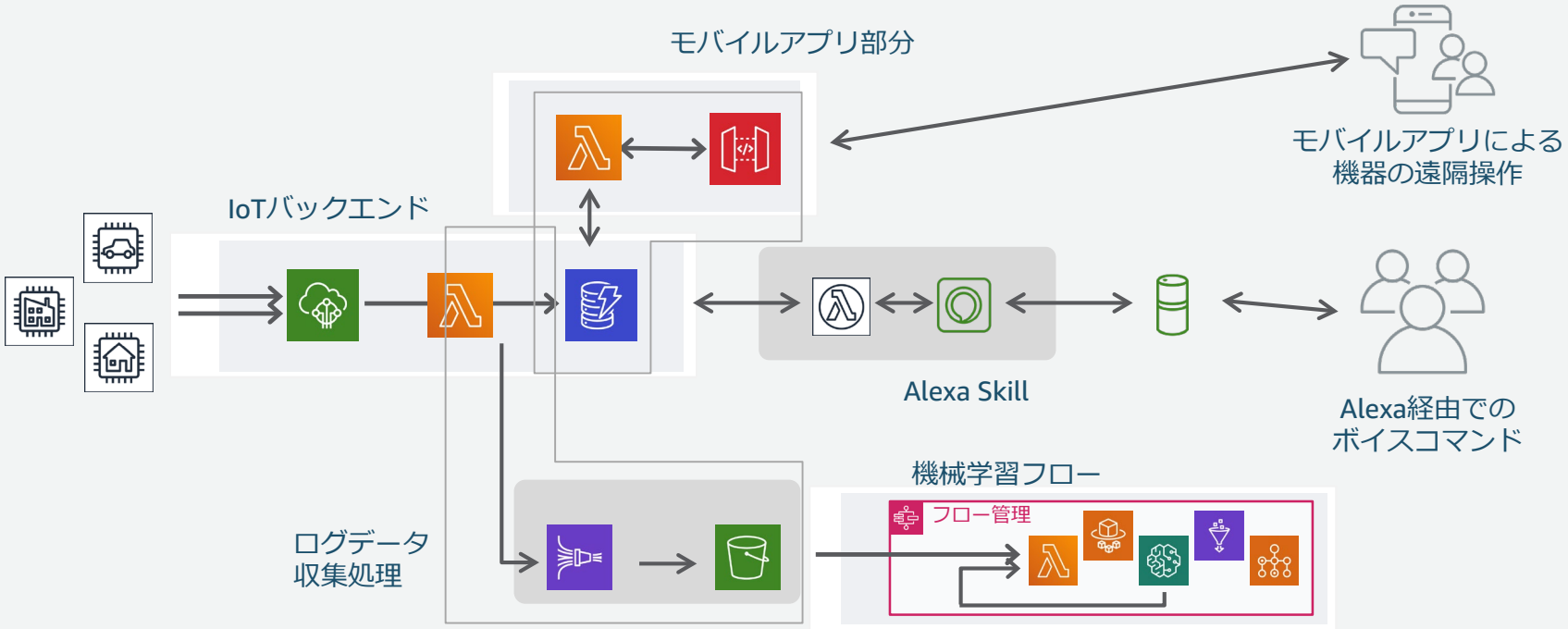
- 自律神経研究の第一人者「順天堂大学医学部 小林弘幸教授」全面監修
- 自律神経の状態を測定し可視化
  - スマホカメラに60秒指を置くだけで1～100の数値でスコア化

