ユーザーガイド

AWS AppConfig

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

AWS AppConfig: ユーザーガイド

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon の商標およびトレードドレスはAmazon 以外の製品およびサービスに使用することはできません。また、お客様に誤解を与える可能性がある形式で、または Amazon の信用を損なう形式で使用することもできません。Amazon が所有していないその他のすべての商標は Amazon との提携、関連、支援関係の有無にかかわらず、それら該当する所有者の資産です。

Table of Contents

とは AWS AppConfig 1
AWS AppConfig ユースケース 2
を使用する利点 AWS AppConfig 2
の AWS AppConfig 仕組み 3
AWS AppConfigの使用を開始 5
SDK 5
の料金 AWS AppConfig 6
AWS AppConfig クォータ 6
セットアップ AWS AppConfig 7
にサインアップする AWS アカウント 7
管理アクセスを持つユーザーを作成する 7
プログラム的なアクセス権を付与する 9
IPv6 サポートについて 10
(推奨) 自動ロールバックのアクセス許可を設定する 11
ステップ 1: CloudWatch アラームに基づいてロールバック用のアクセス許可ポリシーを作成する 12
ステップ 2: CloudWatch アラームに基づいてロールバック用の IAM ロールを作成する 13
ステップ 3: 信頼関係を追加する 14
作成 15
設定プロファイル IAM ロールの概要 16
ネームスペースの作成 18
AWS AppConfig アプリケーションの作成 (コンソール) 19
AWS AppConfig アプリケーションの作成 (コマンドライン) 20
環境の作成 21
AWS AppConfig 環境の作成 (コンソール) 22
AWS AppConfig 環境の作成 (コマンドライン) 23
AWS AppConfigで設定プロファイルを作成します。 25
機能フラグ設定プロファイルを作成します。 28
フリーフォームの設定プロファイルを作成します。 61
非ネイティブデータソースの設定プロファイルの作成 76
デプロイ 78
デプロイ戦略の使用 79
定義済みのデプロイ戦略の使用 81
デプロイ戦略の作成 83

構成のデプロイ ... 87
設定をデプロイする (コンソール) ... 88
設定をデプロイする (コマンドライン) ... 89
CodePipeline を使用したデプロイ ... 93
統合の仕組み ... 93
設定の復元 ... 94
取得中 ... 96
AWS AppConfig エージェントとは ... 97
AWS AppConfig エージェントを使用して設定データを取得する方法 ... 99
での AWS AppConfig エージェントの使用 AWS Lambda ... 99
Amazon EC2 およびオンプレミスマシンでの AWS AppConfig エージェントの使用 ... 190
Amazon ECS および Amazon EKS での AWS AppConfig エージェントの使用 ... 207
機能フラグの取得 ... 225
マニフェストを使用して追加の取得機能を有効にする ... 229
OpenAPI 仕様を使用したクライアントの生成 ... 239
AWS AppConfig エージェントローカル開発モードの使用 ... 242
ブラウザとモバイルの使用に関する考慮事項 ... 247
設定データとフラグの取得 ... 247
認証と Amazon Cognito ... 248
キャッシュ ... 248
セグメンテーション ... 249
帯域幅 (モバイルユースケース) ... 249
その他のフラグのユースケース ... 250
AWS AppConfig エージェントを使用せずに設定データを取得する ... 250
(例) AWS AppConfig APIs を呼び出して設定を取得する ... 252
AWS AppConfig ワークフローの拡張 ... 254
AWS AppConfig 拡張機能について ... 254
ステップ 1: 拡張機能を使用する操作を決定します ... 255
ステップ 2: 拡張機能をいつ実行するかを決める ... 256
ステップ 3: 拡張機能の作成 ... 257
ステップ 4: 設定をデプロイし、拡張機能のアクションが実行されたことを確認する ... 258
AWS が作成した拡張機能での作業 ... 258
Amazon CloudWatch Evidently エクステンションの使用 ... 259
Amazon EventBridge 拡張機能への AWS AppConfig デプロイイベントの使用 ... 259
Amazon SNS 拡張機能への AWS AppConfig デプロイイベントの使用 ... 262
Amazon SQS 拡張機能への AWS AppConfig デプロイイベントの使用 ... 264

Jira 拡張機能の使用 ... 267
チュートリアル: カスタム AWS AppConfig 拡張機能の作成 ... 272
ステップ 1: カスタム AWS AppConfig 拡張機能の Lambda 関数を作成する ... 274
ステップ 2: カスタム AWS AppConfig 拡張機能のアクセス許可を設定する ... 280
ステップ 3: カスタム AWS AppConfig 拡張機能を作成する ... 281
ステップ 4: カスタム拡張機能の AWS AppConfig 拡張機能関連付けを作成する ... 285
コードサンプル ... 287
ホスト設定ストアに保存されているフリーフォーム設定の作成または更新 ... 287
Secrets Manager に保存されているシークレットの設定プロファイルの作成 ... 290
設定プロファイルのデプロイ ... 291
AWS AppConfig エージェントを使用してフリーフォーム設定プロファイルを読み取る ... 296
AWS AppConfig エージェントを使用して特定の機能フラグを読み取る ... 298
AWS AppConfig エージェントを使用してバリアントを含む機能フラグを取得する ... 299
GetLatestConfiguration API アクションを使用してフリーフォーム設定プロファイルを読み取る ... 301
環境のクリーンアップ ... 308
削除保護 ... 314
削除保護チェックをバイパスまたは強制する ... 315
セキュリティ ... 317
最小特権アクセスの実装 ... 317
AWS AppConfigでの静止時のデータの暗号化 ... 318
AWS PrivateLink ... 322
考慮事項 ... 323
インターフェイスエンドポイントの作成 ... 323
エンドポイントポリシーを作成する ... 324
Secrets Manager のキーローテーション ... 325
によってデプロイされた Secrets Manager シークレットの自動ローテーションの設定 AWS AppConfig ... 325
モニタリング ... 327
CloudTrail ログ ... 328
AWS AppConfig CloudTrail のデータイベント ... 329
AWS AppConfig CloudTrail の管理イベント ... 331
AWS AppConfig イベントの例 ... 331
AWS AppConfig データプレーン呼び出しのメトリクスのログ記録 ... 333
CloudWatch メトリクスのアラームを作成します ... 335
自動ロールバックのためのデプロイのモニタリング ... 336

自動ロールバックをモニタリングするための推奨メトリクス 337
ドキュメント履歴 343
..... ccclxix

とは AWS AppConfig

AWS AppConfig 機能フラグと動的設定により、ソフトウェアビルダーはフルコードのデプロイなしで本番環境のアプリケーション動作を迅速かつ安全に調整できます。 AWS AppConfig はソフトウェアリリース頻度を高速化し、アプリケーションの耐障害性を向上させ、緊急の問題をより迅速に対処できます。機能フラグを使用すると、新しい機能をすべてのユーザーに完全にデプロイする前に、徐々にユーザーにリリースし、それらの変更の影響を測定できます。運用フラグと動的設定を使用すると、ブロックリスト、許可リスト、スロットリング制限、ロギングの冗長性を更新したり、その他の運用上の調整を行うことで、実稼働環境の問題に迅速に対応できます。

> **Note**
>
> AWS AppConfig は のツールです AWS Systems Manager。

効率を高め、変更をより早くリリースできます

機能フラグを新機能とともに使用すると、変更内容を本番環境にリリースするプロセスがスピードアップします。機能フラグを使用すると、リリース前に複雑なマージを必要とする長期にわたる開発ブランチに頼る代わりに、トランクベースの開発を使用してソフトウェアを作成できます。機能フラグを使用すると、ユーザーには見えない CI/CD パイプラインでリリース前のコードを安全にロールアウトできます。変更をリリースする準備ができたら、新しいコードをデプロイしなくても機能フラグを更新できます。リリースが完了した後も、フラグはコードのデプロイをロールバックしなくても新しい機能や無効にするブロックスイッチとして機能します。

組み込みの安全機能により、意図しない変更や障害を回避できます

AWS AppConfig には、機能フラグを有効にしたり、アプリケーション障害の原因となる設定データを更新したりしないように、次の安全機能が用意されています。

- バリデーター: バリデーターが、本番環境にデプロイする前に、構文的にもセマンティック的にも正しいことを確認します。
- デプロイ戦略: この戦略を使うと、数分または数時間かけて変更を製品環境にゆっくりとリリースできます。
- モニタリングと自動ロールバック: Amazon CloudWatch と AWS AppConfig 統合して、アプリケーションの変更をモニタリングします。不適切な設定変更が原因でアプリケーションに異常が発生し、その変更が CloudWatch のアラームをトリガーした場合、 AWS AppConfig アプリケーションユーザーへの影響を最小限に抑えるために変更を自動的にロールバックします。

安全でスケーラブルな機能フラグのデプロイ

AWS AppConfig は AWS Identity and Access Management (IAM) と統合され、サービスへのロールベースのきめ細かなアクセスを提供します。 AWS AppConfig また、 は暗号化と監査のために AWS Key Management Service （ AWS KMS) AWS CloudTrail とも統合されます。外部顧客にリリースされる前に、すべての AWS AppConfig 安全管理は、最初に を使用して開発され、大規模なサービスを使用する内部顧客によって検証されました。

AWS AppConfig ユースケース

アプリケーション設定の内容はアプリケーションによって大きく異なる可能性があるにもかかわらず、 では、幅広い顧客のニーズをカバーする以下のユースケース AWS AppConfig がサポートされています。

- 機能フラグとトグル — 管理された環境で、新しい機能を安全に顧客にリリースできます。問題が発生した場合は、変更を即座にロールバックできます。
- アプリケーションのチューニング — アプリケーションの変更を慎重に導入し、その変更による影響を本番環境のユーザーにテストします。
- 許可リストまたは禁止リスト — 新しいコードをデプロイしなくても、プレミアム機能へのアクセスを制御したり、特定のユーザーを即座にブロックできます。
- 一元化された設定ストレージ — すべてのワークロードにわたって設定データを整理し、一貫性のある状態に保ちます。を使用して AWS AppConfig 、 AWS AppConfig ホストされた設定ストア、、 AWS Secrets Manager Systems Manager パラメータストア、または Amazon S3 に保存されている設定データをデプロイできます。

を使用する利点 AWS AppConfig

AWS AppConfig では、組織に次の利点があります。

- 顧客の予期しないダウンタイムを削減します

  AWS AppConfig は、設定を検証するためのルールを作成できるようにすることで、アプリケーションのダウンタイムを軽減します。有効でない設定はデプロイできません。 には、設定を検証するための次の 2 つのオプション AWS AppConfig が用意されています。

  - 構文検証では、JSON スキーマを使用できます。 は、JSON スキーマを使用して設定 AWS AppConfig を検証し、設定変更がアプリケーション要件に準拠していることを確認します。

- セマンティック検証のために、 は所有している AWS Lambda 関数を AWS AppConfig 呼び出して、設定内のデータを検証できます。
- 一連ののターゲットにわたって迅速に変更をデプロイします

  AWS AppConfig は、一元的な場所から設定変更をデプロイすることで、大規模なアプリケーションの管理を簡素化します。 は、 AWS AppConfig ホストされた設定ストア、Systems Manager パラメータストア、Systems Manager (SSM) ドキュメント、および Amazon S3 に保存されている設定 AWS AppConfig をサポートします。EC2 インスタンス AWS AppConfig 、コンテナ、モバイルアプリケーション AWS Lambda、または IoT デバイスでホストされているアプリケーションでを使用できます。

  ターゲットは、Systems Manager SSM エージェントまたは他の Systems Manager ツールに必要な IAM インスタンスプロファイルで設定する必要はありません。つまり、 はアンマネージドインスタンスで AWS AppConfig 動作します。
- 中断することなくアプリケーションを更新する

  AWS AppConfig は、実行時に、重いビルドプロセスやターゲットのサービス停止を行わずに、ターゲットに設定変更をデプロイします。
- アプリケーション全体で変更のデプロイを制御する

  設定変更をターゲットにデプロイする場合、 AWS AppConfig ではデプロイ戦略を使用してリスクを最小限に抑えることができます。デプロイメント戦略により、設定の変更を徐々にフリートに展開できます。デプロイ中に問題が発生した場合は、設定変更が大半のホストに届く前にロールバックできます。

の AWS AppConfig 仕組み

このセクションでは、 AWS AppConfig の仕組みと開始方法の概要を説明します。

1. クラウドで管理したいコード内の設定値を特定します。

   AWS AppConfig アーティファクトの作成を開始する前に、 を使用して動的に管理するコード内の設定データを特定することをお勧めします AWS AppConfig。良い例としては、機能フラグやトグル、許可リストと禁止リスト、ロギングの冗長性、サービス制限、スロットリングルールなどがあります。

   設定データがすでにクラウドに存在する場合は、 AWS AppConfig 検証、デプロイ、拡張機能を活用して、設定データ管理をさらに効率化できます。

2. アプリケーションの名前空間を作成する

   名前空間を作成するには、アプリケーションと呼ばれる AWS AppConfig アーティファクトを作成します。アプリケーションはフォルダのようなシンプルな組織構造です。

3. 環境の作成

   AWS AppConfig アプリケーションごとに、1 つ以上の環境を定義します。環境は、 `Beta`または`Production`環境のアプリケーション、 AWS Lambda 関数、コンテナなどのターゲットの論理グループです。`Web`、`Mobile`、および `Back-end` など、アプリケーションのサブコンポーネントの環境を定義することもできます。

   各環境に対して Amazon CloudWatch のアラームを設定できます。システムは、設定のデプロイ中にアラームをモニタリングします。アラームがトリガーされると、システムは設定をロールバックします。

4. 設定プロファイルの作成

   設定プロファイルには、特に、 がその保存場所とプロファイルタイプで設定データを検索 AWS AppConfig できるようにする URI が含まれます。 AWS AppConfig は、機能フラグとフリーフォーム設定の 2 つの設定プロファイルタイプをサポートしています。機能フラグ設定プロファイルは、 AWS AppConfig ホストされた設定ストアにデータを保存し、URI は単純にですhosted。フリーフォーム設定プロファイルの場合、「」で説明されているように AWS AppConfig、 AWS AppConfig ホストされた設定ストアまたは と統合されている任意の AWS サービスにデータを保存できますでの無料フォーム設定プロファイルの作成 AWS AppConfig。

   設定プロファイルには、設定データが構文的および意味的に正しいことを確認するためのオプションのバリデータを含めることもできます。 は、デプロイを開始するときにバリデータを使用してチェック AWS AppConfig を実行します。エラーが検出されると、デプロイは前の設定データにロールバックします。

5. 設定データをデプロイします

   新しいデプロイを作成する際には、以下を指定できます。

   - アプリケーション ID
   - 設定プロファイル ID
   - 設定バージョン。
   - 設定データをデプロイする環境 ID
   - 変更をどのくらいの速さで反映させたいかを定義するデプロイ戦略 ID

   StartDeployment API アクションを呼び出すと、 は次のタスク AWS AppConfig を実行します。

1. 設定プロファイルのロケーション URI を使用して、基になるデータストアから設定データを取得します。
2. 設定プロファイルに作成したときに指定したバリデータを使用して、設定データが構文的にも意味論的にも正しいことを確認します。
3. データのコピーをキャッシュして、アプリケーションがすぐに取り出せるようにします。このキャッシュされたコピーはデプロイされたデータと呼ばれます。

6. 設定を取得します。

   AWS AppConfig エージェントをローカルホストとして設定し、エージェントに設定の更新 AWS AppConfig をポーリングできます。エージェントは StartConfigurationSession と GetLatestConfiguration API アクションを呼び出し、設定データをローカルにキャッシュします。データを取得するために、アプリケーションは localhost サーバーに HTTP 呼び出しを行います。 AWS AppConfig Agent は、「」で説明されているように、いくつかのユースケースをサポートしていますAWS AppConfig エージェントを使用して設定データを取得する方法。

   AWS AppConfig エージェントがユースケースでサポートされていない場合は、StartConfigurationSession および GetLatestConfiguration API アクションを直接呼び出すことで、 AWS AppConfig 設定の更新をポーリングするようにアプリケーションを設定できます。

AWS AppConfigの使用を開始

以下のリソースは、 を直接操作するのに役立ちます AWS AppConfig。

Amazon Web Services YouTube Channel でさらに多くの AWS 動画をご覧ください。

次のブログは、 AWS AppConfig とその機能の詳細を理解するのに役立ちます。

- 機能フラグ AWS AppConfig の使用
- AWS AppConfig 機能フラグと設定データを検証するためのベストプラクティス

SDK

AWS AppConfig 言語固有の SDKs の詳細については、以下のリソースを参照してください。

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++

- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

の料金 AWS AppConfig

の料金は AWS AppConfig 、設定データと機能フラグの取得に基づいてpay-as-you-goです。エージェントを使用してコスト AWS AppConfig を最適化することをお勧めします。詳細については、「AWS Systems Manager 料金」を参照してください。

AWS AppConfig クォータ

AWS AppConfig エンドポイントとサービスクォータ、およびその他の Systems Manager クォータに関する情報は、「」を参照してくださいAmazon Web Services 全般のリファレンス。

> **Note**
>
> AWS AppConfig 設定を保存するサービスのクォータについては、「」を参照してください設定ストアのクォータと制限について。

セットアップ AWS AppConfig

まだサインアップしていない場合は、 にサインアップ AWS アカウント して管理ユーザーを作成します。

にサインアップする AWS アカウント

がない場合は AWS アカウント、次の手順を実行して作成します。

にサインアップするには AWS アカウント

1. https://portal.aws.amazon.com/billing/signup を開きます。
2. オンラインの手順に従います。

   サインアップ手順の一環として、電話またはテキストメッセージを受け取り、電話キーパッドで検証コードを入力します。

   にサインアップすると AWS アカウント、 AWS アカウントのルートユーザー が作成されます。ルートユーザーには、アカウントのすべての AWS のサービス とリソースへのアクセス権があります。セキュリティベストプラクティスとして、ユーザーに管理アクセス権を割り当て、ルートユーザーアクセスが必要なタスクの実行にはルートユーザーのみを使用するようにしてください。

AWS サインアッププロセスが完了すると、 から確認メールが送信されます。https://aws.amazon.com/ の [マイアカウント] をクリックして、いつでもアカウントの現在のアクティビティを表示し、アカウントを管理することができます。

管理アクセスを持つユーザーを作成する

にサインアップしたら AWS アカウント、日常的なタスクにルートユーザーを使用しないように AWS アカウントのルートユーザー、 を保護し AWS IAM Identity Center、 を有効にして管理ユーザーを作成します。

を保護する AWS アカウントのルートユーザー

1. ルートユーザーを選択し、 AWS アカウント E メールアドレスを入力して、アカウント所有者AWS Management Consoleとして にサインインします。次のページでパスワードを入力します。

ルートユーザーを使用してサインインする方法については、AWS サインイン ユーザーガイドのルートユーザーとしてサインインするを参照してください。

2. ルートユーザーの多要素認証 (MFA) を有効にします。

   手順については、IAM ユーザーガイドの AWS アカウント 「ルートユーザー (コンソール) の仮想 MFA デバイスを有効にする」を参照してください。

管理アクセスを持つユーザーを作成する

1. IAM アイデンティティセンターを有効にします。

   手順については、「AWS IAM Identity Center ユーザーガイド」の「AWS IAM Identity Centerの有効化」を参照してください。

2. IAM アイデンティティセンターで、ユーザーに管理アクセスを付与します。

   を ID ソース IAM アイデンティティセンターディレクトリ として使用する方法のチュートリアルについては、「 AWS IAM Identity Center ユーザーガイド」の「デフォルトを使用してユーザーアクセスを設定する IAM アイデンティティセンターディレクトリ」を参照してください。

管理アクセス権を持つユーザーとしてサインインする

- IAM アイデンティティセンターのユーザーとしてサインインするには、IAM アイデンティティセンターのユーザーの作成時に E メールアドレスに送信されたサインイン URL を使用します。

  IAM Identity Center ユーザーを使用してサインインする方法については、AWS サインイン「 ユーザーガイド」の AWS 「 アクセスポータルにサインインする」を参照してください。

追加のユーザーにアクセス権を割り当てる

1. IAM アイデンティティセンターで、最小特権のアクセス許可を適用するというベストプラクティスに従ったアクセス許可セットを作成します。

   手順については、「AWS IAM Identity Center ユーザーガイド」の「権限設定を作成する」を参照してください。

2. グループにユーザーを割り当て、そのグループにシングルサインオンアクセス権を割り当てます。

手順については、「AWS IAM Identity Center ユーザーガイド」の「グループの結合」を参照してください。

プログラム的なアクセス権を付与する

ユーザーが の AWS 外部で を操作する場合は、プログラムによるアクセスが必要です AWS Management Console。プログラムによるアクセスを許可する方法は、 がアクセスするユーザーのタイプによって異なります AWS。

ユーザーにプログラマチックアクセス権を付与するには、以下のいずれかのオプションを選択します。

プログラマチックアクセス権を必要とするユーザー	目的	方法
ワークフォースアイデンティティ (IAM アイデンティティセンターで管理されているユーザー)	一時的な認証情報を使用して AWS CLI、、 AWS SDKs、または AWS APIs。	使用するインターフェイスの指示に従ってください。 • については AWS CLI、「 AWS Command Line Interface ユーザーガイド」の「 を使用する AWS CLI ように AWS IAM Identity Centerを設定する」を参照してください。 • AWS SDKs、ツール、API については、 AWS APIs「 SDK およびツールリファレンスガイド」の「IAM アイデンティティセンター認証」を参照してください。AWS SDKs
IAM	一時的な認証情報を使用して AWS CLI、、 AWS SDKs、または AWS APIs。	「IAM ユーザーガイド」の「 AWS リソースでの一時的

プログラマチックアクセス権を必要とするユーザー	目的	方法
		な認証情報の使用」の手順に従います。
IAM	(非推奨) 長期認証情報を使用して、AWS CLI、 AWS SDKs、または AWS APIs。	使用するインターフェイスの指示に従ってください。 • については AWS CLI、「 AWS Command Line Interface ユーザーガイド」の「IAM ユーザー認証情報を使用した認証」を参照してください。 • AWS SDKs「 SDK とツールリファレンスガイド」の「長期認証情報を使用した認証」を参照してください。 AWS SDKs • API AWS APIs「IAM ユーザーガイド」の「IAM ユーザーのアクセスキーの管理」を参照してください。

IPv6 サポートについて

すべての AWS AppConfig APIs IPv4 および IPv6 呼び出しを完全にサポートしています。

コントロールプレーン API

コントロールプレーンへの IPv4 および IPv6 デュアルスタック呼び出しには、次のエンドポイントを使用します。

```
appconfig.Region.api.aws
```

例: appconfig.us-east-1.api.aws

IPv4 の場合のみ、次の URL を使用します。

```
appconfig.Region.amazonaws.com
```

データプレーン APIs

データプレーンへのデュアルスタック呼び出しには、次のエンドポイントを使用します。

```
appconfigdata.Region.api.aws
```

例: appconfig.us-east-1.api.aws

IPv4 の場合のみ、次の URL を使用します。

```
appconfigdata.Region.amazonaws.com
```

> **Note**
>
> 詳細については、「AWS 全般のリファレンス」の「AWS AppConfig エンドポイントとクォータ」を参照してください。

(推奨) 自動ロールバックのアクセス許可を設定する

1 つ以上の Amazon CloudWatch アラームに応じて、設定の以前のバージョンにロールバック AWS AppConfig するように を設定できます。CloudWatch アラームに応答するようにデプロイを設定する場合、 AWS Identity and Access Management (IAM) role を指定します。CloudWatch アラームをモニタリングできるように、 はこのロール AWS AppConfig が必要です。この手順はオプションですが強くお勧めします。

> **Note**
>
> 以下の情報に注意してください。
>
> - IAM ロールは 現在のアカウントに属している必要があります。デフォルトでは、 は現在のアカウントが所有するアラームのみをモニタリング AWS AppConfig できます。

