ちょっぴり DIVE DEEP

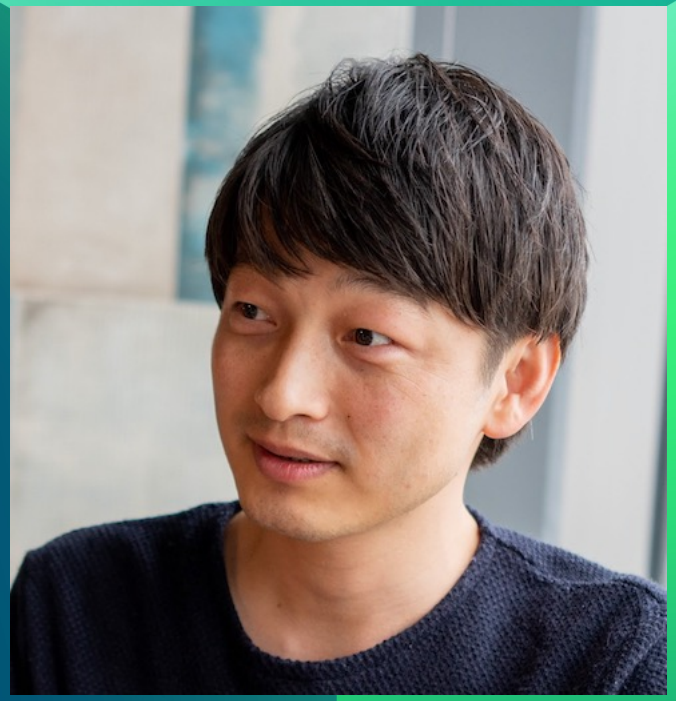
サーバーレス流のちょっぴりお得なデザインパターン講座

Kensuke Shimokawa

Snr. Serverless Specialist
Amazon Web Services Japan

Kensuke Shimokawa

Amazon Web Services Japan
Snr. Serverless Specialist



_kensh

Slides https://speakerdeck.com/_kensh/
Qiita https://qiita.com/_kensh

Agenda

- サーバーレスサービスの紹介
- サーバーレスな Webアプリケーション
- モノリスとの付き合い方
- サーバーレスな ファイル変換処理
- サーバーレスな マルチテナント管理

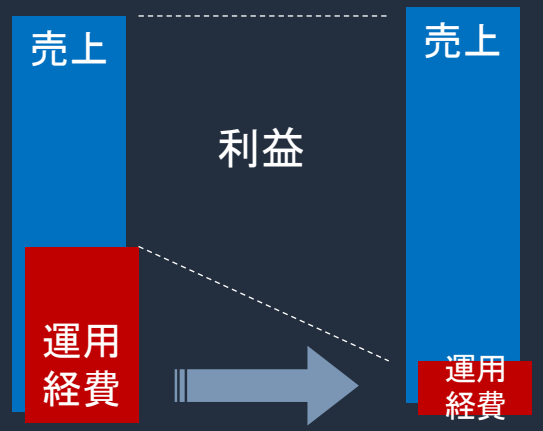
これからお話すること

- サーバーレスサービスの特徴を用いたアプリケーションの考え方
- ビジネスロジックにフォーカスするアーキテクチャの導入
- イベント駆動に処理をすることの大切さ

サーバーレスサービスの紹介

サーバーレスの効能

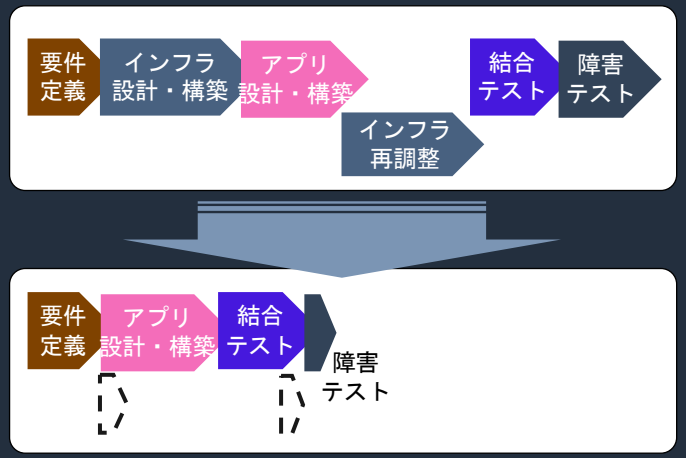
運用工数の削減



運用工数の削減によって

利益増加に貢献

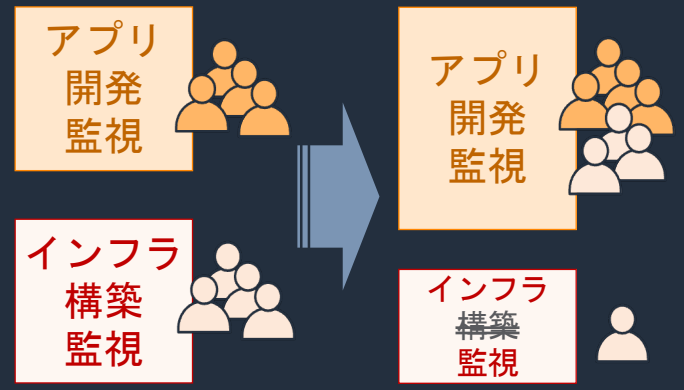
設計工数の削減



インフラ関連の工数を減らし

市場投入スピードを**短縮**

生産性へのシフト

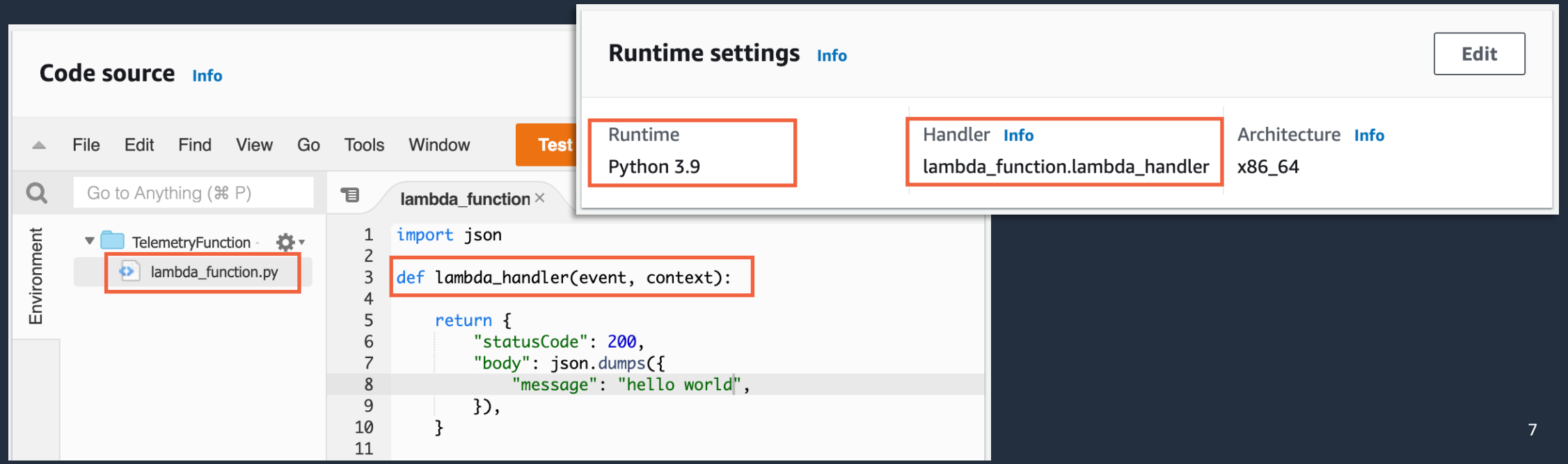


人員を**生産的な施策**に

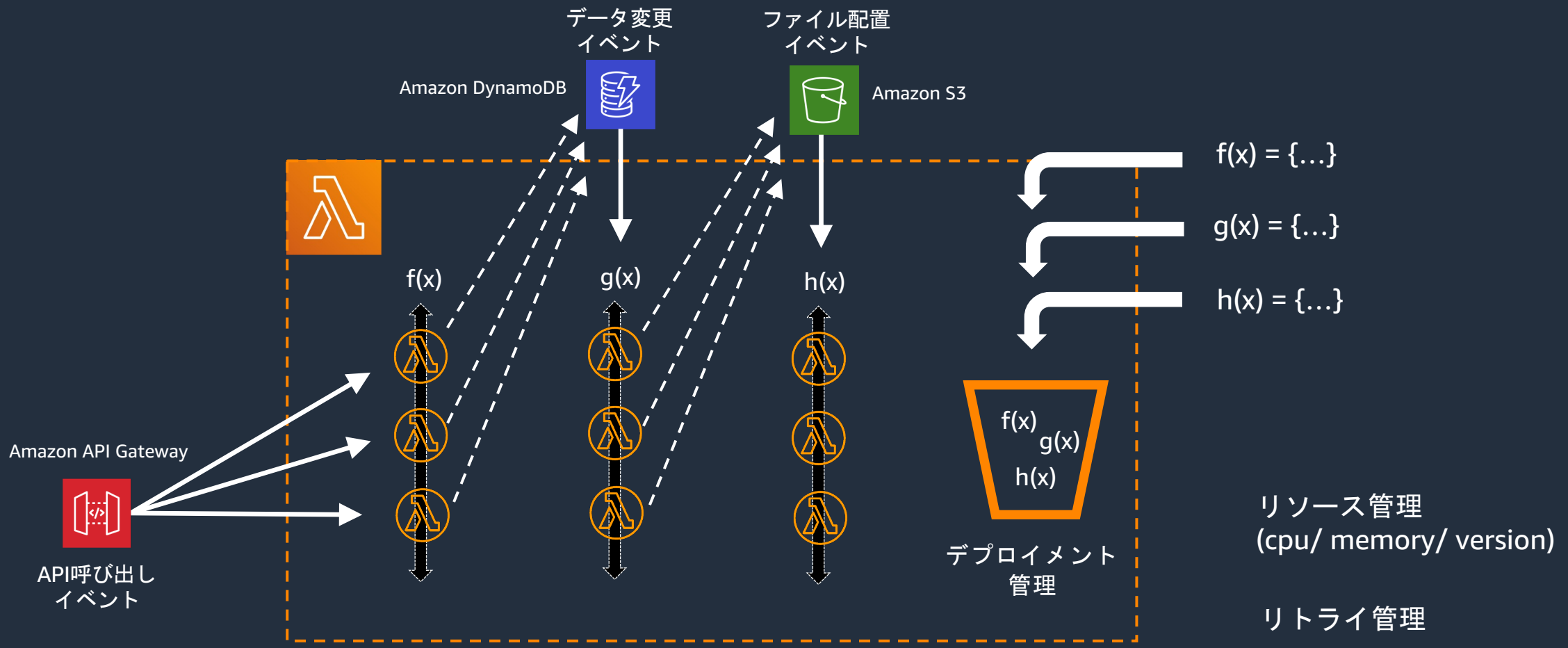
重点的に配置可能

AWS Lambda におけるコーディング

- 対応言語
  - Node.js、Python 、Java、Go、Ruby、C# 、PowerShell
  - サポートされていない言語は、カスタムランタイムを実装することで利用可能
- ハンドラーで呼び出す関数を指定する
  - Python のデフォルトでは lambda_function.lambda_handler