https://carte-ca.jp/



- 開発生産性: **1ケ月**で実装
- 運用効率化: **2人**でシステム運用

IoTアプリケーションの例

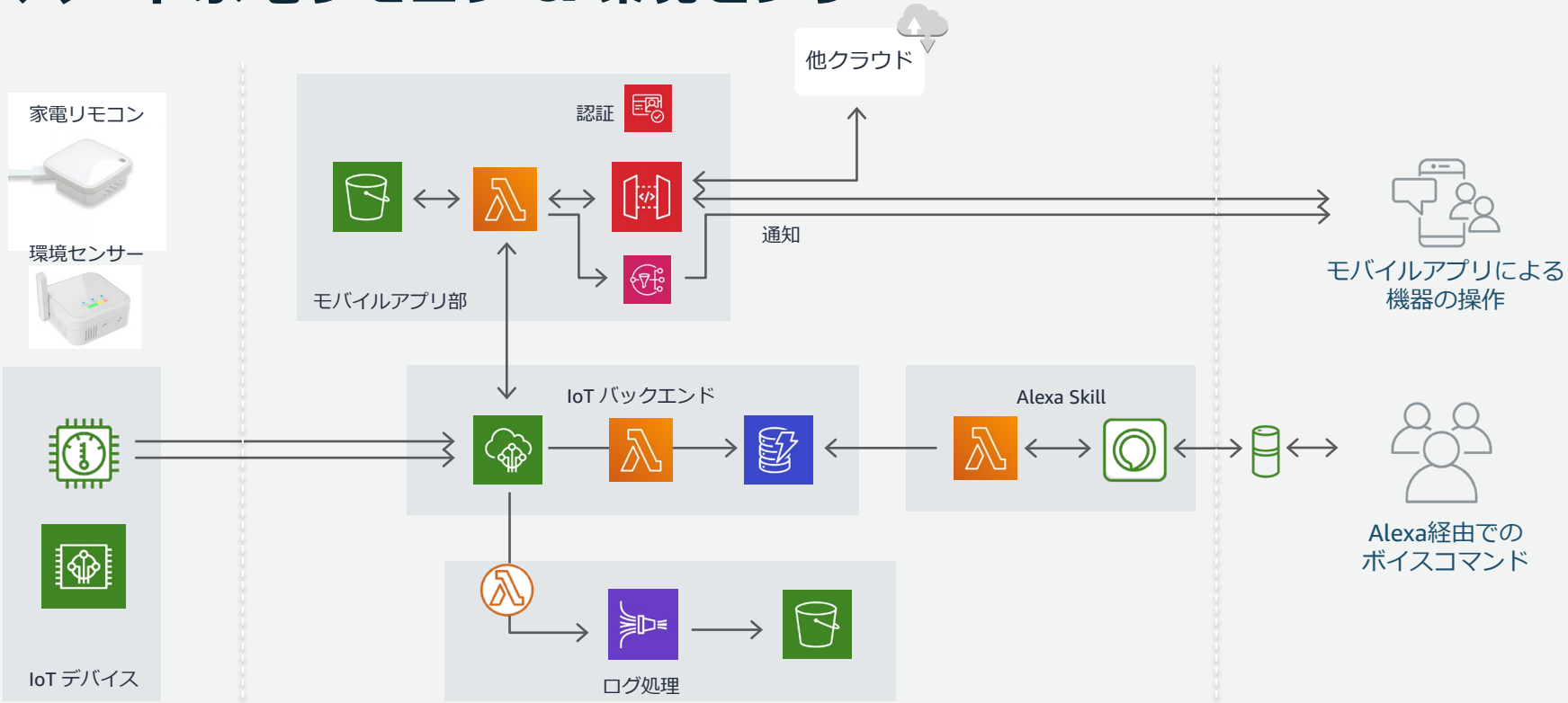


マネージド
業務注力

スケーラビリティ
（機会損失防止）

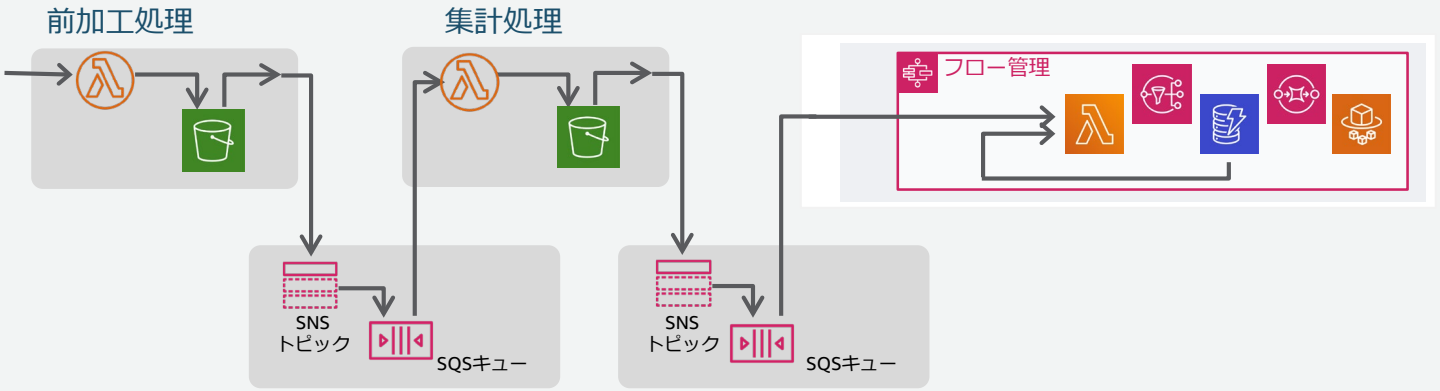
コスト最適化

ラトックシステム様
スマート家電リモコン & 環境センサー



aws

データ処理パイプライン



イベント駆動の業務処理連携パターン

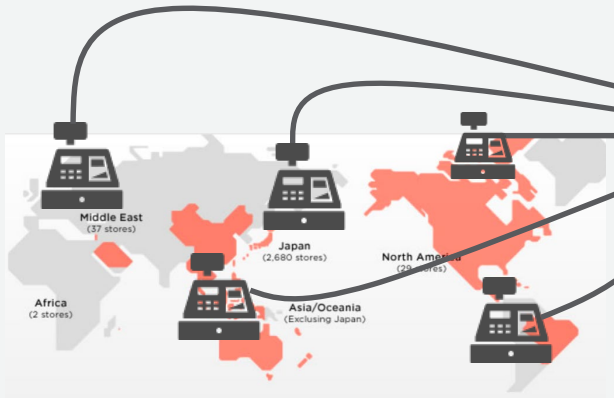
マネージド
業務注力

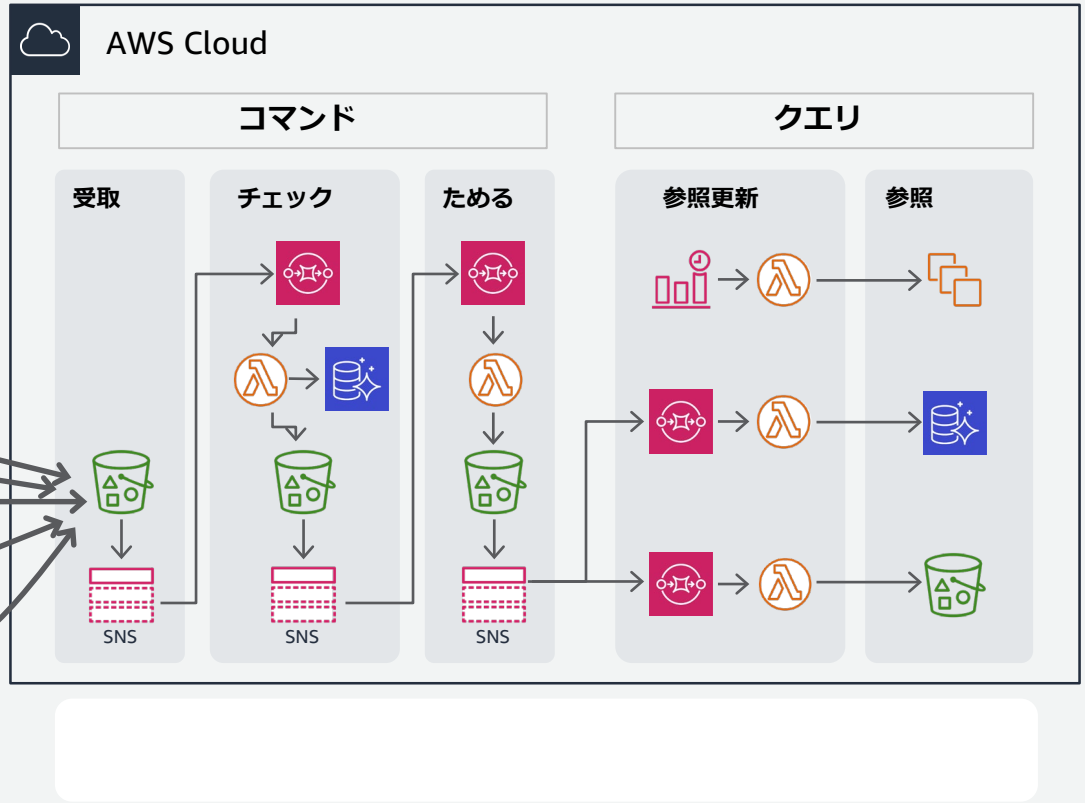
マネージド
自動リソース管理

変更容易性

ダイソー様
サーバーレスによるPOSデータ処理

- 5,000 を超える店舗、70,000 以上の商品点数（2018/02時点）
- 今後のデータ増加に自動でリソース拡張





aws

補足情報：3つのサーバーレス関連Webページ

Project 責任者向け
ビジネス価値とは?
- 事例 Pickup
- IDC調査レポート

amzn.to/2WeZ1xz

アーキテクト向け
ユースケースパターン
- 16 のパターン
- 組み合わせ活用例

amzn.to/2UJT4bB

これからの開発者向け
サーバーレス技術情報
- ハンズオン・技術資料
- 開発環境、Tips...

amzn.to/2WeZuQl

まとめ

- 本日ご紹介した16のユースケースパターンは、いずれも実システムにおいて稼働実績のあるサービスの組み合わせ事例です。

- 構築したいシステムの利用目的が、これらのユースケースのいずれか、または複数に合致する場合は、これらのパターン構成を参考にシステムを設計してみてはいかがでしょう。

- アーキテクチャは目的に従って機能要件、非機能要件を定義し、要件と制約を満たすために処理方式を選択します。そのため、唯一絶対のアーキテクチャはありませんが、より良いアーキテクチャを構築するために先人の知恵を借りるのは良い方法ではないでしょうか。

Q&A

お答えできなかっ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