- モニタリングするメトリクスと、自動ロールバック AWS AppConfig 用に を設定する方法については、「」を参照してください自動ロールバックのためのデプロイのモニタリング。

次の手順を使用して、 が CloudWatch アラームに基づいて AWS AppConfig ロールバックできるようにする IAM ロールを作成します。このセクションには、以下の手順が含まれます。

1. ステップ 1: CloudWatch アラームに基づいてロールバック用のアクセス許可ポリシーを作成する
2. ステップ 2: CloudWatch アラームに基づいてロールバック用の IAM ロールを作成する
3. ステップ 3: 信頼関係を追加する

ステップ 1: CloudWatch アラームに基づいてロールバック用のアクセス許可ポリシーを作成する

次の手順を使用して、 `DescribeAlarms` API アクションを呼び出す AWS AppConfig アクセス許可を付与する IAM ポリシーを作成します。

CloudWatch アラームに基づいてロールバック用の IAM アクセス許可ポリシーを作成するには

1. https://console.aws.amazon.com/iam/ で IAM コンソール を開きます。
2. ナビゲーションペインで ポリシーを選択してから ポリシーの作成を選択します。
3. ポリシーの作成ページで、JSON タブを選択します。
4. JSON タブのデフォルトのコンテンツを次のアクセス許可ポリシーに置き換え、次へ: タグ を選択します。

> **Note**
>
> CloudWatch 複合アラームに関する情報を返すには、DescribeAlarms API オペレーションに、ここに示すように * アクセス許可を割り当てる必要があります。`DescribeAlarms` のアクセス許可の範囲が狭い場合、複合アラームに関する情報を返すことはできません。

```
{
        "Version": "2012-10-17",
```

```
        "Statement": [
            {
                "Effect": "Allow",
                "Action": [
                    "cloudwatch:DescribeAlarms"
                ],
                "Resource": "*"
            }
        ]
    }
```

5. このロールのタグを入力し、次へ: 確認 を選択します。
6. 確認 ページで、名前フィールドに「**SSMCloudWatchAlarmDiscoveryPolicy**」を入力します。
7. ポリシーの作成を選択します。システムによってポリシーページに戻ります。

ステップ 2: CloudWatch アラームに基づいてロールバック用の IAM ロールを作成する

次の手順を使用して、IAM ロールを作成し、前の手順で作成したポリシーをそのロールに割り当てます。

CloudWatch アラームに基づいてロールバック用の IAM ロールを作成するには

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションペインで ロール を選択し、続いて ロールを作成する を選択します。
3. 信頼されたエンティティの種類を選択 の下で、AWS サービス ( ) を選択します。
4. このロールを使用するサービスを選択 のすぐ下で、EC2: お客様に代わって EC2 インスタンスが AWS サービスを呼び出すことができるようにするを選択し、次へ: アクセス許可 を選択します。
5. アタッチされたアクセス許可ポリシー ページで、SSMCloudWatchAlarmDiscoveryPolicy を検索します。
6. このポリシーを選択し、次へ: タグ を選択します。
7. このロールのタグを入力し、次へ: 確認 を選択します。
8. ロールの作成ページで、ロール名 フィールドに「**SSMCloudWatchAlarmDiscoveryRole**」を入力し、ロールの作成 を選択します。
9. ロール ページで、作成したロールを選択します。概要 ページが開きます。

ステップ 3: 信頼関係を追加する

次の手順を使用して、先ほど作成したロールが AWS AppConfigを信頼するように設定します。

の信頼関係を追加するには AWS AppConfig

1. 作成したロールの 概要 ページで 信頼関係 タブを選択し、信頼関係の編集 を選択します。
2. 次の例に示すように、「`appconfig.amazonaws.com`」のみを含めるようにポリシーを編集します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "appconfig.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

3. 信頼ポリシーの更新 を選択します。

で機能フラグとフリーフォーム設定データを作成する AWS AppConfig

このセクションのトピックは、 AWS AppConfig内にある以下のタスクを実行するのに役立ちます。これらのタスクでは設定データのデプロイにおいて重要なアーティファクトが作成されます。

1. アプリケーションの名前空間を作成する

   アプリケーション名前空間を作成するには、アプリケーションと呼ばれる AWS AppConfig アーティファクトを作成します。アプリケーションはフォルダのようなシンプルな組織構造です。

2. 環境の作成

   AWS AppConfig アプリケーションごとに、1 つ以上の環境を定義します。環境は、 `Beta`または `Production`環境内のアプリケーションなど、 AWS AppConfig ターゲットの論理デプロイグループです。アプリケーションの `AWS Lambda functions`、`Containers`、`Web`、`Mobile` および `Back-end` といったコンポーネントを含む、アプリケーションのサブコンポーネントの環境を定義することもできます。

   環境ごとにAmazon CloudWatchアラームを設定して、問題のある設定変更を自動的にロールバックできます。システムは、設定のデプロイ中にアラームをモニタリングします。アラームがトリガーされると、システムは設定をロールバックします。

3. 設定プロファイルの作成

   設定データは、アプリケーションの動作に影響する設定のコレクションです。設定プロファイルには、 がその保存場所に設定データを配置 AWS AppConfig できるようにする URI と設定タイプが含まれます。 は次のタイプの設定プロファイル AWS AppConfig をサポートします。

   - 機能フラグ: 機能フラグを使用して、アプリケーション内の機能を有効または無効にしたり、フラグ属性を使用してアプリケーション機能のさまざまな特性を設定したりできます。 は、AWS AppConfig ホストされた設定ストアに、フラグとフラグ属性に関するデータとメタデータを含む機能フラグ形式で機能フラグ設定 AWS AppConfig を保存します。機能フラグ設定の URI は、シンプルに `hosted` です。
   - フリーフォーム設定: フリーフォーム設定では、次のいずれかの AWS のサービス および Systems Manager ツールにデータを保存できます。
     - AWS AppConfig ホストされた設定ストア
     - Amazon Simple Storage Service

- AWS CodePipeline
- AWS Secrets Manager
- AWS Systems Manager (SSM) パラメータストア
- SSM ドキュメントストア

> **Note**
>
> 可能な限り、ホストされた設定ストアで設定データをホストすることをお勧めします。ホスト AWS AppConfig された設定ストアは、ほとんどの機能と機能強化を提供します。

4. (オプションですが推奨) マルチバリアント機能フラグを作成する

   AWS AppConfig は基本的な機能フラグを提供します。この機能フラグは (有効になっている場合) リクエストごとに特定の設定データのセットを返します。ユーザーセグメンテーションとトラフィック分割のユースケースをより適切にサポートするために、 には、リクエストに対して返す可能性のあるフラグ値のセットを定義できるマルチバリアント機能フラグ AWS AppConfig も用意されています。マルチバリアントフラグには、異なるステータス (有効または無効) を設定することもできます。バリアントで設定されたフラグをリクエストする場合、アプリケーションは一連のユーザー定義ルールに対して AWS AppConfig を評価するコンテキストを提供します。リクエストで指定されたコンテキストとバリアントに定義されたルールに応じて、 は異なるフラグ値をアプリケーションに AWS AppConfig 返します。

トピック

- 設定プロファイル IAM ロールの概要
- AWS AppConfigでアプリケーションの名前空間を作成します。
- AWS AppConfigでのアプリケーション環境を作成します
- AWS AppConfigで設定プロファイルを作成します。

設定プロファイル IAM ロールの概要

AWS AppConfigを使用して、設定データへのアクセスを提供する IAM ロールを作成できます。または自分でIAM ロールを作成します。を使用してロールを作成する場合 AWS AppConfig、システムはロールを作成し、選択した設定ソースのタイプに応じて、次のいずれかのアクセス許可ポリシーを指定します。

設定ソースは Secrets Manager シークレットです

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue"
             ],
            "Resource": [
                "arn:aws:secretsmanager:AWS #####:account_ID:secret:secret_name-a1b2c3"
            ]
        }
    ]
}
```

設定ソースはパラメータストアパラメータ

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ssm:GetParameter"
            ],
            "Resource": [
                "arn:aws:ssm:AWS #####:account_ID:parameter/parameter_name"
            ]
        }
    ]
    }
```

設定ソースは SSM ドキュメント

```
{

    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
```

```
                "ssm:GetDocument"
            ],
            "Resource": [
                "arn:aws:ssm:AWS #####:account_ID:document/document_name"
            ]
        }
    ]
}
```

を使用してロールを作成すると AWS AppConfig、システムはロールに対して次の信頼関係も作成します。

```
{

  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "appconfig.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

AWS AppConfigでアプリケーションの名前空間を作成します。

このセクションの手順は、アプリケーションと呼ばれる AWS AppConfig アーティファクトを作成するのに役立ちます。アプリケーションは組織構造であり、アプリケーションの名前空間を識別するフォルダのようなものです。この組織構造は、実行可能なコードの単位と関係しています。例えば、MyMobileApp というアプリケーションを作成して、ユーザーがインストールしたモバイルアプリケーションの構成データを整理および管理できます。 AWS AppConfig を使用して機能フラグまたはフリーフォーム設定データをデプロイおよび取得する前に、これらのアーティファクトを作成する必要があります。

次の手順では、拡張機能を機能フラグ設定プロファイルに関連付けるオプションを提供します。拡張機能は、設定を作成またはデプロイする AWS AppConfig ワークフローのさまざまな時点でロジックまたは動作を挿入する機能を強化します。詳細については、「AWS AppConfig 拡張機能について」を参照してください。

> **Note**
>
> AWS CloudFormation を使用して、アプリケーション、環境、設定プロファイル、デプロイ、デプロイ戦略、ホスト設定バージョンなどの AWS AppConfig アーティファクトを作成できます。詳細については、「AWS CloudFormation ユーザーガイド」の「AWS AppConfig リソースタイプのリファレンス」を参照してください。

トピック

- AWS AppConfig アプリケーションの作成 (コンソール)
- AWS AppConfig アプリケーションの作成 (コマンドライン)

AWS AppConfig アプリケーションの作成 (コンソール)

コンソールを使用して AWS AppConfig アプリケーションを作成するには、 AWS Systems Manager 次の手順に従います。

アプリケーションを作成する方法

1. https://console.aws.amazon.com/systems-manager/appconfig/ で AWS Systems Manager コンソールを開きます。
2. ナビゲーションペインで、[アプリケーション] を選択し、[アプリケーションを作成] を選択します。
3. 名前 に、アプリケーションの名前を入力します。
4. 説明 に、アプリケーションに関する情報を入力します。
5. (オプション) [拡張機能] セクションで、リストから拡張機能を選択します。詳細については、「AWS AppConfig 拡張機能について」を参照してください。
6. (オプション) [タグ] セクションで、キーとオプションの値を入力します。1 つのリソースに対して最大 50 個のタグを指定できます。
7. アプリケーションの作成 を選択します。

AWS AppConfig はアプリケーションを作成し、環境タブを表示します。AWS AppConfigでのアプリケーション環境を作成します に進みます。

AWS AppConfig アプリケーションの作成 (コマンドライン)

次の手順では、 AWS CLI (Linux または Windows の場合) または AWS Tools for PowerShell を使用して AWS AppConfig アプリケーションを作成する方法について説明します。

アプリケーションをステップバイステップで作成するには

1. を開きます AWS CLI。
2. アプリケーションを作成するには、次のコマンドを実行します。

   Linux

   ```
   aws appconfig create-application \
     --name A_name_for_the_application \
     --description A_description_of_the_application \
     --tags User_defined_key_value_pair_metadata_for_the_application
   ```

   Windows

   ```
   aws appconfig create-application ^
     --name A_name_for_the_application ^
     --description A_description_of_the_application ^
     --tags User_defined_key_value_pair_metadata_for_the_application
   ```

   PowerShell

   ```
   New-APPCApplication `
     -Name Name_for_the_application `
     -Description Description_of_the_application `
     -Tag Hashtable_type_user_defined_key_value_pair_metadata_for_the_application
   ```

   システムが以下のような情報をレスポンスします。

   Linux

   ```
   {
      "Id": "Application ID",
      "Name": "Application name",
      "Description": "Description of the application"
   ```

```
}
```

Windows

```
{
   "Id": "Application ID",
   "Name": "Application name",
   "Description": "Description of the application"
}
```

PowerShell

```
ContentLength    : Runtime of the command
Description      : Description of the application
HttpStatusCode   : HTTP Status of the runtime
Id               : Application ID
Name             : Application name
ResponseMetadata : Runtime Metadata
```

AWS AppConfigでのアプリケーション環境を作成します

AWS AppConfig アプリケーションごとに、1 つ以上の環境を定義します。環境は、 `Beta` または `Production`環境のアプリケーション、 AWS Lambda 関数、コンテナなど、AppConfig ターゲットの論理デプロイグループです。`Web`、`Mobile`、および `Back-end` など、アプリケーションのサブコンポーネントの環境を定義することもできます。 各環境に対して Amazon CloudWatch のアラームを設定できます。システムは、設定のデプロイ中にアラームをモニタリングします。アラームがトリガーされると、システムは設定をロールバックします。

開始する前に

が CloudWatch アラームに応答して設定 AWS AppConfig をロールバックできるようにする場合は、が CloudWatch アラームに AWS AppConfig 応答できるようにするアクセス許可を持つ AWS Identity and Access Management (IAM) ロールを設定する必要があります。このロールは、次の手順で選択します。詳細については、「(推奨) 自動ロールバックのアクセス許可を設定する」を参照してください。

トピック

- AWS AppConfig 環境の作成 (コンソール)

- AWS AppConfig 環境の作成 (コマンドライン)

AWS AppConfig 環境の作成 (コンソール)

コンソールを使用して AWS AppConfig 環境を作成するには、 AWS Systems Manager 次の手順に従います。

環境を作成する方法

1. https://console.aws.amazon.com/systems-manager/appconfig/ で AWS Systems Manager コンソールを開きます。
2. ナビゲーションペインで、[アプリケーション] を選択し、アプリケーションの名前を選択して詳細ページを開きます。
3. [環境] タブ、[環境の作成] の順に選択します。
4. 名前 に、環境の名前を入力します。
5. 説明 に、環境に関する情報を入力します。
6. (オプション) Monitors セクションで、IAM ロールフィールドを選択し、アラームをモニタリングするメトリクス`cloudwatch:DescribeAlarms`で を呼び出すアクセス許可を持つ IAM ロールを選択します。
7. CloudWatch アラームリストに、モニタリングする Amazon リソースネーム (ARNs) の 1 つ以上のメトリクスを入力します。これらのメトリクスのいずれかが `ALARM`状態になった場合は、設定デプロイを AWS AppConfig ロールバックします。推奨されるメトリクスの詳細については、「自動ロールバックのためのデプロイのモニタリング」を参照してください。
8. (オプション) [拡張機能の関連付け] セクションで、リストから拡張機能を選択します。詳細については、「AWS AppConfig 拡張機能について」を参照してください。
9. (オプション) [タグ] セクションで、キーとオプションの値を入力します。1 つのリソースに対して最大 50 個のタグを指定できます。
10. 環境の作成を選択します。

AWS AppConfig は環境を作成し、環境の詳細ページを表示します。AWS AppConfigで設定プロファイルを作成します。 に進みます。

AWS AppConfig 環境の作成 (コマンドライン)

次の手順では、 AWS CLI (Linux または Windows) または AWS Tools for PowerShell を使用して AWS AppConfig 環境を作成する方法について説明します。

環境をステップバイステップで作成する

1. を開きます AWS CLI。
2. 以下のコマンドを実行して、環境を作成します。

   Linux

```
aws appconfig create-environment \
  --application-id The_application_ID \
  --name A_name_for_the_environment \
  --description A_description_of_the_environment \
  --monitors
 "AlarmArn=ARN_of_the_Amazon_CloudWatch_alarm,AlarmArnRole=ARN_of_the_IAM
 role_for_AWS AppConfig_to_monitor_AlarmArn" \
  --tags User_defined_key_value_pair_metadata_of_the_environment
```

   Windows

```
aws appconfig create-environment ^
  --application-id The_application_ID ^
  --name A_name_for_the_environment ^
  --description A_description_of_the_environment ^
  --monitors
 "AlarmArn=ARN_of_the_Amazon_CloudWatch_alarm,AlarmArnRole=ARN_of_the_IAM
 role_for_AWS AppConfig_to_monitor_AlarmArn" ^
  --tags User_defined_key_value_pair_metadata_of_the_environment
```

   PowerShell

```
New-APPCEnvironment `
  -Name Name_for_the_environment `
  -ApplicationId The_application_ID
  -Description Description_of_the_environment `
  -Monitors
 @{"AlarmArn=ARN_of_the_Amazon_CloudWatch_alarm,AlarmArnRole=ARN_of_the_IAM
 role_for_AWS AppConfig_to_monitor_AlarmArn"} `
```

```
  -Tag Hashtable_type_user_defined_key_value_pair_metadata_of_the_environment
```

システムが以下のような情報をレスポンスします。

Linux

```
{
   "ApplicationId": "The application ID",
   "Id": "The_environment ID",
   "Name": "Name of the environment",
   "State": "The state of the environment",
   "Description": "Description of the environment",

   "Monitors": [
      {
         "AlarmArn": "ARN of the Amazon CloudWatch alarm",
         "AlarmRoleArn": "ARN of the IAM role for AppConfig to monitor AlarmArn"
      }
   ]
}
```

Windows

```
{
   "ApplicationId": "The application ID",
   "Id": "The environment ID",
   "Name": "Name of the environment",
   "State": "The state of the environment"
   "Description": "Description of the environment",

   "Monitors": [
      {
         "AlarmArn": "ARN of the Amazon CloudWatch alarm",
         "AlarmRoleArn": "ARN of the IAM role for AppConfig to monitor AlarmArn"
      }
   ]
}
```

PowerShell

```
ApplicationId     : The application ID
```

```
ContentLength     : Runtime of the command
Description       : Description of the environment
HttpStatusCode    : HTTP Status of the runtime
Id                : The environment ID
Monitors          : {ARN of the Amazon CloudWatch alarm, ARN of the IAM role for
 AppConfig to monitor AlarmArn}
Name              : Name of the environment
Response Metadata : Runtime Metadata
State             : State of the environment
```

AWS AppConfigで設定プロファイルを作成します。 に進みます。

AWS AppConfigで設定プロファイルを作成します。

設定データは、アプリケーションの動作に影響する設定のコレクションです。設定プロファイルには、 がその保存場所に設定データを配置 AWS AppConfig できるようにする URI と設定タイプが含まれます。 は次のタイプの設定プロファイル AWS AppConfig をサポートします。

- 機能フラグ: 機能フラグを使用して、アプリケーション内の機能を有効または無効にしたり、フラグ属性を使用してアプリケーション機能のさまざまな特性を設定したりできます。 は、 AWS AppConfig ホストされた設定ストアに、フラグとフラグ属性に関するデータとメタデータを含む機能フラグ形式で機能フラグ設定 AWS AppConfig を保存します。機能フラグ設定の URI は、シンプルに `hosted` です。
- フリーフォーム設定: フリーフォーム設定では、次のいずれかの AWS のサービス および Systems Manager ツールにデータを保存できます。
  - AWS AppConfig ホストされた設定ストア
  - Amazon Simple Storage Service
  - AWS CodePipeline
  - AWS Secrets Manager
  - AWS Systems Manager (SSM) パラメータストア
  - SSM ドキュメントストア

> **Note**
>
> 可能な限り、ホストされた設定ストアで設定データをホストすることをお勧めします。ホスト AWS AppConfig された設定ストアは、ほとんどの機能と機能強化を提供します。

さまざまなタイプの設定データと、機能フラグで使用する方法、または設定プロファイルを使用しない方法を理解するのに役立つ設定データサンプルをいくつか紹介します。

機能フラグ設定データ

次の機能フラグ設定データは、モバイル決済とデフォルト決済を地域ごとに有効または無効にします。

JSON

```
{
  "allow_mobile_payments": {
    "enabled": false
  },
  "default_payments_per_region": {
    "enabled": true
  }
}
```

YAML

```
---
allow_mobile_payments:
  enabled: false
default_payments_per_region:
  enabled: true
```

オペレーション設定データ

次のフリーフォームの設定データでは、アプリケーションのリクエスト処理方法に制限が課されます。

JSON

```
{
```

```
  "throttle-limits": {
    "enabled": "true",
    "throttles": [
      {
        "simultaneous_connections": 12
      },
      {
        "tps_maximum": 5000
      }
    ],
    "limit-background-tasks": [
      true
    ]
  }
}
```

YAML

```
---
throttle-limits:
  enabled: 'true'
  throttles:
  - simultaneous_connections: 12
  - tps_maximum: 5000
  limit-background-tasks:
  - true
```

アクセス制御リスト設定データ

以下のアクセス制御リストのフリーフォーム設定データでは、アプリケーションにアクセスできるユーザーまたはグループを指定します。

JSON

```
{
  "allow-list": {
    "enabled": "true",
    "cohorts": [
      {
        "internal_employees": true
      },
      {
```

```
        "beta_group": false
      },
      {
        "recent_new_customers": false
      },
      {
        "user_name": "Jane_Doe"
      },
      {
        "user_name": "John_Doe"
      }
    ]
  }
}
```

YAML