AWS Lambda のスケールの仕組み

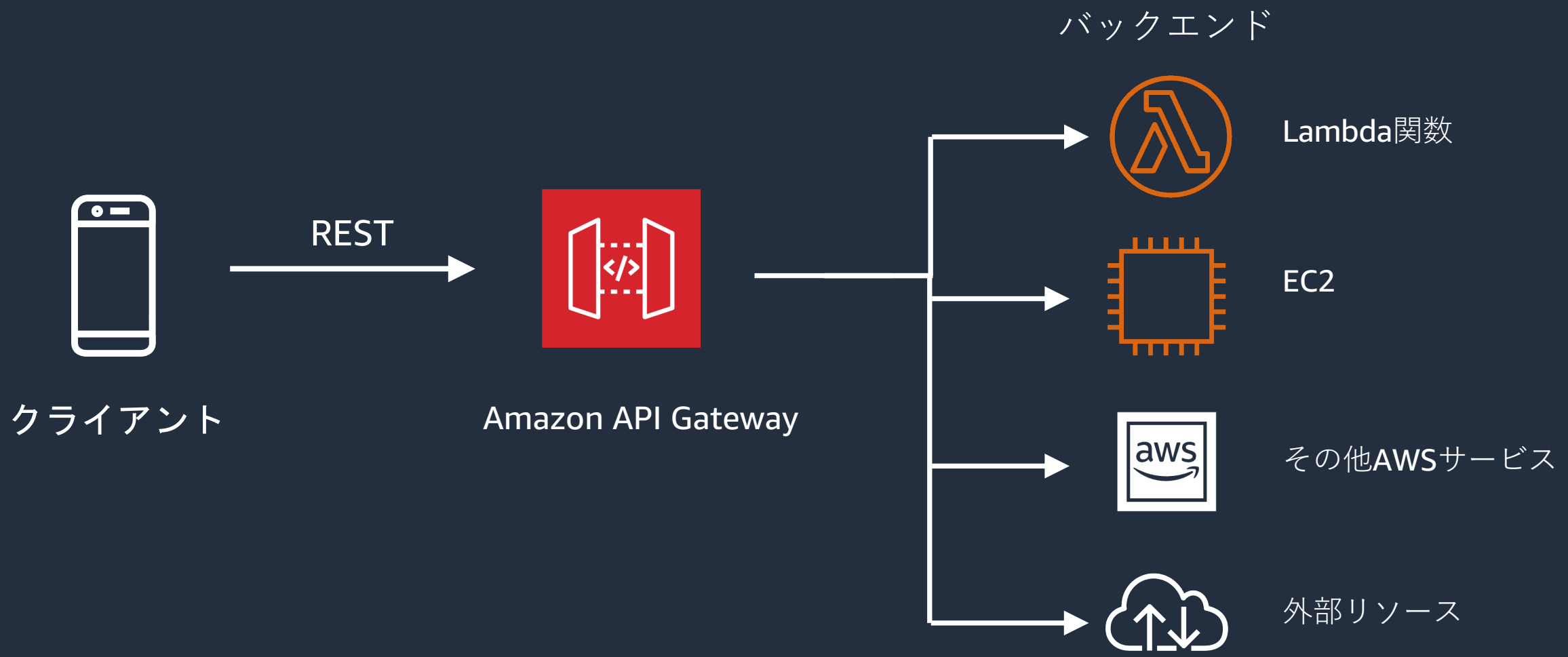


リソース管理
(cpu/ memory/ version)

リトライ管理

ログ出力
トレース管理

Amazon API Gateway の豊富なバックエンド統合

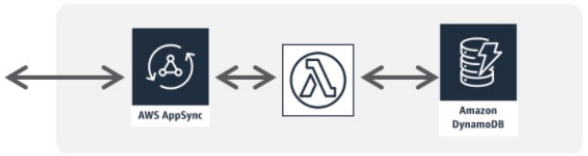


サーバーレスなWebアプリケーション

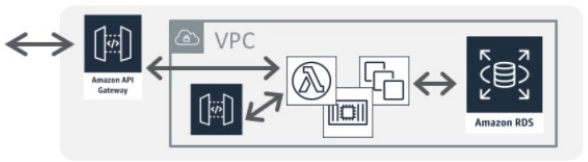
形で考えるサーバーレス設計

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
関連資料 | Tutorial | Tutorial（中級編）
テンプレートから始める → こちら



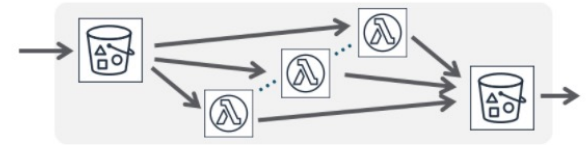
リアルタイムモバイル / オフライン対応
AWS マンガ | ワークショップ
Solution リンク



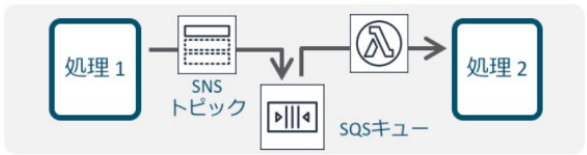
業務系 API / グループ企業間 API
Tutorials | Private API 記事 | 関連事例
OpenAPI の利用

Push 配信系・インタラクティブ API
関連リンク | AppRepositoryサンプル
解説動画（英語）

画像処理 / シンプルなデータ加工
Tutorial | 関連事例 | Solution リンク
[New] S3 Object Lambda
テンプレートから始める → こちら



分散並列処理（like MapReduce）
関連事例1 | 関連事例2
RefArch

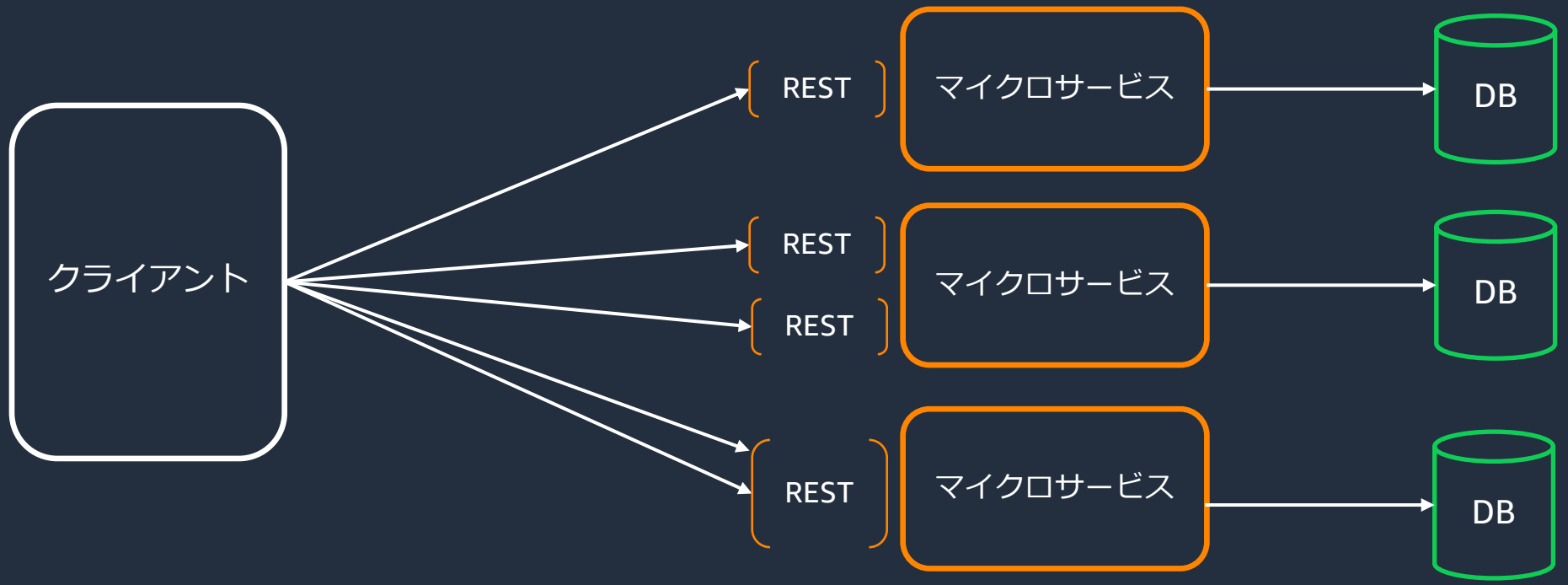


イベント駆動の業務処理連携
Tutorial | イベント駆動アーキテクチャ
FIFOデモ（順序保証、重複排除）
テンプレートから始める → こちら

アプリケーションフロー処理
Tutorial（Workflow / エラー処理）
短時間・高速フロー処理向け Express
[New] ローコードの処理フロー Tutorial

