

stripe

AWS Dev Day 2022 Japan

EventBridgeとAmplify + Stripeで 「イベント駆動」な Webサービスを開発する

@hidetaka_dev

Nov 2022

今日のトピック

- AWS AmplifyとStripeを利用して、
SaaSやECのフロントエンド開発をスムーズにできる
- 決済やサブスクリプションなど、
顧客・契約の状態と連動したWFも、Amazon EventBridgeでスムーズに
- AWS AppSync・AWS Step Functionsを活用して、
決済・サブスクに関する運用効率化(決済OPS)をはじめよう

岡本 秀高 (@hidetaka_dev)

- **Stripe Developer Advocate**
(ex-developer in Digitalcube)
- **JavaScript / TypeScript developer**
 - **AWS Lambda / CDK**
 - **Next.js / React**
 - **WordPress / Alexa / etc**
- **AWS Samurai 2017 /**
AWS Community Day APAC 2017
- QiitaでStripeに関する
Dev blogを週2/3本で更新中
 - <https://qiita.com/hideokamoto>
 - **年間120記事ペース**



今日のトピック

- AWS AmplifyとStripeを利用して、SaaSやECのフロントエンド開発をスムーズにできる
- 決済やサブスクリプションなど、顧客・契約の状態と連動したWFも、Amazon EventBridgeでスムーズに
- AWS AppSync・AWS Step Functionsを活用して、決済・サブスクに関する運用効率化(決済OPS)をはじめよう

SaaSやECなどのサービス開発を、Amplifyで効率化

- サービスの**GUI**提供に必要な機能が、**AWS Amplify**には揃っている
 - 認証・認可
 - RESTまたはGraphQL API
 - ホスティング・SSL
 - CLIやSDK, UI Component, Figmaなどのツール
 - etc..
- サービスの「**コア機能以外**」を、**AWS Amplify**と関連ツールで効率的に実装できる

ビジネスに欠かせない「課金機能」をAPI as a Serviceで

- Stripeなど、決済やサブスクリプションに特化したSaaS
 - PCI DSS準拠
 - サービスの停止率が低い(99.999% uptime for the last 90 days)
- サブスクリプションの契約プランや割引、請求など「お金に関わる機能要件」の構築・保守コストを、SaaSで抑える
- AWSのAPIを呼び出すように、StripeのAPIをアプリ・システムから利用して課金・請求管理機能を実装