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]

Appendix

参照情報

- 動的 Web/モバイルバックエンド
  - AWSサーバーレス多層アーキテクチャ
    https://d1.awsstatic.com/International/ja_JP/Whitepapers/AWS-Serverless-Multi-Tier-Architectures_JA.pdf
  - チュートリアル
    https://aws.amazon.com/jp/serverless/build-a-web-app/
  - チュートリアル（中級編）
    https://aws.amazon.com/jp/getting-started/projects/build-modern-app-fargate-lambda-dynamodb-python/
  - テンプレートから始める
    https://console.aws.amazon.com/lambda#/create/application/view?applicationId=web-backend

参照情報

- リアルタイムモバイル/オフライン対応
  - 関連資料
    https://d1.awsstatic.com/webinars/jp/pdf/services/20180523_AWS-BlackBelt_AppSync.pdf
- 業務系 API/グループ企業間API
  - Private API 記事
    https://aws.amazon.com/blogs/compute/introducing-amazon-api-gateway-private-endpoints/
  - VPC Lambda 記事
    https://aws.amazon.com/jp/blogs/news/announcing-improved-vpc-networking-for-aws-lambda-functions/
  - RDS+Lambda 記事
    https://aws.amazon.com/jp/blogs/news/onlineseminar-rds-lambda-doc-qa/
  - 関連事例
    https://kabu.com/company/pressrelease/20180807_1.html

参照情報

- Push 配信系・インタラクティブAPI
  - 関連リンク
    https://aws.amazon.com/jp/blogs/news/announcing-websocket-apis-in-amazon-api-gateway/
  - AppRepositoryサンプル
    https://serverlessrepo.aws.amazon.com/applications/arn:aws:serverlessrepo:us-east-1:729047367331:applications~simple-websockets-chat-app

参照情報

- 画像処理/シンプルなデータ加工
    - Tutorial
      https://docs.aws.amazon.com/lambda/latest/dg/with-s3-example.html
    - 関連事例
      https://speakerdeck.com/sho3334/images-object-zozo
    - Solution リンク
      https://aws.amazon.com/solutions/serverless-image-handler/
    - テンプレートから始める
      https://console.aws.amazon.com/lambda#/create/application/view?applicationId=file-processing

参照情報

- 分散並列処理
  - 関連事例1
    https://d1.awsstatic.com/events/jp/2017/summit/devday/D4T8-4.pdf
  - 関連事例2
    https://www.slideshare.net/AmazonWebServices/serverless-design-patterns-for-rethinking-traditional-enterprise-application-approaches-aws-public-sector-summit-2017/18
  - RefArch
    https://github.com/awslabs/lambda-refarch-mapreduce

