Engineer

サーバーレス

実行に使うサーバーを自分たちで管理せずにコードを動かす方式

どのようなものか

サーバーレスは、チームがアプリケーションを書いてデプロイし、その下にある機械の運用、拡張、保守を事業者に任せるコードの実行方式です。CNCF Serverless Working Groupのホワイトペーパーでは、サーバーレスコンピューティングという長い名称を使い、チームの誰もサーバーを管理せずにアプリケーションを構築し、動かすという考え方だと説明しています。

名前は実態を大げさに表しています。ホワイトペーパーも「サーバーレスプラットフォームの実行には、引き続きサーバーが必要です」と明言しています。サーバーも運用作業もなくなりません。変わるのは担当者です。チームは、機械のサイズを選ぶ、OSを入れる、パッチを適用する、実行するコピー数を決めるといった作業をしなくなります。渡すのは、コードまたはコンテナーイメージと、それをいつ実行するかの記述です。

ホワイトペーパーは、この分野を2つに分けています。Functions-as-a-service、略してFaaSは、何かが起きたときに小さなロジックを動かすイベント駆動の処理です。関数を起動するものをトリガーと呼び、通常はHTTPリクエストの到着やデータベースの変更といったイベントをきっかけにします。もう一方がBackend-as-a-service、略してBaaSです。APIを介して使う第三者のサービスが、アプリケーションの機能の一部を丸ごと置き換えます。関数は状態を持たず短時間で終わることが期待されるため、状態はBaaS側に置きます。

2つの製品が、その幅を示しています。AWS Lambdaは、サーバーの用意や管理をせずにコードを実行するサーバーレスのコンピューティングサービスと説明されており、元来の形はハンドラーとトリガーからなるイベント関数です。Azure Container Appsは、事業者がインフラを運用し、アプリケーションのコンテナーを実行するサービスとされています。前者が受け取るのは関数、後者はコンテナー全体で、どちらも機械の管理は事業者に任せます。

概念を判断に生かす

「サーバーレスとは？」を実際の判断に生かしましょう。その概念を、開発または購入するシステムに結び付けて考えるお手伝いをします。 実際の使用量に合わせてサーバーを構成し、月額費用を事前に把握したうえで、クラウドへの移行もクラウドからの移転も進めます。

サービスを見る お問い合わせ →

用語の背後にある前提を検証する

解説とリンク先の実績を参考に、プロジェクトについての質問を整理してください。ツールを選ぶ前に、前提を一緒に検討できます。 Google Cloud・AWSでDockerアプリ20個を運用 — 20 デプロイし監視したコンテナアプリケーション

Google Cloud・AWSでDockerアプリ20個を運用 お問い合わせ →

なぜ重要なのか

チームが検討する最初の理由は、課金方式です。ホワイトペーパーは、サーバーレスの料金を従量制と説明し、コードが動いている時間だけ課金され、何も動いていなければ課金されないとしています。Lambdaのイベント関数は、リクエスト数に加え、ギガバイト秒で測る実行時間に応じて課金されます。1日に1時間だけ忙しいサービスなら、その1時間の料金を払い、残りの23時間待機するサーバーの料金は払いません。

2つ目の理由は、増減の両方向への拡張です。Lambdaはトラフィックの増減に応じて実行環境を作成・破棄し、ホワイトペーパーも、仕事がなければインスタンス数がゼロになる関数を説明しています。Microsoftによると、Container AppsはHTTPトラフィック、イベント、CPUやメモリーの負荷に応じて拡張でき、ほとんどのアプリケーションはゼロまで縮小できます。

3つ目は、チームが手放せる作業です。AWSは、Lambdaが代わりに行う仕事として、サーバー保守、処理能力の用意、拡張、パッチ適用を挙げています。小さなチームでは、こうした雑務だけで何もリリースできないまま1週間が埋まることもあります。サーバーレスプラットフォームでは、これらは事業者の責任になります。

ホワイトペーパーの用途例から、適した処理が分かります。データベースの変更への反応、機器から届くセンサー入力の処理、データストリームの処理、通常ならcronジョブにするような短い定期処理です。MicrosoftがあげるContainer Appsの用途には、APIエンドポイント、バックグラウンド処理ジョブ、マイクロサービスもあります。

同じホワイトペーパーは、サーバーレスが最初は最も経済的に見える可能性にも注意を促し、採用前に費用全体を検討するよう求めています。利用量に応じた課金が有利なのは、断続的に集中し、それ以外の大半は動かない処理です。機械を