```
---
allow-list:
  enabled: 'true'
  cohorts:
  - internal_employees: true
  - beta_group: false
  - recent_new_customers: false
  - user_name: Jane_Doe
  - user_name: Ashok_Kumar
```

トピック

- で機能フラグ設定プロファイルを作成する AWS AppConfig
- での無料フォーム設定プロファイルの作成 AWS AppConfig
- 非ネイティブデータソースの設定プロファイルの作成

で機能フラグ設定プロファイルを作成する AWS AppConfig

機能フラグを使用して、アプリケーション内の機能を有効または無効にしたり、フラグ属性を使用してアプリケーション機能のさまざまな特性を設定したりできます。 は、フラグとフラグ属性に関するデータとメタデータを含む機能フラグ形式で、 AWS AppConfig ホストされた設定ストアに機能フラグ設定 AWS AppConfig を保存します。

> **Note**
>
> 機能フラグ設定プロファイルを作成するときは、設定プロファイルワークフローの一部として基本的な機能フラグを作成できます。 は、マルチバリアント機能フラグ AWS AppConfig もサポートしています。マルチバリアント機能フラグを使用すると、リクエストに対して返す可能性のあるフラグ値のセットを定義できます。バリアントで設定されたフラグをリクエストする場合、アプリケーションは一連のユーザー定義ルールに対して AWS AppConfig を評価するコンテキストを提供します。リクエストで指定されたコンテキストとバリアントに定義されたルールに応じて、 は異なるフラグ値をアプリケーションに AWS AppConfig 返します。
> マルチバリアント機能フラグを作成するには、まず設定プロファイルを作成し、次に設定プロファイル内のフラグを編集してバリアントを追加します。詳細については、「マルチバリアント機能フラグの作成」を参照してください。

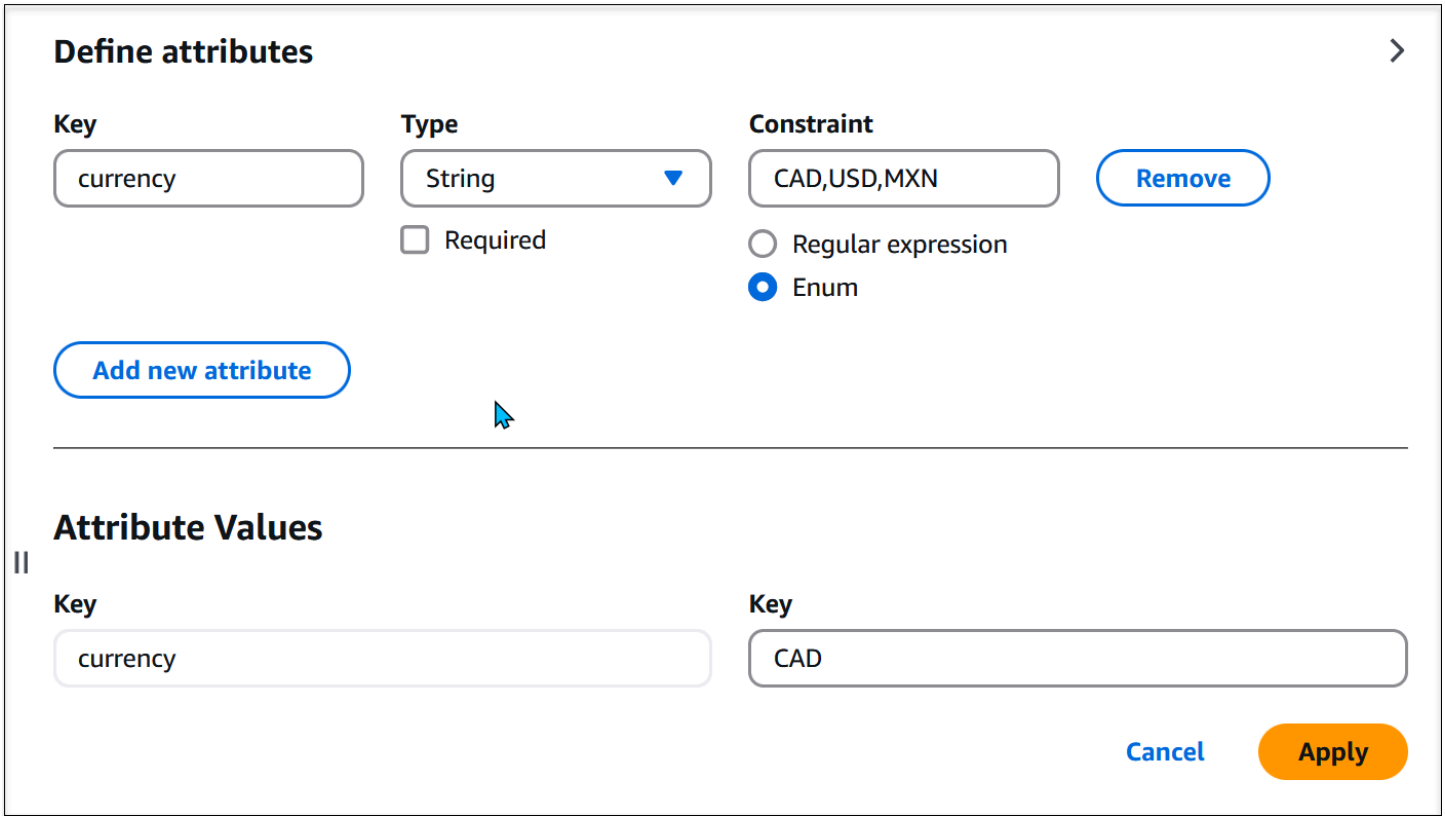
トピック

- 機能フラグ属性について
- 機能フラグ設定プロファイルの作成 (コンソール)
- 機能フラグ設定プロファイルの作成 (コマンドライン)
- マルチバリアント機能フラグの作成
- のタイプリファレンスについて AWS.AppConfig.FeatureFlags
- 以前の機能フラグバージョンを新しいバージョンに保存

機能フラグ属性について

機能フラグ設定プロファイルを作成する場合、または既存の設定プロファイル内に新しいフラグを作成する場合、フラグの属性と対応する制約を指定できます。属性は、機能フラグに関連するプロパティを表すために、機能フラグに関連付けるフィールドです。属性は、フラグキーとフラグの `enable` または `disable` の値を使用してアプリケーションに配信されます。

制約により、予期しない属性値がアプリケーションにデプロイされないようにします。次のイメージは例を示しています。



> **Note**
>
> フラグ属性に関する以下の情報に注意してください。
>
> - 属性名には、「有効」という単語を残します。「有効」という名前の機能フラグ属性は作成できません。他の予約語はありません。
> - 機能フラグの属性は、このフラグが有効になっている場合にのみ `GetLatestConfiguration` レスポンスに含まれます。
> - 特定のフラグのフラグ属性キーは一意である必要があります。

AWS AppConfig は、次のタイプのフラグ属性とそれに対応する制約をサポートします。

タイプ	制約事項	説明
文字列	正規表現	文字列の正規表現パターン
	列挙型	文字列に許容される値のリスト

タイプ	制約事項	説明
数値	最小値	属性の最小数値
	最大	属性の最大数値
ブール	なし	なし
文字列配列	正規表現	配列の要素の正規表現パターン
	列挙型	配列の要素に許容される値のリスト
数値配列	最小値	配列の要素の最小数値
	最大	配列の要素の最大数値

機能フラグ設定プロファイルの作成 (コンソール)

AWS AppConfig コンソールを使用して AWS AppConfig 機能フラグ設定プロファイルを作成するには、次の手順に従います。設定プロファイルを作成するときに、基本的な機能フラグも作成できます。

設定プロファイルを作成する方法

1. https://console.aws.amazon.com/systems-manager/appconfig/ で AWS Systems Manager コンソールを開きます。
2. ナビゲーションペインで、[アプリケーション] を選択し、AWS AppConfigでアプリケーションの名前空間を作成します。 で作成したアプリケーションを選択します。
3. [設定プロファイルと機能フラグ] タブで、[設定を作成] を選択します。
4. [設定オプション] セクションで、[機能フラグ] を選択します。
5. [設定プロファイル] セクションで、[設定プロファイル名] に名前を入力します。
6. (オプション) [説明] を展開し、説明を入力します。
7. (オプション) [追加オプション] を展開し、必要に応じて以下を完了します。

    a. 暗号化リストで、リストから AWS Key Management Service （ AWS KMS) キーを選択します。このカスタマーマネージドキーを使用すると、 AWS AppConfig ホストされた設定スト

アで新しい設定データバージョンを暗号化できます。このキーの詳細については、「のセキュリティ AWS AppConfig」の「AWS AppConfig はカスタマーマネージャーキーをサポートします」を参照してください。

b. [タグ] セクションで、[新しいタグを追加] を選択し、キーとオプションの値を指定します。

8. [次へ] を選択します。
9. [機能フラグ定義] セクションの [フラグ名] に名前を入力します。
10. [フラグキー] には、フラグ識別子を入力して、同じ設定プロファイル内のフラグを区別します。同じ設定プロファイル内のフラグに同じキーを設定することはできません。フラグを作成した後、フラグ名は編集できますが、フラグキーは編集できません。
11. (オプション) [説明] を展開し、このフラグに関する情報を入力します。
12. Select これは短期フラグであり、オプションでフラグを無効化または削除する必要がある日付を選択します。 AWS AppConfig は非推奨日にフラグを無効にしません。
13. (オプション) [機能フラグ属性] セクションで、[属性を定義] を選択します。属性を使用すると、フラグ内に追加の値を指定できます。属性と制約の詳細については、「機能フラグ属性について」を参照してください。

    a. [キー] には、フラグキーを指定し、[タイプ] リストからそのタイプを選択します。[値] および [制約] フィールドでサポートされているオプションについては、属性に関する前述のセクションを参照してください。

    b. 必須の値 を選択して、属性値が必須かどうかを指定します。

    c. 属性を追加するには、[属性を定義] を選択します。

14. [機能フラグ値] セクションで、[有効] を選択してフラグを有効にします。該当する場合、同じトグルを使用して、指定された廃止日に達したときにフラグを無効にします。
15. [次へ] を選択します。
16. [確認して保存] ページで、フラグの詳細を確認してから [保存してデプロイを続ける] を選択します。

AWS AppConfigへの機能フラグと設定データのデプロイ に進みます。

機能フラグ設定プロファイルの作成 (コマンドライン)

次の手順では、 AWS Command Line Interface (Linux または Windows) または Tools for Windows PowerShell を使用して AWS AppConfig 機能フラグ設定プロファイルを作成する方法について説明します。設定プロファイルを作成するときに、基本的な機能フラグも作成できます。

機能フラグの設定を作成する

1. を開きます AWS CLI。
2. タイプ に `AWS.AppConfig.FeatureFlags` を指定して、機能フラグの設定プロファイルを作成します。設定プロファイルでは、ロケーション URI に `hosted` を使用する必要があります。

   Linux

   ```
   aws appconfig create-configuration-profile \
     --application-id APPLICATION_ID \
     --name CONFIGURATION_PROFILE_NAME \
     --location-uri hosted \
     --type AWS.AppConfig.FeatureFlags
   ```

   Windows

   ```
   aws appconfig create-configuration-profile ^
     --application-id APPLICATION_ID ^
     --name CONFIGURATION_PROFILE_NAME ^
     --location-uri hosted ^
     --type AWS.AppConfig.FeatureFlags
   ```

   PowerShell

   ```
   New-APPCConfigurationProfile `
     -Name CONFIGURATION_PROFILE_NAME `
     -ApplicationId APPLICATION_ID `
     -LocationUri hosted `
     -Type AWS.AppConfig.FeatureFlags
   ```

3. 機能フラグの設定データを作成します。データは JSON 形式であり、`AWS.AppConfig.FeatureFlags` JSON スキーマに準拠していることが必要です。スキーマの詳細については、「のタイプリファレンスについて AWS.AppConfig.FeatureFlags」を参照してください。
4. `CreateHostedConfigurationVersion` API を使用して、機能フラグの設定データを AWS AppConfigに保存します。

Linux

```
aws appconfig create-hosted-configuration-version \
  --application-id APPLICATION_ID \
  --configuration-profile-id CONFIGURATION_PROFILE_ID \
  --content-type "application/json" \
  --content file://path/to/feature_flag_configuration_data.json \
  --cli-binary-format raw-in-base64-out
```

Windows

```
aws appconfig create-hosted-configuration-version ^
  --application-id APPLICATION_ID ^
  --configuration-profile-id CONFIGURATION_PROFILE_ID ^
  --content-type "application/json" ^
  --content file://path/to/feature_flag_configuration_data.json ^
  --cli-binary-format raw-in-base64-out
```

PowerShell

```
New-APPCHostedConfigurationVersion `
  -ApplicationId APPLICATION_ID `
  -ConfigurationProfileId CONFIGURATION_PROFILE_ID `
  -ContentType "application/json" `
  -Content file://path/to/feature_flag_configuration_data.json
```

コマンドは、Content パラメータに指定されたコンテンツをディスクからロードします。コンテンツは次の例のようになります。

```
{
    "flags": {
        "ui_refresh": {
            "name": "UI Refresh"
        }
    },
    "values": {
        "ui_refresh": {
            "enabled": false,
            "attributeValues": {
                "dark_mode_support": true
```

```
            }
        }
    },
    "version": "1"
}
```

システムが以下のような情報をレスポンスします。

Linux

```
{
   "ApplicationId"          : "ui_refresh",
   "ConfigurationProfileId" : "UI Refresh",
   "VersionNumber"          : "1",
   "ContentType"            : "application/json"
}
```

Windows

```
{
   "ApplicationId"          : "ui_refresh",
   "ConfigurationProfileId" : "UI Refresh",
   "VersionNumber"          : "1",
   "ContentType"            : "application/json"
}
```

PowerShell

```
ApplicationId          : ui_refresh
ConfigurationProfileId : UI Refresh
VersionNumber          : 1
ContentType            : application/json
```

`service_returned_content_file` には、 AWS AppConfig 生成されたメタデータを含む設定データが含まれています。

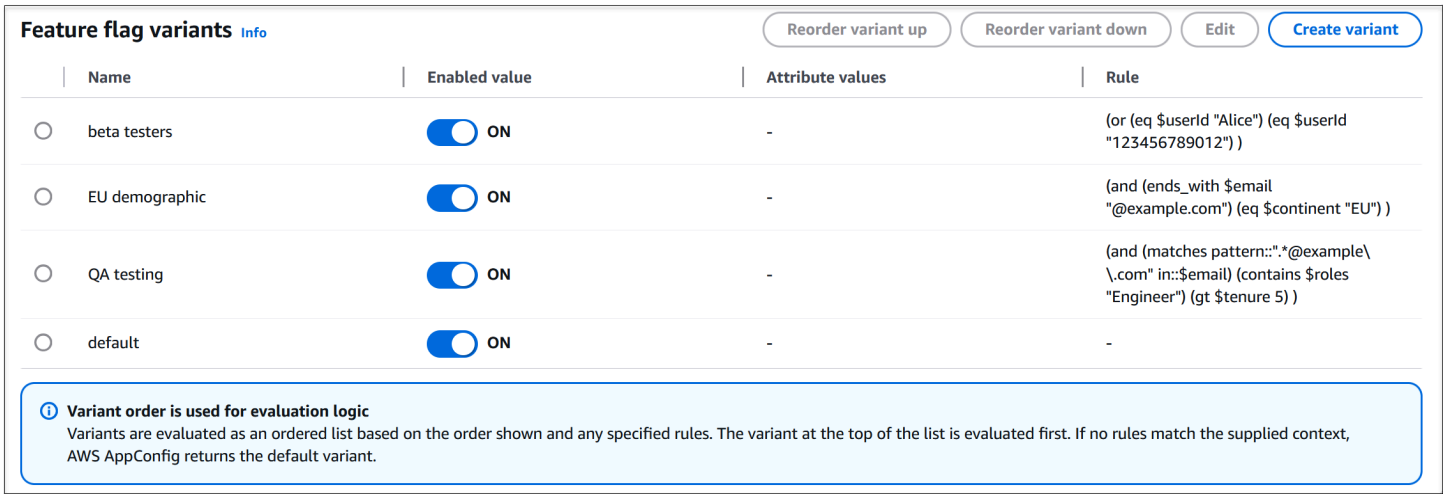
> **Note**
>
> ホストされた設定バージョンを作成すると、 AWS AppConfig はデータが JSON `AWS.AppConfig.FeatureFlags` スキーマに準拠していることを確認します。 AWS

AppConfig さらに、 は、データ内の各機能フラグ属性が、これらの属性に定義した制約を満たしていることを確認します。

マルチバリアント機能フラグの作成

機能フラグバリアントを使用すると、リクエストに対して返す可能性のあるフラグ値のセットを定義できます。マルチバリアントフラグには、異なるステータス (有効または無効) を設定することもできます。バリアントで設定されたフラグをリクエストする場合、アプリケーションは一連のユーザー定義ルールに対して AWS AppConfig を評価するコンテキストを提供します。リクエストで指定されたコンテキストとバリアントに定義されたルールに応じて、 は異なるフラグ値をアプリケーションに AWS AppConfig 返します。

次のスクリーンショットは、3 つのユーザー定義バリアントとデフォルトのバリアントを持つ機能フラグの例を示しています。



トピック

- マルチバリアント機能フラグの概念と一般的なユースケースを理解する
- マルチバリアント機能フラグルールについて
- マルチバリアント機能フラグの作成

マルチバリアント機能フラグの概念と一般的なユースケースを理解する

機能フラグバリアントをよりよく理解するために、このセクションでは、フラグバリアントの概念と一般的なユースケースについて説明します。

概念

- 機能フラグ: アプリケーションの機能の動作を制御するために使用される AWS AppConfig 設定タイプ。フラグには、ステータス (有効または無効) と、任意の文字列、数値、ブール値、または配列値を含むオプションの属性セットがあります。
- 機能フラグバリアント: 機能フラグに属するステータス値と属性値の特定の組み合わせ。機能フラグには複数のバリアントがある場合があります。
- バリアントルール: 機能フラグバリアントを選択するために使用されるユーザー定義の式。各バリアントには、それを返すかどうかを決定するために が AWS AppConfig 評価する独自のルールがあります。
- デフォルトバリアント: 他のバリアントが選択されていない場合に返される特別なバリアント。すべてのマルチバリアント機能フラグにはデフォルトのバリアントがあります。

  デフォルトのバリアントは、バリアントの順序で最後に設定する必要があり、ルールを関連付けることはできません。最後に定義されていない場合、マルチバリアントフラグを作成しようとする`BadRequestException`と、 は AWS AppConfig を返します。
- コンテキスト: 設定の取得時に AWS AppConfig に渡されるユーザー定義のキーと値。コンテキスト値は、ルール評価中に、返す機能フラグバリアントを選択するために使用されます。

> **Note**
>
> AWS AppConfig エージェントはバリアントルールを評価し、指定されたコンテキストに基づいてリクエストに適用されるルールを決定します。マルチバリアント機能フラグの取得の詳細については、「」を参照してください基本およびマルチバリアント機能フラグの取得。

一般的なユースケース

このセクションでは、機能フラグバリアントの 2 つの一般的なユースケースについて説明します。

ユーザーセグメンテーション

ユーザーセグメンテーションは、特定の属性に基づいてユーザーを分割するプロセスです。例えば、フラグバリアントを使用して、ユーザー ID、地理的位置、デバイスタイプ、または購入頻度に基づいて、一部のユーザーには機能を公開し、他のユーザーには公開しないようにすることができます。

購入頻度の例を使用して、コマースアプリケーションが顧客ロイヤルティを向上させる機能をサポートしているとします。フラグバリアントを使用すると、ユーザーが最後に何かを購入した日時に基づいて、ユーザーに表示されるさまざまなインセンティブタイプを設定できます。新規ユーザーには顧

客になることを促すために小さな割引を提供し、リピーターの顧客には新しいカテゴリから何かを購入するとより大きな割引を提供することができます。

トラフィック分割

トラフィック分割とは、定義したコンテキスト値に基づいて、ランダムでありながら一貫性のあるフラグバリアントを選択するプロセスです。例えば、少数のユーザー (ユーザー ID で識別) に特定のバリアントを表示する実験を実行することができます。または、ロールアウト全体で一貫したユーザーエクスペリエンスを維持しながら、機能が最初にユーザーの 5%、次に 15%、次に 40%、次に 100% に公開される段階的な機能ロールアウトを実行することもできます。

実験例を使用すると、フラグバリアントを使用して、アプリケーションのホームページで主なアクションの新しいボタンスタイルをテストし、クリック数が増えるかどうかを確認できます。実験では、トラフィック分割ルールを使用してフラグバリアントを作成し、5% のユーザーを選択して新しいスタイルを表示できます。デフォルトのバリアントは、既存のスタイルを表示し続ける必要があるユーザーを示します。実験が成功した場合、パーセンテージ値を増やすことも、そのバリアントをデフォルトにすることもできます。

マルチバリアント機能フラグルールについて

機能フラグバリアントを作成するときは、そのバリアントのルールを指定します。ルールは、コンテキスト値を入力として受け取り、ブール値を出力として生成する式です。例えば、アカウント ID で識別されるベータユーザーのフラグバリアントを選択するルールを定義して、ユーザーインターフェイスの更新をテストできます。このシナリオでは、次の操作を行います。

1. UI Refresh という新しい機能フラグ設定プロファイルを作成します。
2. ui_refresh という新しい機能フラグを作成します。
3. バリアントを追加するには、作成後に機能フラグを編集します。
4. BetaUsers という新しいバリアントを作成して有効にします。
5. リクエストコンテキストのアカウント ID が、新しいベータエクスペリエンスの表示が承認されたアカウント ID のリストにある場合にバリアントを選択する BetaUsers のルールを定義します。
6. デフォルトのバリアントのステータスが Disabled に設定されていることを確認します。

> **Note**
> バリアントは、コンソールで定義されている順序に基づいて順序付きリストとして評価されます。リストの先頭にあるバリアントが最初に評価されます。指定されたコンテキストに一致するルールがない場合、 はデフォルトのバリアント AWS AppConfig を返します。

が機能フラグリクエスト AWS AppConfig を処理すると、最初に AccountID (この例の場合) を含む指定されたコンテキストを BetaUsers バリアントと比較します。コンテキストが BetaUsers のルールと一致する場合、 はベータエクスペリエンスの設定データを AWS AppConfig 返します。コンテキストにアカウント ID が含まれていない場合、またはアカウント ID が 123 以外にある場合、 はデフォルトルールの設定データを AWS AppConfig 返します。これは、ユーザーが本番環境で現在のエクスペリエンスを表示することを意味します。

> **Note**
> マルチバリアント機能フラグの取得については、「」を参照してください基本およびマルチバリアント機能フラグの取得。

マルチバリアント機能フラグのルールの定義

バリアントルールは、1 つ以上のオペランドと演算子で構成される式です。オペランドは、ルールの評価中に使用される特定の値です。オペランド値は、リテラル数値や文字列などの静的値、またはコンテキストで見つかった値や別の式の結果などの変数のいずれかにすることができます。「より大きい」などの演算子は、値を生成するオペランドに適用されるテストまたはアクションです。バリアントルール式を有効にするには、「true」または「false」のいずれかを生成する必要があります。

オペランド

タイプ	説明	例
String	UTF-8 文字のシーケンス。二重引用符で囲まれています。	`"apple", "###ë# ##š##"`
整数	64 ビットの整数値。	`-7, 42`

タイプ	説明	例
浮動小数点数	64 ビットの IEEE-754 浮動小数点値。	`3.14, 1.234e-5`
Timestamp	日付と時刻の形式に関する W3C ノートで説明されている特定の時刻。	`2012-03-04T05:06:07-08:00, 2024-01`
ブール値	true または false 値。	`true, false`
コンテキスト値	ルール評価中にコンテキストから取得される $*key* 形式のパラメータ化された値。	`$country, $userId`

比較演算子

演算子	説明	例
eq	コンテキスト値が指定された値と等しいかどうかを判断します。	`(eq $state "Virginia")`
gt	コンテキスト値が指定された値より大きいかどうかを判断します。	`(gt $age 65)`
gte	コンテキスト値が指定された値以上かどうかを判断します。	`(gte $age 65)`
lt	コンテキスト値が指定された値よりも小さいかどうかを判断します。	`(lt $age 65)`

演算子	説明	例
lte	コンテキスト値が指定された値以下かどうかを判断します。	`(lte $age 65)`

論理演算子

演算子	説明	例
and	両方のオペランドが true かどうかを判断します。	`(and (eq $state "Virginia") (gt $age 65) )`
or	少なくとも 1 つのオペランドが true かどうかを判断します。	`(or (eq $state "Virginia") (gt $age 65) )`
not	式の値を反転します。	`(not (eq $state "Virginia"))`

カスタム演算子

演算子	説明	例
begins_with	コンテキスト値が特定のプレフィックスで始まるかどうかを判断します。	`(begins_with $state "A")`

演算子	説明	例
ends_with	コンテキスト値が特定のプレフィックスで終わるかどうかを判断します。	`(ends_with $email "amazon.com")`
contains	コンテキスト値に特定の部分文字列が含まれているかどうかを判断します。	`(contains $promoCode "WIN")`
in	コンテキスト値が定数のリストに含まれるかどうかを判断します。	`(in $userId ["123", "456"])`
matches	コンテキスト値が特定の正規表現パターンと一致するかどうかを判断します。	`(matches in::$greeting pattern::"h.*y")`
exists	コンテキストキーに値が提供されたかどうかを判断します。	`(exists key::"country")`

演算子	説明	例
split	指定されたコンテキスト値の一貫したハッシュ (複数可) に基づいて、トラフィックの特定の割合について true と評価します。のsplit仕組みの詳細については、このトピックの次のセクション「」を参照してください分割演算子について。 seed はオプションのプロパティであることに注意してください。seed を指定しない場合、ハッシュはローカルで一貫しており、そのフラグに対してトラフィックは一貫して分割されますが、同じコンテキスト値を受け取る他のフラグはトラフィックを異なる方法で分割する可能性があります。seed が指定されている場合、各一意の値によって、機能フラグ、設定プロファイル、および AWS アカウント間でトラフィックが一貫して分割されることが保証されます。	`(split pct::10 by::$userId seed::"abc")`

分割演算子について

次のセクションでは、さまざまなシナリオで使用した場合のsplitオペレータの動作について説明します。リマインダーとして、 は指定されたコンテキスト値の一貫したハッシュに基づいて、特定の割合のトラフィックtrueについて splitを評価します。これをよりよく理解するには、 を 2 つのバリアントで分割する次のベースラインシナリオを検討してください。

