AWS 決定ガイド

AWS Fargate または AWS Lambda?

AWS Fargate または AWS Lambda?: AWS 決定ガイド

Table of Contents

AWS Fargate または AWS Lambda?

違いを理解し、自分に合ったものを選択する

目的	または がサーバーレスコンピューティングサービスのニーズ AWS Fargate AWS Lambda を満たしているかどうかを確認するには。
最終更新日	2024 年 11 月 15 日
対象サービス	• AWS Fargate • AWS Lambda

序章

サーバーレスコンピューティングサービス AWS Fargate として AWS Lambda または を選択するかどうかを調べる前に、おそらくより広範なコンピューティングサービス (AWS コンピューティングサービスの選択決定ガイドを参照) を検討し、次の 2 つの選択肢に絞り込んでいるはずです。 AWS

- 運用オーバーヘッドの削減: Lambda と Fargate の両方がサーバー管理を抽象化し、パッチ適用、メンテナンス、キャパシティプランニングの必要性を減らします。
- Pay-per-use: 実際に使用するコンピューティングリソースに対してのみ支払うため、可変ワークロードのコストを削減できる可能性があります。
- デプロイの高速化: 通常、EC2 インスタンスのプロビジョニングと設定よりもデプロイ時間が短縮されます。
- 組み込みの高可用性: どちらのサービスもインフラストラクチャの冗長性を自動的に処理します。
- コンプライアンスの簡素化: 攻撃対象領域を減らし、セキュリティ機能を組み込み、コンプライアンス作業を容易にします。
- コードに集中する: 開発者は、インフラストラクチャを管理するよりもアプリケーションコードの作成に集中できます。

Lambda と Fargate はどちらもサーバーレスオプションですが、それらには大きな違いがあります。

AWS Fargate はコンテナ用のサーバーレスコンピューティングエンジンで、主に Amazon ECS で使用されます。インフラストラクチャを自動的に管理するため、コンテナ化されたアプリケーションの

デプロイとスケーリングに集中できます。Fargate は、長時間実行されるアプリケーション、マイクロサービス、またはバッチ処理に最適です。リソース割り当て (CPU、メモリ) をきめ細かく制御する必要があり、基盤となるサーバーの管理を避けたい場合に最適です。

AWS Lambda は、イベントに応じてコードを自動的に実行し、基盤となるコンピューティングリソースを管理するサーバーレスコンピューティングサービスです。Amazon S3 にアップロードされたファイルの処理、HTTP リクエストへの応答、スケジュールされたタスクの実行など、イベント駆動型アプリケーションに最適です。Lambda は、イベントに応じて自動的にスケーリングし、大量のデータをリアルタイムで処理できるため、ストリーム処理およびデータ処理アプリケーションにも適しています。Lambda は、Amazon Kinesis や Amazon DynamoDB などのソースからのデータストリームを処理できるため、インフラストラクチャを管理することなく、効率的でサーバーレスなデータ変換、フィルタリング、分析が可能になります。Lambda は存続期間の短いタスク (最大 15 分) 向けに設計されており、リクエスト数と実行時間に基づいて請求されるため、散発的なワークロードに対してコスト効率が向上します。

プロジェクトにイベント駆動型の短期間のタスクや予測不可能なワークロードが含まれる場合は、が適している AWS Lambda 可能性があります。特定のリソースニーズでコンテナ化されたアプリケーションを実行する必要がある場合 (または永続的なプロセスが必要な場合) AWS Fargate は、より適切です。

次の表は、これらのサービスの違いの一部をより詳細に示しています。

機能	AWS Fargate	AWS Lambda
実行モデル	コンテナベースのサーバーレスコンピューティング	イベント駆動型のサーバーレス関数
サポートされている言語	コンテナで実行できる任意の言語	サポートされている言語: Node.js、Python、Java、C#、Go、Ruby、PowerShell。カスタムランタイムを構築して、選択した言語で AWS Lambda 関数を実装することもできます。
ユースケース	長時間実行されるコンテナ化されたアプリケーション	短期間のイベント駆動型タスク