一日中動かし続ける処理なら、計算は変わります。ホワイトペーパーがIaaSやPaaSより難しくなる場合があるとするデバッグも、費用の一部です。

実際の活用例

AWSのPythonハンドラーのドキュメントには、注文を受け取り、テキストの領収書を書き、Amazon S3バケットへ保存する例があります。短くすると、次のようになります。

```
import os

import boto3

# Runs in the Init phase, once per execution environment.
s3_client = boto3.client("s3")


def lambda_handler(event, context):
    order_id = event["Order_id"]
    bucket_name = os.environ.get("RECEIPT_BUCKET")
    receipt_content = (
        f"OrderID: {order_id}\n"
        f"Amount: ${event['Amount']}\n"
        f"Item: {event['Item']}"
    )
    key = f"receipts/{order_id}.txt"
    s3_client.put_object(Bucket=bucket_name, Key=key, Body=receipt_content)
    return {"statusCode": 200, "message": "Receipt processed successfully"}
```

ハンドラーが取る引数は、ちょうど2つです。イベントは処理対象のデータを運び、JSONイベントはPythonの辞書として届くため、この例は次の形の入力を想定しています。

```
{"Order_id": "12345", "Amount": 199.99, "Item": "Wireless Headphones"}
```

コンテキストは、呼び出し自体に関する情報を運びます。ファイル名が `lambda_function.py` なら、ハンドラー名は `lambda_function.lambda_handler` です。これはLambdaコンソールが新しい関数に与える既定値です。バケット名はコード内ではなく環境変数から取得しており、AWSが運用設定を渡す方法として勧める形です。また、コードに認証情報はありません。Lambdaが独自の環境変数を通じてSDKへ渡します。

クライアントを置く場所が重要です。SnapStartやManaged Instancesを使わないオンデマンドのイベント関数では、Lambdaの実行環境はInit、Invoke、Shutdownの3段階を通ります。ハンドラーの外のコードはInitで1回動き、AWSは最初のInitの試行を10秒に制限しています。その後、ハンドラーは関数のタイムアウト内でイベントごとに動きます。次のイベントに実行環境を再利用する場合をAWSはウォームスタートと呼びます。このとき、Initで作ったクライアントは残っているため、作り直しません。

サーバーレスが関数を意味するとは限りません。当サイトのAzure環境をコードで用意する実績記事には、Azure Container Appsへデプロイした `api` と `www` という2つのサービスが記載されており、コンテナーイメージはプロジェクト専用のAzure Container Registryに置いています。MicrosoftはContainer Appsをコンテナー向けのサーバーレスプラットフォームと呼んでいるため、この2サービスが、このAzure環境でサーバーレスに当たる部分です。どの環境も、パラメーターを受け取る同じ一式のBicepテンプレートで記述しています。そのため、記事では新しい環境の追加をプルリクエストとして扱えます。

逆方向へ進むのが適切な場合もあります。インフラを見直し、処理をAWSから移した記事は、事業の利用量を大きく上回る規模で、不要なサービスも残るAWS構成から始まります。使っていないサービスを削除し、残りの処理能力を調整し、処理をAWSの外へ移しました。サーバーレスの課金は、使っていない処理能力への支払いをやめる1つの方法です。未使用のインフラを見直して取り除くのも別の方法です。どちらが安いかを判断するには、実際の負荷と費用を比較する必要があります。この記事が記録しているのは変更内容であり、同一条件での費用比率ではありません。

問題が起きるところ

サーバーレスでの失敗の多くは、常時稼働する小さなサーバーのように関数を扱うことから起きます。しかし、関数はそのようなものではありません。

コールドスタートで一部のリクエストが遅れます。 準備済みの環境がなければ、Lambdaはコードをダウンロードし、環境を用意し、Initのコードを実行してから、イベントをハンドラーへ渡します。AWSによると、コールドスタートは通常、呼び出しの1%未満に影響し、100ミリ秒未満から1秒超の時間がかかります。開発用やテスト用の関数でより多く発生し、Initの時間にも課金されます。対策は、Initを小さく保ち、必要なものだけをインポートし、めったに使わないオブジェクトは必要時に読み込むことです。遅延が問題になる場合は、プロビジョンドコンカレンシーで指定数の環境を事前に初期化しておけます。