```
A: (split by::$uniqueId pct::20)
C: <no rule>
```

予想どおり、uniqueId値のランダムなセットを指定すると、次のようなディストリビューションが生成されます。

```
A: 20%
C: 80%
```

3 番目のバリアントを追加しても、次のように同じ分割パーセンテージを使用する場合：

```
A: (split by::$uniqueId pct::20)
B: (split by::$uniqueId pct::20)
C: <default>
```

最終的に次のディストリビューションになります。

```
A: 20%
B: 0%
C: 80%
```

この予期しないディストリビューションが発生する可能性があるのは、各バリアントルールが順番に評価され、最初の一致によって返されるバリアントが決定されるためです。ルール A が評価されると、uniqueId値の 20% がそれと一致するため、最初のバリアントが返されます。次に、ルール B が評価されます。ただし、2 番目の分割ステートメントに一致するすべてのuniqueId値はバリアントルール A によって既に一致しているため、B に一致する値はありません。代わりにデフォルトのバリアントが返されます。

次に、3 番目の例を考えてみましょう。

```
A: (split by::$uniqueId pct::20)
B: (split by::$uniqueId pct::25)
C: <default>
```

前の例と同様に、uniqueId値の最初の 20% はルール A に一致します。バリアントルール B の場合、すべてのuniqueId値の 25% は一致しますが、以前に一致したルール A のほとんどは一致しま

す。これにより、バリアント B の合計の 5% が残り、残りはバリアント C を受け取ります。ディストリビューションは次のようになります。

```
A: 20%
B: 5%
C: 75%
```

seed プロパティの使用

seed プロパティを使用すると、分割演算子が使用されている場所に関係なく、特定のコンテキスト値に対してトラフィックが一貫して分割されるようにできます。seed を指定しない場合、ハッシュはローカルで一貫しており、そのフラグに対してトラフィックは一貫して分割されますが、同じコンテキスト値を受け取る他のフラグはトラフィックを異なる方法で分割する可能性があります。seed が指定されている場合、各一意の値によって、機能フラグ、設定プロファイル、および AWS アカウント間でトラフィックが一貫して分割されることが保証されます。

通常、お客様は、同じコンテキストプロパティでトラフィックを分割するときに、フラグ内のバリアント間で同じseed値を使用します。ただし、別のシード値を使用するのが理にかなっている場合があります。ルール A と B に異なるシードを使用する例を次に示します。

```
A: (split by::$uniqueId pct::20 seed::"seed_one")
B: (split by::$uniqueId pct::25 seed::"seed_two")
C: <default>
```

前述のように、一致するuniqueId値の 20% がルール A に一致します。つまり、値の 80% がフォールスルーされ、バリアントルール B に対してテストされます。シードが異なるため、A に一致する値と B に一致する値の間に相関はありません。ただし、分割するuniqueId値は 80% のみで、その数に一致するルール B の 25% と 75% はそうではありません。これは次のディストリビューションに当てはまります。

```
A: 20%
B: 20% (25% of what falls through from A, or 25% of 80%)
C: 60%
```

マルチバリアント機能フラグの作成

このセクションの手順を使用して、機能フラグのバリアントを作成します。

[開始する前に]

次の重要な情報に注意してください。

- 既存の機能フラグを編集してバリアントを作成できます。新しい設定プロファイルを作成するときに、新しい機能フラグのバリアントを作成することはできません。まず、新しい設定プロファイルを作成するワークフローを完了する必要があります。設定プロファイルを作成したら、設定プロファイル内の任意のフラグにバリアントを追加できます。新しい設定プロファイルを作成する方法については、「で機能フラグ設定プロファイルを作成する AWS AppConfig」を参照してください。
- Amazon EC2、Amazon ECS、Amazon EKS コンピューティングプラットフォームの機能フラグバリアントデータを取得するには、 AWS AppConfig エージェントバージョン 2.0.4416 以降を使用する必要があります。
- パフォーマンス上の理由から、 AWS CLI SDK は を呼び出してバリアントデータを取得 AWS AppConfig しません。 AWS AppConfig エージェントの詳細については、「」を参照してくださいAWS AppConfig エージェントを使用して設定データを取得する方法。
- 機能フラグバリアントを作成するときは、そのバリアントのルールを指定します。ルールは、リクエストコンテキストを入力として受け取り、ブール結果を出力として生成する式です。バリアントを作成する前に、フラグバリアントルールでサポートされているオペランドと演算子を確認してください。バリアントを作成する前にルールを作成できます。詳細については、「マルチバリアント機能フラグルールについて」を参照してください。

トピック

- マルチバリアント機能フラグの作成 (コンソール)
- マルチバリアント機能フラグの作成 (コマンドライン)

マルチバリアント機能フラグの作成 (コンソール)

次の手順では、 AWS AppConfig コンソールを使用して既存の設定プロファイルのマルチバリアント機能フラグを作成する方法について説明します。既存の機能フラグを編集してバリアントを作成することもできます。

マルチバリアント機能フラグを作成するには

1. https://console.aws.amazon.com/systems-manager/appconfig/ で AWS Systems Manager コンソールを開きます。
2. ナビゲーションペインで、[アプリケーション] を選択し、アプリケーションを選択します。
3. [設定プロファイルと機能フラグ] タブで、既存の機能フラグ設定プロファイルを選択します。
4. [フラグ] セクションで、[新しいフラグの追加] を選択します。

5. [機能フラグ定義] セクションの [フラグ名] に名前を入力します。
6. [フラグキー] には、フラグ識別子を入力して、同じ設定プロファイル内のフラグを区別します。同じ設定プロファイル内のフラグに同じキーを設定することはできません。フラグを作成した後、フラグ名は編集できますが、フラグキーは編集できません。
7. (オプション) [説明] フィールドに、このフラグに関する情報を入力します。
8. [バリアント] セクションで、[マルチバリアントフラグ] を選択します。
9. (オプション) [機能フラグ属性] セクションで、[属性を定義] を選択します。属性を使用すると、フラグ内に追加の値を指定できます。属性と制約の詳細については、「機能フラグ属性について」を参照してください。

   a. [キー] には、フラグキーを指定し、[タイプ] リストからそのタイプを選択します。[値] および [制約] フィールドでサポートされているオプションについては、属性に関する前述のセクションを参照してください。
   b. 必須の値 を選択して、属性値が必須かどうかを指定します。
   c. 属性を追加するには、[属性を定義] を選択します。
   d. 属性の変更を保存するには、[適用] を選択します。
10. [機能フラグバリアント] セクションで、[バリアントを作成] を選択します。

    a. [バリアント名] に名前を入力します。
    b. [有効化された値] トグルを使用してバリアントを有効にします。
    c. [ルール] テキストボックスにルールを入力します。
    d. [バリアントを作成] > [上にバリアントを作成] または [下にバリアントを作成] オプションを使用して、このフラグに追加のバリアントを作成します。
    e. [デフォルトバリアント] セクションで、[有効化された値] トグルを使用してデフォルトバリアントを有効にします。必要に応じて、ステップ 10 で定義された属性の値を指定します。
    f. [Apply] (適用) を選択します。
11. フラグとそのバリアントの詳細を確認し、[フラグを作成] を選択します。

新規の機能フラグとバリアントをデプロイする方法の詳細については、「AWS AppConfigへの機能フラグと設定データのデプロイ」を参照してください。

マルチバリアント機能フラグの作成 (コマンドライン)

次の手順では、 AWS Command Line Interface (Linux または Windows) または Tools for Windows PowerShell を使用して、既存の設定プロファイルのマルチバリアント機能フラグを作成する方法について説明します。既存の機能フラグを編集してバリアントを作成することもできます。

[開始する前に]

AWS CLIを使用して多変量機能フラグを作成する前に、以下のタスクを完了してください。

- 機能フラグ設定プロファイルを作成します。詳細については、「で機能フラグ設定プロファイルを作成する AWS AppConfig」を参照してください。
- AWS CLIを最新バージョンに更新します。詳細については、「AWS Command Line Interface ユーザーガイド」の「AWS CLIの最新バージョンをインストールまたは更新する」を参照してください。

マルチバリアント機能フラグを作成するには

1. 作成するマルチバリアントフラグの詳細を指定する設定ファイルをローカルマシンに作成します。ファイル拡張子 (`.json`) を付けてこのファイルを保存します。ファイルは `AWS.AppConfig.FeatureFlags` JSON スキーマに準拠している必要があります。設定ファイルのスキーマの内容は、次のようになります。

```
{
  "flags": {
    "FLAG_NAME": {
      "attributes": {
          "ATTRIBUTE_NAME": {
          "constraints": {
            "type": "CONSTRAINT_TYPE"
          }
        }
      },
      "description": "FLAG_DESCRIPTION",
      "name": "VARIANT_NAME"
    }
  },
  "values": {
    "VARIANT_VALUE_NAME": {
      "_variants": [
        {
```

```
          "attributeValues": {
            "ATTRIBUTE_NAME": BOOLEAN
          },
          "enabled": BOOLEAN,
          "name": "VARIANT_NAME",
          "rule": "VARIANT_RULE"
        },
        {
          "attributeValues": {
            "ATTRIBUTE_NAME": BOOLEAN
          },
          "enabled": BOOLEAN,
          "name": "VARIANT_NAME",
          "rule": "VARIANT_RULE"
        },
        {
          "attributeValues": {
            "ATTRIBUTE_NAME": BOOLEAN
          },
          "enabled": BOOLEAN,
          "name": "VARIANT_NAME",
          "rule": "VARIANT_RULE"
        },
        {
          "attributeValues": {
            "ATTRIBUTE_NAME": BOOLEAN
          },
          "enabled": BOOLEAN,
          "name": "VARIANT_NAME",
          "rule": "VARIANT_RULE"
        }
      ]
    }
  },
  "version": "VERSION_NUMBER"
}
```

以下は、3 つのバリアントとデフォルトのバリアントの例です。

```
{
  "flags": {
    "ui_refresh": {
      "attributes": {
```

```
        "dark_mode_support": {
          "constraints": {
            "type": "boolean"
          }
        }
      },
      "description": "A release flag used to release a new UI",
      "name": "UI Refresh"
    }
  },
  "values": {
    "ui_refresh": {
      "_variants": [
        {
          "attributeValues": {
            "dark_mode_support": true
          },
          "enabled": true,
          "name": "QA",
          "rule": "(ends_with $email \"qa-testers.mycompany.com\")"
        },
        {
          "attributeValues": {
            "dark_mode_support": true
          },
          "enabled": true,
          "name": "Beta Testers",
          "rule": "(exists key::\"opted_in_to_beta\")"
        },
        {
          "attributeValues": {
            "dark_mode_support": false
          },
          "enabled": true,
          "name": "Sample Population",
          "rule": "(split pct::10 by::$email)"
        },
        {
          "attributeValues": {
            "dark_mode_support": false
          },
          "enabled": false,
          "name": "Default Variant"
        }
```

```
      ]
    }
  },
  "version": "1"
}
```

2. `CreateHostedConfigurationVersion` API を使用して、機能フラグの設定データを AWS AppConfigに保存します。

   Linux

   ```
   aws appconfig create-hosted-configuration-version \
     --application-id APPLICATION_ID \
     --configuration-profile-id CONFIGURATION_PROFILE_ID \
     --content-type "application/json" \
     --content file://path/to/feature_flag_configuration_data.json \
     --cli-binary-format raw-in-base64-out \
     outfile
   ```

   Windows

   ```
   aws appconfig create-hosted-configuration-version ^
     --application-id APPLICATION_ID ^
     --configuration-profile-id CONFIGURATION_PROFILE_ID ^
     --content-type "application/json" ^
     --content file://path/to/feature_flag_configuration_data.json ^
     --cli-binary-format raw-in-base64-out ^
     outfile
   ```

   PowerShell

   ```
   New-APPCHostedConfigurationVersion `
     -ApplicationId APPLICATION_ID `
     -ConfigurationProfileId CONFIGURATION_PROFILE_ID `
     -ContentType "application/json" `
     -Content file://path/to/feature_flag_configuration_data.json `
     -Raw
   ```

   `service_returned_content_file` には、 AWS AppConfig 生成されたメタデータを含む設定データが含まれています。

> **Note**
>
> ホストされた設定バージョンを作成すると、 AWS AppConfig はデータが `AWS.AppConfig.FeatureFlags` JSON スキーマに準拠していることを確認します。 AWS AppConfig さらに、 は、データ内の各機能フラグ属性が、それらの属性に定義した制約を満たしていることを確認します。

のタイプリファレンスについて AWS.AppConfig.FeatureFlags

`AWS.AppConfig.FeatureFlags` JSON スキーマを、機能フラグの設定データを作成するためのリファレンスとして使用します。

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "definitions": {
    "flagSetDefinition": {
      "type": "object",
      "properties": {
        "version": {
          "$ref": "#/definitions/flagSchemaVersions"
        },
        "flags": {
          "$ref": "#/definitions/flagDefinitions"
        },
        "values": {
          "$ref": "#/definitions/flagValues"
        }
      },
      "required": ["version"],
      "additionalProperties": false
    },
    "flagDefinitions": {
      "type": "object",
      "patternProperties": {
        "^[a-z][a-zA-Z\\d_-]{0,63}$": {
          "$ref": "#/definitions/flagDefinition"
        }
      },
      "additionalProperties": false
    },
```

```
    "flagDefinition": {
      "type": "object",
      "properties": {
        "name": {
          "$ref": "#/definitions/customerDefinedName"
        },
        "description": {
          "$ref": "#/definitions/customerDefinedDescription"
        },
        "_createdAt": {
          "type": "string"
        },
        "_updatedAt": {
          "type": "string"
        },
        "_deprecation": {
          "type": "object",
          "properties": {
            "status": {
              "type": "string",
              "enum": ["planned"]
            },
            "date": {
              "type": "string",
              "format": "date"
            }
          },
         "additionalProperties": false
        },
        "attributes": {
          "$ref": "#/definitions/attributeDefinitions"
        }
      },
      "additionalProperties": false
    },
    "attributeDefinitions": {
      "type": "object",
      "patternProperties": {
        "^[a-z][a-zA-Z\\d_-]{0,63}$": {
          "$ref": "#/definitions/attributeDefinition"
        }
      },
      "maxProperties": 25,
      "additionalProperties": false
```

```
    },
    "attributeDefinition": {
      "type": "object",
      "properties": {
        "description": {
          "$ref": "#/definitions/customerDefinedDescription"
        },
        "constraints": {
          "oneOf": [
            { "$ref": "#/definitions/numberConstraints" },
            { "$ref": "#/definitions/stringConstraints" },
            { "$ref": "#/definitions/arrayConstraints" },
            { "$ref": "#/definitions/boolConstraints" }
          ]
        }
      },
      "additionalProperties": false
    },
    "flagValues": {
      "type": "object",
      "patternProperties": {
        "^[a-z][a-zA-Z\\d_-]{0,63}$": {
          "$ref": "#/definitions/flagValue"
        }
      },
      "additionalProperties": false
    },
    "flagValue": {
      "type": "object",
      "properties": {
        "enabled": {
          "type": "boolean"
        },
        "_createdAt": {
          "type": "string"
        },
        "_updatedAt": {
          "type": "string"
        },
        "_variants": {
          "type": "array",
          "maxLength": 32,
          "items": {
            "$ref": "#/definitions/variant"
```

```
          }
        }
      },
      "patternProperties": {
        "^[a-z][a-zA-Z\\d_-]{0,63}$": {
          "$ref": "#/definitions/attributeValue",
          "maxProperties": 25
        }
      },
      "additionalProperties": false
    },
    "attributeValue": {
      "oneOf": [
        { "type": "string", "maxLength": 1024 },
        { "type": "number" },
        { "type": "boolean" },
        {
          "type": "array",
          "oneOf": [
            {
              "items": {
                "type": "string",
                "maxLength": 1024
              }
            },
            {
              "items": {
                "type": "number"
              }
            }
          ]
        }
      ],
      "additionalProperties": false
    },
    "stringConstraints": {
      "type": "object",
      "properties": {
        "type": {
          "type": "string",
          "enum": ["string"]
        },
        "required": {
          "type": "boolean"
```

```
        },
        "pattern": {
          "type": "string",
          "maxLength": 1024
        },
        "enum": {
          "type": "array",
          "maxLength": 100,
          "items": {
            "oneOf": [
              {
                "type": "string",
                "maxLength": 1024
              },
              {
                "type": "integer"
              }
            ]
          }
        }
      },
      "required": ["type"],
      "not": {
        "required": ["pattern", "enum"]
      },
      "additionalProperties": false
    },
    "numberConstraints": {
      "type": "object",
      "properties": {
        "type": {
          "type": "string",
          "enum": ["number"]
        },
        "required": {
          "type": "boolean"
        },
        "minimum": {
          "type": "integer"
        },
        "maximum": {
          "type": "integer"
        }
      },
```

```
      "required": ["type"],
      "additionalProperties": false
    },
    "arrayConstraints": {
      "type": "object",
      "properties": {
        "type": {
          "type": "string",
          "enum": ["array"]
        },
        "required": {
          "type": "boolean"
        },
        "elements": {
          "$ref": "#/definitions/elementConstraints"
        }
      },
      "required": ["type"],
      "additionalProperties": false
    },
    "boolConstraints": {
      "type": "object",
      "properties": {
        "type": {
          "type": "string",
          "enum": ["boolean"]
        },
        "required": {
          "type": "boolean"
        }
      },
      "required": ["type"],
      "additionalProperties": false
    },
    "elementConstraints": {
      "oneOf": [
        { "$ref": "#/definitions/numberConstraints" },
        { "$ref": "#/definitions/stringConstraints" }
      ]
    },
    "variant": {
      "type": "object",
      "properties": {
        "enabled": {
```

```
          "type": "boolean"
        },
        "name": {
          "$ref": "#/definitions/customerDefinedName"
        },
        "rule": {
          "type": "string",
          "maxLength": 16384
        },
        "attributeValues": {
          "type": "object",
          "patternProperties": {
            "^[a-z][a-zA-Z\\d_-]{0,63}$": {
              "$ref": "#/definitions/attributeValue"
            }
          },
          "maxProperties": 25,
          "additionalProperties": false
        }
      },
      "required": ["name", "enabled"],
      "additionalProperties": false
    },
    "customerDefinedName": {
      "type": "string",
      "pattern": "^[^\\n]{1,64}$"
    },
    "customerDefinedDescription": {
      "type": "string",
      "maxLength": 1024
    },
    "flagSchemaVersions": {
      "type": "string",
      "enum": ["1"]
    }
  },
  "type": "object",
  "$ref": "#/definitions/flagSetDefinition",
  "additionalProperties": false
}
```

> **Important**
>
> 機能フラグ設定データを取得するには、アプリケーションが GetLatestConfiguration API を呼び出す必要があります。GetConfiguration は廃止されたため、呼び出しても機能フラグ設定データを取得できません。詳細については、AWS AppConfig API リファレンスの GetLatestConfiguration を参照してください。

アプリケーションが GetLatestConfiguration を呼び出して、新しくデプロイされた設定を受信すると、機能フラグと属性を定義する情報が削除されます。簡略化された JSON には、指定された各フラグキーに一致するキーマップが含まれています。簡略化された JSON には、enabled 属性の true または false のマッピング値も含まれています。フラグが 、enabledからtrueに設定されている場合、そのフラグの属性も存在することになる。この JSON スキーマは、JSON 出力の形式を記述します。

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "type": "object",
  "patternProperties": {
    "^[a-z][a-zA-Z\\d_-]{0,63}$": {
      "$ref": "#/definitions/attributeValuesMap"
    }
  },
  "maxProperties": 100,
  "additionalProperties": false,
  "definitions": {
    "attributeValuesMap": {
      "type": "object",
      "properties": {
        "enabled": {
          "type": "boolean"
        }
      },
      "required": ["enabled"],
      "patternProperties": {
        "^[a-z][a-zA-Z\\d_-]{0,63}$": {
          "$ref": "#/definitions/attributeValue"
        }
      },
      "maxProperties": 25,
      "additionalProperties": false
```

```
    },
    "attributeValue": {
      "oneOf": [
        { "type": "string","maxLength": 1024 },
        { "type": "number" },
        { "type": "boolean" },
        {
          "type": "array",
          "oneOf": [
            {
              "items": {
                "oneOf": [
                  {
                    "type": "string",
                    "maxLength": 1024
                  }
                ]
              }
            },
            {
              "items": {
                "oneOf": [
                  {
                    "type": "number"
                  }
                ]
              }
            }
          ]
        }
      ],
      "additionalProperties": false
    }
  }
}
```

以前の機能フラグバージョンを新しいバージョンに保存

機能フラグを更新すると、 は変更を新しいバージョン AWS AppConfig に自動的に保存します。以前の機能フラグバージョンを使用する場合は、それをドラフトバージョンにコピーしてから保存する必要があります。以前のフラグバージョンに変更を編集して保存するには、新しいバージョンに保存する必要があります。

以前の機能フラグバージョンを編集して新しいバージョンに保存するには

1. https://console.aws.amazon.com/systems-manager/appconfig/ で AWS Systems Manager コンソールを開きます。
2. ナビゲーションペインで、アプリケーションを選択し、編集して新しいバージョンに保存する機能フラグを持つアプリケーションを選択します。
3. 設定プロファイルと機能フラグタブで、編集して新しいバージョンに保存する機能フラグを持つ設定プロファイルを選択します。
4. 機能フラグタブで、バージョンリストを使用して、編集して新しいバージョンに保存するバージョンを選択します。
5. コピーからドラフトバージョンを選択します。
6. バージョンラベルフィールドに、新しいラベルを入力します (オプション、ただし推奨) 。
7. バージョンの説明フィールドに、新しい説明を入力します (オプションですが、推奨) 。
8. バージョンの保存 を選択します。
9. デプロイの開始 を選択して、新しいバージョンをデプロイします。

での無料フォーム設定プロファイルの作成 AWS AppConfig

設定データは、アプリケーションの動作に影響する設定のコレクションです。設定プロファイルには、 がその保存場所に設定データを見つけ AWS AppConfig られるようにする URI と設定タイプが含まれます。フリーフォーム設定プロファイルを使用すると、 AWS AppConfig ホストされた設定ストア、または次のいずれかの AWS のサービス および Systems Manager ツールにデータを保存できます。

場所	サポートされているファイルの種類
AWS AppConfig ホスト設定ストア	を使用して追加した場合は、YAML、JSON、およびテキスト AWS Management Console。AWS AppConfig CreateHostedConfigurationVersion API アクションを使用して追加された場合の任意のファイルタイプ。
Amazon Simple Storage Service (Amazon S3)	いずれか
AWS CodePipeline	パイプライン (サービスによって定義される)

場所	サポートされているファイルの種類
AWS Secrets Manager	シークレット (サービスによって定義される)
AWS Systems Manager パラメータストア	標準で安全な文字列パラメータ (Parameter Store で定義される)
AWS Systems Manager ドキュメントストア (SSM ドキュメント)	YAML、JSON、テキスト

設定プロファイルには、設定データが構文的および意味的に正しいことを確認するためのオプションのバリデータを含めることもできます。 は、デプロイの開始時にバリデータを使用してチェック AWS AppConfig を実行します。エラーが検出されると、設定のターゲットに変更が加えられる前にデプロイが停止します。

> **Note**
>
> 可能な限り、ホストされた設定ストアで設定データをホストすることをお勧めします。ホスト AWS AppConfig された設定ストアは、ほとんどの機能と機能強化を提供します。

AWS AppConfig ホストされた設定ストアまたは SSM ドキュメントに保存されているフリーフォーム設定の場合、設定プロファイルの作成時に Systems Manager コンソールを使用してフリーフォーム設定を作成できます。このプロセスについては、このトピックの後半で説明します。

Parameter Store、Secrets Manager または Amazon S3 に格納されているフリーフォーム設定の場合は、まずパラメータ、シークレット、またはオブジェクトを作成してから、関連する設定ストアに格納する必要があります。設定データを保存したら、このトピックの手順を使用して設定プロファイルを作成できます。

トピック

- バリデータを理解する
- 設定ストアのクォータと制限について
- AWS AppConfig ホストされた設定ストアについて
- Amazon S3 に保存されている設定について
- AWS AppConfig フリーフォーム設定プロファイルの作成 (コンソール)
- AWS AppConfig フリーフォーム設定プロファイルの作成 (コマンドライン)

バリデータを理解する

設定プロファイルを作成する場合、最大 2 つのバリデータを指定できます。バリデータは、設定データが構文的かつ意味的に正しいことを保証します。バリデータを使用する場合は、設定プロファイルを作成する前に作成する必要があります。 は、次のタイプのバリデータ AWS AppConfig をサポートしています。

- AWS Lambda 関数: 機能フラグとフリーフォーム設定でサポートされています。
- JSON スキーマ: 無料フォーム設定でサポートされています (JSON スキーマに対して機能フラグ AWS AppConfig を自動的に検証します) 。

トピック

- AWS Lambda 関数の検証
- JSON スキーマバリデータ

AWS Lambda 関数の検証

Lambda 関数バリデータは、次のイベントスキーマで設定する必要があります。 AWS AppConfig はこのスキーマを使用して Lambda 関数を呼び出します。内容は base64 でエンコードされた文字列で、URI は文字列です。