スケーリング	必要なタスク数に基づく自動スケーリング	リクエストあたりの自動スケーリング
コールドスタート	35 秒 ~ 2 分	100 ミリ秒 ~ 2 秒
実行時間の制限	ハード制限なし	最大 15 分
メモリの割り当て	最大 120 GiB	最大 10 GiB
CPU 割り当て	最大 16 個の vCPU	メモリに比例、最大 6 vCPU
ネットワーク	VPC で実行、ENIsを使用可	Hyperplane を使用して AWS AWS マネージド VPC で実行することも、カスタマーマネージド VPC にアタッチすることもできます
ステート管理	Fargate のコンテナは、コンテナが実行されている限りリクエスト間で状態を維持できるため、外部ストレージを必要とせずにセッションの処理、データのキャッシュ、インメモリ状態を維持できます。重要なデータには外部ストレージが推奨されます。	ステートレス設計 (ステートは Amazon S3、Amazon DynamoDB、Amazon EFS など、外部で管理する必要があります)
コンテナのサポート	コンテナをサポート	コンテナサポートの制限 (コンテナイメージのデプロイ経由)
オーケストレーション	Amazon ECS との統合	オーケストレーションは不要
料金モデル	使用される vCPU とメモリの 1 秒あたりの請求	呼び出しごとおよび継続時間 (GB 秒)
同時実行数の制限	クラスターの容量に基づく	デフォルトでは 1000 件の同時実行 (増やすことができます)

イベント駆動型の呼び出し	追加セットアップが必要	さまざまな AWS イベントソースのネイティブサポート
コールドスタートの緩和	Seekable OCI を使用したイメージの遅延ロードにより、Fargate タスクの開始を高速化できます	プロビジョニングされた同時実行が利用可能
パッケージサイズ制限	特定の制限なし (設定されたエフェメラルストレージによって制限されるコンテナサイズ、最大 200 GiB)	レイヤーを含む 250 MB 解凍、コンテナイメージデプロイ用 10GB

Fargate と Lambda の違い

Fargate と Lambda のさまざまな主要領域の違いについて説明します。

Languages supported

Fargate: AWS Fargate はコンテナオーケストレーションサービスです。つまり、Docker コンテナにパッケージ化できるプログラミング言語またはランタイム環境をサポートします。この柔軟性により、開発者はアプリケーションのニーズに合ったほぼすべての言語、フレームワーク、またはライブラリを使用できます。Python、Java、Node.js、Go、.NET、Ruby、PHP、またはカスタム言語や環境を使用している場合でも、Fargate はコンテナにカプセル化されている限りそれらを実行できます。この幅広い言語サポートにより、Fargate はレガシーシステム、多言語マイクロサービス、最新のクラウドネイティブアプリケーションなど、さまざまなアプリケーションの実行に最適です。

Lambda: は、Fargate と比較して、イベント駆動型関数用に特別に設計された、より限られた言語のネイティブサポート AWS Lambda を提供します。現時点では、Lambda は次の言語を正式にサポートしています。

- Node.js
- Python
- Java
- Go
- Ruby

- C#
- PowerShell

Lambda はカスタムランタイムもサポートしているため、独自の言語やランタイム環境を導入できますが、ネイティブにサポートされているオプションを使用するよりも多くのセットアップと管理が必要です。コンテナイメージから Lambda 関数をデプロイする場合は、 AWS OS 専用のベースイメージを使用し、イメージに Rust ランタイムクライアントを含めることで、Rust で関数を記述できます。 AWSが提供するランタイムインターフェイスクライアントがない言語を使用している場合は、独自の言語を作成する必要があります。