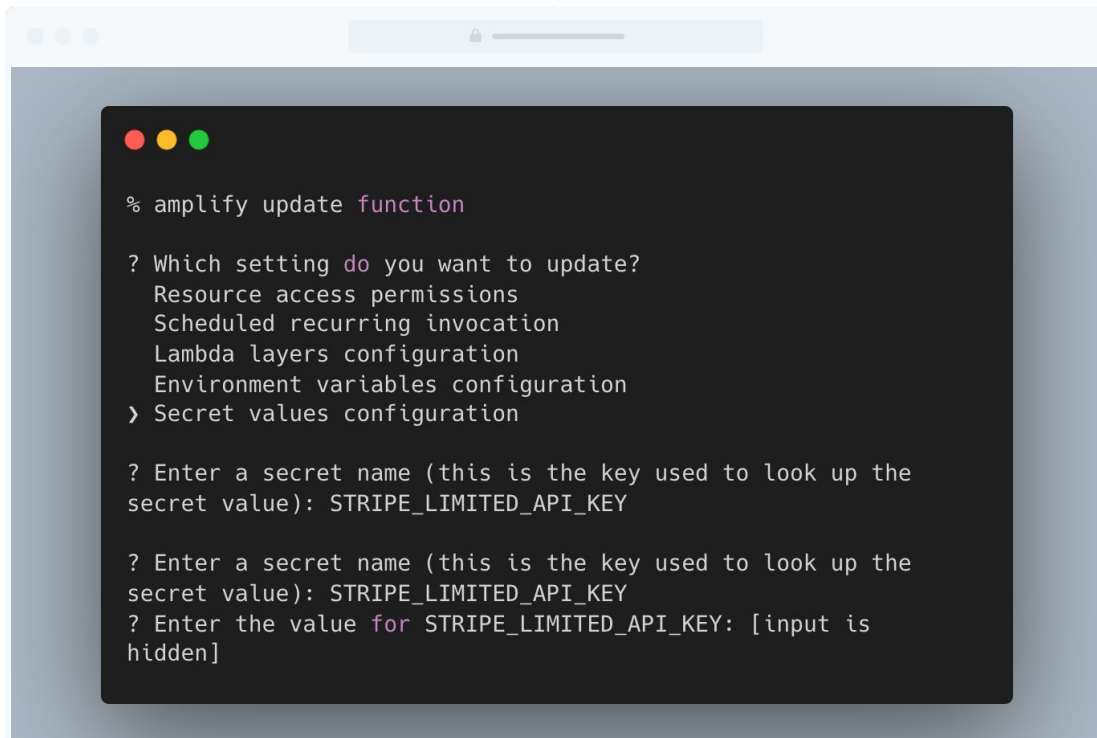
Amplify CLIで、安全にAPIキーを管理

StripeのAPIキー3種

- Publishable API Key
- Secret API Key
- Limited API Key

Secret / Limited API keyは、
Secrets Managerで
安全に管理する

Amplify CLIを利用して、
APIキーの保管と利用を
手軽に実現する



```
% amplify update function

? Which setting do you want to update?
  Resource access permissions
  Scheduled recurring invocation
  Lambda layers configuration
  Environment variables configuration
  > Secret values configuration

? Enter a secret name (this is the key used to look up the
secret value): STRIPE_LIMITED_API_KEY

? Enter a secret name (this is the key used to look up the
secret value): STRIPE_LIMITED_API_KEY
? Enter the value for STRIPE_LIMITED_API_KEY: [input is
hidden]
```

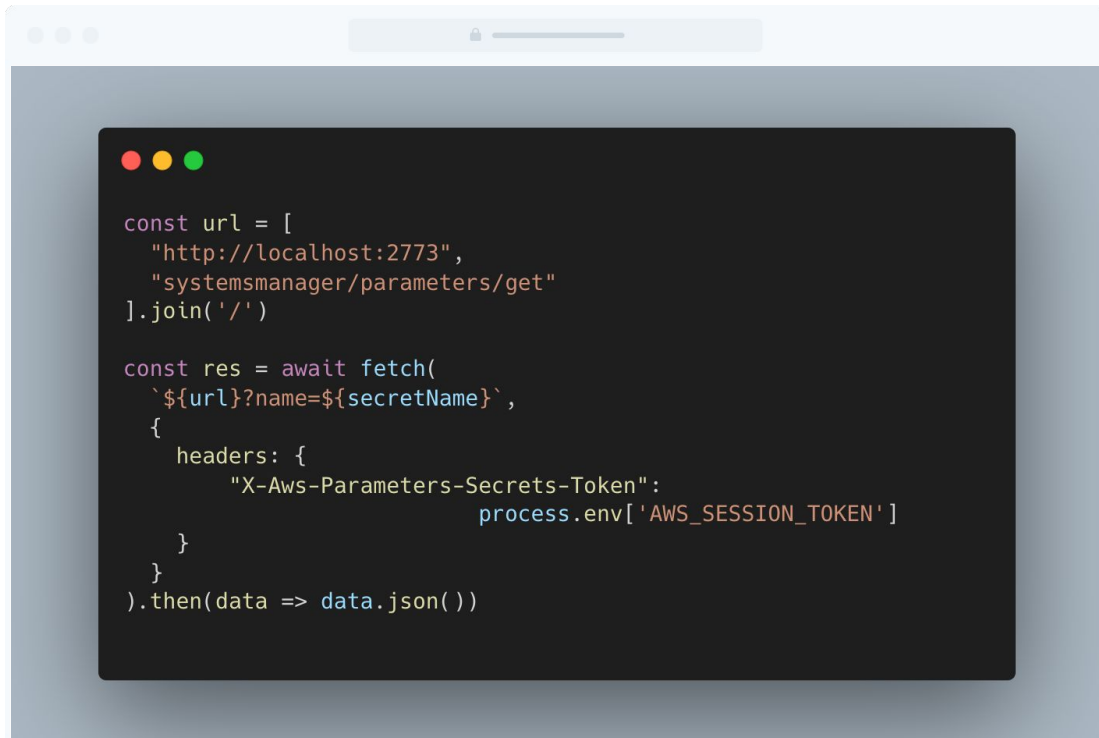
Tips: Secretの取得はLambda Layerで

layer:AWS-Parameters-and-Secrets-Lambda-Extensionが登場 (2022/10)

LayerをLambdaに登録すると、HTTPリクエストでSecretが取得できる

環境変数で、キャッシュのTTLやタイムアウトなども設定可能

https://docs.aws.amazon.com/ja_jp/secretsmanager/latest/userguide/retrieving-secrets_lambda.html#retrieving-secrets_lambda_env-var



```
const url = [
  "http://localhost:2773",
  "systemsmanager/parameters/get"
].join('/')

const res = await fetch(
  `${url}?name=${secretName}`,
  {
    headers: {
      "X-Aws-Parameters-Secrets-Token":
        process.env['AWS_SESSION_TOKEN']
    }
  }
).then(data => data.json())
```


Stripeの顧客情報と、Amplifyのユーザー情報の紐付け

Stripeは、同じメールアドレスで複数のCustomerデータが作れる

Stripe Customer IDとCognito User PoolsのIDをマッピングして重複を回避

手早い方法は、Metadataでのマッピング

検索性を求める場合、DynamoDBでテーブルを用意

stripe

The screenshot shows the AWS Amplify console interface for a user. At the top, there's a section for 'ユーザー属性 (4) 情報' (User Attributes (4) Information) with an '編集' (Edit) button. Below this is a search bar with the placeholder text 'プロパティまたは値でフィルタリング' (Filter by property or value). The main part of the screenshot is a table listing user attributes:

属性名	値	タイプ
custom:stripeCustomerId	cus_K7zqpFI2KRVA5h	オプション
email	[REDACTED]	必須
preferred_username	dev_hide_11	オプション
sub	691363fb-a3cc-436f-a208-7f426123567d	必須

Below the table, there's a section for 'グループメンバーシップ (0) 情報' (Group Membership (0) Information) with a description: 'このユーザーのグループメンバーシップを表示および編集します。' (Display and edit the group membership of this user). At the bottom, there are two buttons: 'ユーザーをグループから削除' (Remove user from group) and 'ユーザーをグループに追加' (Add user to group).

IDの同期は、Stripe -> CognitoでSyncしよう

Cognitoには、
情報更新時のWebhookがない

StripeのWebhookで、
顧客データ更新を受け付ける

Customer metadataから
Cognitoのuser idを取得、
AWS SDKで更新処理

DynamoDBを使う場合は、
DynamoDB StreamでもOK

stripe

```
17
18 // This is your Stripe CLI webhook secret for testing your endpoint locally.
19 const endpointSecret = "whsec_c287ea07024c1f49210362efd3e980c770cd18467bf05ad3a9c531
20
21 app.post('/webhook', express.raw({type: 'application/json'}), (request, response) =>
22   const sig = request.headers['stripe-signature'];
23
24   let event;
25
26   try {
27     event = stripe.webhooks.constructEvent(request.body, sig, endpointSecret);
28   } catch (err) {
29     response.status(400).send(`Webhook Error: ${err.message}`);
30     return;
31   }
32
33   // Handle the event
34   console.log(`Unhandled event type ${event.type}`);
35
36   // Return a 200 response to acknowledge receipt of the event
37   response.send();
38 });
```

Tips: ユーザーDBを作るか作らないか

- **いずれ必要になる可能性が高い**
 - Metadataでの紐付けでは難しいことへの対応
 - 1ユーザーに、**複数のCustomerやSubscription**を紐づけ
 - 親子アカウントやチームなどの**関係性**ができる場合
 - Stripe APIにアクセスする**時間の省略**
 - Stripe Webhookで、**事前にDBへ必要情報をSync**する
 - 顧客からの取得(GET)系を**AWS内で完結**させる
- 実行したいクエリ・ユースケースに応じて、利用するDBを選択

Tips: UIの表示系を、契約に応じて制御する

- 3つの方法
 - Stripe APIから都度取得する
 - 実装の手間は少ない
 - HTTPSリクエストが出るので、少し時間がかかる
 - Stripe Webhook経由で、**CognitoのUser Metadata**にもたせる
 - Amplify SDKでクライアント側からデータが取れる
 - リスト系データの保存が難しい
 - Stripe Webhook経由で、**ユーザーDB**にもたせる
 - 柔軟性はもっとも高い

AWS AmplifyとStripeでフロントエンドを効率的に開発

- Stripeを使って、サブスクリプションの契約管理ができる
 - ただし契約系以外のUIはStripeのみで提供できない
- **AWS Amplify**を使えば、サービスのUIから認証系、UI Component、アクセス管理にCI / CDと**SaaSのフロントに必要なものが揃う**
- セキュリティのために、Stripeのサーバー側APIキーは**Secrets Store**へ
- StripeのCustomerとCognito Userの紐付け方法は、アプリ・サービスの状況や要件に応じて決定しよう

今日のトピック

- AWS AmplifyとStripeを利用して、SaaSやECのフロントエンド開発をスムーズにできる
- **決済やサブスクリプションなど、顧客・契約の状態と連動したWFも、Amazon EventBridgeでスムーズに**
- AWS AppSync・AWS Step Functionsを活用して、**決済・サブスクに関する運用効率化(決済OPS)をはじめよう**