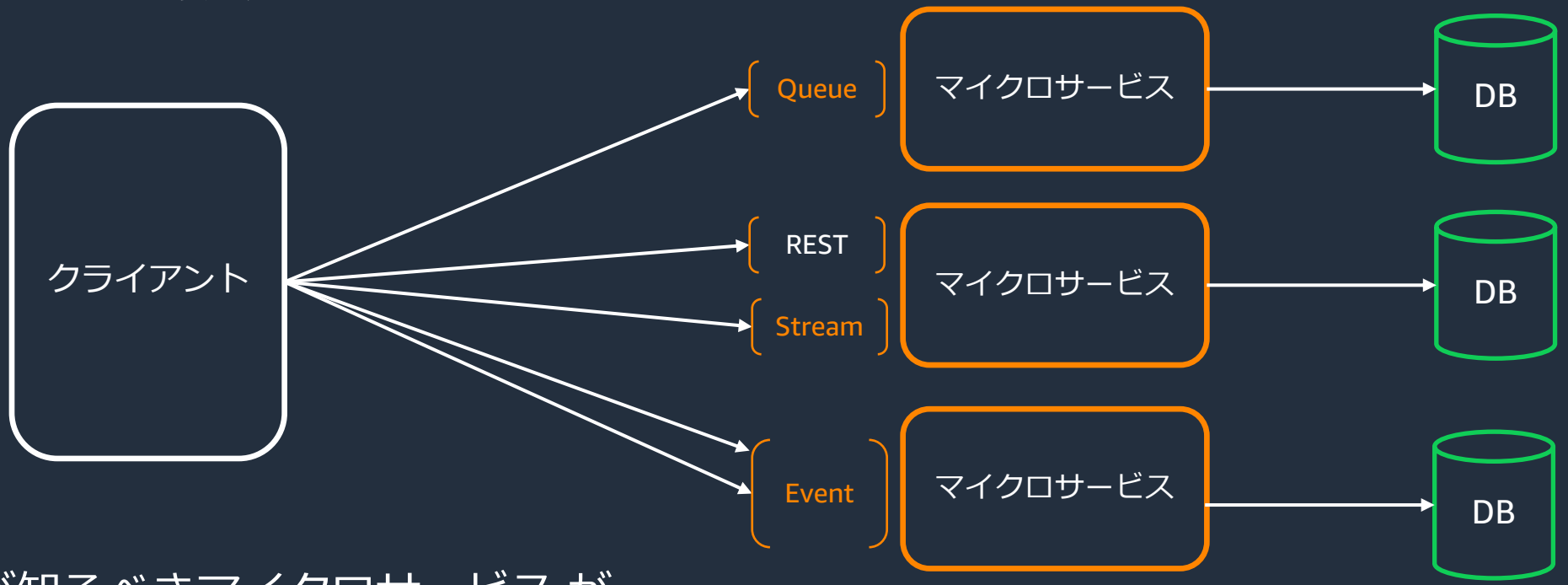
https://aws.amazon.com/jp/serverless/patterns/serverless-pattern/

よくある実装



よくある実装

クライアントは、REST や Queue、Streamで
マイクロサービス と直接会話すると

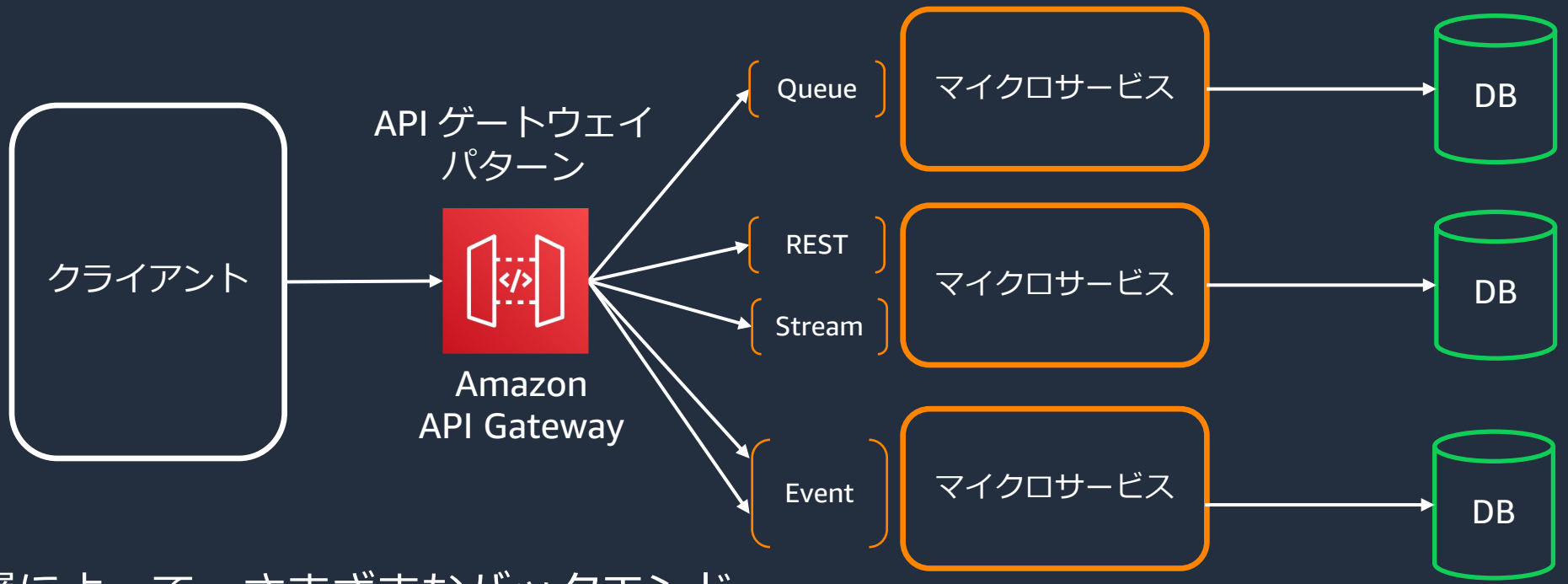


クライアントが知るべきマイクロサービス が
どんどん増えて行く

課題

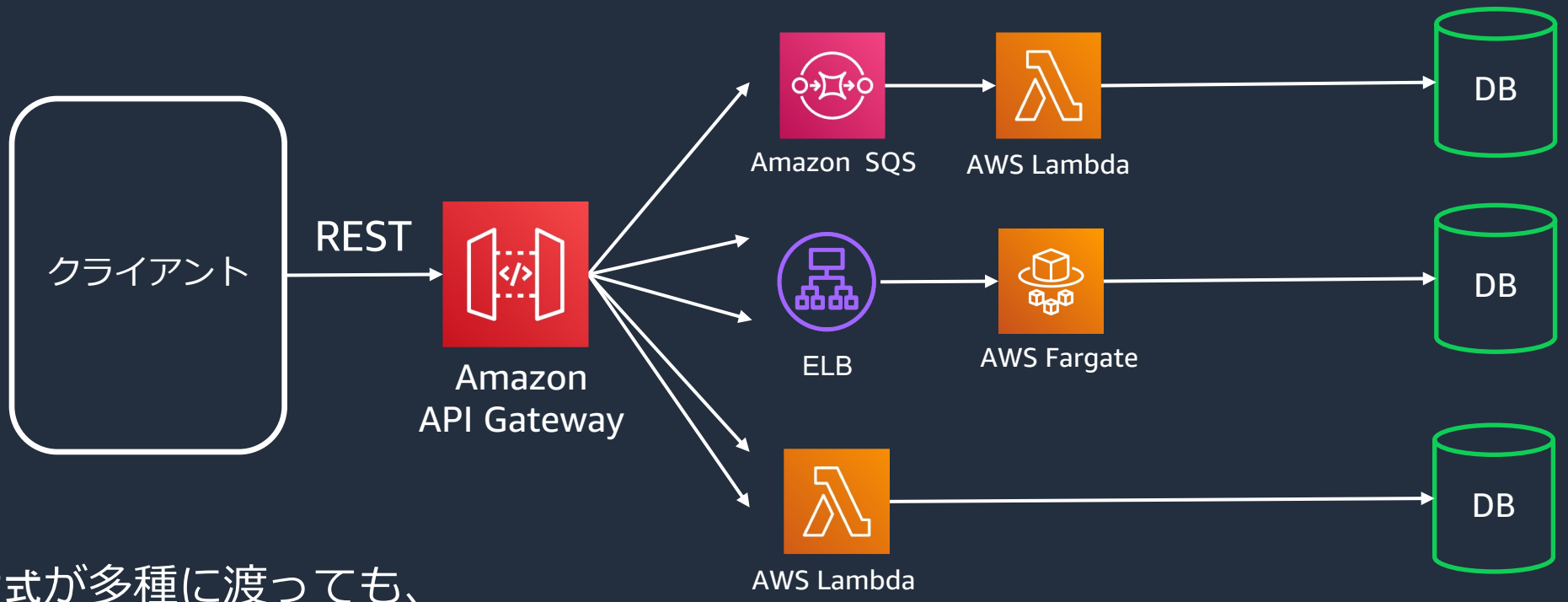
- マイクロサービス 分割には成功したが、サービス が増えて行くにつれてクライアントがサービス エンドポイントに直接アクセスするのがつらい
  - 連携先への接続方式が千差万別
  - クライアント側がバックエンドの能力を個々に把握する必要がある
    - 可用性、回復性、スループットなど
  - サービス 側の REST 契約が破棄されると直接クライアント影響が出る

API ゲートウェイパターンを導入



ゲートウェイ層によって、さまざまなバックエンド
サービスの呼び出しを統一された APIで統合

API ゲートウェイパターンを導入



連携先の接続方式が多種に渡っても、
クライアントからは REST でアクセス可能とする

課題の改善策

- マイクロサービス 分割には成功したが、サービス が増えて行くにつれてクライアントがサービス エンドポイントに直接アクセスするのがつらい
  - 連携先の接続方式が千差万別
    - API Gateway は多種多様なサービスと統合
  - クライアント側がバックエンドの能力を個々に把握する必要がある
    - 連携先の能力に応じて、API Gateway のスループットを調整できる
  - サービス 側の REST 契約が破棄されると直接クライアント影響が出る
    - API Gateway 層で（ある程度）影響を吸収できる(e.g. template利用)