```
{
    "applicationId": "The application ID of the configuration profile being
 validated",
    "configurationProfileId": "The ID of the configuration profile being validated",
    "configurationVersion": "The version of the configuration profile being validated",
    "content": "Base64EncodedByteString",
    "uri": "The configuration uri"
}
```

AWS AppConfig は、レスポンスで Lambda `X-Amz-Function-Error`ヘッダーが設定されていることを確認します。Lambda は、関数が例外をスローした場合にこのヘッダーを設定します。`X-Amz-Function-Error` の詳細については、「AWS Lambda デベロッパーガイド」の「エラー処理と AWS Lambdaでの自動再試行」を参照してください。

検証を成功させる Lambda 応答コードの簡単な例を以下に示します。

```
import json
```

```
def handler(event, context):
     #Add your validation logic here
     print("We passed!")
```

検証を失敗させる Lambda 応答コードの簡単な例を以下に示します。

```
def handler(event, context):
     #Add your validation logic here
     raise Exception("Failure!")
```

設定パラメータが素数の場合にのみ有効とする、別の例を次に示します。

```
function isPrime(value) {
    if (value < 2) {
        return false;
    }

    for (i = 2; i < value; i++) {
        if (value % i === 0) {
            return false;
        }
    }

    return true;
}

exports.handler = async function(event, context) {
    console.log('EVENT: ' + JSON.stringify(event, null, 2));
    const input = parseInt(Buffer.from(event.content, 'base64').toString('ascii'));
    const prime = isPrime(input);
    console.log('RESULT: ' + input + (prime ? ' is' : ' is not') + ' prime');
    if (!prime) {
        throw input + "is not prime";
    }
}
```

AWS AppConfig は、 `StartDeployment`および `ValidateConfigurationActivity` API オペレーションを呼び出すときに検証 Lambda を呼び出します。Lambda を呼び出すには、`appconfig.amazonaws.com` アクセス許可を提供する必要があります。詳細については、AWS 「サービスへの関数アクセスの付与 AWS AppConfig 」を参照してください。Lambda の実行時間の検証は、起動レイテンシーを含めて 15 秒に制限されます。

JSON スキーマバリデータ

SSM ドキュメントで設定を作成する場合は、その設定の JSON スキーマを指定または作成する必要があります。JSON スキーマは、アプリケーション構成設定ごとに許可されるプロパティを定義します。この JSON スキーマは、新規または更新された構成設定がアプリケーションに必要なベストプラクティスに準拠するようにするための一連のルールのように機能します。以下はその例です。

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "$id$",
  "description": "BasicFeatureToggle-1",
  "type": "object",
  "additionalProperties": false,
  "patternProperties": {
      "[^\\s]+$": {
          "type": "boolean"
      }
  },
  "minProperties": 1
}
```

SSM ドキュメントから設定を作成すると、その設定がスキーマの要件を満たしているかどうかをシステムが自動的に検証します。そうでない場合は、 AWS AppConfig は、検証エラーを返します。

> **Important**
>
> JSON スキーマ検証ツールについては、次の重要事項に留意してください。
>
> - SSM ドキュメントに保存された設定データは、設定をシステムに追加する前に、関連付けられた JSON スキーマに対して検証する必要があります。SSM パラメータには検証方法は必要ありませんが、 を使用して、新規または更新された SSM パラメータ設定の検証チェックを作成することをお勧めします AWS Lambda。
> - SSM ドキュメントの設定では `ApplicationConfiguration` ドキュメントタイプを使用します。対応する JSON スキーマは `ApplicationConfigurationSchema` ドキュメントタイプを使用します。
> - AWS AppConfig は、インラインスキーマの JSON スキーマバージョン 4.X をサポートしています。アプリケーション設定で JSON スキーマの異なるバージョンが必要な場合は、Lambda バリデータを作成する必要があります。

設定ストアのクォータと制限について

でサポートされている設定ストア AWS AppConfig には、次のクォータと制限があります。

	AWS AppConfig ホスト設定ストア	Amazon S3	Systems Manager Parameter Store	AWS Secrets Manager	Systems Manager ドキュメント・ストア	AWS CodePipeline
設定サイズの制限	デフォルト 2 MB、最大 4 MB	2 MB S3 ではなく AWS AppConfig によって強制される	4 KB (無料利用枠)/8 KB (詳細パラメータ)	64 KB	64 KB	2 MB CodePipeline ではなく AWS AppConfig によって強制される
リソースストレージの制限	1 GB	無制限	10,000 パラメータ (無料利用枠)/100,000 パラメータ (詳細パラメータ)	500,000	500 ドキュメント	アプリケーションごとの設定プロファイルの数によって制限される (アプリケーションあたり 100 プロファイル)
サーバー側の暗号化	はい	SSE-S3、SSE-KMS	はい	はい	なし	はい
AWS CloudFormation のサポート	はい	データの作成または更新用ではありません	はい	はい	なし	はい

	AWS AppConfig ホスト設定ストア	Amazon S3	Systems Manager Parameter Store	AWS Secrets Manager	Systems Manager ドキュメント・ストア	AWS CodePipeline
料金表	空き	「Amazon S3 の料金」を参照してください	「AWS Systems Manager 料金表」を参照してください	「AWS Secrets Manager 料金表」を参照してください	空き	「AWS CodePipeline 料金表」を参照してください

AWS AppConfig ホストされた設定ストアについて

AWS AppConfig には、内部設定ストアまたはホスト設定ストアが含まれています。設定は 2 MB 以下である必要があります。 AWS AppConfig ホストされた設定ストアには、他の設定ストアオプションよりも以下の利点があります。

- Amazon Simple Storage Service (Amazon S3) やパラメータストアなど、他のサービスをセットアップして設定する必要はありません。
- 設定ストアを使用するには AWS Identity and Access Management 、(IAM) アクセス許可を設定する必要はありません。
- 設定を YAML、JSON、またはテキストドキュメントとして保存できます。
- ストアを使用してもコストは発生しません。
- 構成プロファイルを作成したら、作成した構成をストアに追加できます。

Amazon S3 に保存されている設定について

設定は、Amazon Simple Storage Service (Amazon S3) バケットに保存できます。設定プロファイルの作成時には、バケット内の 1 つの S3 オブジェクトの URI を指定します。オブジェクトを取得するアクセス許可を付与 AWS AppConfig する AWS Identity and Access Management (IAM) ロールの Amazon リソースネーム (ARN) も指定します。Amazon S3 オブジェクトの設定プロファイルを作成する場合は、次の制限事項に注意してください。

制限	詳細
サイズ	S3 オブジェクトとして保存できる設定のサイズは最大 1 MB です。
オブジェクト暗号化	設定プロファイルは、SSE-S3 と SSE-KMS で暗号化されたオブジェクトを対象にすることができます。
ストレージクラス	AWS AppConfig は、S3 ストレージクラス `STANDARD`、、`INTELLIGENT_TIERING` `REDUCED_REDUNDANCY` 、`STANDARD_IA` 、および をサポートしています`ONEZONE_IA` 。すべての S3 Glacier クラス (`GLACIER` および `DEEP_ARCHIVE`) はサポートしていません。
バージョニング	AWS AppConfig では、S3 オブジェクトがバージョニングを使用する必要があります。

Amazon S3 オブジェクトとして保存する設定のアクセス許可の設定

S3 オブジェクトとして保存されている設定の設定プロファイルを作成するときは、オブジェクトを取得する AWS AppConfig アクセス許可を付与する IAM ロールの ARN を指定する必要があります。このロールには、以下のアクセス許可が含まれている必要があります。

S3 オブジェクトに対するアクセス許可

- s3:GetObject
- s3:GetObjectVersion

S3 オブジェクトを一覧表示するアクセス許可

s3:ListAllMyBuckets

オブジェクトを保存する S3 バケットへのアクセス許可

- s3:GetBucketLocation

- s3:GetBucketVersioning
- s3:ListBucket
- s3:ListBucketVersions

が S3 オブジェクトに保存されている設定を取得 AWS AppConfig できるようにするロールを作成するには、次の手順を実行します。

S3 オブジェクトにアクセスするための IAM ポリシーの作成

次の手順を使用して、 が S3 オブジェクトに保存されている設定 AWS AppConfig を取得できるようにする IAM ポリシーを作成します。

S3 オブジェクトにアクセスするための IAM ポリシーを作成するには

1. https://console.aws.amazon.com/iam/ で IAM コンソール を開きます。
2. ナビゲーションペインで ポリシーを選択してから ポリシーの作成を選択します。
3. ポリシーの作成 ページで、JSON タブを選択します。
4. S3 バケットおよび設定オブジェクトについての情報を、次のサンプルポリシーで更新します。次に、そのポリシーを JSON タブのテキストフィールドに貼り付けます。*#########*を、ユーザー自身の情報に置き換えます。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:GetObjectVersion"
      ],
      "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/my-configurations/my-
configuration.json"
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetBucketLocation",
        "s3:GetBucketVersioning",
        "s3:ListBucketVersions",
        "s3:ListBucket"
```

```
        ],
        "Resource": [
          "arn:aws:s3:::amzn-s3-demo-bucket"
        ]
      },
      {
        "Effect": "Allow",
        "Action": "s3:ListAllMyBuckets",
        "Resource": "*"
      }
    ]
}
```

5. [ポリシーの確認] を選択します。
6. レビューポリシー ページの 名前 ボックスに名前を入力し、続いて説明を入力します。
7. ポリシーの作成 を選択します。ロールページが再度表示されます。

S3 オブジェクトにアクセスするための IAM ロールの作成

次の手順を使用して、 が S3 オブジェクトに保存されている設定を取得 AWS AppConfig できるようにする IAM ロールを作成します。

Amazon S3 オブジェクトにアクセスするための IAM ポリシーを作成するには

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションペインで ロールを選択してから、ロールを作成する を選択します。
3. 信頼するエンティティのタイプを選択 で AWS サービス ( ) を選択します。
4. ユースケースの選択 セクションの 一般的ユースケース で EC2 を選択して、次へ: アクセス許可 を選択します。
5. アクセス権限ポリシーをアタッチする ページで、検索ボックスに前の手順で作成したポリシーの名前を入力します。
6. ポリシーを選択して、次へ: タグ を選択します。
7. タグ (オプショナル) の追加 ページでキーと任意の値を入力して、次へ: 確認 を選択します。
8. 確認 ページの ロール名 フィールドに名前を入力し、続いて説明を入力します。
9. ロールを作成するを選択します。ロールページが再度表示されます。

10. ロール ページで作成したロールを選択して、概要 ページを開きます。ロール名 と ロール ARN を書き留めます。このロール ARN は、このトピックで後述する設定プロファイルの作成時に指定します。

信頼関係の作成

次の手順を使用して、先ほど作成したロールが AWS AppConfigを信頼するように設定します。

信頼関係を追加するには

1. 作成したロールの 概要 ページで 信頼関係 タブを選択し、信頼関係の編集 を選択します。
2. 次の例に示すように、`"ec2.amazonaws.com"` を削除して `"appconfig.amazonaws.com"` を追加します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "appconfig.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

3. 信頼ポリシーの更新 を選択します。

AWS AppConfig フリーフォーム設定プロファイルの作成 (コンソール)

AWS Systems Manager コンソールを使用して AWS AppConfig 、フリーフォーム設定プロファイルと (オプションで) フリーフォーム設定を作成するには、次の手順に従います。

フリーフォーム設定プロファイルを作成するには

1. https://console.aws.amazon.com/systems-manager/appconfig/ で AWS Systems Manager コンソールを開きます。
2. ナビゲーションペインで、[アプリケーション] を選択し、AWS AppConfigでアプリケーションの名前空間を作成します。 で作成したアプリケーションを選択します。

3. [設定プロファイルと機能フラグ] タブを選択し、[設定を作成] を選択します。
4. [設定オプション] セクションで、[フリーフォーム設定] を選択します。
5. [設定プロファイル名] に設定プロファイルの名前を入力します。
6. (オプション) [説明] を展開し、説明を入力します。
7. (オプション) [追加オプション] を展開し、必要に応じて以下を完了します。
   a. [拡張機能を関連付ける] セクションで、リストから拡張機能を選択します。
   b. [タグ] セクションで、[新しいタグを追加] を選択し、キーとオプションの値を指定します。
8. [次へ] を選択します。
9. [設定データを指定する] ページの [設定の定義] セクションで、オプションを選択します。
10. 次の表の説明に従って、選択したオプションのフィールドに入力します。

選択したオプション	詳細
AWS AppConfig ホスト設定	[テキスト]、[JSON]、または [YAML] のいずれかを選択し、フィールドに設定を入力します。この手順のステップ 12 に進みます。
Amazon S3 オブジェクト	[S3 オブジェクトソース] フィールドにオブジェクト URI を入力し、この手順のステップ 11 に進みます。
AWS CodePipeline	[次へ] を選択し、この手順のステップ 12 に進みます。
Secrets Manager シークレット	リストからシークレットを選択し、この手順のステップ 11 に進みます。
AWS Systems Manager parameter	リストからパラメータを選択し、この手順のステップ 11 に進みます。
AWS Systems Manager document	1. リストからドキュメントを選択するか、[新しい文書を作成] を選択します。 2. [新しい文書を作成] を選択した場合は、[ドキュメント名] に名前を入力します。必要に応じて、[バージョン名] を展開し、ド

選択したオプション	詳細
	キュメントバージョンの名前を入力します。 3. [アプリケーション設定スキーマ] セクションで、リストを使用して JSON スキーマを選択するか、[スキーマを作成] を選択します。スキーマの作成 を選択すると、Systems Manager は スキーマの作成 ページを開きます。スキーマの詳細を入力し、[アプリケーション設定スキーマを作成] を選択します。 4. コンテンツ セクションで、YAML または JSON を選択し、フィールドに設定データを入力します。

11. サービスロールセクションで、新しいサービスロールを選択して、設定データへのアクセスを提供する IAM ロール AWS AppConfig を作成します。 は、前に入力した名前に基づいてロール名フィールド AWS AppConfig を自動的に入力します。または、[既存のサービスロール] を選択します。ロール ARN リストを使用してロールを選択します。
12. オプションで、[検証ツールを追加] ページで、[JSON スキーマ] または [AWS Lambda] を選択します。JSON スキーマ を選択した場合、フィールドに JSON スキーマを入力します。AWS Lambda を選択した場合は、リストから関数 Amazon リソースネーム (ARN) とバージョンを選択します。

> **Important**
>
> SSM ドキュメントに保存された設定データは、設定をシステムに追加する前に、関連付けられた JSON スキーマに対して検証する必要があります。SSM パラメータには検証方法は必要ありませんが、 を使用して、新規または更新された SSM パラメータ設定の検証チェックを作成することをお勧めします AWS Lambda。

13. [次へ] を選択します。
14. [確認して保存] ページで、[保存してデプロイを続ける] を選択します。

> **Important**
>
> の設定プロファイルを作成した場合は AWS CodePipeline、デプロイプロバイダー AWS AppConfig として を指定するパイプラインを CodePipeline で作成する必要があります。AWS AppConfigへの機能フラグと設定データのデプロイ を実行する必要はありません。ただし、「AWS AppConfig エージェントを使用せずに設定データを取得する」で説明されているように、アプリケーション設定の更新を受け取るようにクライアントを設定する必要があります。デプロイプロバイダー AWS AppConfig として を指定するパイプラインの作成については、 AWS CodePipeline ユーザーガイドの「チュートリアル: デプロイプロバイダー AWS AppConfig として を使用するパイプラインを作成する」を参照してください。

AWS AppConfigへの機能フラグと設定データのデプロイ に進みます。

AWS AppConfig フリーフォーム設定プロファイルの作成 (コマンドライン)

次の手順では、 AWS CLI (Linux または Windows) または AWS Tools for PowerShell を使用して AWS AppConfig フリーフォーム設定プロファイルを作成する方法について説明します。必要に応じて、 AWS CloudShell を使用して以下のコマンドを実行できます。詳細については、「 AWS CloudShellユーザーガイド」の「What is AWS CloudShell ?」(とは ?) を参照してください。

> **Note**
>
> ホストされた設定ストアで AWS AppConfig ホストされているフリーフォーム設定の場合は、場所 URI `hosted`に を指定します。

を使用して設定プロファイルを作成するには AWS CLI

1. を開きます AWS CLI。
2. 次のコマンドを実行して、フリーフォーム設定プロファイルを作成します。

   Linux

```
aws appconfig create-configuration-profile \
  --application-id APPLICATION_ID \
  --name NAME \
  --description CONFIGURATION_PROFILE_DESCRIPTION \
  --location-uri CONFIGURATION_URI or hosted \
```

```
  --retrieval-role-arn IAM_ROLE_ARN \
  --tags TAGS \
  --validators "Content=SCHEMA_CONTENT or LAMBDA_FUNCTION_ARN,Type=JSON_SCHEMA
 or LAMBDA"
```

Windows

```
aws appconfig create-configuration-profile ^
  --application-id APPLICATION_ID ^
  --name NAME ^
  --description CONFIGURATION_PROFILE_DESCRIPTION ^
  --location-uri CONFIGURATION_URI or hosted  ^
  --retrieval-role-arn IAM_ROLE_ARN ^
  --tags TAGS ^
  --validators "Content=SCHEMA_CONTENT or LAMBDA_FUNCTION_ARN,Type=JSON_SCHEMA
 or LAMBDA"
```

PowerShell