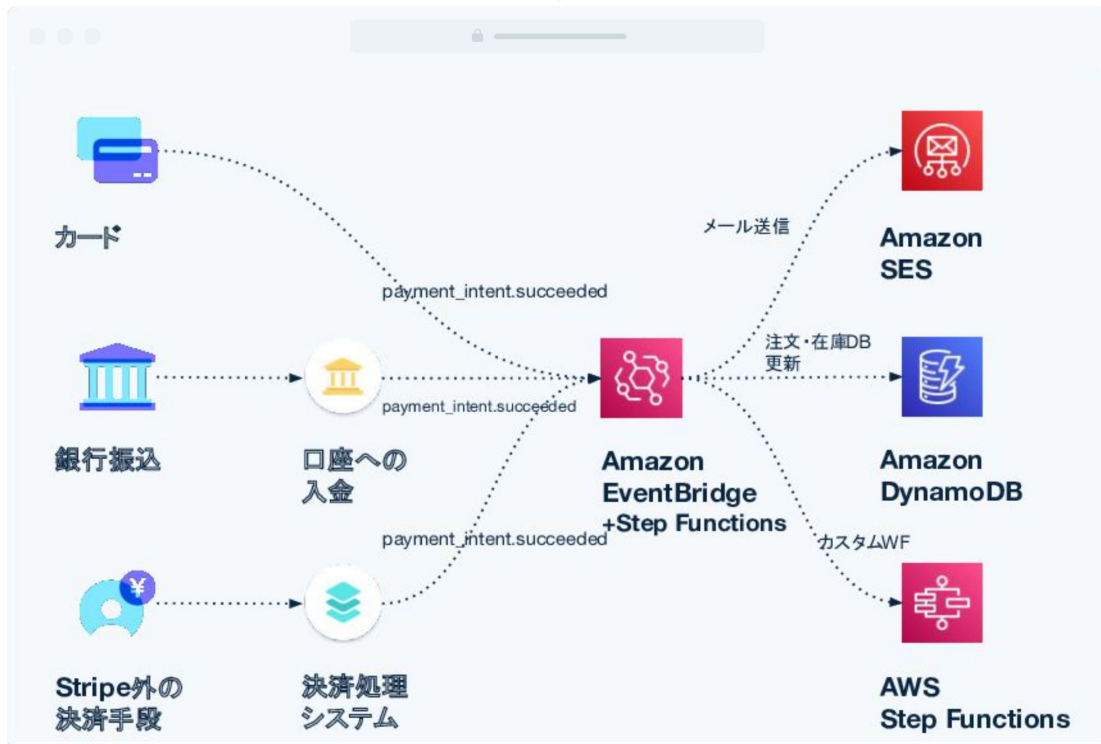
なぜイベント駆動にするのか

同期的に処理できないことに対応するシステムを作るため

- ・銀行振込・コンビニ決済
- ・サブスクの契約更新
- ・未払いの検知・入金確認

顧客やシステムの「状態」を「イベント」としてシステムに通知する仕組み

Stripeでは、150種類以上のWebhookイベントをサポート



システムもサブスクリプションも、「運用」が大切

- サブスクリプションは、「契約し続けてもらう」必要がある
 - 使わないサブスクサービスは、解約される
 - 競合他社への乗り換え:「もっと安いor便利なサービスにしよう」
 - 需要の消失:「子どもが成長したので、もう使わない」
- システムも、「動き続けてもらう」必要がある
 - 環境の変化で、「変更してないコード」も時に動かなくなる
 - SaaSやAWSのAPI仕様変更
 - OS・ランタイム・ライブラリの更新や統廃合

EventBridgeなどのメッセージサービスに、状態を集約

- StripeのイベントやAWSリソースのイベントを投入する
 - EventBridgeは「どのイベントを待ち受けるか」で設定できる
 - ただし強い速達性やスケールを求める場合は、**SNSやSQS**が向いているケースもある
- 「この状態になったら」「このワークフローを実行する」で設計
 - 例: 決済に失敗したら、EC2インスタンスを停止する
 - 例: CloudFormationの立ち上げに失敗したら、顧客に返金する