モノリスとの付き合い方

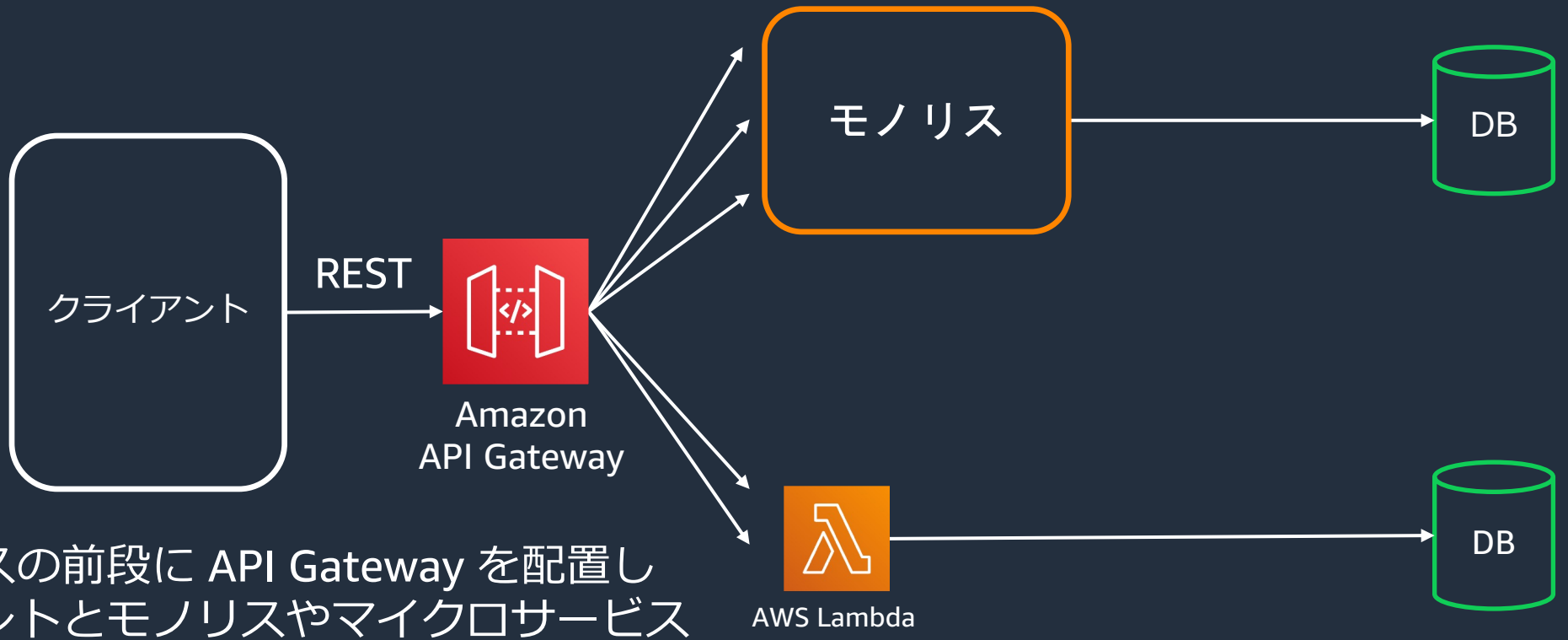
モノリス

既存のモノリスをどのようにモダンにしていくか？

課題

- 既存資産（モノリス）があり、全てをマイクロサービス に置き換えることはできない
  - モノリスとマイクロサービス を共存させたい

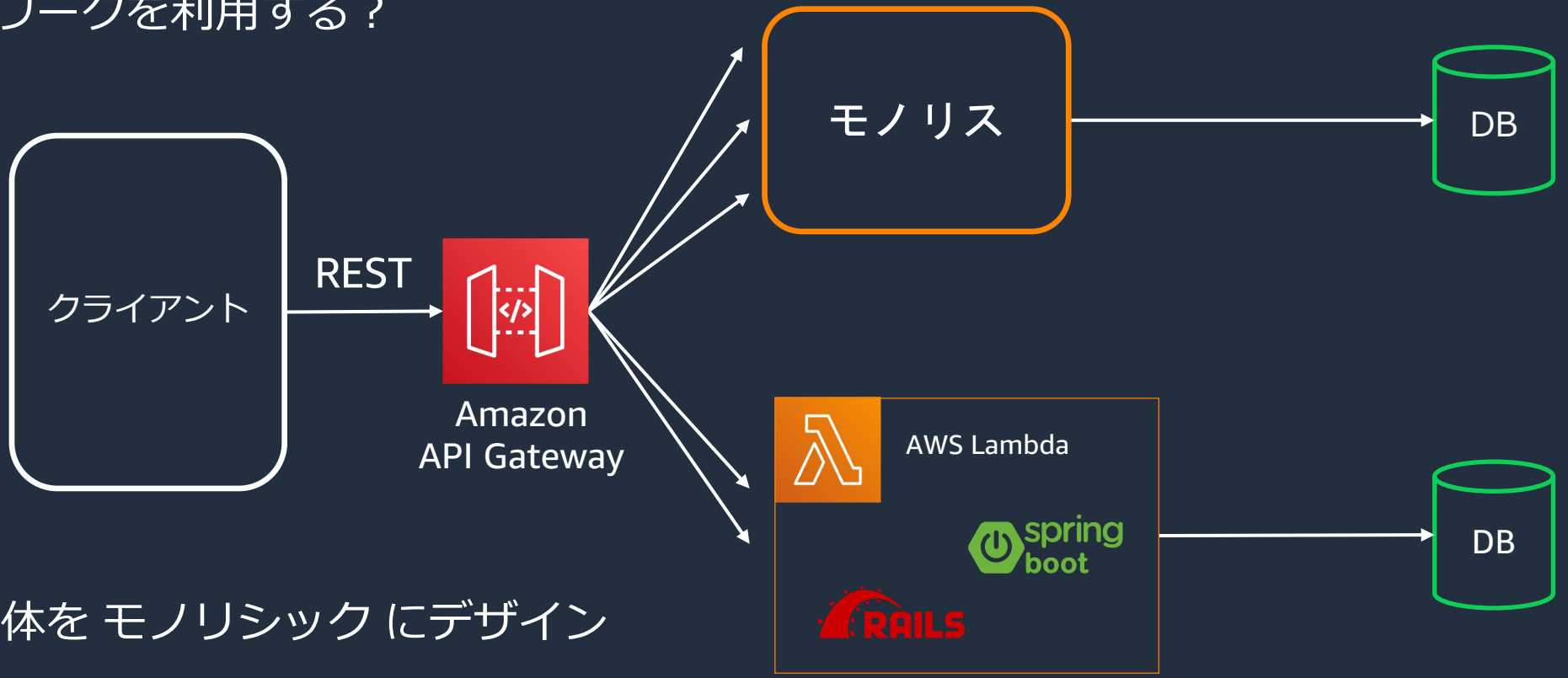
モノリスを分割



既存のモノリスの前段に API Gateway を配置して、クライアントとモノリスやマイクロサービスとの繋がりを疎結合にすることができる

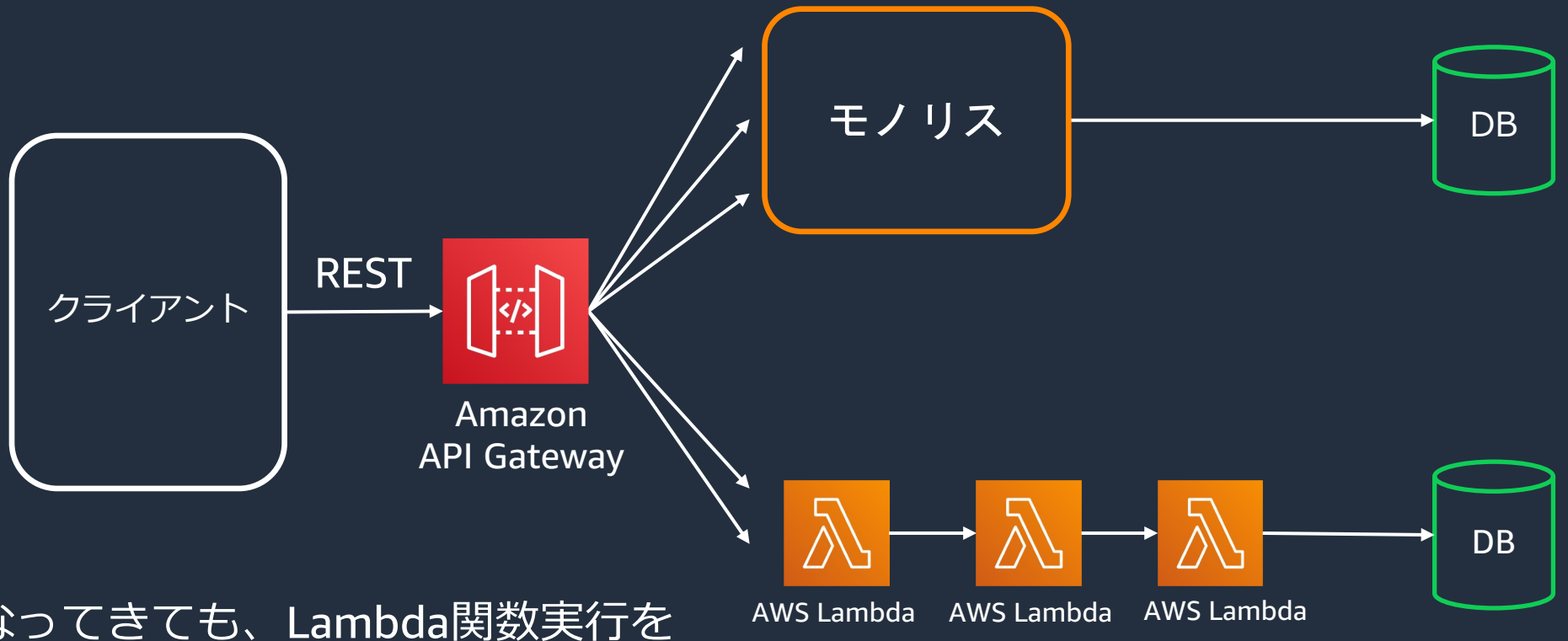
モノリスを分割

Lambda 関数の実装が複雑になってきたら
WEB フレームワークを利用する？



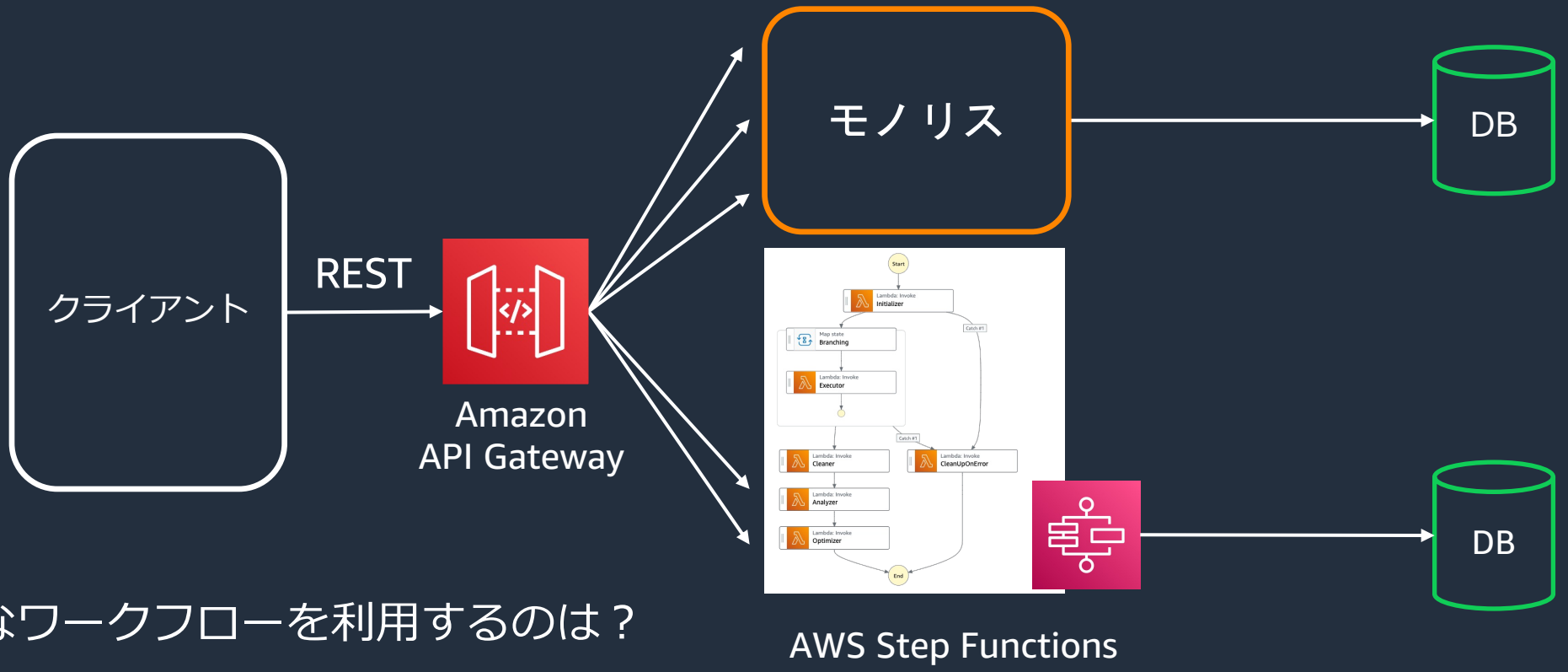
Lambda関数自体を モノリシック にデザイン
するのも、、、

モノリスを分割



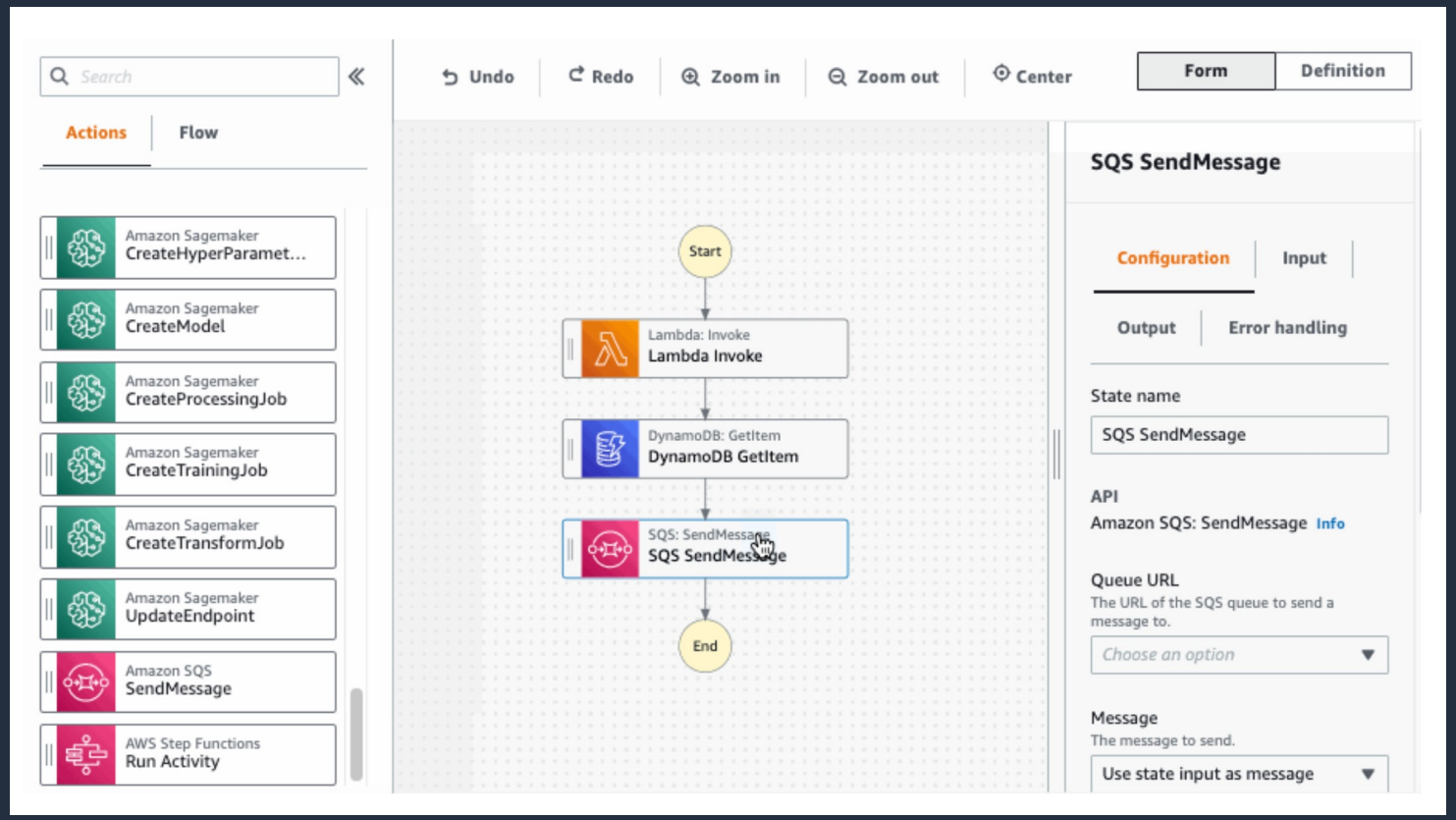
処理が複雑になってきても、Lambda関数実行を
直列にするのはバッドプラクティス

モノリスを分割



サーバーレスなワークフローを利用するのは？

サーバーレス ワークフロー / AWS Step Functions



Step Functions Workflow Studio によるワークフローの構築

※ Express Workflow では、実行開始レートは毎秒 100,000 以上 をサポート

課題の改善策

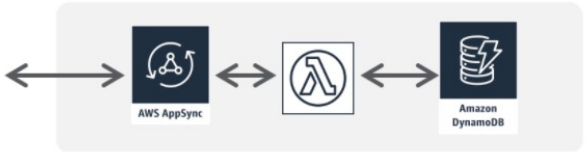
- 既存資産（モノリス）があり、全てをマイクロサービス に置き換えることはできない
  - モノリスとマイクロサービス を共存させたい
    - API Gateway でバックエンドを抽象化する