```
New-APPCConfigurationProfile `
  -Name NAME `
  -ApplicationId APPLICATION_ID `
  -Description CONFIGURATION_PROFILE_DESCRIPTION `
  -LocationUri CONFIGURATION_URI or hosted `
  -RetrievalRoleArn IAM_ROLE_ARN `
  -Tag TAGS `
  -Validators "Content=SCHEMA_CONTENT or LAMBDA_FUNCTION_ARN,Type=JSON_SCHEMA
 or LAMBDA"
```

Important

次の重要な情報に注意してください。

- の設定プロファイルを作成した場合は AWS CodePipeline、デプロイプロバイダー AWS AppConfig として を指定するパイプラインを CodePipeline で作成する必要があります。AWS AppConfigへの機能フラグと設定データのデプロイ を実行する必要はありません。ただし、「AWS AppConfig エージェントを使用せずに設定データを取得する」で説明されているように、アプリケーション設定の更新を受け取るようにクライアントを設定する必要があります。デプロイプロバイダー AWS AppConfig として を指定するパイプラインの作成については、 AWS CodePipeline ユーザーガイドの「チュートリアル: デプロイ

プロバイダー AWS AppConfig として を使用するパイプラインを作成する」を参照してください。

- AWS AppConfig ホストされた設定ストアで設定を作成した場合は、CreateHostedConfigurationVersion API オペレーションを使用して設定の新しいバージョンを作成できます。この API オペレーション AWS CLI の詳細とサンプルコマンドを表示するには、AWS CLI 「 コマンドリファレンス」のcreate-hosted-configuration-version」を参照してください。

AWS AppConfigへの機能フラグと設定データのデプロイ に進みます。

非ネイティブデータソースの設定プロファイルの作成

AWS AppConfig は、ほとんどのデータストアからの設定データのデプロイをサポートしています。ネイティブでは、 AWS AppConfig は次のサービスに保存されている設定データのデプロイをサポートしています。

- AWS AppConfig ホストされた設定ストア
- Amazon S3
- AWS Secrets Manager
- AWS Systems Manager パラメータストア
- Systems Manager ドキュメントストア
- AWS CodePipeline

設定データが でネイティブにサポートされていない場所に保存されている場合は AWS AppConfig、AWS AppConfig 拡張機能を作成して、ソースからデータを取得できます。たとえば、 拡張機能を使用すると AWS AppConfig 、Amazon Relational Database Service (Amazon RDS)、Amazon DynamoDB (DynamoDB)、GitHub、GitLab、またはローカルリポジトリに保存されている設定データを取得できます。拡張機能を実装することで、アプリケーションとコンピューティング環境 AWS AppConfig のセキュリティと DevOps の機能強化を活用できます。この方法は、設定データをレガシーシステムから に移行するときにも使用できます AWS AppConfig。

で AWS AppConfig ネイティブにサポートされていないデータソースの設定プロファイルを作成するには、以下のプロセスまたはアクションが必要です。

1. データソースからデータを取得する AWS Lambda 関数を作成します。Lambda 関数がデータソースにアクセスできる限り、 AWS AppConfig 拡張機能はデータを取得できます。

2. Lambda 関数を呼び出すカスタム AWS AppConfig 拡張機能を作成します。詳細については、「チュートリアル: カスタム AWS AppConfig 拡張機能の作成」を参照してください。
3. AWS AppConfig 自由形式の設定プロファイルを作成します。具体的には、AWS AppConfig ホストされた設定定義を使用する設定プロファイルを作成します。設定プロファイルは、Lambda 関数がソースから設定を取得した後、一時データストアとして機能します。アプリケーションは、AWS AppConfig ホストされた設定ストアから設定データを取得します。詳細については、「での無料フォーム設定プロファイルの作成 AWS AppConfig」を参照してください。
4. `PRE_CREATE_HOSTED_CONFIGURATION_VERSION` アクションポイントを使用してトリガーする拡張機能の関連付けを作成します。詳細については、「ステップ 4: カスタム拡張機能の AWS AppConfig 拡張機能関連付けを作成する」を参照してください。

設定後、アプリケーションが設定データの新しいバージョンをリクエストすると、Lambda は設定データを取得し、設定プロファイルにプルします。 AWS AppConfig 次に、 は設定プロファイルとサードパーティーデータを保存します。

準備ができたら、他のタイプの設定データと同様に、設定プロファイルをアプリケーションにデプロイできます。

> **Note**
>
> 既存の設定データに沿ってサードパーティーデータを挿入するか、設定データの内容全体にサードパーティーデータのみを含めるかを選択できます。データを他の既存のデータに合わせる場合、そのロジックはサードパーティーソースからデータをインポートする Lambda 関数の一部である必要があります。

レガシーおよび自社開発の設定サービス AWS AppConfig から への移行

の使用を開始し AWS AppConfig 、別のシステムにレガシー設定データまたは機能フラグがある場合は、このトピックで前述したプロセスを使用してレガシーシステムから移行できます AWS AppConfig。レガシーシステムからデータを取得してデプロイする拡張機能を構築できます AWS AppConfig。 AWS AppConfig この方法で を使用すると、レガシーデータストアを引き続き使用しながら、すべての安全ガードレールの制御と利点が得られます。

AWS AppConfigへの機能フラグと設定データのデプロイ

機能フラグとフリーフォームの設定データを扱うために必要なアーティファクトを作成したら、新しいデプロイを作成できるようになります。新しいデプロイを作成するときは、以下の情報を指定します。

- アプリケーション ID
- 設定プロファイル ID
- 設定バージョン。
- 設定データをデプロイする環境 ID
- 変更をどのくらいの速さで反映させたいかを定義するデプロイ戦略 ID
- カスタマーマネージドキーを使用してデータを暗号化するための AWS Key Management Service (AWS KMS) キー ID。

StartDeployment API アクションを呼び出すと、 は次のタスク AWS AppConfig を実行します。

1. 設定プロファイルのロケーション URI を使用して、基になるデータストアから設定データを取得します。
2. 設定プロファイルに作成したときに指定したバリデータを使用して、設定データが構文的にも意味論的にも正しいことを確認します。
3. データのコピーをキャッシュして、アプリケーションがすぐに取り出せるようにします。このキャッシュされたコピーはデプロイされたデータと呼ばれます。

Amazon CloudWatch アラームに基づくデプロイ戦略と自動ロールバックを組み合わせて使用することで、設定データの AWS AppConfig デプロイによってアプリケーションにエラーが発生する状況を軽減できます。デプロイ戦略を使うと、数分または数時間かけて変更を本番環境にゆっくりとリリースできます。設定後、デプロイ中に 1 つ以上の CloudWatch アラームがアラーム状態になると、 は設定データを以前のバージョン AWS AppConfig に自動的にロールバックします。デプロイ戦略の詳細については、「デプロイ戦略の使用」を参照してください。自動ロールバックの詳細については、「自動ロールバックのためのデプロイのモニタリング」を参照してください。

トピック

- デプロイ戦略の使用
- 構成のデプロイ

- CodePipeline を使用した AWS AppConfig 設定のデプロイ
- 設定の復元

デプロイ戦略の使用

デプロイ戦略を使うと、数分または数時間かけて変更を本番環境にゆっくりとリリースできます。AWS AppConfig デプロイ戦略は、設定デプロイの以下の重要な側面を定義します。

<table>
<tr><th>設定</th><th>説明</th></tr>
<tr><td>デプロイタイプ</td><td>デプロイタイプは、設定のデプロイ方法またはロールアウト方法を定義します。 は線形デプロイタイプと指数デプロイタイプ AWS AppConfig をサポートします。

• 線形: このタイプでは、 はデプロイ全体に均等に分散された増加係数の増分でデプロイ AWS AppConfig を処理します。以下に、20% の直線的成長を使用して 10 時間でデプロイのタイムラインの例を示します：

<table>
<tr><th>経過時間</th><th>デプロイの進捗状況</th></tr>
<tr><td>0 時間</td><td>0%</td></tr>
<tr><td>2 時間</td><td>20%</td></tr>
<tr><td>4 時間</td><td>40%</td></tr>
<tr><td>6 時間</td><td>60%</td></tr>
<tr><td>8 時間</td><td>80%</td></tr>
<tr><td>10 時間</td><td>100%</td></tr>
</table>
• 指数: このタイプの場合、 AWS AppConfig は次の式を使用してデプロイを指数的に処理</td></tr>
</table>

設定	説明

	します。G*(2^N)。この式で、G はユーザーによって指定されたステップパーセンテージで、N は設定がすべてのターゲットにデプロイされるまでのステップ数です。たとえば、増加係数に 2 を指定すると、次のように構成がロールアウトされます。
	
	`2*(2^0)`
	`2*(2^1)`
	`2*(2^2)`
	
	デプロイは、ターゲットの 2%、ターゲットの 4%、ターゲットの 8%、のようにロールアウトされ、設定がすべてのターゲットにデプロイされるまで続行されます。

ステップパーセンテージ (増加係数)	この設定では、デプロイの各ステップでターゲットとする発信者の割合を指定します。 **Note** SDK および AWS AppConfig API リファレンス では、`step percentage` を `growth factor` と呼んでいます。
デプロイ時間	この設定では、 がホストに AWS AppConfig デプロイする時間を指定します。これはタイムアウト値ではありません。これは、デプロイが間隔を置いて処理される時間枠です。

設定	説明
ベイク時間	この設定では、設定がターゲットの 100% にデプロイされてから、デプロイが完了したと見なされるまでに、Amazon CloudWatch アラームを AWS AppConfig モニタリングする時間を指定します。この間にアラームがトリガーされた場合、 AWS AppConfig はデプロイをロールバックします。CloudWatch アラームに基づいてロールバック AWS AppConfig するには、 のアクセス許可を設定する必要があります。詳細については、「(推奨) 自動ロールバックのアクセス許可を設定する」を参照してください。

に含まれている事前定義された戦略を選択する AWS AppConfig か、独自の戦略を作成できます。

トピック

- 定義済みのデプロイ戦略の使用
- デプロイ戦略の作成

定義済みのデプロイ戦略の使用

AWS AppConfig には、設定を迅速にデプロイするための事前定義されたデプロイ戦略が含まれています。設定をデプロイするときには、独自の戦略を作成する代わりに、次のいずれかを選択できます。

デプロイ戦略	説明
AppConfig.Linear: 6 分ごとに 20 パーセント	AWS 推奨： この戦略では、6 分ごとに設定をすべてのターゲットの20%にデプロイし、30 分間のデプロイを行います。システムは Amazon CloudWatch アラームを 30 分間監視します。この時間内にアラームが受信されなければ、デプロイは完了です。この間にアラームがトリガーされた場

デプロイ戦略	説明
	合、 はデプロイを AWS AppConfig ロールバックします。 この戦略は、 AWS ベストプラクティスに沿っており、長期間とベイク時間のため、デプロイの安全性をさらに重視しているため、本番環境のデプロイに使用することをお勧めします。
AppConfig.Canary10Percent20Minutes	AWS 推奨： この戦略では、20 分間にわたって 10% の増加係数を使用し、デプロイを指数関数的に処理します。システムは CloudWatch アラームを 10 分間監視します。この時間内にアラームが受信されなければ、デプロイは完了です。この間にアラームがトリガーされた場合、 はデプロイを AWS AppConfig ロールバックします。 この戦略は、設定デプロイの AWS ベストプラクティスと一致するため、本番環境のデプロイに使用することをお勧めします。
AppConfig.AllAtOnce	クイック: この戦略では、すべてのターゲットにただちに設定をデプロイします。システムは CloudWatch アラームを 10 分間監視します。この時間内にアラームが受信されなければ、デプロイは完了です。この間にアラームがトリガーされた場合、 AWS AppConfig はデプロイをロールバックします。

デプロイ戦略	説明
AppConfig.Linear50PercentEvery30Seconds	テスト中/デモンストレーション: この戦略では、30 秒ごとに設定をすべてのターゲットの半分にデプロイし、1 分間のデプロイを行います。システムは、Amazon CloudWatch アラームを 1 分間監視します。この時間内にアラームが受信されなければ、デプロイは完了です。この間にアラームがトリガーされた場合、 はデプロイを AWS AppConfig ロールバックします。 この戦略では、継続時間と処理時間が短いため、テストまたはデモンストレーションの目的でのみ使用することをお勧めします。

デプロイ戦略の作成

定義済みのデプロイ戦略のいずれかを使用しない場合は、独自のデプロイ戦略を作成できます。最大 20 のデプロイ戦略を作成できます。設定をデプロイするときに、アプリケーションおよび環境に最適なデプロイ戦略を選択できます。

AWS AppConfig デプロイ戦略の作成 (コンソール)

AWS Systems Manager コンソールを使用して AWS AppConfig デプロイ戦略を作成するには、次の手順に従います。

デプロイ戦略を作成するには

1. AWS Systems Manager コンソールを https://console.aws.amazon.com/systems-manager/appconfig/://https://https://https://https://https
2. ナビゲーションペインで、[デプロイ戦略] タブを選択し、[デプロイ戦略を作成] を選択します。
3. 名前 に、デプロイ戦略の名前を入力します。
4. 説明 に、デプロイ戦略に関する情報を入力します。
5. デプロイタイプ で、タイプを選択します。

6. ステップパーセンテージ で、デプロイの各ステップでターゲットとする発信者の割合を選択します。
7. デプロイ時間 に、デプロイの合計期間を分または時間単位で入力します。
8. ベイク時間 には、デプロイの次のステップに進む前、またはデプロイを完了とみなす前に Amazon CloudWatch アラームをモニタリングする合計時間を分または時間単位で入力します。
9. タグ セクションで、キーとオプションの値を入力します。1 つのリソースに対して最大 50 個のタグを指定できます。
10. デプロイ戦略の作成 を選択します。

> **Important**
>
> の設定プロファイルを作成した場合は AWS CodePipeline、デプロイプロバイダー AWS AppConfig として を指定するパイプラインを CodePipeline で作成する必要があります。構成のデプロイ を実行する必要はありません。ただし、「AWS AppConfig エージェントを使用せずに設定データを取得する」で説明されているように、アプリケーション設定の更新を受け取るようにクライアントを設定する必要があります。をデプロイプロバイダー AWS AppConfig として指定するパイプラインの作成については、「 ユーザーガイド」の「チュートリアル: デプロイプロバイダー AWS AppConfig として を使用するパイプラインの作成」を参照してください。 AWS CodePipeline

構成のデプロイ に進みます。

AWS AppConfig デプロイ戦略(コマンドライン)の作成

次の手順では、 AWS CLI (Linux または Windows の場合) または AWS Tools for PowerShell を使用して AWS AppConfig デプロイ戦略を作成する方法について説明します。

デプロイ戦略をステップバイステップで作成するには

1. を開きます AWS CLI。
2. 以下のコマンドを実行して、デプロイ戦略を作成します。

   Linux

```
aws appconfig create-deployment-strategy \
  --name A_name_for_the_deployment_strategy \
  --description A_description_of_the_deployment_strategy \
```

```
  --deployment-duration-in-minutes Total_amount_of_time_for_a_deployment_to_last
 \
  --final-bake-time-in-minutes Amount_of_time_AWS
 AppConfig_monitors_for_alarms_before_considering_the_deployment_to_be_complete
 \
  --growth-
factor The_percentage_of_targets_to_receive_a_deployed_configuration_during_each_interva
 \
  --growth-
type The_linear_or_exponential_algorithm_used_to_define_how_percentage_grows_over_time
 \
  --replicate-
to To_save_the_deployment_strategy_to_a_Systems_Manager_(SSM)_document \
  --tags User_defined_key_value_pair_metadata_of_the_deployment_strategy
```

Windows

```
aws appconfig create-deployment-strategy ^
  --name A_name_for_the_deployment_strategy ^
  --description A_description_of_the_deployment_strategy ^
  --deployment-duration-in-minutes Total_amount_of_time_for_a_deployment_to_last
 ^
  --final-bake-time-in-minutes Amount_of_time_AWS
 AppConfig_monitors_for_alarms_before_considering_the_deployment_to_be_complete
 ^
  --growth-
factor The_percentage_of_targets_to_receive_a_deployed_configuration_during_each_interva
 ^
  --growth-
type The_linear_or_exponential_algorithm_used_to_define_how_percentage_grows_over_time
 ^
  --name A_name_for_the_deployment_strategy ^
  --replicate-
to To_save_the_deployment_strategy_to_a_Systems_Manager_(SSM)_document ^
  --tags User_defined_key_value_pair_metadata_of_the_deployment_strategy
```

PowerShell

```
New-APPCDeploymentStrategy `
  --Name A_name_for_the_deployment_strategy `
  --Description A_description_of_the_deployment_strategy `
  --DeploymentDurationInMinutes Total_amount_of_time_for_a_deployment_to_last `
```

```
  --FinalBakeTimeInMinutes Amount_of_time_AWS
 AppConfig_monitors_for_alarms_before_considering_the_deployment_to_be_complete
 `
  --
GrowthFactor The_percentage_of_targets_to_receive_a_deployed_configuration_during_each_i
 `
  --
GrowthType The_linear_or_exponential_algorithm_used_to_define_how_percentage_grows_over_
 `
  --
ReplicateTo To_save_the_deployment_strategy_to_a_Systems_Manager_(SSM)_document
 `
  --
Tag Hashtable_type_User_defined_key_value_pair_metadata_of_the_deployment_strategy
```

システムが以下のような情報をレスポンスします。

Linux

```
{
   "Id": "Id of the deployment strategy",
   "Name": "Name of the deployment strategy",
   "Description": "Description of the deployment strategy",
   "DeploymentDurationInMinutes": "Total amount of time the deployment lasted",
   "GrowthType": "The linear or exponential algorithm used to define how
 percentage grew over time",
   "GrowthFactor": "The percentage of targets that received a deployed
 configuration during each interval",
   "FinalBakeTimeInMinutes": "The amount of time AWS AppConfig monitored for
 alarms before considering the deployment to be complete",
   "ReplicateTo": "The Systems Manager (SSM) document where the deployment
 strategy is saved"
}
```

Windows

```
{
   "Id": "Id of the deployment strategy",
   "Name": "Name of the deployment strategy",
   "Description": "Description of the deployment strategy",
   "DeploymentDurationInMinutes": "Total amount of time the deployment lasted",
```

```
  "GrowthType": "The linear or exponential algorithm used to define how
 percentage grew over time",
  "GrowthFactor": "The percentage of targets that received a deployed
 configuration during each interval",
  "FinalBakeTimeInMinutes": "The amount of time AWS AppConfig monitored for
 alarms before considering the deployment to be complete",
  "ReplicateTo": "The Systems Manager (SSM) document where the deployment
 strategy is saved"
}
```

PowerShell

```
ContentLength                : Runtime of the command
DeploymentDurationInMinutes : Total amount of time the deployment lasted
Description                  : Description of the deployment strategy
FinalBakeTimeInMinutes       : The amount of time AWS AppConfig monitored for
 alarms before considering the deployment to be complete
GrowthFactor                 : The percentage of targets that received a deployed
 configuration during each interval
GrowthType                   : The linear or exponential algorithm used to define
 how percentage grew over time
HttpStatusCode               : HTTP Status of the runtime
Id                           : The deployment strategy ID
Name                         : Name of the deployment strategy
ReplicateTo                  : The Systems Manager (SSM) document where the
 deployment strategy is saved
ResponseMetadata             : Runtime Metadata
```

構成のデプロイ

機能フラグとフリーフォーム設定データを操作するために必要なアーティファクトを作成したら、AWS Management Console、、 AWS CLIまたは SDK を使用して新しいデプロイを作成できます。でデプロイを開始すると、StartDeployment API オペレーションが AWS AppConfig 呼び出されます。この呼び出しには、デプロイする AWS AppConfig アプリケーションの ID、環境、構成プロファイル、および構成データバージョン（オプショナル）が含まれます。この呼び出しには、使用するデプロイ戦略の ID も含まれます。ID は、構成データのデプロイ方法を決定します。

に保存されているシークレット AWS Secrets Manager、カスタマーマネージドキーで暗号化された Amazon Simple Storage Service (Amazon S3) オブジェクト、またはカスタマーマネージドキーで暗号化された Parameter Store AWS Systems Manager に保存されている安全な文字列パ

ラメータをデプロイする場合は、 `KmsKeyIdentifier` パラメータの値を指定する必要があります。設定が暗号化されていない場合、または で暗号化されている場合は AWS マネージドキー、`KmsKeyIdentifier`パラメータの値を指定する必要はありません。

> **Note**
>
> `KmsKeyIdentifier` に指定する値は、カスタマーマネージド型キーである必要があります。これは設定の暗号化に使用したキーと同じである必要はありません。
> を使用してデプロイを開始する場合`KmsKeyIdentifier`、 AWS Identity and Access Management (IAM) プリンシパルにアタッチされたアクセス許可ポリシーで `kms:GenerateDataKey`オペレーションを許可する必要があります。

AWS AppConfig は、すべてのホストへのディストリビューションを監視し、ステータスを報告します。ディストリビューションが失敗した場合、 は設定を AWS AppConfig ロールバックします。

> **Note**
>
> 環境には、一度に 1 つの設定のみデプロイできます。ただし、1 つの設定をそれぞれ異なる環境に同時にデプロイすることができます。

設定をデプロイする (コンソール)

コンソールを使用して AWS AppConfig 設定を AWS Systems Manager デプロイするには、次の手順を使用します。

コンソールを使用して設定をデプロイするには

1. AWS Systems Manager コンソールを https://console.aws.amazon.com/systems-manager/appconfig/://https://https://https://https://https://https
2. ナビゲーションペインで、[アプリケーション] を選択し、AWS AppConfigでアプリケーションの名前空間を作成します。 で作成したアプリケーションを選択します。
3. [環境] タブで環境のラジオボタンを入力し、[詳細を表示] を選択します。
4. デプロイの開始 を選択します。
5. 設定 で、リストから設定を選択します。
6. 設定のソースに応じて、バージョンリストを使用して、デプロイするバージョンを選択します。

7. デプロイ戦略 で、リストから戦略を選択します。
8. (オプション)[デプロイの説明] に説明を入力します。
9. その他の暗号化オプションについては、リストから AWS Key Management Service キーを選択します。
10. (オプション) [タグ] セクションで [新しいタグを追加] を選択し、キーとオプションの値を入力します。1 つのリソースに対して最大 50 個のタグを指定できます。
11. デプロイの開始 を選択します。

設定をデプロイする (コマンドライン)

次の手順では、 AWS CLI (Linux または Windows の場合) または AWS Tools for PowerShell を使用して AWS AppConfig 設定をデプロイする方法について説明します。

設定をステップバイステップでデプロイする

1. を開きます AWS CLI。
2. 次のコマンドを実行して、設定をデプロイします。

   Linux

```
aws appconfig start-deployment \
  --application-id The_application_ID \
  --environment-id The_environment_ID \
  --deployment-strategy-id The_deployment_strategy_ID \
  --configuration-profile-id The_configuration_profile_ID \
  --configuration-version The_configuration_version_to_deploy \
  --description A_description_of_the_deployment \
  --tags User_defined_key_value_pair_metadata_of_the_deployment
```

   Windows

```
aws appconfig start-deployment ^
  --application-id The_application_ID ^
  --environment-id The_environment_ID ^
  --deployment-strategy-id The_deployment_strategy_ID ^
  --configuration-profile-id The_configuration_profile_ID ^
  --configuration-version The_configuration_version_to_deploy ^
  --description A_description_of_the_deployment ^
```

```
  --tags User_defined_key_value_pair_metadata_of_the_deployment
```

PowerShell

```
Start-APPCDeployment `
  -ApplicationId The_application_ID `
  -ConfigurationProfileId The_configuration_profile_ID `
  -ConfigurationVersion The_configuration_version_to_deploy `
  -DeploymentStrategyId The_deployment_strategy_ID `
  -Description A_description_of_the_deployment `
  -EnvironmentId The_environment_ID `
  -Tag Hashtable_type_user_defined_key_value_pair_metadata_of_the_deployment
```

システムが以下のような情報をレスポンスします。

Linux

```
{
   "ApplicationId": "The ID of the application that was deployed",
   "EnvironmentId" : "The ID of the environment",
   "DeploymentStrategyId": "The ID of the deployment strategy that was
 deployed",
   "ConfigurationProfileId": "The ID of the configuration profile that was
 deployed",
   "DeploymentNumber": The sequence number of the deployment,
   "ConfigurationName": "The name of the configuration",
   "ConfigurationLocationUri": "Information about the source location of the
 configuration",
   "ConfigurationVersion": "The configuration version that was deployed",
   "Description": "The description of the deployment",
   "DeploymentDurationInMinutes": Total amount of time the deployment lasted,
   "GrowthType": "The linear or exponential algorithm used to define how
 percentage grew over time",
   "GrowthFactor": The percentage of targets to receive a deployed configuration
 during each interval,
   "FinalBakeTimeInMinutes": Time AWS AppConfig monitored for alarms before
 considering the deployment to be complete,
   "State": "The state of the deployment",

   "EventLog": [
      {
         "Description": "A description of the deployment event",
```

```
         "EventType": "The type of deployment event",
         "OccurredAt": The date and time the event occurred,
         "TriggeredBy": "The entity that triggered the deployment event"
      }
   ],

   "PercentageComplete": The percentage of targets for which the deployment is
 available,
   "StartedAt": The time the deployment started,
   "CompletedAt": The time the deployment completed
}
```

Windows

```
{
   "ApplicationId": "The ID of the application that was deployed",
   "EnvironmentId" : "The ID of the environment",
   "DeploymentStrategyId": "The ID of the deployment strategy that was
 deployed",
   "ConfigurationProfileId": "The ID of the configuration profile that was
 deployed",
   "DeploymentNumber": The sequence number of the deployment,
   "ConfigurationName": "The name of the configuration",
   "ConfigurationLocationUri": "Information about the source location of the
 configuration",
   "ConfigurationVersion": "The configuration version that was deployed",
   "Description": "The description of the deployment",
   "DeploymentDurationInMinutes": Total amount of time the deployment lasted,
   "GrowthType": "The linear or exponential algorithm used to define how
 percentage grew over time",
   "GrowthFactor": The percentage of targets to receive a deployed configuration
 during each interval,
   "FinalBakeTimeInMinutes": Time AWS AppConfig monitored for alarms before
 considering the deployment to be complete,
   "State": "The state of the deployment",

   "EventLog": [
      {
         "Description": "A description of the deployment event",
         "EventType": "The type of deployment event",
         "OccurredAt": The date and time the event occurred,
         "TriggeredBy": "The entity that triggered the deployment event"
      }
```

```
    ],

    "PercentageComplete": The percentage of targets for which the deployment is
 available,
    "StartedAt": The time the deployment started,
    "CompletedAt": The time the deployment completed
}
```

PowerShell