Event-driven invocation

Lambda は、本質的にイベント駆動型コンピューティング用に設計されています。Lambda 関数は、データの変更、ユーザーアクション、スケジュールされたタスクなど、さまざまなイベントに応じてトリガーされます。Amazon S3 (ファイルのアップロード時に関数を呼び出すなど）、DynamoDB (データ更新時にトリガーするなど）、API Gateway (HTTP リクエストの処理など) AWS のサービスなど、多くの とシームレスに統合されます。Lambda イベント駆動型アーキテクチャは、永続的なコンピューティングリソースを必要とせずにイベントにすぐに応答する必要があるアプリケーションに最適です。

Fargate はネイティブにイベント駆動型ではありませんが、追加の定型ロジックを使用すると、Amazon SQS や Kinesis などのイベントソースと統合できます。Lambda はこの統合ロジックの大部分を処理しますが、これらのサービスの APIs を使用してこの統合を自分で実装する必要があります。

Runtime/use cases

Fargate はコンテナ化されたアプリケーションを実行するように設計されており、コンテナの CPU、メモリ、ネットワーク設定を定義できる柔軟なランタイム環境を提供します。Fargate はコンテナベースのモデルで動作するため、長時間実行されるプロセス、永続的なサービス、および特定のランタイム要件を持つアプリケーションをサポートします。Fargate のコンテナは実行時間に制限がないため、無期限に実行できるため、継続的に稼働する必要があるアプリケーションに最適です。

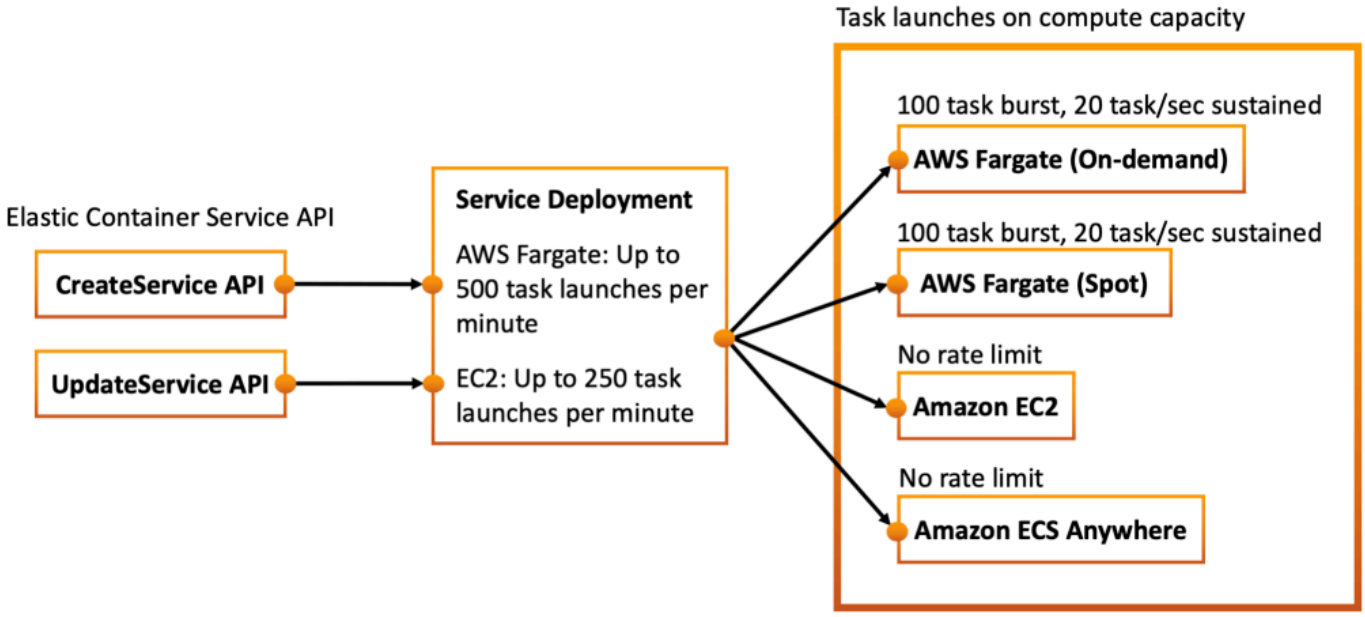
一方、Lambda は、存続期間の短いイベント駆動型タスク用に最適化されています。Lambda 関数は、最大実行時間を 15 分に制限するステートレス環境で実行されます。これにより、Lambda はファイル処理、リアルタイムデータストリーミング、HTTP リクエスト処理などのシナリオに適しています。タスクは短く、長時間実行されるプロセスは必要ありません。