    - クライアントからはモノリスを意識させない

サーバーレスな ファイル変換処理

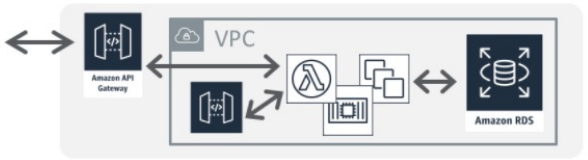
形で考えるサーバーレス設計

代表的な適用シーン/ユースケースと実装形

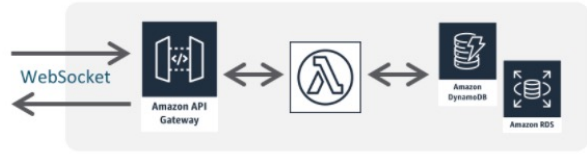
動的 Web / モバイルバックエンド
関連資料 | Tutorial | Tutorial（中級編）
テンプレートから始める → こちら



リアルタイムモバイル / オフライン対応
AWS マンガ | ワークショップ
Solution リンク

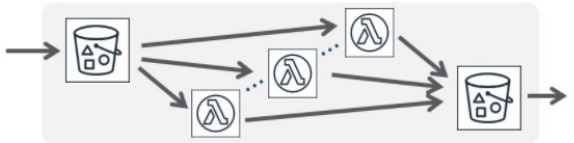


業務系 API / グループ企業間 API
Tutorials | Private API 記事 | 関連事例
OpenAPI の利用

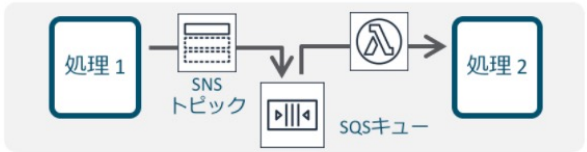


Push 配信系・インタラクティブ API
関連リンク | AppRepositoryサンプル
解説動画（英語）

画像処理 / シンプルなデータ加工
Tutorial | 関連事例 | Solution リンク
[New] S3 Object Lambda
テンプレートから始める → こちら



分散並列処理（like MapReduce）
関連事例1 | 関連事例2
RefArch

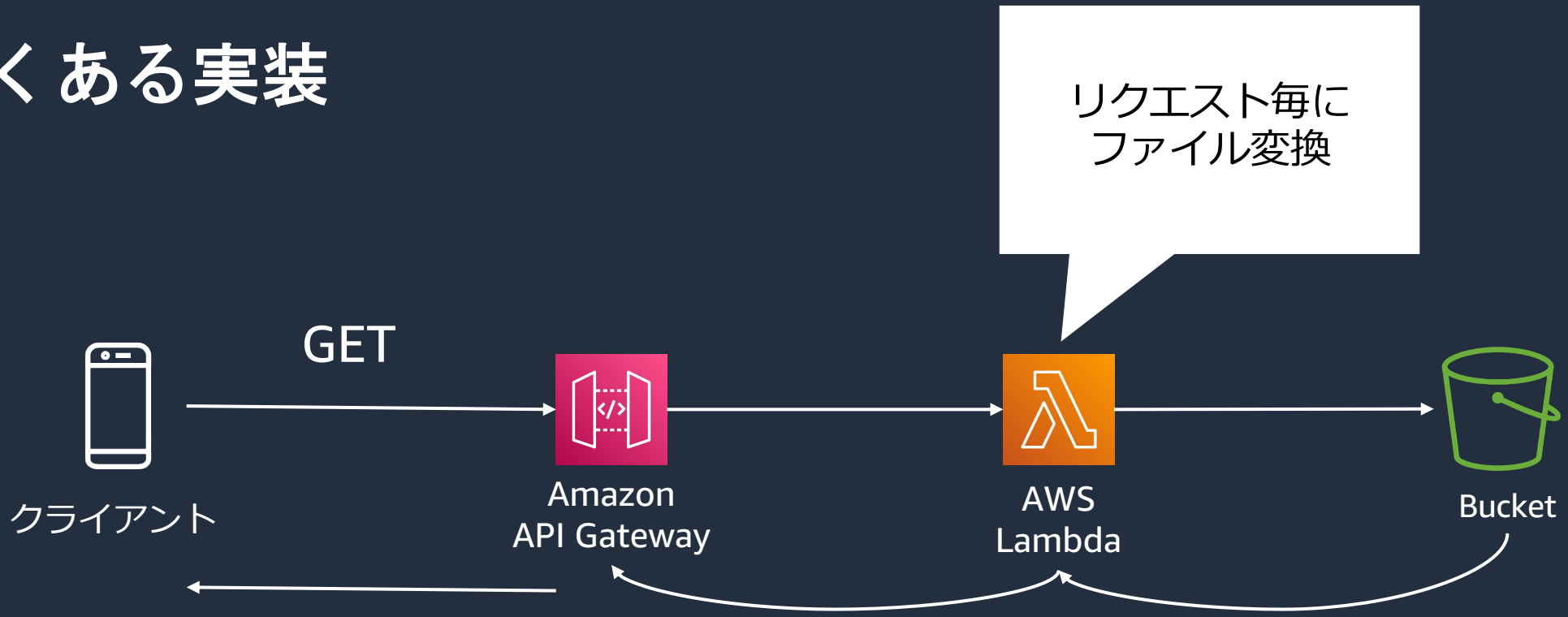


イベント駆動の業務処理連携
Tutorial | イベント駆動アーキテクチャ
FIFOデモ（順序保証、重複排除）
テンプレートから始める → こちら

アプリケーションフロー処理
Tutorial（Workflow / エラー処理）
短時間・高速フロー処理向け Express
[New] ローコードの処理フロー Tutorial