```
ApplicationId               : The ID of the application that was deployed
CompletedAt                 : The time the deployment completed
ConfigurationLocationUri    : Information about the source location of the
 configuration
ConfigurationName           : The name of the configuration
ConfigurationProfileId      : The ID of the configuration profile that was
 deployed
ConfigurationVersion        : The configuration version that was deployed
ContentLength               : Runtime of the deployment
DeploymentDurationInMinutes : Total amount of time the deployment lasted
DeploymentNumber            : The sequence number of the deployment
DeploymentStrategyId        : The ID of the deployment strategy that was
 deployed
Description                 : The description of the deployment
EnvironmentId               : The ID of the environment that was deployed
EventLog                    : {Description : A description of the deployment
 event, EventType : The type of deployment event, OccurredAt : The date and time
 the event occurred,
         TriggeredBy : The entity that triggered the deployment event}
FinalBakeTimeInMinutes      : Time AWS AppConfig monitored for alarms before
 considering the deployment to be complete
GrowthFactor                : The percentage of targets to receive a deployed
 configuration during each interval
GrowthType                  : The linear or exponential algorithm used to define
 how percentage grew over time
HttpStatusCode              : HTTP Status of the runtime
PercentageComplete          : The percentage of targets for which the deployment
 is available
ResponseMetadata            : Runtime Metadata
StartedAt                   : The time the deployment started
State                       : The state of the deployment
```

CodePipeline を使用した AWS AppConfig 設定のデプロイ

AWS AppConfig は (AWS CodePipeline CodePipeline) の統合デプロイアクションです。CodePipeline はフルマネージド型の継続的デリバリーサービスで、アプリケーションとインフラストラクチャの更新を迅速かつ高い信頼性で行うために、パイプラインのリリースを自動化します。CodePipeline はお客様が定義したリリースモデルに基づき、コードチェンジがあった場合のフェーズの構築、テスト、およびデプロイを自動化します。詳細については、「What is AWS CodePipeline?」を参照してください。

AWS AppConfig と CodePipeline の統合には、次の利点があります。

- オーケストレーションを管理するために CodePipeline を使用するお客様は、コードベース全体をデプロイすることなく、アプリケーションに設定変更をデプロイする軽量な手段が利用できるようになりました。
- AWS AppConfig を使用して設定デプロイを管理するが、 が現在のコードまたは設定ストアをサポート AWS AppConfig していないため、制限されているお客様は、追加のオプションを利用できるようになりました。CodePipeline は AWS CodeCommit、、GitHub、BitBucket (いくつか) をサポートしています。

> **Note**
>
> AWS AppConfig CodePipeline との統合は、CodePipeline が利用可能な AWS リージョン でのみサポートされています。

統合の仕組み

まず、CodePipeline をセットアップして設定します。これには、CodePipeline がサポートするコードストアに設定を追加することが含まれます。次に、次のタスクを実行して AWS AppConfig 環境を設定します。

- 「名前空間と設定プロファイルを作成する」
- 定義済みのデプロイ戦略の選択または独自のデプロイ戦略の作成

これらのタスクが完了したら、デプロイプロバイダー AWS AppConfig として を指定するパイプラインを CodePipeline に作成します。作成できたら、設定を変更して、CodePipeline コードストアにアップロードします。新しい設定をアップロードすると、CodePipeline で新しいデプロイが

自動的に開始されます。デプロイが完了したら、変更を確認できます。をデプロイプロバイダー AWS AppConfig として指定するパイプラインの作成については、 AWS CodePipeline ユーザーガイドの「チュートリアル: デプロイプロバイダー AWS AppConfig として を使用するパイプラインの作成」を参照してください。

設定の復元

デプロイ中、不正な形式または誤った設定データが原因でアプリケーションにエラーが発生する状況を軽減するには、自動ロールバックを使用する (デプロイ中にアラームがトリガーされた場合) か、設定データを以前のバージョンに戻す (デプロイが正常に完了した場合) 必要があります。

自動ロールバックでは、 AWS AppConfig デプロイ戦略と Amazon CloudWatch アラームの組み合わせを使用できます。設定すると、デプロイ中に 1 つ以上の CloudWatch アラームが `ALARM`状態になると、 は設定データを以前のバージョン AWS AppConfig に自動的にロールバックするため、アプリケーションの停止やエラーを防止できます。開始するには、「(推奨) 自動ロールバックのアクセス許可を設定する」を参照してください。

> **Note**
>
> また、デプロイが進行中に StopDeployment API オペレーションを呼び出すことで、設定をロールバックすることもできます。

正常に完了したデプロイの場合、 は、StopDeployment API オペレーションで `AllowRevert`パラメータを使用して設定データを以前のバージョンに戻す AWS AppConfig こともサポートします。一部のお客様では、デプロイが成功した後に以前の設定に戻すと、データはデプロイ前と同じであることが保証されます。また、元に戻すとアラームモニターも無視され、アプリケーションの緊急事態中にロールフォワードが進行するのを防ぐことができます。

> **Important**
>
> `AllowRevert` パラメータを有効に`StopDeployment`して を呼び出すと、デプロイが過去 72 時間以内に成功した場合にのみ、 AWS AppConfig はデプロイを元に戻します。72 時間が経過すると、デプロイを元に戻すことはできません。新しいデプロイを作成する必要があります。

以下は、さまざまな状況に基づく `StopDeployment` 機能の内訳です。

1. 進行中のデプロイで StopDeployment が呼び出された場合、結果のデプロイ状態は ROLLED_BACK になります。
2. 進行中のデプロイで StopDeployment (でAllowRevert) が呼び出された場合、結果のデプロイ状態は になりますROLLED_BACK。
3. 完了したデプロイで StopDeployment が呼び出された場合、BadRequestException がスローされます。
4. 完了したデプロイで StopDeployment (でAllowRevert) が呼び出された場合、結果のデプロイ状態は になりますREVERTED。
5. 72 時間後に完了したデプロイで StopDeployment (でAllowRevert) が呼び出されると、BadRequestException がスローされます。

を使用して AWS CLI 、 AllowRevertパラメータを使用して StopDeployment オペレーションを呼び出すことができます。AllowRevert パラメータを含む AWS CLI コマンドの例を次に示します。

```
aws appconfig stop-deployment \
    --application-id 339ohji \
    --environment-id 54j1r29 \
    --deployment-number 2 \
    --allow-revert
```

AWS AppConfigで機能フラグと設定データを取得します。

アプリケーションは、 AWS AppConfig Data サービスを使用して設定セッションを確立することで、機能フラグとフリーフォーム設定データを取得します。 AWS AppConfig エージェントを使用して設定データを取得することをお勧めします。エージェント (または Lambda コンピューティング環境の AWS AppConfig エージェント Lambda 拡張機能) は、ユーザーに代わって一連の API コールとセッショントークンを管理します。大まかに言うと、プロセスは次のように機能します。

1. AWS AppConfig エージェントをローカルホストとして設定し、エージェントに設定の更新 AWS AppConfig をポーリングします。
2. エージェントは StartConfigurationSession と GetLatestConfiguration API アクションを呼び出し、設定データをローカルにキャッシュします。
3. データを取得するために、アプリケーションは localhost サーバーに HTTP 呼び出しを行います。AWS AppConfig エージェントは、「」で説明されているように、いくつかのユースケースをサポートしていますAWS AppConfig エージェントを使用して設定データを取得する方法。

希望に応じて、これらの API アクションを手動で呼び出して設定を取得できます。API プロセスは次のように機能します。

1. アプリケーションは `StartConfigurationSession` API アクションを使用して設定セッションを確立します。次に、セッションのクライアントは定期的に `GetLatestConfiguration` を呼び出し、最新の利用可能なデータを確認して取得します。
2. を呼び出すと`StartConfigurationSession`、コードはセッションが追跡する AWS AppConfig アプリケーション、環境、および設定プロファイルの識別子 (ID または名前) を送信します。
3. これに応じて、 はセッションのクライアントに渡す `InitialConfigurationToken` AWS AppConfig を提供し、そのセッション`GetLatestConfiguration`を初めて呼び出すときに を使用します。
4. `GetLatestConfiguration` を呼び出すと、クライアントコードは最新の `ConfigurationToken` 値を送信し、応答を受信します。
   - `NextPollConfigurationToken` :次回の`GetLatestConfiguration`への次の呼び出しで使用する`ConfigurationToken`値。
   - 設定:セッション用の最新データ。クライアントにすでに最新バージョンの設定がある場合は、この値は空になる可能性があります。

内容

- AWS AppConfig エージェントとは
- AWS AppConfig エージェントを使用して設定データを取得する方法
- AWS AppConfig ブラウザとモバイルの使用に関する考慮事項
- AWS AppConfig エージェントを使用せずに設定データを取得する

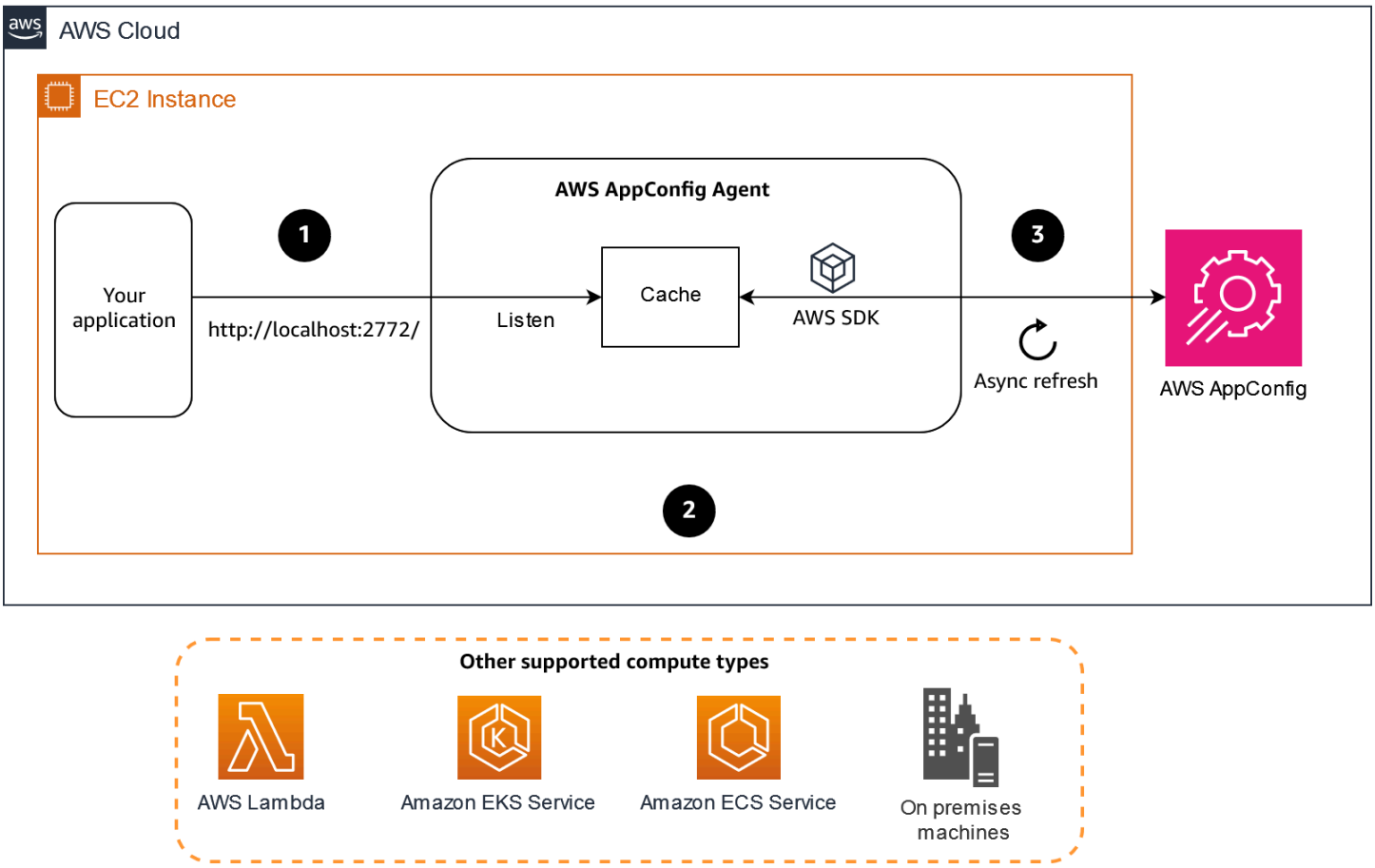
AWS AppConfig エージェントとは

AWS AppConfig エージェントは、設定データを取得するための Amazon が開発したマネージドプロセスです AWS AppConfig。エージェントを使用すると、設定データをローカルにキャッシュし、データ AWS AppConfig プレーンサービスを更新のために非同期的にポーリングできます。このキャッシュ/ポーリングプロセスにより、レイテンシーとコストを最小限に抑えながら、アプリケーションで設定データを常に利用できるようになります。エージェントは設定データを取得する唯一の方法ではなく AWS AppConfig、推奨される方法です。エージェントは、次の方法でアプリケーションの処理と管理を強化します。

- エージェントは、 AWS Identity and Access Management (IAM) プリンシパルを使用して設定データのローカルキャッシュを管理することで、 AWS AppConfig ユーザーに代わって を呼び出します。ローカルキャッシュから設定データを取得することで、アプリケーションが設定データを管理するために必要となるコードの更新が少なくなり、設定データをミリ秒単位で取得でき、呼び出しを妨げるネットワークの問題による影響を受けなくなります。
- エージェントは、 AWS AppConfig 機能フラグを取得して解決するためのネイティブエクスペリエンスを提供します。
- エージェントはすぐにキャッシュ戦略、ポーリング間隔、ローカル設定データの可用性に関するベストプラクティスを提供すると同時に、以降のサービスコールに必要な設定トークンを追跡します。
- バックグラウンドで実行中、エージェントは設定 AWS AppConfig データの更新のためにデータプレーンサービスを定期的にポーリングします。アプリケーションは、ポート 2772 (カスタマイズ可能なデフォルトポート値) で localhost に接続し、HTTP GET を呼び出してデータを取得できます。

> **Note**
>
> AWS AppConfig エージェントは、サービスが設定データを初めて取得したときにデータをキャッシュします。このため、データを取得する最初の呼び出しは、それ以降の呼び出しよりも時間がかかります。

次の図は、 AWS AppConfig エージェントの仕組みを示しています。



1. アプリケーションはエージェントに設定データをリクエストします。
2. エージェントはインメモリキャッシュからデータを返します。
3. エージェントは、事前定義された頻度で最新の設定データを AWS AppConfig サービスに非同期的にポーリングします。最新の設定データは、常にメモリ内のキャッシュに保存されます。

AWS AppConfig エージェントを使用して設定データを取得する方法

AWS AppConfig エージェントは、 AWS AppConfig 機能フラグまたはフリーフォーム設定データを取得するために推奨される方法です。エージェントは、Amazon EC2、Amazon ECS、Amazon EKS、Lambda など、あらゆる形式の AWS コンピューティングでサポートされています。エージェントの初期セットアップが完了したら、エージェントを使用して設定データを取得する方が、 AWS AppConfig APIsを直接呼び出すよりも簡単です。エージェントはベストプラクティスを自動的に実装し、設定を取得するための API 呼び出しが少なくな AWS AppConfig るため、 の使用コストを削減できる場合があります。

トピック

- での AWS AppConfig エージェントの使用 AWS Lambda
- Amazon EC2 およびオンプレミスマシンでの AWS AppConfig エージェントの使用
- Amazon ECS および Amazon EKS での AWS AppConfig エージェントの使用
- 基本およびマルチバリアント機能フラグの取得
- マニフェストを使用して追加の取得機能を有効にする
  - 複数のアカウントから設定を取得するように AWS AppConfig エージェントを設定する
  - ディスクに設定のコピーを書き込むように AWS AppConfig エージェントを設定する
- OpenAPI 仕様を使用したクライアントの生成
- AWS AppConfig エージェントローカル開発モードの使用

での AWS AppConfig エージェントの使用 AWS Lambda

AWS Lambda 拡張機能は、Lambda 関数の機能を強化するコンパニオンプロセスです。拡張機能は、関数が呼び出される前に開始し、関数と並行して実行し、関数の呼び出しが処理された後も引き続き実行できます。本質的に、Lambda 拡張機能は Lambda 呼び出しと並行して実行されるクライアントのようなものです。この並列クライアントは、ライフサイクル中の任意の時点で関数とインターフェイスできます。

Lambda 関数で AWS AppConfig 機能フラグやその他の動的設定データを使用する場合は、 AWS AppConfig エージェント Lambda 拡張機能を Lambda 関数のレイヤーとして追加することをお勧めします。これにより、機能フラグの呼び出しが簡単になり、拡張機能自体には、コストを削減 AWS AppConfig しながら の使用を簡素化するベストプラクティスが含まれています。コストを削減する

には、 AWS AppConfig サービスへの API コールが少なくなり、Lambda 関数の処理時間が短くなります。Lambda 拡張機能の詳細については、「AWS Lambda デベロッパーガイド」の「Lambda 拡張機能」を参照してください。

> **Note**
>
> AWS AppConfig は の機能です AWS Systems Manager。料金は、設定が呼び出されて受信された回数に基づいています。 AWS AppConfig https://aws.amazon.com/systems-manager/pricing/Lambda が複数のコールドスタートを実行し、新しい設定データを頻繁に取得すると、コストが増加します。

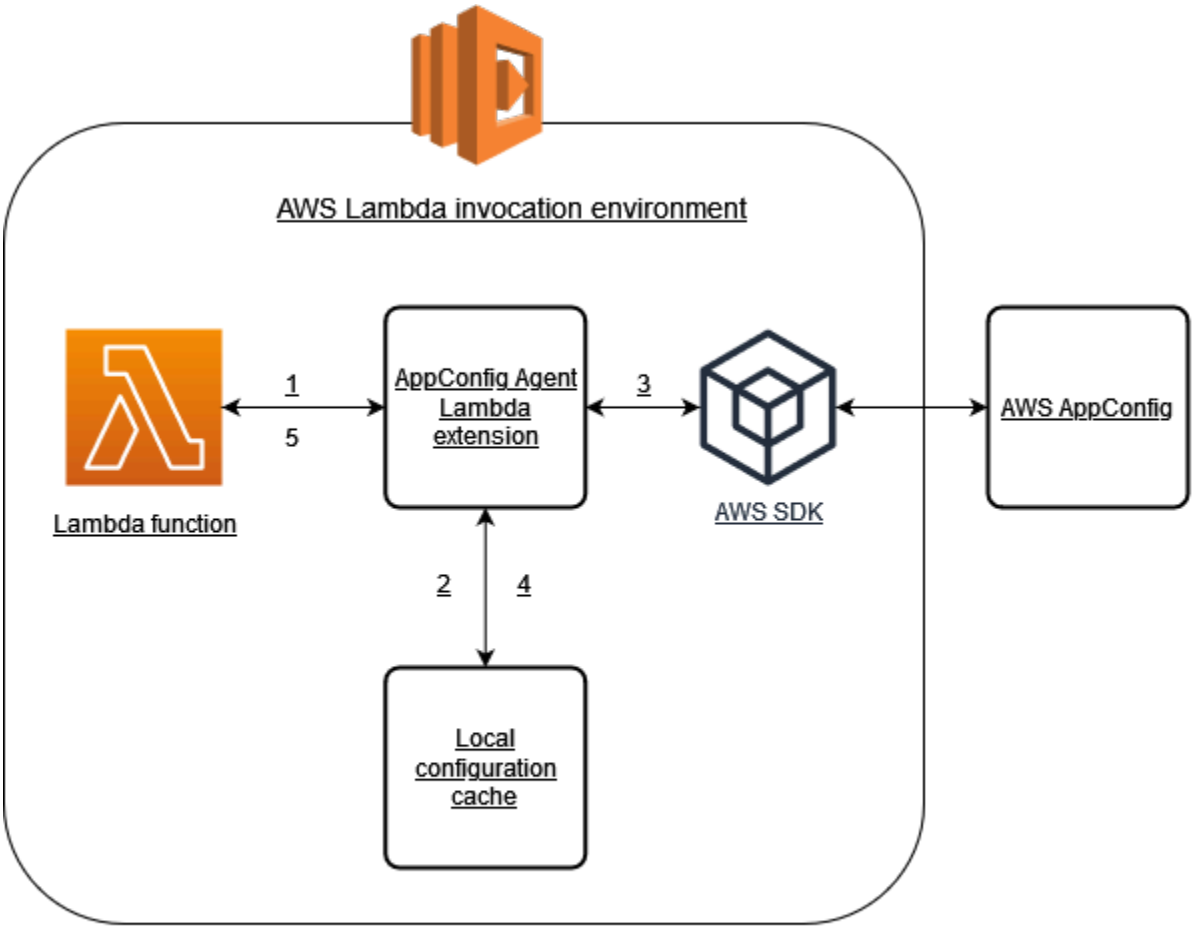
トピック

- AWS AppConfig エージェント Lambda 拡張機能の仕組みを理解する
- AWS AppConfig エージェント Lambda 拡張機能の追加
- AWS AppConfig エージェント Lambda 拡張機能の設定
- AWS AppConfig Agent Lambda 拡張機能の利用可能なバージョンについて

AWS AppConfig エージェント Lambda 拡張機能の仕組みを理解する

AWS AppConfig を使用して Lambda 拡張機能のない Lambda 関数の設定を管理する場合は、StartConfigurationSession および GetLatestConfiguration API アクションと統合して、設定の更新を受け取るように Lambda 関数を設定する必要があります。

AWS AppConfig エージェント Lambda 拡張機能を Lambda 関数と統合すると、このプロセスが簡素化されます。拡張機能は、 AWS AppConfig サービスの呼び出し、取得されたデータのローカルキャッシュの管理、次のサービス呼び出しに必要な設定トークンの追跡、バックグラウンドでの設定更新の定期的なチェックを行います。次の図は、その仕組みを示しています。



1. AWS AppConfig エージェント Lambda 拡張機能を Lambda 関数のレイヤーとして設定します。
2. 設定データにアクセスするために、関数は で実行されている HTTP エンドポイントで AWS AppConfig 拡張機能を呼び出します`localhost:2772`。
3. 拡張機能は、設定データのローカルキャッシュを保持します。データがキャッシュにない場合、拡張機能は AWS AppConfig を呼び出して設定データを取得します。
4. サービスから設定を受信すると、拡張機能はローカルキャッシュに設定を保存し、Lambda 関数に渡します。
5. AWS AppConfig エージェント Lambda 拡張機能は、設定データの更新をバックグラウンドで定期的にチェックします。Lambda 関数が呼び出されるたびに、拡張機能は、設定を取得してからの経過時間をチェックします。経過時間が設定されたポーリング間隔より大きい場合、拡張機能は AWS AppConfig を呼び出して新しくデプロイされたデータをチェックし、変更があった場合はローカルキャッシュを更新し、経過時間をリセットします。

> **Note**
>
> - Lambda は、関数が必要とする同時実行レベルに対応する個別のインスタンスをインスタンス化します。各インスタンスは分離され、設定データの独自のローカルキャッシュが保持されます。Lambda インスタンスと同時実行の詳細については、「Lambda 関数の同時実行数の管理」を参照してください。
> - 更新された設定をデプロイした後、設定変更が Lambda 関数に表示されるまでにかかる時間は AWS AppConfig、デプロイに使用したデプロイ戦略と、拡張機能用に設定したポーリング間隔によって異なります。

AWS AppConfig エージェント Lambda 拡張機能の追加

AWS AppConfig エージェント Lambda 拡張機能を使用するには、拡張機能を Lambda に追加する必要があります。これは、 AWS AppConfig エージェント Lambda 拡張機能を Lambda 関数にレイヤーとして追加するか、Lambda 関数で拡張機能をコンテナイメージとして有効にすることで実行できます。

> **Note**
>
> AWS AppConfig 拡張機能はランタイムに依存しず、すべてのランタイムをサポートします。

[開始する前に]

AWS AppConfig エージェント Lambda 拡張機能を有効にする前に、次の操作を行います。

- 設定を AWS AppConfigに外部化できるように、Lambda 関数での設定を整理します。
- 機能フラグやフリーフォーム設定データなど、 AWS AppConfig アーティファクトと設定データを作成します。詳細については、「で機能フラグとフリーフォーム設定データを作成する AWS AppConfig」を参照してください。
- Lambda `appconfig:GetLatestConfiguration` 関数実行ロールで使用される AWS Identity and Access Management (IAM) ポリシーに `appconfig:StartConfigurationSession`と を追加します。詳細については、「AWS Lambda デベロッパーガイド」の「AWS Lambda 実行ロール」を参照してください。詳細については、「サービス承認リファレンス」の「 AWS AppConfig の AWS AppConfig アクション、リソース、および条件キー」を参照してください。

レイヤーと AWS AppConfig ARN を使用してエージェント Lambda 拡張機能を追加します。

AWS AppConfig エージェント Lambda 拡張機能を使用するには、拡張機能を Lambda 関数にレイヤーとして追加します。関数にレイヤーを追加する方法については、「AWS Lambda デベロッパーガイド」の「拡張機能の設定」を参照してください。 AWS Lambda コンソールの拡張機能の名前は AWS-AppConfig-Extension です。また、拡張機能を Lambda にレイヤーとして追加する場合は、Amazon リソースネーム (ARN) を指定する必要があることにも注意してください。プラットフォームと Lambda を作成した AWS リージョン 場所に対応する次のいずれかのリストから ARN を選択します。

- x86-64 プラットフォーム
- ARM64 プラットフォーム

拡張機能を関数に追加する前にテストする場合は、次のコード例を使用して拡張機能が機能することを確認できます。

```
import urllib.request


def lambda_handler(event, context):
    url = f'http://localhost:2772/applications/application_name/
environments/environment_name/configurations/configuration_name'
    config = urllib.request.urlopen(url).read()
    return config
```

これをテストするには、Python 用の新しい Lambda 関数を作成し、拡張機能を追加して Lambda 関数を実行します。Lambda 関数を実行すると、 AWS AppConfig Lambda 関数は http://localhost:2772 パスに指定した設定を返します。Lambda 関数の作成については、「AWS Lambda デベロッパーガイド」の「コンソールで Lambda の関数の作成」を参照してください。

> **Important**
>
> AWS AppConfig エージェント Lambda 拡張機能の AWS Lambda ログデータをログに表示できます。ログエントリには `appconfig agent` というプレフィックスが付けられています。以下に例を示します。
>
> ```
> [appconfig agent] 2024/05/07 04:19:01 ERROR retrieve failure for
> 'SourceEventConfig:SourceEventConfigEnvironment:SourceEventConfigProfile':
> StartConfigurationSession: api error AccessDenied: User:
> ```