Lambda では、ランタイム環境がより抽象化され、基盤となるインフラストラクチャに対する制御が少なくなります。Lambda のステートレスな性質は、各関数の呼び出しが独立しており、呼び出し間で保持する必要がある状態やデータは外部 (データベースやストレージサービスなど) で管理する必要があることを意味します。

Scaling

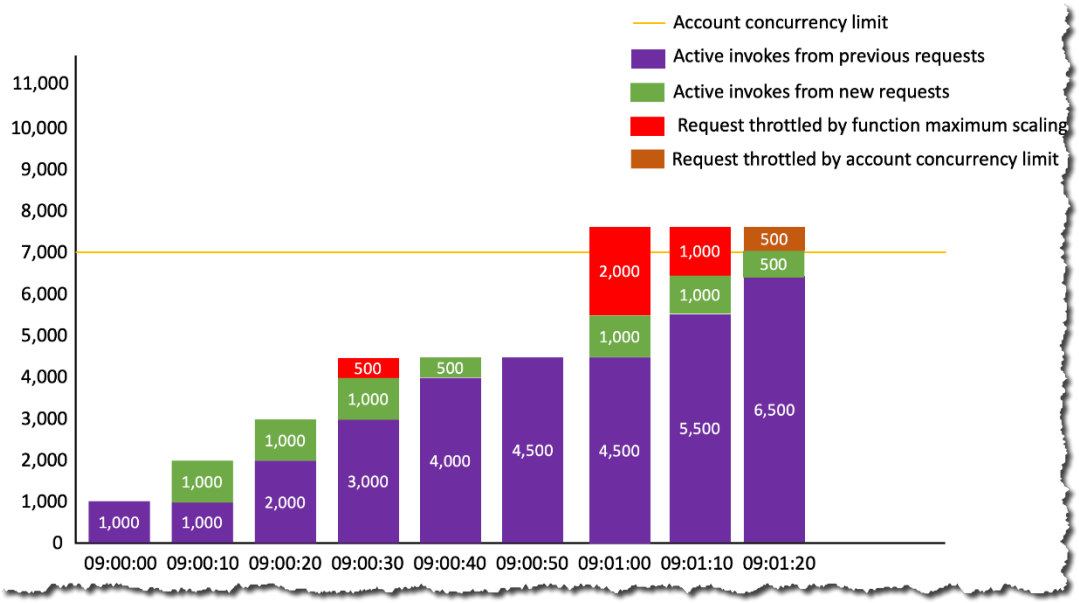
Fargate は、コンテナオーケストレーションサービス (Amazon ECS) で定義された目的の状態に基づいて実行中のコンテナの数を調整することでスケーリングします。このスケーリングは、Amazon EC2 Auto Scaling を使用して手動または自動で実行できます。このブログ記事では、その方法について詳しく説明します。

Fargate では、各コンテナは分離された環境で実行され、スケーリングには追加のコンテナを起動するか、負荷に基づいてコンテナを停止する必要があります。Amazon ECS サービススケジューラは、ウェブおよびその他の長時間実行されるサービスに対して、サービスごとに 1 分未満で最大 500 個のタスクを起動できます。