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

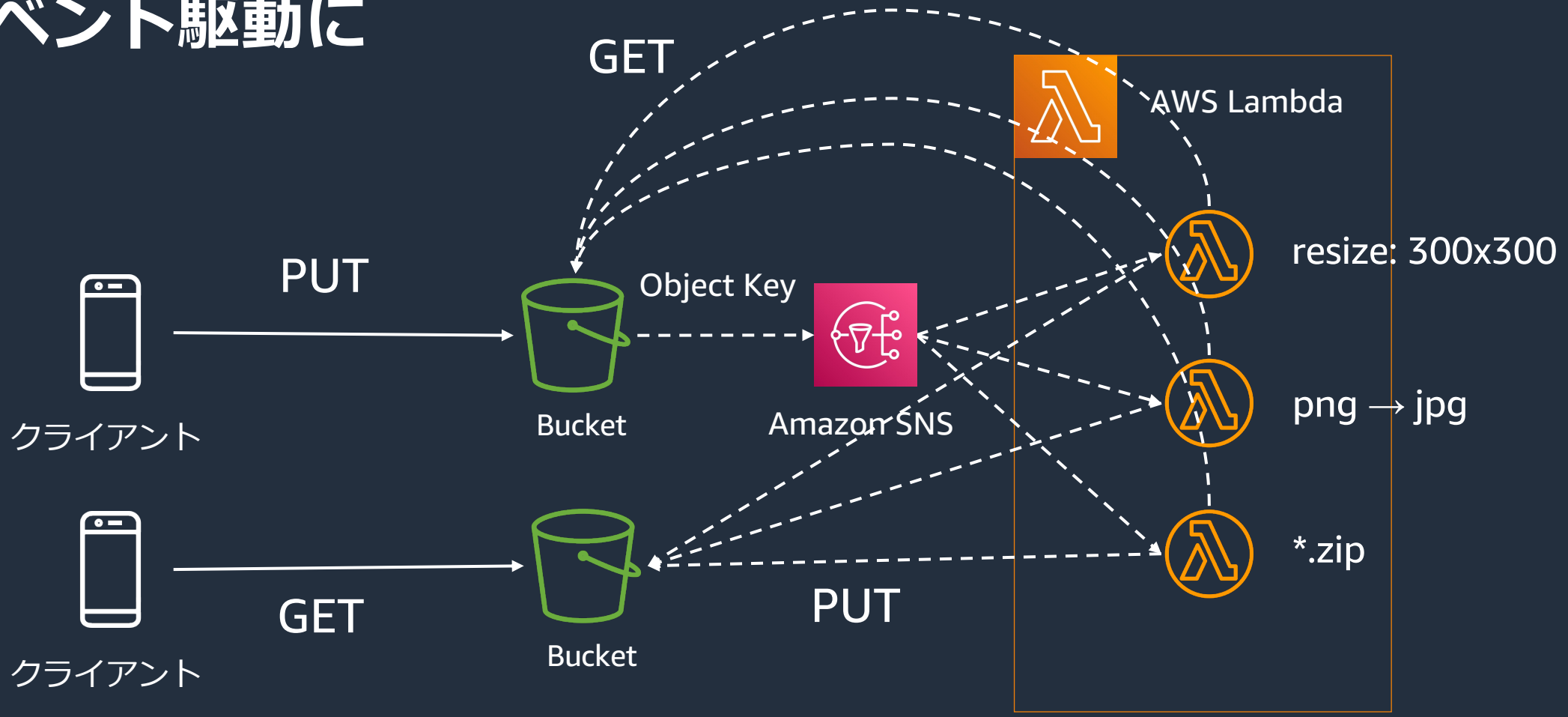
よくある実装



課題

- リクエストごとに処理結果を返すと処理時間や処理回数が増える
- 多数の変換バリエーションがあり、今後増える可能性もある

イベント駆動に



課題の改善策

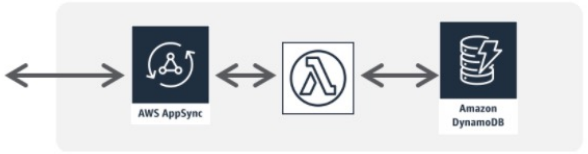
- リクエストごとに処理結果を返すと処理時間や処理回数が増える
  - ファイルが生成されたタイミングで処理する
- 多数の変換バリエーションがあり、今後増える可能性もある
  - 多数のバリエーションは ファン・アウト で処理する

サーバーレスな マルチテナント管理

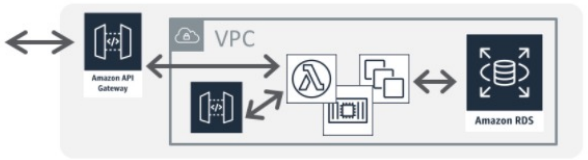
形で考えるサーバーレス設計

代表的な適用シーン/ユースケースと実装形

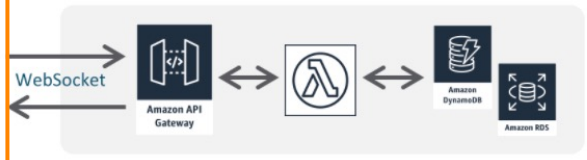
動的 Web / モバイルバックエンド
関連資料 | Tutorial | Tutorial（中級編）
テンプレートから始める → こちら



リアルタイムモバイル / オフライン対応
AWS マンガ | ワークショップ
Solution リンク

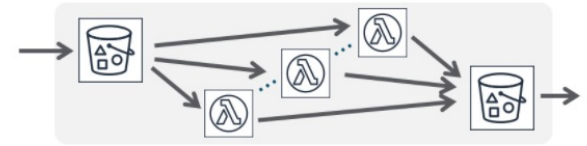


業務系 API / グループ企業間 API
Tutorials | Private API 記事 | 関連事例
OpenAPI の利用

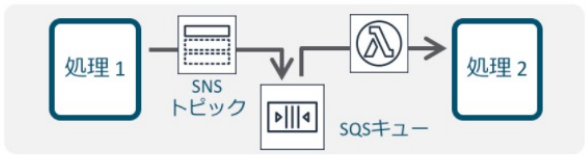


Push 配信系・インタラクティブ API
関連リンク | AppRepositoryサンプル
解説動画（英語）

画像処理 / シンプルなデータ加工
Tutorial | 関連事例 | Solution リンク
[New] S3 Object Lambda
テンプレートから始める → こちら



分散並列処理（like MapReduce）
関連事例1 | 関連事例2
RefArch



イベント駆動の業務処理連携
Tutorial | イベント駆動アーキテクチャ
FIFOデモ（順序保証、重複排除）
テンプレートから始める → こちら

アプリケーションフロー処理
Tutorial（Workflow / エラー処理）
短時間・高速フロー処理向け Express
[New] ローコードの処理フロー Tutorial

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

課題

- あるテナントが別のテナントの UX に影響を与えないようにしたい
- 様々なテナントに、異なるレベルのパフォーマンスを提供したい
- インバウンドのリクエストレートを適切に制限したい

ソリューション となる二つのパーツ

Throttling

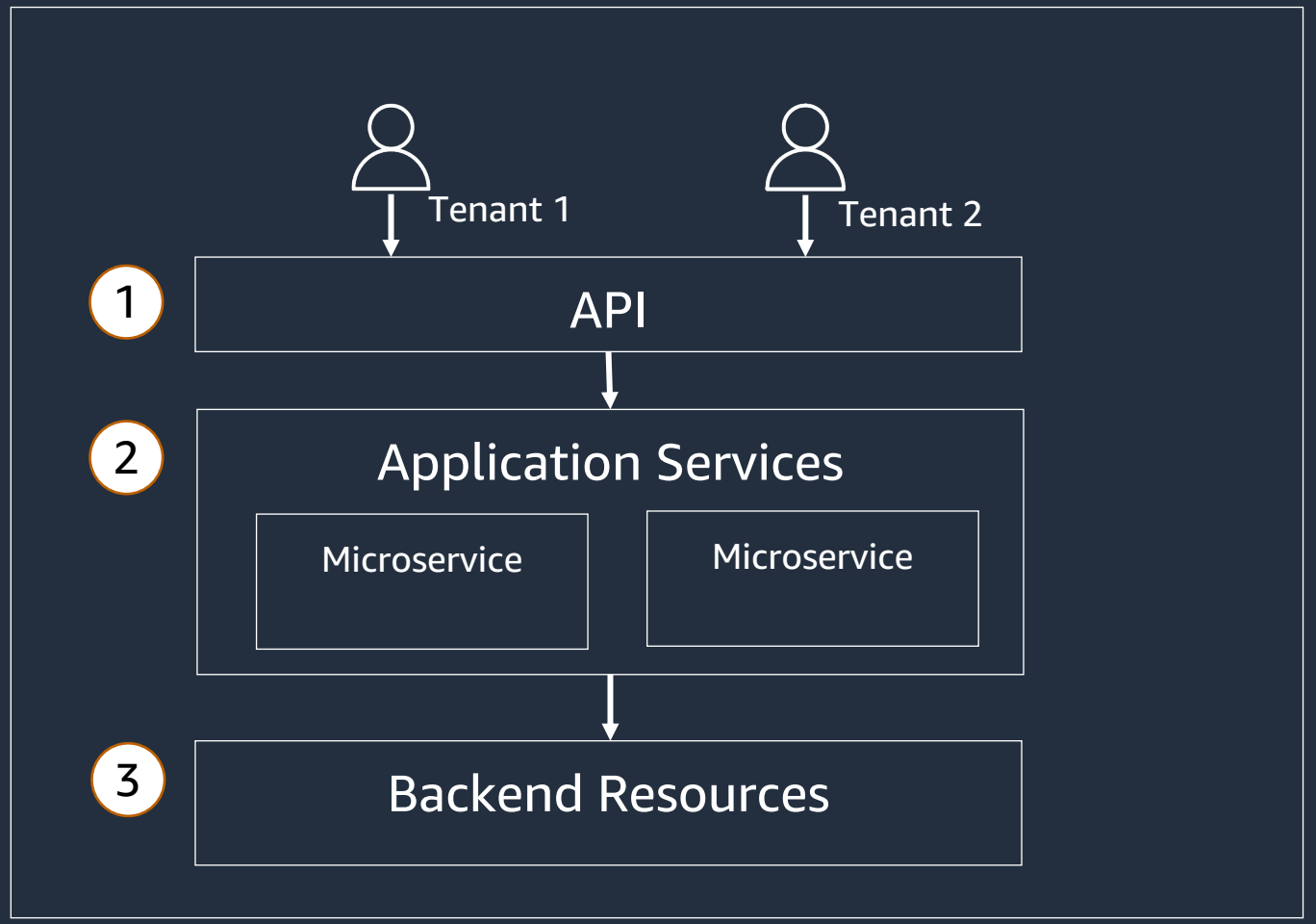
- 処理するリクエストの最大数/秒を制限
- リクエストスパイクから バックエンドを保護