Amazon EventBridgeでWebhook連携を効率化

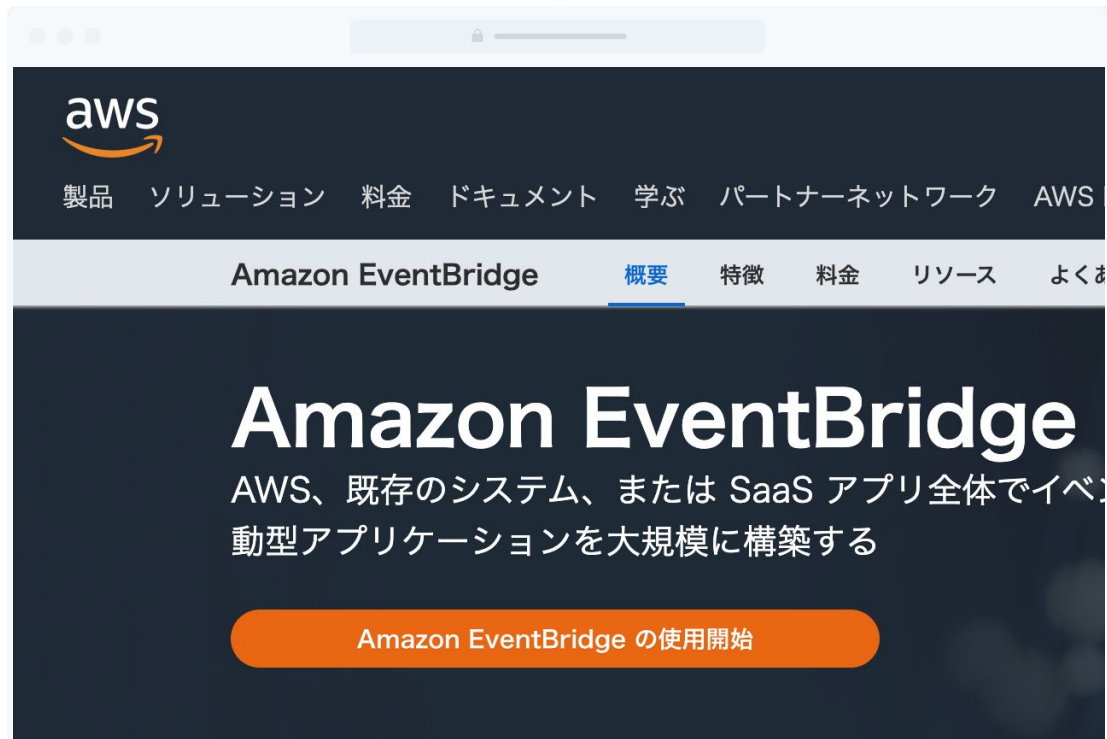
StripeのWebhook APIは、
署名検証などの実装が必要

Amazon EventBridgeの
クイックスタートで、
Webhookの受信部分を
CloudFormationから
セットアップできる

Stripe -> AWS連携を
実質ノーコードで実現

<https://aws.amazon.com/jp/about-aws/whats-new/2022/08/amazon-eventbridge-supports-receiving-events-github-stripe-twilio-using-webhooks/>

stripe



Stripeのイベントから「契約・顧客の状態」を知る

Stripe Webhookのイベントは
リソースの詳細情報が見れる

JSONで構造化(ネスト)した
データがくるので、
Stripe API Docsを参照に
読み解く必要がある

読み解くと、
EventBridgeのフィルターで
複雑な条件検索が可能になる

```
{
  "version": "0",
  "id": "123-456-789",
  "detail-type": "invoice.updated",
  "source": "stripe.com",
  "account": "0123456789",
  "time": "2022-09-07T06:23:31Z",
  "region": "us-east-1",
  "resources": [],
  "detail": {
    "id": "evt_XXXXXXXXXX",
    "object": "event",
    "api_version": "2020-08-27",
    "created": 1662531810,
    "data": {
      "object": {
        "id": "in_XXXXXXXXXX",
        "object": "invoice",
        "account_country": "JP",
        "account_name": "テストアカウント",

```

EventBridgeでのフィルターの例

クイックスタート利用の場合

Sourceでstripe.comを指定

detail-typeで
Stripeのリソース名を指定

detailで、JSON構造に対応した
条件を書くと、絞り込み

右の例:
「5000円以上の請求書」に
関する全てのイベント



```
{
  "source": [{
    "prefix": "stripe.com"
  }],
  "detail-type": [{
    "prefix": "invoice"
  }],
  "detail": {
    "data": {
      "object": {
        "total": [{
          "numeric": [ ">", 5000 ]
        }]
      }
    }
  }
}
```

EventBridgeに「状態」を集約し、システムの起点にする

- AWS内のAPIイベントや、StripeのイベントをEventBridgeへ
- 顧客・契約・システムの**状態**と、その状態に対する**処理**をEventBridgeのルールとして**明示的に定義**
 - CDKやCloudFormationで、コード管理も可能
- 「XXの場面では、この作業を」という「**運用タスク(OPS)**」を、手続き・処理として整理して、AWSで**自動化・効率化**しよう

今日のトピック

- AWS AmplifyとStripeを利用して、SaaSやECのフロントエンド開発をスムーズにできる
- 決済やサブスクリプションなど、顧客・契約の状態と連動したWFも、Amazon EventBridgeでスムーズに
- **AWS AppSync・AWS Step Functionsを活用して、決済・サブスクに関する運用効率化(決済OPS)をはじめよう**

変化した「状態」をどう処理するか？

- EventBridgeで、「この状態になったら、処理を起動」は定義できる
- 「どのような処理を、どの順番で実行するか？」
 - 複数のデータを、繰り返し処理する
 - 失敗した時の、リカバリーや返金フロー
 - 別のリソースが「特定の状態」になることを待機する