Lambda の場合、同時実行数は、 AWS Lambda 関数が同時に処理している処理中のリクエストの数です。これは、利用可能なコンピューティングリソースとネットワークリソースがある限り、各 Fargate タスクが同時リクエストを処理できる Fargate の同時実行とは異なります。Lambda は、同時実行リクエストごとに、実行環境の個別のインスタンスをプロビジョニングします。関数が受け取るリクエストが増えると、Lambda が実行環境数のスケーリングを自動的に処理し、これはアカウントの同時実行上限に達するまで行われます。デフォルトでは、Lambda は のすべての関数で同時実行数の合計を 1,000 に制限し AWS リージョン、必要に応じてクォータの引き上げをリクエストできます。

リージョン内の各 Lambda 関数の同時実行スケーリングレートは、アカウントの最大同時実行数まで、10 秒ごとに 1,000 実行インスタンスです。このブログで説明されているように、10 秒間のリクエスト数が 1,000 を超える場合、追加のリクエストはスロットリングされます。次のグラフは、アカウントの同時実行数を 7000 と仮定して Lambda スケーリングがどのように機能するかを示しています。



Cold start and cold-start mitigation

Lambda ではコールドスタートが発生することがあります。コールドスタートは、しばらくアイドル状態になった後に関数が呼び出されたときに発生します。コールドスタート中、Lambda サービスはランタイム、依存関係、関数コードのロードなど、新しい実行環境を初期化する必要があります。このプロセスでは、特に初期化時間が長い言語 (Java、C# など) でレイテンシーが発生する可能性があります。コールドスタートは、特に低レイテンシーの応答を必要とするアプリケーションのパフォーマンスに影響を与える可能性があります。

Lambda のコールドスタートを軽減するために、いくつかの戦略を使用できます。

- 関数サイズを最小化する: 関数パッケージのサイズとその依存関係を減らすと、初期化に必要な時間を短縮できます。
- メモリ割り当てを増やす: メモリ割り当てを増やすと CPU 容量が増加し、初期化時間が短縮される可能性があります。
- 関数をウォームに保つ: Lambda 関数を定期的に呼び出す (CloudWatch Events を使用するなど) と、関数をアクティブに保ち、コールドスタートの可能性を減らすことができます。
- Lambda SnapStart: Lambda SnapStart for Java 関数を使用して起動時間を短縮します。

- プロビジョニングされた同時実行数: この機能は、指定された数の関数インスタンスをウォームに保ち、リクエストを処理する準備ができているため、コールドスタートのレイテンシーが短縮されます。ただし、プロビジョニングされたインスタンスがリクエストをアクティブに処理していない場合でも、プロビジョニングされたインスタンスの料金が発生するため、コストが増加します。

Fargate は通常、Lambda と同じ方法でコールドスタートの影響を受けません。Fargate タスクの開始にかかる時間は、イメージレジストリからタスクで定義されたコンテナイメージをプルするのにかかる時間と直接相関します。Fargate は、Seekable OCI (SOCI) でインデックス化されたコンテナイメージの遅延ロードもサポートしています。SOCI を使用してコンテナイメージを遅延ロードすると、Fargate で Amazon ECS タスクを起動するのにかかる時間が短縮されます。Fargate は、必要な期間アクティブのままのコンテナを実行します。つまり、常にリクエストを処理する準備ができています。ただし、スケーリングイベントに応じて新しいコンテナを起動する必要がある場合、コンテナの初期化に多少の遅延が生じることがありますが、通常、これは Lambda コールドスタートと比較してそれほど重要ではありません。

Memory and CPU options

Fargate は、コンテナ化されたアプリケーションのメモリと CPU リソースの両方をきめ細かく制御します。Fargate でタスクを起動するときに、アプリケーションのニーズに基づいて CPU とメモリの正確な要件を指定できます。CPU とメモリの割り当ては独立しているため、ワークロードに最適な組み合わせを選択できます。例えば、設定に応じて、コンテナごとに 0.25 vCPUs から 16 vCPUs までの CPU 値と 0.5 GB から 120 GB までのメモリを選択できます。

この柔軟性は、メモリを大量に消費するデータベースや CPU バウンド計算タスクなど、特定のパフォーマンス特性を必要とするアプリケーションの実行に最適です。Fargate を使用すると、リソースの割り当てを最適化してコストとパフォーマンスを効果的にバランスさせることができます。