Quota

- 時間あたりの API の全体的な使用量を制限
- テナントによるシステムの過剰使用から サービス を保護

Throttling をどこで発生させるか？

1. 「フロントドア」でスロットル
2. マイクロサービスでのスロットル
3. リソースへのアクセスを制限

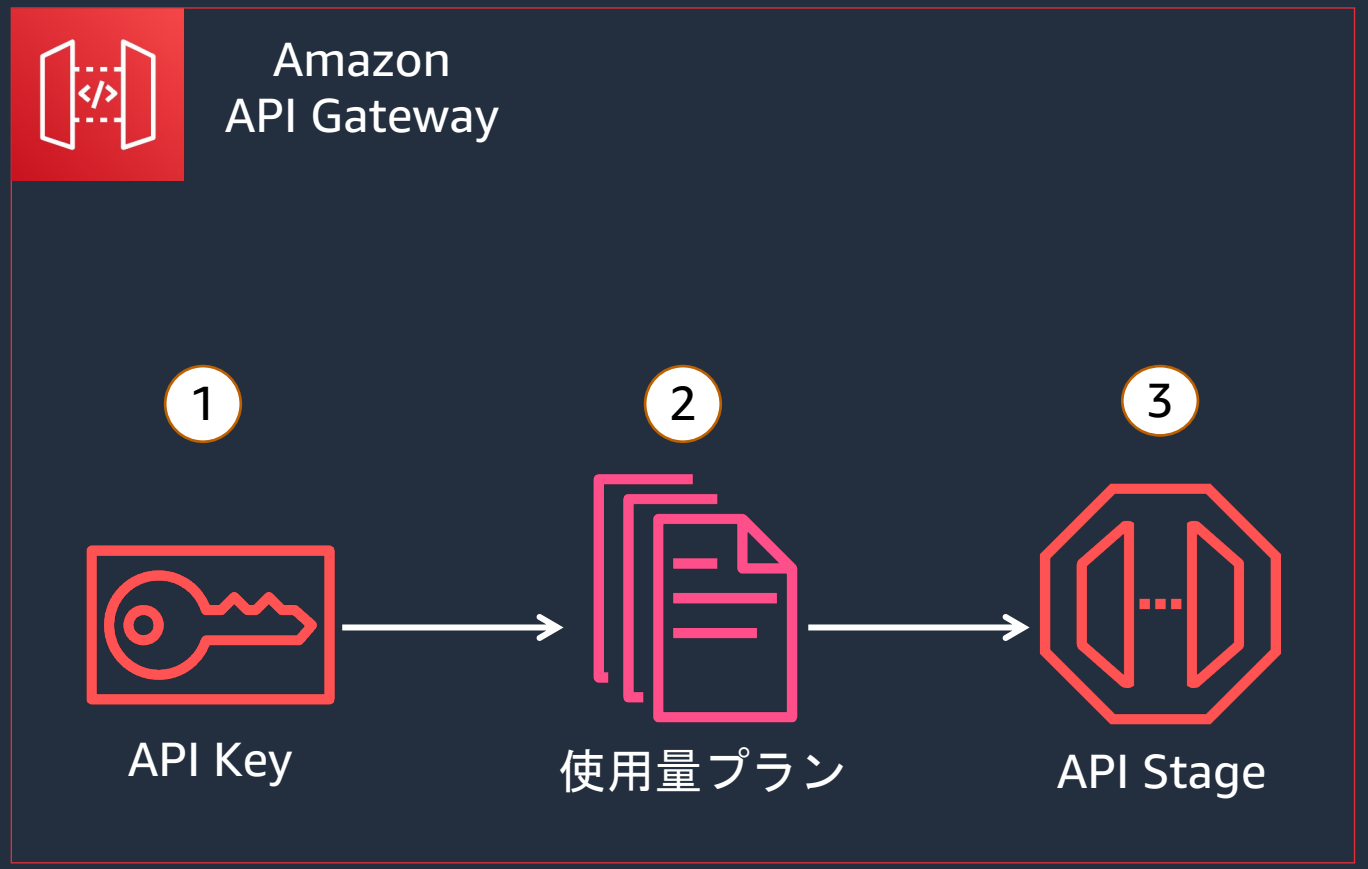


Amazon API Gateway の場合どうするか

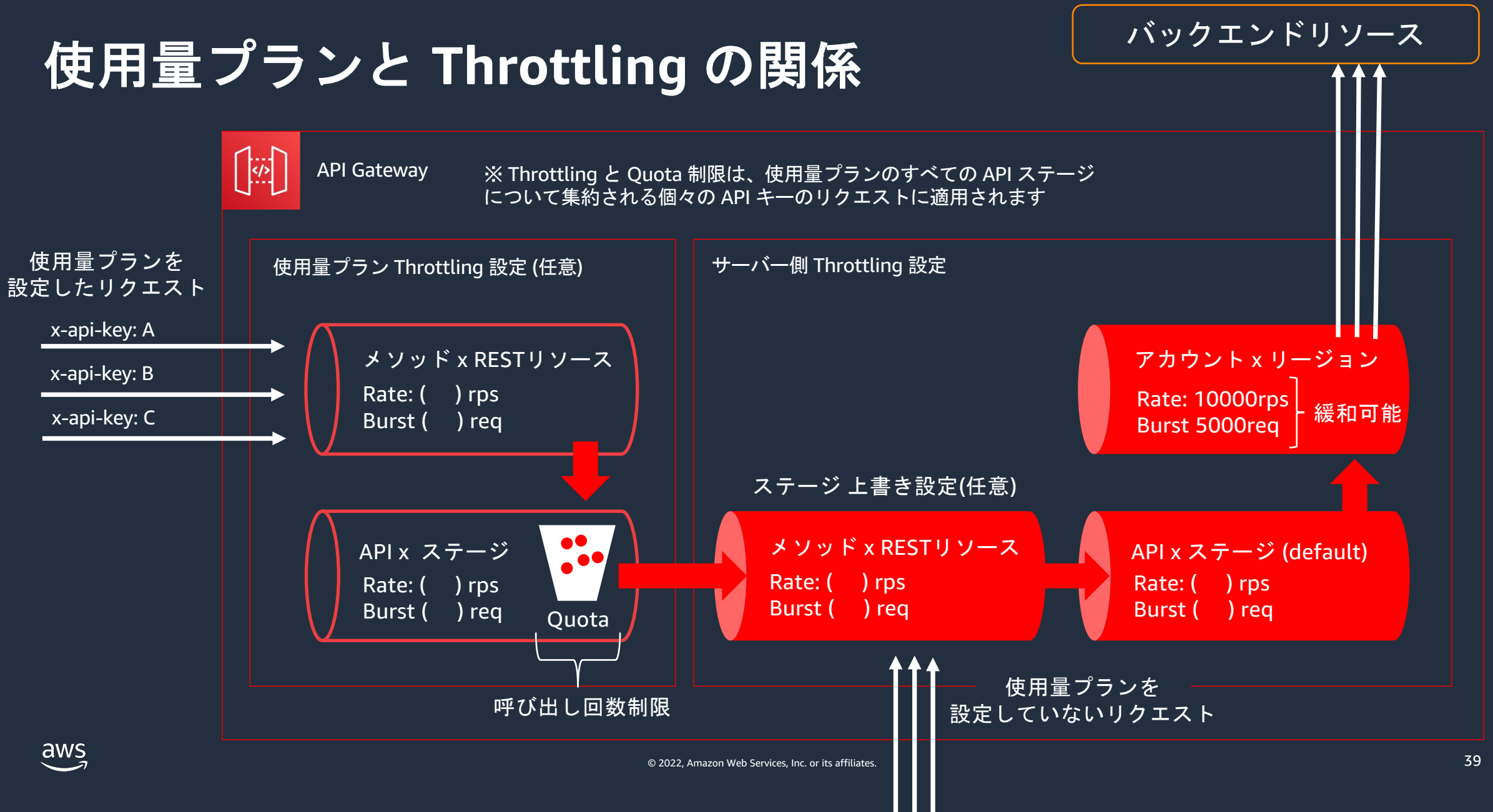
1. API キーは、クライアント リクエストを識別するための一意のキー

2. 使用量プランは、デプロイされた API ステージ/メソッドに誰がアクセスできるか、およびアクセスできる量と速度を指定可能

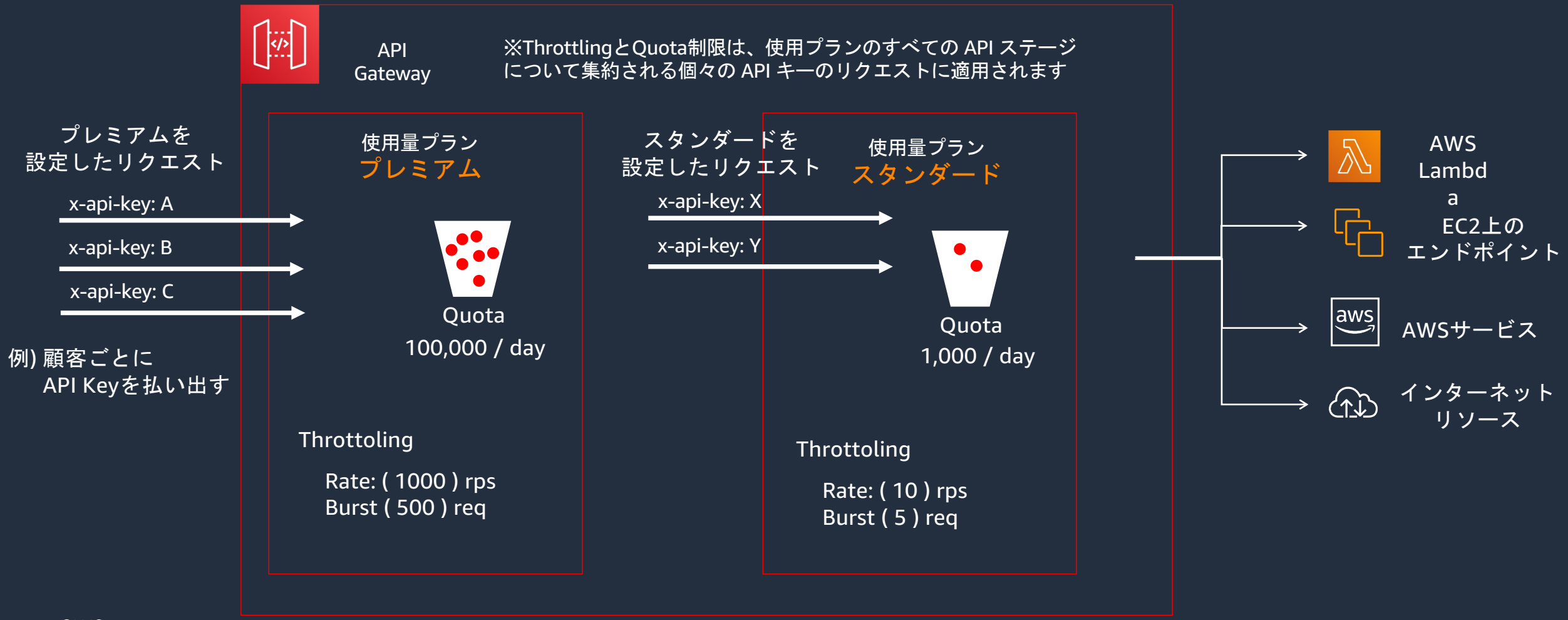
3. API ステージは、デプロイされた API の独立した環境



使用量プランと Throttling の関係



使用量プランと API Key の関係



課金モデル (料金プラン課金)

- 初期利用は無料でサービス提供、サービス認知向上やサービスの機能向上を行う（フィードバックを得る）
- さらなる機能利用に対しては有料プランに誘導（フリーミアム）

例)

※単一のAPI Keyで複数のAPI販売に利用できます。

課金モデル (Pay as you go = 従量課金)

使用実績の確認 (CLI)

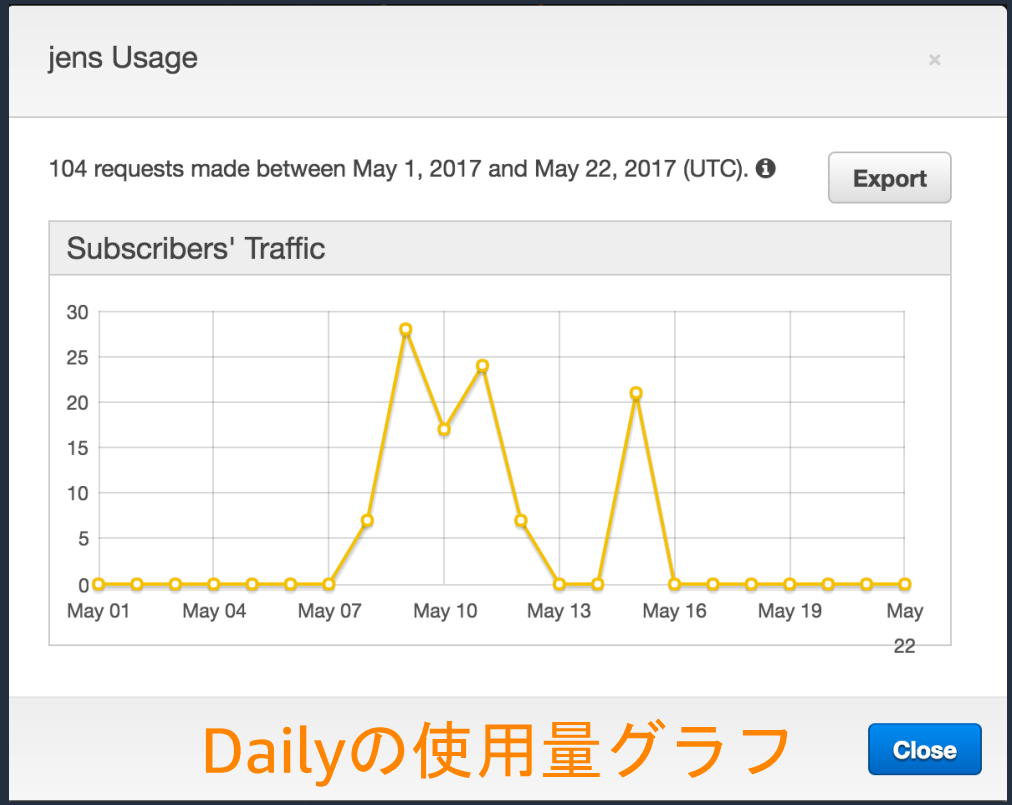
```
aws apigateway get-usage ¥
  --usage-plan-id 0qrdhs ¥
  --key-id tn3gh64l3b ¥
  --start-date "2022-10-01" ¥
  --end-date "2022-10-31" ¥
  --no-paginate
{
    "usagePlanId": "0qrdhs",
    "startDate": "2022-10-01",
    "endDate": "2022-10-31",
    "items": {
        "tn3gh64l3b": [
            [0, 100],
            [0, 100],
             (snipped)
            [10, 90],
            [80, 20],
            [80, 20]
        ]
    }
}
```

API Key単位

[Dailyの使用量, Dailyの残量]

※直近の最長90日までの実績が取得可能

使用実績の確認 (Management Console)