環境内に残した状態が漏れたり消えたりします。 呼び出しはそれぞれ独立して動き、環境が再利用されることも、されないこともあります。AWSはランタイム更新のため、トラフィックが継続していても数時間ごとに環境を入れ替えます。一方、クラッシュやタイムアウト後にリセットしても、`/tmp` ディレクトリーは消去しません。そのため、あるイベントのデータが別のイベントの処理中に現れたり、予告なく消えたりします。対策は、利用者のデータやイベントを環境内に残さず、永続化が必要なものはデータベースかストレージサービスへ置くことです。

同じイベントを2回処理します。 関数には重複イベントが届くことがあります。AWSの推奨事項は、各イベントを検証し、重複を元のイベントと同じように扱う冪等なコードを求めています。上の例では領収書のキーが注文IDから決まるため、2回目の実行は別の領収書を作らず、同じオブジェクトを上書きします。メール送信やカードへの課金など、安全に繰り返せない副作用については、何を実行済みかを別途記録する必要があります。

上限に達すると処理が途中で終わります。 Lambda Managed Instances以外のイベント関数では、1回の呼び出しで実行できるのは最大900秒、つまり15分です。同期リクエストとバッファリングされる同期応答はそれぞれ6 MB、非同期リクエストは1 MB、環境変数は合計4 KBに制限されます。上限を超えると、遅くなるのではなくエラーになります。対策は、最初から制約を織り込んで設計することです。長いジョブは段階に分け、大きなデータはストレージに置いて参照だけを渡し、収まらない処理はコンテナープラットフォームや仮想マシンへ移します。

入口で受け付けられても、同時実行枠が尽きます。 Lambdaの既定のクォータはリージョンごとに1,000同時実行ですが、API Gatewayの既定値は毎秒10,000リクエストです。そのため、入口では受け付けられても、背後の関数では実行しきれないリクエストが生じます。新規アカウントは同時実行数とメモリーのクォータが低い状態で始まり、AWSが利用状況に応じて自動的に引き上げます。対策は、公開前に引き上げを申請し、予想するピーク負荷でテストすることです。

自分を起動する関数が料金を膨らませます。 関数が自身を呼び出したり、自身を再び呼ぶ処理を開始したりすると、大量の呼び出しが発生する可能性があり、その呼び出しはすべて課金されます。AWSは再帰的な呼び出しを避けるよう勧めています。予想外の量が発生した場合は、コードを直すまで、関数の予約済み同時実行数を直ちに `0` へ設定し、すべての呼び出しを抑制します。

プログラミングモデルによって、1つのプラットフォームに縛られます。 ホワイトペーパーは、プラットフォームのプログラミングモデル、イベントとメッセージングのインターフェース、BaaS製品をロックインの原因に挙げています。対策は、業務ロジックを普通の関数に置き、ハンドラーはそれを呼ぶだけにすることです。また、将来の移行が現実的な可能性としてあるなら、どのレジストリーからでも標準的なコンテナーイメージを実行できるContainer Appsのようなプラットフォームを優先します。

参考資料

- CNCF Serverless Whitepaper v1.0 — ワーキンググループによるサーバーレス、FaaS、BaaSの定義と欠点を説明しています。
- AWS Lambdaとは — Lambdaが代わりに管理するもの、関数の種類、課金方式を説明しています。
- PythonでLambdaイベント関数のハンドラーを定義する — 領収書の例の全体、ハンドラーの命名、コードの推奨事項を説明しています。
- Lambda実行環境のライフサイクルを理解する — Init、Invoke、Shutdownの各段階、コールドスタート、環境の再利用を説明しています。
- Lambdaのクォータ — 同時実行数、メモリー、タイムアウト、ペイロードの上限と、その引き上げ方を説明しています。
- Azure Container Appsの概要 — Microsoftのコンテナー向けサーバーレスプラットフォームと、ゼロまで縮小する仕組みを説明しています。

用語集

実際の制約に合わせて選ぶ

デプロイの流れ、運用上のリスク、緊急対応の担当者をお知らせください。検査、復旧手段、明確な責任の所在を伴う変更の範囲を定められます。

料金を見る お問い合わせ →

用語を実務に結び付ける

「サーバーレスとは？」がお客様の組織にどう当てはまるかをご相談ください。
現在のシステムの具体例が一つあれば、話を始められます。

サービスを見る

welcome@engineer.company +45 50 10 61 84 デンマーク、コペンハーゲン

このページを持ち歩く

連絡先カード

心を込めて丁寧に制作： D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

著作権 © 2025 – 現在 · Engineer ApS · すべての権利を留保します。

印刷元： https://engineer.company/ja/glossary/serverless/