Lambda では、メモリと CPU がリンクされ、選択したメモリ量に比例して CPU が自動的に割り当てられます。128 MB から 10 GB までのメモリ割り当てを 1 MB 単位で選択できます。CPU はメモリ、最大 6 vCPU でスケールします。つまり、メモリ設定が高いほど CPU 電力が増えますが、CPU 割り当て自体を直接制御することはできません。

このモデルはシンプルに設計されており、開発者は CPU 設定を管理することなくメモリ設定をすばやく調整できます。ただし、CPU リソースとメモリリソースの間で特定のバランスを必要とするワークロードでは柔軟性が低い場合があります。Lambda のモデルは、メモリのニーズに基づいて簡単にスケーリングしたいタスクに適していますが、複雑または高度に固有のリソース需要があるアプリケーションには適していない場合があります。

Networking

Fargate にタスクをデプロイすると、タスクは Amazon VPC (Amazon Virtual Private Cloud) で実行され、ネットワーク環境を完全に制御できます。これには、セキュリティグループ、ネットワークアクセスコントロールリスト (ACLs)、ルーティングテーブルの設定が含まれます。各 Fargate タスクは、専用のプライベート IP アドレスを持つ独自のネットワークインターフェイスを取得し、必要に応じてパブリック IP アドレスを割り当てることができます。

Fargate は、ロードバランシング (AWS Elastic Load Balancing を使用) 、VPC ピアリング、VPC AWS のサービス 内の他の への直接アクセスなどの高度なネットワーク機能をサポートしています。は、インターネットを経由することなく AWS のサービス、サポートされている への安全なプライベート接続 AWS PrivateLink にも使用できます。

デフォルトでは、Lambda 関数は、ネットワークインターフェイスまたは IP アドレスを直接制御することなく、マネージドネットワーク環境で実行されます。ただし、Lambda は AWS Hyperplane を使用してカスタマーマネージド VPC にアタッチできるため、VPC 内のリソースへのアクセスを制御できます。

Lambda 関数をカスタマーマネージド VPC にアタッチすると、VPC のセキュリティグループとサブネット設定を継承するため、同じ VPC 内の他の AWS のサービス (RDS データベースなど) と安全にやり取りできます。

Lambda サービスは、Network Function Virtualization プラットフォームを使用して、Lambda VPC からお客様の VPC に NAT 機能を提供します。これにより、Lambda 関数が作成または更新されたポイントで必要な Elastic Network Interface (ENI) が設定されます。また、アカウントの ENI を複数の実行環境で共有できるため、Lambda は関数をスケールするときに制限されたネットワークリソースをより効率的に使用できます。

ENI は枯渇するリソースであり、リージョンごとに 250 ENI というソフト制限があるため、VPC アクセス用に Lambda 関数を設定する場合は、Elastic Network Interface の使用状況をモニタリングする必要があります。同じ AZ および同じセキュリティグループの Lambda 関数は ENIs を共有できます。一般的に、Lambda で同時実行数の制限を増やす場合は、Elastic Network Interface を増やす必要があるかどうかを評価する必要があります。制限に達すると、VPC 対応の Lambda 関数の呼び出しがスロットリングされます。

Pricing model

Fargate の料金は、コンテナに割り当てられたリソース、特にタスクごとに選択した vCPU とメモリに基づいています。コンテナが使用する CPU とメモリに対して、1 秒あたり 1 分の最低料金が請求されます。コストは、アプリケーションが消費するリソースに直接関連しています。つまり、アプリケーションがリクエストをアクティブに処理しているかどうかにかかわらず、プ

ロビジョニングした分に対して料金が発生します。Fargate は、特定のリソース設定を必要とする予測可能なワークロードに最適で、割り当てられたリソースを調整することでコストを最適化できます。さらに、データ転送、ストレージ、ネットワークなどの関連サービス (VPC、Elastic Load Balancing など) には追加料金が発生する場合があります。