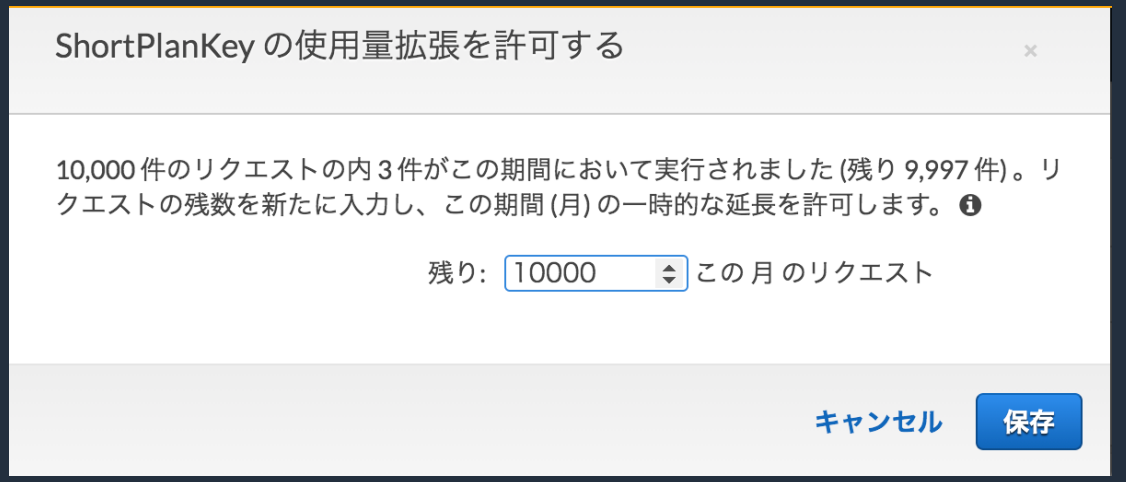


API KeyごとにQuota拡張できる

例)

月単位で Quota を設定し払い出した API Key を利用している顧客が、当月の使用量を使い果たした場合など

- ビジネスによっては追加費用を払えば、月内の利用 Quota を確保するオプションを設置することもできる



AWS Lambda の無料利用枠

フリーミアムを
設計しやすい

AWS Lambda の無料利用枠には、1 か月あたり 100 万件の無料リクエストと、1 か月あたり 40 万 GB-s のコンピューティングタイムが含まれており、x86 および Graviton2 プロセッサの両方を搭載した機能を利用できます。Lambdaはまた、毎月の使用量が一定の基準値を超えたオンデマンド期間に対して、段階的な価格オプションを提供しています。AWS Lambda は Compute Savings Plans でカバーされます。これは、1 年または 3 年の期間での一貫したコンピューティング使用量 (USD/時間で測定) を契約するかわりに、 Amazon Elastic Compute Cloud (Amazon EC2)、AWS Fargate、および Lambda を低価格でご利用いただけるようになる柔軟な料金モデルです。Compute Savings Plans を使うと、AWS Lambda で最大 17% の節約が可能になります。節約効果は使用時間と Provisioned Concurrency に適用されます。 詳細はこちら »

課題の改善策

- あるテナントが別のテナントの UX に影響を与えないようにしたい
  - API Key を分離することにより、テナント間の独立レベルをコントロール
- 様々なテナントに、様々なレベルのパフォーマンスを提供したい
  - 使用量プランを使い、プランごとに異なるパフォーマンス量を定義
- インバウンドのリクエストレートを適切に制限したい
  - API Key を含むリクエスト以外を拒否できる

まとめ

- AWS Lambda や Amazon API Gateway といったサーバーレスサービスを使うことで、ユーザーはインフラストラクチャー管理のことを意識することなく、ビジネスに直結したロジックの開発に集中することが可能に
- サーバーレスならではのアーキテクチャデザインを考え、アーキテクチャのパターンを組み合わせることで、より現実的なサービス構築が可能に

Happy Coding !

Thank you!

Kensuke Shimokawa

_kensh

Slides https://speakerdeck.com/_kensh/
Qiita https://qiita.com/_kensh