参照情報

- イベント駆動の業務処理連携
  - SQS 連携機能
    https://aws.amazon.com/jp/serverless/patterns/sqs-lambda/
  - サンプルコード
    https://docs.aws.amazon.com/lambda/latest/dg/with-sqs-create-package.html
  - テンプレートから始める
    https://console.aws.amazon.com/lambda#/create/application/view?applicationId=queue-processing
- アプリケーションフロー処理
  - Tutorial Workflow
    https://aws.amazon.com/getting-started/tutorials/create-a-serverless-workflow-step-functions-lambda/
  - エラー処理
    https://aws.amazon.com/getting-started/tutorials/handle-serverless-application-errors-step-functions-lambda/
  - 短期間・高速処理のためのオプション
    https://aws.amazon.com/jp/about-aws/whats-new/2019/12/introducing-aws-step-functions-express-workflows/

参照情報

- 流入データの連続処理
  - 関連資料
    https://d1.awsstatic.com/International/ja_JP/Whitepapers/Serverless_Streaming_Architecture_Best_Practices_JA.pdf
  - RefArch
    https://github.com/aws-samples/lambda-refarch-streamprocessing
  - Tutorial1
    https://docs.aws.amazon.com/lambda/latest/dg/with-kinesis-example.html
  - Tutorial2
    https://aws.amazon.com/jp/getting-started/projects/build-serverless-real-time-data-processing-app-lambda-kinesis-s3-dynamodb-cognito-athena/

参照情報

- IoT バックエンド
  - 関連資料
    https://d1.awsstatic.com/International/ja_JP/Whitepapers/Serverless_Streaming_Architecture_Best_Practices_JA.pdf
  - 関連事例
    https://aws.amazon.com/jp/solutions/case-studies/sony/
  - RefArch
    https://github.com/aws-samples/lambda-refarch-iotbackend
  - 関連 Solution1
    https://aws.amazon.com/solutions/smart-product-solution/
  - 関連Solution2
    https://aws.amazon.com/solutions/aws-connected-vehicle-solution/
  -

参照情報

- チャットボット / Alexa スキル
  - Alexa スキル開発
    https://developer.amazon.com/alexa-skills-kit/
  - RefArch
    https://github.com/aws-samples/lambda-refarch-image-moderation-chatbot
  - Solution リンク
    https://aws.amazon.com/solutions/serverless-bot-framework/
- データ変更トリガー処理
  - 活用例
    https://aws.amazon.com/jp/blogs/news/anomaly-detection-on-amazon-dynamodb-streams-using-the-amazon-sagemaker-random-cut-forest-algorithm/
  - Tutorial
    https://docs.aws.amazon.com/lambda/latest/dg/with-ddb-example.html

参照情報

- ログデータ収集処理
  - 関連記事
    https://aws.amazon.com/jp/blogs/news/analyze-and-visualize-your-vpc-network-traffic-using-amazon-kinesis-and-amazon-athena/
  - 関連事例
    https://www.youtube.com/watch?v=rsZjU5g_yXI
  - 関連 Solution
    https://aws.amazon.com/solutions/implementations/game-analytics-pipeline/
  - データ変換ブループリント
    https://docs.aws.amazon.com/firehose/latest/dev/data-transformation.html

参照情報

- データレイク周りのデータ加工
  - Solution リンク
    https://aws.amazon.com/answers/big-data/data-lake-solution/
  - DB Loader
    https://github.com/awslabs/aws-lambda-redshift-loader
  - より包括的なソリューション
    https://aws.amazon.com/lake-formation/
- 機械学習/ETLデータパイプライン
  - 関連記事1
    https://aws.amazon.com/jp/blogs/news/orchestrate-multiple-etl-jobs-using-aws-step-functions-and-aws-lambda/
  - 関連記事2
    https://aws.amazon.com/jp/blogs/news/automated-and-continuous-deployment-of-amazon-sagemaker-models-with-aws-step-functions/
  - 機能紹介動画
    https://www.youtube.com/watch?v=dNb5jVffzPs
  - 関連事例
    https://www.slideshare.net/shoujishirotori/the-design-for-serverless-etl-pipeline-489

参照情報

- スケジュール・ジョブ/SaaS イベント
  - 関連 Doc
    https://docs.aws.amazon.com/lambda/latest/dg/with-scheduled-events.html
  - Template
    https://docs.aws.amazon.com/lambda/latest/dg/with-scheduledevents-example-use-app-spec.html
  - 活用 Solution
    https://aws.amazon.com/solutions/fraud-detection-using-machine-learning/
  - テンプレートから始める
    https://console.aws.amazon.com/lambda#/create/application/view?applicationId=scheduled

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