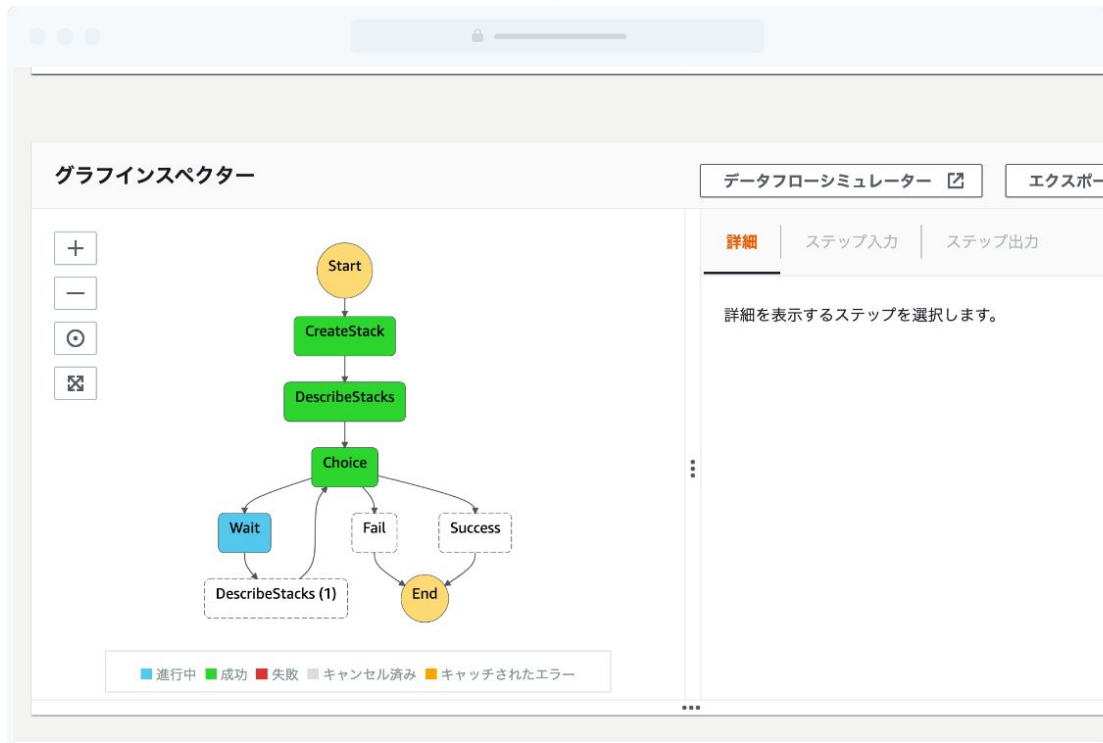
AWS StepFunctionsで、手続き処理をローコードに

ワークフローを設計・実行

AWSのAPI呼び出しが可能
ローコードにWFが作れる

ChoiceやWaitで、
条件分岐や待機も可能

- ・失敗したら、
10秒後にリトライ
- ・3回失敗したら、
キャンセルして返金



Stripe + Step Functionsで、簡易WPホスティングWS

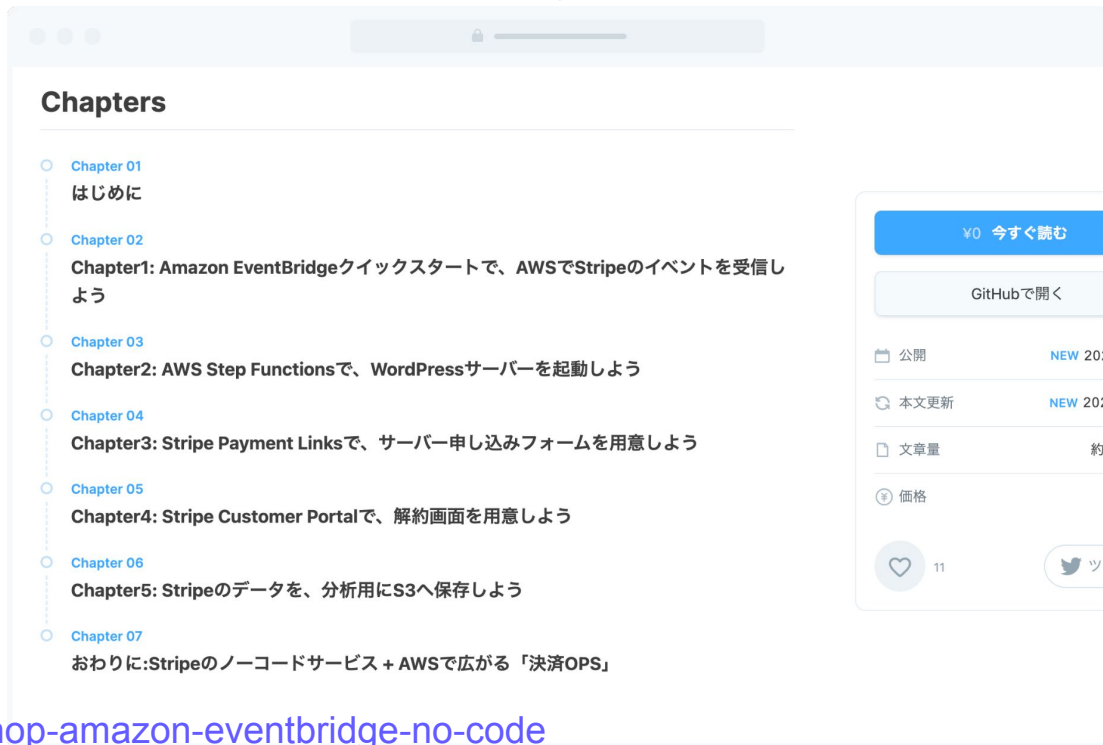
10月に開催したAWS連携WS

Stripe Webhookで
サブスク契約完了を通知

EventBridge -> StepFunctions

StepFunctionsから、
CloudFormationを起動

同様の仕組みで、
解約時のリソース削除も対応



<https://zenn.dev/stripe/books/workshop-amazon-eventbridge-no-code>

ワークフローはGUIで設定可能

Workflow Studio

ドラッグ&ドロップで
処理ステップの編集が可能

AWS API呼び出しの引数は
APIパラメータから設定

入力/出カタブで、
ステップに渡すデータの
編集や指定が可能

The screenshot displays the AWS Workflow Studio interface. On the left, a workflow diagram is shown on a grid background. It starts with a yellow circle labeled 'Start', followed by three rectangular task boxes: 'CloudFormation: CreateStack CreateStack', 'CloudFormation: DescribeStacks DescribeStacks', and 'DynamoDB: PutItem DynamoDB PutItem'. Below these is a 'Choice state Choice' box with a question mark icon. A transition arrow labeled '\$.stack.status == "ROLLBACK"' points from the Choice state to a 'Fail state Fail' box with a red 'X' icon. On the right, a configuration panel is visible with tabs for '設定' (Settings), '入力' (Input), '出力' (Output), and 'エラー処理' (Error Handling). The '設定' tab is active, showing the state name 'DynamoDB PutItem', the API 'DynamoDB: PutItem', the integration type 'Optimized', and a JSON block for API parameters:

```
1 {
2   "TableName": "StripeWSContract",
3   "Item": {
4     "subscription_id": {
5       "S.$": "$.subscription_id"
6     },
7     "stack_name": {
8       "S.$": "$.stack.name"
9     }
10  }
```

Workflow Studioで作って、JSONで共有・管理する

ステートマシンをJSONに

GUIで設計して、
完成したらJSONを取得

CDKやCloudFormation化で
完成したWFのコード管理も

コード管理の現場でも、
設計はWorkflow Studioで
効率化することが可能

```
{
  "Comment": "A description of my state machine",
  "StartAt": "CreateStack",
  "States": {
    "CreateStack": {
      "Type": "Task",
      "Parameters": {
        "StackName.$": "States.Format('LAMP-{}', $.detail.created)",
        "TemplateURL": "https://s3-external-1.amazonaws.com/cloudformation-templates-...",
        "Parameters": [
          {
            "ParameterKey": "KeyName",
            "ParameterValue": "for-stripe-workshop"
          },
          {
            "ParameterKey": "DBName",
            "ParameterValue": "MyDatabase"
          }
        ]
      }
    }
  }
}
```

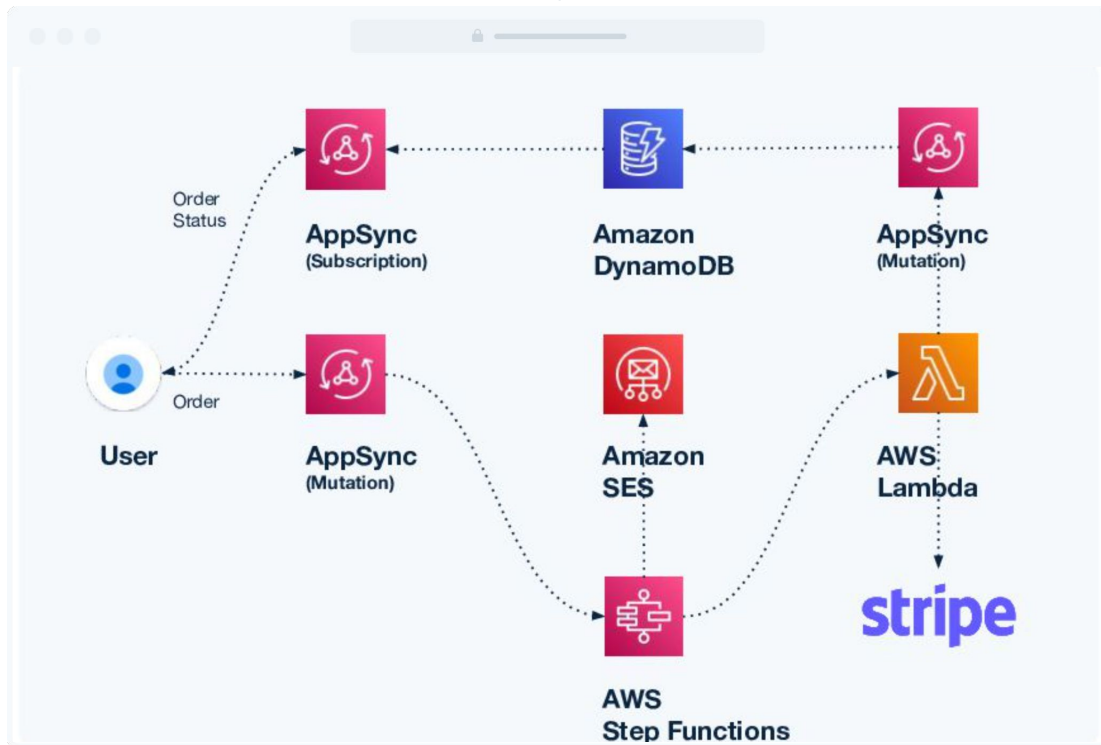
ユーザーとのコミュニケーションにAppSync

イベント駆動では、
ユーザーが「今の状態」を
把握しにくい

APIポーリングでも可能

AppSyncの
Mutation / Subscriptionで
状態変化をLiveに通知できる

MutationはLambdaまたは
EventBridge API Destinationで



EventBridgeで、外部APIを呼び出す (API Destination)