Lambda には、イベント駆動型およびpay-per-executionという異なる料金構造があります。関数が受信したリクエストの数と各実行時間に基づいて課金され、ミリ秒単位で測定されます。Lambda は、関数に割り当てるメモリの量も考慮します。コストは、使用するメモリと実行時間に基づいてスケーリングされます。料金モデルには無料利用枠が含まれており、1 か月あたり 100 万件の無料リクエストと 400,000 GB 秒のコンピューティング時間を提供するため、Lambda は少量の散発的なワークロードに対して特にコスト効率が高くなります。

Lambda 料金モデルは、予測不可能なトラフィックパターンやバーストトラフィックパターンのアプリケーションに最適です。実際の関数の呼び出しと実行時間に対してのみ料金を支払うため、アイドル状態の容量をプロビジョニングしたり料金を支払う必要はありません。

使用アイテム

AWS Fargate と の選択基準について読んだので AWS Lambda、ニーズに合ったサービスを選択し、以下の情報を使用してそれぞれの使用を開始できます。

AWS Fargate

- Fargate 起動タイプの Amazon ECS Linux タスクを作成する方法について説明します。

  Linux タスクに Fargate 起動タイプを使用して AWS Fargate 、 で Amazon ECS の使用を開始します。

  ガイドを見る
- Fargate 起動タイプの Amazon ECS Windows タスクを作成する方法について説明します。

  Windows タスクに Fargate 起動タイプを使用して AWS Fargate 、 で Amazon ECS の使用を開始します。

  ガイドを見る
- Fargate と Amazon EKS の開始方法

  このガイドでは、Amazon EKS クラスター AWS Fargate を使用して でポッドの実行を開始する方法について説明します。

ガイドを見る

- AWS Fargate 料金

  このガイドでは、vCPU、メモリ、ストレージ、オペレーティングシステムの設定が AWS Fargate 料金にどのように影響するかを理解します。

  ガイドを見る

- AWS Fargate よくある質問

  AWS Fargate 機能に関する一般的な質問に対する回答と、実装のベストプラクティスを取得します。

  ガイドを見る

AWS Lambda

- サーバーレスファイル処理アプリケーションを作成する

  Amazon SNS をセットアップして使用するstep-by-stepのチュートリアル。トピックの作成、トピックへのエンドポイントのサブスクライブ、メッセージの発行、アクセス許可の設定などのトピックについて説明します。

  ガイドを見る

- Serverless Developer Guide

  このガイドは、サーバーレスアプリケーション開発と、クラウドアプリケーションの中核となるアプリケーションパターンを作成するためにさまざまな AWS のサービス がどのように連携するかをより概念的に理解するのに役立ちます。

  ガイドを見る

- サーバーレスランド

  このサイトには、 AWS Serverless の最新情報、ブログ、動画、コード、学習リソースがまとめられています。低コストでフルマネージド型のサーバーレスアーキテクチャで自動的にスケールするアプリケーションを使用および構築する方法について説明します。

  サイトの詳細

- AWS Lambda 料金

このガイドを使用して、関数の使用状況と設定に基づいて費用を見積もり、コストを最適化します。これには、 AWS Lambda とアーキテクチャのコストを 1 回の見積りで計算するための料金計算ツールが含まれています。

ガイドを見る

- AWS Lambda よくある質問

  AWS Lambda 機能に関する一般的な質問に対する回答と、実装のベストプラクティスを取得します。

  ガイドを見る

AWS Fargate または AWS Lambdaのドキュメント履歴

次の表に、この決定ガイドの重要な変更点を示します。このガイドの更新に関する通知については、RSS フィードをサブスクライブできます。

変更	説明	日付
初回リリース	決定ガイドの初回リリース。	2024 年 11 月 15 日

翻訳は機械翻訳により提供されています。提供された翻訳内容と英語版の間で齟齬、不一致または矛盾がある場合、英語版が優先します。