日本語名: API送信先

EventBridgeのルールで、
外部APIを呼び出せる

入力トランスフォーマーで
リクエストBodyのJSON定義も

エラーハンドルや
データ処理を加えたい場合は
引き続きLambdaを用意しよう



API 送信先の詳細

名前
送信先の名前を入力します。この名前はアカウントで一意的にする必要があります。

GraphQLdemo

数字、小文字/大文字、.(ピリオド)、-(ハイフン)、_(アンダーバー)を含め、最大 64 文字まで使用できます。

説明 - オプション
送信先の説明を入力します。

これは説明です

最大 512 文字。

API 送信先エンドポイント [情報](#)
ターゲットとして呼び出す URL エンドポイント (例: パートナーサービスによって生成された有効なエンドポイント)。URL は HTTPS で始まる必要があります。ターゲットの HttpParameters から設定するパスパラメータのワイルドカードとして「*」を含めることができます。

https://u7dno2igqfdxvxdvpsc32rf4.appsync-api.us-east-1.amazonaws.com/graphql

HTTP メソッド
呼び出しエンドポイントに使用する HTTP メソッド (GET、POST、PUT など) を選択します。

POST

Webhookで、ユーザーとの関係を構築しよう

- システムもサブスクリプションも、リリースor契約からが本番
- それぞれのWebhookイベントで、「システム上で何が起きたか」「契約が今どうなっているか」をシステムや人間が簡単に把握・対処できるようになる
- SaaSのAPI / Webhookからの情報を収集して、QuickSightなどを使ったビジネス分析も

**うまくいった施策・システムを
横に展開 or SaaS化する**

Stripe Appsで、自動化・効率化の「横展開」

- Webhookを利用した「仕組み化」に成功すると、
成功施策の横展開やパッケージ化が可能になることも
- **Stripe Apps**で、
StripeとWebhookの処理システムとの連携や配布が簡単に
 - **必要なもの**
 - Webhookを処理するAPIシステム(AWSで構築)
 - React + TypeScript + 専用UI FWでダッシュボードウィジェット

Stripe Appsの例: RPAツールとStripeで顧客データ連携

The screenshot shows a mobile application interface for '顧客リスト' (Customer List). The app header includes navigation icons and a search bar labeled 'アプリ内検索'. Below the header, there is a description: '会社名、担当者名、連絡先などお客様の情報を登録するアプリです。キーワード検索、地域等の条件での絞り込み、リストのCSV出力等が可能です。' (This app registers customer information such as company name, contact name, and contact information. It supports keyword search, narrowing down by conditions such as region, and CSV export of the list.)

The main content area features a table with the following columns: レコード番号 (Record Number), 会社名 (Company Name), 部署名 (Department Name), 担当者名 (Contact Name), and 住所 (Address). The table displays 5 records, with the first one highlighted. The interface also includes a '顧客一覧' (Customer List) button and a filter icon.

レコード番号	会社名	部署名	担当者名	住所
20	金都運総研	情報システム部	下山 達士	岐阜県
19	林田商会	ソリューション営業グ...	森 惇	埼玉県
18	板橋電子株式会社	経理部	末永 妃里	神奈川県
17	新山物産	営業本部第一営業部	金子 真帆	大阪府
16	岩下税理士事務所	情報システム部	佐々木 樹里	兵庫県

決済システム / 決済OPSからはじめるDX

- **守りの決済と攻めの決済**
 - **守りの決済: 顧客が正しく・ストレスなく決済できる環境づくり**
 - 決済・請求システムの安定稼働
 - 金額や取引内容の正確性
 - 不正利用の予防やスムーズな対処
 - **攻めの決済: 顧客がよりお得に、便利にサービスを利用できる仕組み**
 - 契約期間・利用累計額に応じた優待や提案
 - 従量課金プランや休会など、ユースケースにあったプランの企画
 - 決済データを元にしたレコメンドや在庫予測

決済システム / 決済OPSからはじめるDX

- AWSのマネージドサービスとSaaSで、
システムの**安定稼働**や**正確さ**などの「**守り**」を固めよう
- EventBridge / Webhookを活用した「**状態の把握**」で、
顧客や契約、システムに応じた柔軟な「**攻めの運用**」を目指そう
- イベント駆動で疎結合なシステムとして構築し、
Stripe Appsなどで「**成功したシステム**」をas a **Service**化しよう