```
 arn:aws:sts::0123456789:assumed-role/us-east-1-LambdaRole/
extension1 is not authorized to perform: sts:AssumeRole on resource:
 arn:aws:iam::0123456789:role/test1 (retry in 60s)
```

AWS AppConfig エージェント Lambda 拡張機能の設定

拡張機能を設定するには、次の AWS Lambda 環境変数を変更します。詳細については、「 AWS Lambda デベロッパーガイド」のAWS Lambda 「環境変数の使用」を参照してください。

設定データのプリフェッチ

環境変数 `AWS_APPCONFIG_EXTENSION_PREFETCH_LIST` は、関数の起動時間を大幅に改善できます。 AWS AppConfig エージェント Lambda 拡張機能が初期化されると、Lambda が関数の初期化を開始し、ハンドラーを呼び出す AWS AppConfig 前に、 から指定された設定を取得します。場合によっては、関数が要求する前に、設定データがローカルキャッシュで既に利用可能になっていることがあります。

プリフェッチ機能を使用するには、設定データに対応するパスに、環境変数の値を設定します。例えば、設定がそれぞれ「my_application」、「my_environment」、「my_configuration_data」という名前のアプリケーション、環境、および設定プロファイルに対応している場合、パスは `/applications/my_application/environments/my_environment/configurations/my_configuration_data` のようになります。設定項目をカンマ区切りのリストとして列挙することで、複数の設定項目を指定できます (カンマを含むリソース名がある場合は、リソースの名前ではなく ID 値を使用します)。

別のアカウントから設定データへアクセスします

AWS AppConfig エージェント Lambda 拡張機能は、データにアクセス許可を付与する IAM ロールを指定することで、別のアカウントから設定データを取得できます。これを設定するには、次の手順に従います。

1. AWS AppConfig が設定データを管理するために使用されるアカウントで、Lambda 関数を実行しているアカウントに `appconfig:StartConfigurationSession` および `appconfig:GetLatestConfiguration`アクションへのアクセス権と、 AWS AppConfig 設定リソースに対応する部分的または完全な ARNs を付与する信頼ポリシーを持つロールを作成します。
2. Lambda 関数を実行するアカウントで、ステップ 1 で作成したロールの ARN `AWS_APPCONFIG_EXTENSION_ROLE_ARN` を含む環境変数を Lambda 関数に追加します。

3. （オプショナル）必要に応じて、AWS_APPCONFIG_EXTENSION_ROLE_EXTERNAL_ID 環境変数を使用して外部 ID を指定できます。同様に、AWS_APPCONFIG_EXTENSION_ROLE_SESSION_NAME セッション名は環境変数を使用して設定できます。

> **Note**
>
> 以下の情報に注意してください。
>
> - AWS AppConfig エージェント Lambda 拡張機能は、1 つのアカウントからのみデータを取得できます。IAM ロールを指定した場合、拡張子は Lambda 関数が実行されているアカウントから設定データを取得できません。
> - AWS Lambda は、Amazon CloudWatch Logs を使用して AWS AppConfig 、エージェント Lambda 拡張機能と Lambda 関数に関する情報をログに記録します。
> - 次の表に、サンプル値の列を示します。モニターの解像度によっては、テーブルの下部までスクロールし、右側までスクロールして列を表示する必要がある場合があります。

環境変数	詳細	デフォルト値	サンプル値
AWS_APPCONFIG_EXTENSION_HTTP_PORT	この環境変数は、拡張機能をホストするローカル HTTP サーバーが実行されるポートを指定します。	2772	2772
AWS_APPCONFIG_EXTENSION_LOG_LEVEL	この環境変数は、エージェントがログに記録する詳細レベルを指定します。各レベルには、現在レベルとそれより上位のすべてのレベルが含まれます。値は大文字と小文字を区別し	情報	trace デバッグ 情報 warn エラー fatal

環境変数	詳細	デフォルト値	サンプル値
	ません。詳細度が最も低いログレベルは、trace、、debug、i よび で すnone。trace ログには、エージェントに関するタイミング情報を含む詳細情報が含まれます。		なし
AWS_APPCONFIG_EXTENSION_MAX_CONNECTIONS	この環境変数は、拡張機能が AWS AppConfigから設定を取得するために使用する最大接続数を設定します。	3	3

環境変数	詳細	デフォルト値	サンプル値
AWS_APPCONFIG_EXTENSION_POLL_INTERVAL_SECONDS	この環境変数は、エージェントが AWS AppConfig 更新された設定データをポーリングする頻度を制御します。間隔の秒数を指定できます。時間単位で数値を指定することもできます。s は秒、m は分、h は時間です。単位が指定されなかった場合、エージェントはデフォルトで秒になります。たとえば、60 、60 秒、1m の場合、ポーリング間隔は同じになります。	45	45 45 秒 5m 1 時間

環境変数	詳細	デフォルト値	サンプル値
AWS_APPCONFIG_EXTENSION_POLL_TIMEOUT_MILLIS	この環境変数は、キャッシュ内のデータを更新する AWS AppConfig ときに、拡張機能が からのレスポンスを待機する最大時間をミリ秒単位で制御します。 AWS AppConfig が指定された時間内に応答しない場合、拡張機能はこのポーリング間隔をスキップし、以前に更新されたキャッシュデータを返します。	3000 ミリ秒	3000 300 ミリ秒 5 秒
AWS_APPCONFIG_EXTENSION_PREFETCH_LIST	この環境変数は、エージェントが起動 AWS AppConfig するとすぐにリクエストする設定データを指定します。複数の設定識別子をカンマ区切りリストで指定できます。から設定データをプリフェッチ AWS AppConfig すると、関数のコールドスタート時間を大幅に短縮できます。	なし	MyApp:MyEnv:MyConfig abcd123:efgh456:ijkl789 MyApp:MyEnv:Config1、MyApp:MyEnv:Config2

環境変数	詳細	デフォルト値	サンプル値
AWS_APPCONFIG_EXTENSION_PROXY_HEADERS	この環境変数は、AWS_APPCONFIG_EXTENSION_PROXY_URL 環境変数で参照されるプロキシに必要なヘッダーを指定します。値は、コンマで区切られたヘッダーの一覧です。	なし	ヘッダー: 値 h1: v1、h2: v2
AWS_APPCONFIG_EXTENSION_PROXY_URL	この環境変数は、AWS AppConfig 拡張機能から への接続に使用するプロキシ URL を指定します AWS のサービス。HTTPSおよび HTTP URLs がサポートされています。	なし	http://localhost:7474 https://my-proxy.example.com
AWS_APPCONFIG_EXTENSION_ROLE_ARN	この環境変数は、設定を取得するために AWS AppConfig 拡張機能が引き受ける必要があるロールに対応する IAM ロール ARN を指定します。	なし	arn:aws:iam::123456789012:role/MyRole
AWS_APPCONFIG_EXTENSION_ROLE_EXTERNAL_ID	この環境変数は、想定されるロール ARN と組み合わせて使用する外部 ID を指定します。	なし	MyExternalId

環境変数	詳細	デフォルト値	サンプル値
`AWS_APPCONFIG_EXTENSION_ROLE_SESSION_NAME`	この環境変数は、引き受ける IAM ロールの認証情報に関連付けるセッション名を指定します。	なし	AWSAppConfigAgentSession
`AWS_APPCONFIG_EXTENSION_SERVICE_REGION`	この環境変数は、拡張機能が AWS AppConfig サービスを呼び出すために使用する代替リージョンを指定します。未定義の場合、拡張機能は現在のリージョンのエンドポイントを使用します。	なし	us-east-1 eu-west-1
`AWS_APPCONFIG_EXTENSION_MANIFEST`	この環境変数は、マルチアカウントの取得やディスクへの設定の保存など、設定ごとに追加の機能を利用するように AWS AppConfig エージェントを設定します。これらの機能の詳細については、「マニフェストを使用して追加の取得機能を有効にする」を参照してください。	なし	マニフェストとして AWS AppConfig 設定を使用する場合: `MyApp:MyEnv:MyManifestConfig` 。 ディスクからマニフェストをロードする場合: `file:/path/to/manifest.json`

環境変数	詳細	デフォルト値	サンプル値
AWS_APPCONFIG_EXTENSION_WAIT_ON_MANIFEST	この環境変数は、起動が完了する前にマニフェストが処理されるまで待機するように AWS AppConfig エージェントを設定します。	真	真 false

AWS AppConfig Agent Lambda 拡張機能の利用可能なバージョンについて

このトピックには、 AWS AppConfig エージェント Lambda 拡張機能のバージョンに関する情報が含まれています。 AWS AppConfig エージェント Lambda 拡張機能は、x86-64 および ARM64 (Graviton2) プラットフォーム用に開発された Lambda 関数をサポートします。正しく動作するには、Lambda 関数が現在ホスト AWS リージョン されている に特定の Amazon リソースネーム (ARN) を使用するように設定する必要があります。 AWS リージョン および ARN の詳細は、このセクションの後半で確認できます。

> **Important**
>
> AWS AppConfig エージェント Lambda 拡張機能に関する以下の重要な詳細に注意してください。
>
> - `GetConfiguration` API アクションは 2022 年 1 月 28 日に廃止されました。設定データを受信する呼び出しでは、`StartConfigurationSession` と `GetLatestConfiguration` API を使用する必要があります 2022 年 1 月 28 日より前に作成された AWS AppConfig エージェント Lambda 拡張機能のバージョンを使用している場合は、新しい APIs へのアクセス許可を設定する必要があります。詳細については、「AWS AppConfig エージェントを使用せずに設定データを取得する」を参照してください。
> - AWS AppConfig は、 にリストされているすべてのバージョンをサポートしています古い拡張機能バージョン。拡張機能を利用するには、定期的に最新バージョンに更新することをお勧めします。

トピック

- AWS AppConfig エージェント Lambda 拡張機能リリースノート
- Lambda 拡張機能のバージョン番号を検索します。
- x86-64 プラットフォーム
- ARM64 プラットフォーム
- 古い拡張機能バージョン

AWS AppConfig エージェント Lambda 拡張機能リリースノート

次の表に、 AWS AppConfig Lambda 拡張機能の最新バージョンに加えられた変更を示します。

バージョン	開始日	メモ
2037 年 2 月 0 日	05/12/2025	そのエージェントのバージョンを返すシンプルなヘルスチェックを公開する`/ping`パスを追加しました。マイナーな機能強化とバグ修正も含まれています。
2.0.1079	12/12/2024	軽微な機能強化とバグ修正。
2.0.719	08/08/2024	軽微な機能強化とバグ修正。
2.0.678	07/23/2024	機能フラグターゲット、バリアント、分割をサポートする機能強化。詳細については、「マルチバリアント機能フラグの作成」を参照してください。
2.0.501	07/01/2024	軽微な機能強化とバグ修正。
2.0.358	12/01/2023	以下の取得機能のサポートが追加されました。 • マルチアカウント取得: プライマリまたは取得 AWS アカウント から AWS

バージョン	開始日	メモ
		AppConfig エージェントを使用して、複数のベンダーアカウントから設定データを取得します。 • ディスクへの設定コピーの書き込み: AWS AppConfig エージェントを使用して、ディスクに設定データを書き込みます。この機能を使用すると、ディスクから設定データを読み取るアプリケーションを使用しているお客様は、 AWS AppConfig と統合できます。
2.0.181	08/14/2023	イスラエル (テルアビブ) il-central-1 のサポートが追加されました AWS リージョン。

バージョン	開始日	メモ
2.0.165	02/21/2023	軽微なバグを修正。 AWS Lambda コンソールを使用して、拡張機能の使用を特定のランタイムバージョンに制限しなくなりました。以下の AWS リージョンにサポートを追加しました。 • 中東 (UAE) me-central-1 • アジアパシフィック (ハイデラバード) ap-south-2 • アジアパシフィック (メルボルン) ap-southeast-4 • 欧州 (スペイン) eu-south-2 • 欧州 (チューリッヒ) eu-central-2
2.0.122	08/23/2022	トンネリングプロキシのサポートが追加され、`AWS_APPCONFIG_EXTENSION_PROXY_URL` 環境変数と `AWS_APPCONFIG_EXTENSION_PROXY_HEADERS` 環境変数を使用して設定できるようになりました。.NET 6 をランタイムとして追加しました。環境変数の詳細については、「AWS AppConfig エージェント Lambda 拡張機能の設定」を参照してください。
2.0.58	05/03/2022	Lambda でのグラビトン2 (ARM64) プロセッサーのサポートが改善されました。

バージョン	開始日	メモ
2.0.45	03/15/2022	1 つの機能フラグを呼び出すサポートが追加されました。以前は、顧客は設定プロファイルにグループ化された機能フラグを呼び出し、その応答をクライアント側で解析する必要がありました。このリリースでは、顧客は HTTP localhost エンドポイントを呼び出すときに`flag=<flag-name>` パラメータを使用して 個々のフラグの値を取得できるようになりました。また、Graviton2 (ARM64) プロセッサの初期サポートも追加されました。

Lambda 拡張機能のバージョン番号を検索します。

次の手順を使用して、現在設定されている AWS AppConfig Agent Lambda 拡張機能のバージョン番号を見つけます。正しく動作するには、Lambda 関数が現在ホスト AWS リージョン されている に特定の Amazon リソースネーム (ARN) を使用するように設定する必要があります。

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/lambda/ で AWS Lambda コンソールを開きます。
2. `AWS-AppConfig-Extension` レイヤーを追加する Lambda 関数を選択します。
3. レイヤーエリアで、レイヤーの追加を選択します。
4. レイヤーの選択セクションで、レイヤーリストからAWS-AppConfig-Extensionを選択します。AWS
5. バージョンリストを使用してバージョン番号を選択します。
6. 追加 を選択します。
7. [テスト] タブを使用して機能をテストします。

8. テストが完了したら、ログ出力を表示します。実行の詳細セクションで AWS AppConfig エージェント Lambda 拡張機能バージョンを見つけます。 このバージョンは、バージョンに必要な URL と一致する必要があります。

x86-64 プラットフォーム

拡張機能を Lambda にレイヤーとして追加する場合は、ARN を指定する必要があります。Lambda を AWS リージョン 作成した に対応する ARN を次の表から選択します。これらの ARN は、x86-64 プラットフォーム向けに開発された Lambda 関数用です。

バージョン 2.0.2037

リージョン	ARN
米国東部 (バージニア北部)	`arn:aws:lambda:us-east-1:027255383542:layer:AWS-AppConfig-Extension:207`
米国東部 (オハイオ)	`arn:aws:lambda:us-east-2:728743619870:layer:AWS-AppConfig-Extension:162`
米国西部 (北カリフォルニア)	`arn:aws:lambda:us-west-1:958113053741:layer:AWS-AppConfig-Extension:258`
米国西部 (オレゴン)	`arn:aws:lambda:us-west-2:359756378197:layer:AWS-AppConfig-Extension:262`
カナダ (中部)	`arn:aws:lambda:ca-central-1:039592058896:layer:AWS-AppConfig-Extension:152`
カナダ西部 (カルガリー)	`arn:aws:lambda:ca-west-1:436199621743:layer:AWS-AppConfig-Extension:57`

リージョン	ARN
欧州 (フランクフルト)	arn:aws:lambda:eu-central-1:066940009817:layer:AWS-AppConfig-Extension:189
欧州 (チューリッヒ)	arn:aws:lambda:eu-central-2:758369105281:layer:AWS-AppConfig-Extension:106
欧州 (アイルランド)	arn:aws:lambda:eu-west-1:434848589818:layer:AWS-AppConfig-Extension:189
欧州 (ロンドン)	arn:aws:lambda:eu-west-2:282860088358:layer:AWS-AppConfig-Extension:133
欧州 (パリ)	arn:aws:lambda:eu-west-3:493207061005:layer:AWS-AppConfig-Extension:162
欧州 (ストックホルム)	arn:aws:lambda:eu-north-1:646970417810:layer:AWS-AppConfig-Extension:259
欧州 (ミラノ)	arn:aws:lambda:eu-south-1:203683718741:layer:AWS-AppConfig-Extension:140
欧州 (スペイン)	arn:aws:lambda:eu-south-2:586093569114:layer:AWS-AppConfig-Extension:102
中国 (北京)	arn:aws-cn:lambda:cn-north-1:615057806174:layer:AWS-AppConfig-Extension:133

リージョン	ARN
中国 (寧夏)	arn:aws-cn:lambda:cn-northwest-1:615084187847:layer:AWS-AppConfig-Extension:131
アジアパシフィック (香港)	arn:aws:lambda:ap-east-1:630222743974:layer:AWS-AppConfig-Extension:142
アジアパシフィック (東京)	arn:aws:lambda:ap-northeast-1:980059726660:layer:AWS-AppConfig-Extension:155
アジアパシフィック (ソウル)	arn:aws:lambda:ap-northeast-2:826293736237:layer:AWS-AppConfig-Extension:165
アジアパシフィック (大阪)	arn:aws:lambda:ap-northeast-3:706869817123:layer:AWS-AppConfig-Extension:159
アジアパシフィック (シンガポール)	arn:aws:lambda:ap-southeast-1:421114256042:layer:AWS-AppConfig-Extension:156
アジアパシフィック (シドニー)	arn:aws:lambda:ap-southeast-2:080788657173:layer:AWS-AppConfig-Extension:199
アジアパシフィック (ジャカルタ)	arn:aws:lambda:ap-southeast-3:418787028745:layer:AWS-AppConfig-Extension:150
アジアパシフィック (メルボルン)	arn:aws:lambda:ap-southeast-4:307021474294:layer:AWS-AppConfig-Extension:78

リージョン	ARN
アジアパシフィック (マレーシア)	arn:aws:lambda:ap-southeast-5:631746059939:layer:AWS-AppConfig-Extension:55
アジアパシフィック (ムンバイ)	arn:aws:lambda:ap-south-1:554480029851:layer:AWS-AppConfig-Extension:175
アジアパシフィック (ハイデラバード)	arn:aws:lambda:ap-south-2:489524808438:layer:AWS-AppConfig-Extension:104
南米 (サンパウロ)	arn:aws:lambda:sa-east-1:000010852771:layer:AWS-AppConfig-Extension:215
アフリカ (ケープタウン)	arn:aws:lambda:af-south-1:574348263942:layer:AWS-AppConfig-Extension:152
イスラエル (テルアビブ)	arn:aws:lambda:il-central-1:895787185223:layer:AWS-AppConfig-Extension:81
中東 (アラブ首長国連邦)	arn:aws:lambda:me-central-1:662846165436:layer:AWS-AppConfig-Extension:120
中東 (バーレーン)	arn:aws:lambda:me-south-1:559955524753:layer:AWS-AppConfig-Extension:154
AWS GovCloud (米国東部)	arn:aws-us-gov:lambda:us-gov-east-1:946561847325:layer:AWS-AppConfig-Extension:110

リージョン	ARN
AWS GovCloud (米国西部)	`arn:aws-us-gov:lambda:us-gov-west-1:946746059096:layer:AWS-AppConfig-Extension:109`

ARM64 プラットフォーム

拡張機能を Lambda にレイヤーとして追加する場合は、ARN を指定する必要があります。Lambda を AWS リージョン 作成した に対応する ARN を次の表から選択します。これらの ARN は ARM64 プラットフォーム用に開発された Lambda 関数用です。

バージョン 2.0.2037

リージョン	ARN
米国東部 (バージニア北部)	`arn:aws:lambda:us-east-1:027255383542:layer:AWS-AppConfig-Extension-Arm64:140`
米国東部 (オハイオ)	`arn:aws:lambda:us-east-2:728743619870:layer:AWS-AppConfig-Extension-Arm64:114`
米国西部 (北カリフォルニア)	`arn:aws:lambda:us-west-1:958113053741:layer:AWS-AppConfig-Extension-Arm64:135`
米国西部 (オレゴン)	`arn:aws:lambda:us-west-2:359756378197:layer:AWS-AppConfig-Extension-Arm64:164`
カナダ (中部)	`arn:aws:lambda:ca-central-1:039592058896:layer:AWS-AppConfig-Extension-Arm64:72`

リージョン	ARN
カナダ西部 (カルガリー)	arn:aws:lambda:ca-west-1:436199621743:layer:AWS-AppConfig-Extension-Arm64:47
欧州 (フランクフルト)	arn:aws:lambda:eu-central-1:066940009817:layer:AWS-AppConfig-Extension-Arm64:132
欧州 (チューリッヒ)	arn:aws:lambda:eu-central-2:758369105281:layer:AWS-AppConfig-Extension-Arm64:64
欧州 (アイルランド)	arn:aws:lambda:eu-west-1:434848589818:layer:AWS-AppConfig-Extension-Arm64:127
欧州 (ロンドン)	arn:aws:lambda:eu-west-2:282860088358:layer:AWS-AppConfig-Extension-Arm64:85
欧州 (パリ)	arn:aws:lambda:eu-west-3:493207061005:layer:AWS-AppConfig-Extension-Arm64:81
欧州 (ストックホルム)	arn:aws:lambda:eu-north-1:646970417810:layer:AWS-AppConfig-Extension-Arm64:118
欧州 (ミラノ)	arn:aws:lambda:eu-south-1:203683718741:layer:AWS-AppConfig-Extension-Arm64:68
欧州 (スペイン)	arn:aws:lambda:eu-south-2:586093569114:layer:AWS-AppConfig-Extension-Arm64:63

リージョン	ARN
アジアパシフィック (香港)	arn:aws:lambda:ap-east-1:630222743974:layer:AWS-AppConfig-Extension-Arm64:70
アジアパシフィック (東京)	arn:aws:lambda:ap-northeast-1:980059726660:layer:AWS-AppConfig-Extension-Arm64:108
アジアパシフィック (ソウル)	arn:aws:lambda:ap-northeast-2:826293736237:layer:AWS-AppConfig-Extension-Arm64:73
アジアパシフィック (大阪)	arn:aws:lambda:ap-northeast-3:706869817123:layer:AWS-AppConfig-Extension-Arm64:74
アジアパシフィック (シンガポール)	arn:aws:lambda:ap-southeast-1:421114256042:layer:AWS-AppConfig-Extension-Arm64:108
アジアパシフィック (シドニー)	arn:aws:lambda:ap-southeast-2:080788657173:layer:AWS-AppConfig-Extension-Arm64:142
アジアパシフィック (ジャカルタ)	arn:aws:lambda:ap-southeast-3:418787028745:layer:AWS-AppConfig-Extension-Arm64:87
アジアパシフィック (メルボルン)	arn:aws:lambda:ap-southeast-4:307021474294:layer:AWS-AppConfig-Extension-Arm64:63
アジアパシフィック (マレーシア)	arn:aws:lambda:ap-southeast-5:631746059939:layer:AWS-AppConfig-Extension-Arm64:30

リージョン	ARN
アジアパシフィック (ムンバイ)	arn:aws:lambda:ap-south-1:554480029851:layer:AWS-AppConfig-Extension-Arm64:117
アジアパシフィック (ハイデラバード)	arn:aws:lambda:ap-south-2:489524808438:layer:AWS-AppConfig-Extension-Arm64:62
南米 (サンパウロ)	arn:aws:lambda:sa-east-1:000010852771:layer:AWS-AppConfig-Extension-Arm64:103
アフリカ (ケープタウン)	arn:aws:lambda:af-south-1:574348263942:layer:AWS-AppConfig-Extension-Arm64:80
中東 (アラブ首長国連邦)	arn:aws:lambda:me-central-1:662846165436:layer:AWS-AppConfig-Extension-Arm64:76
中東 (バーレーン)	arn:aws:lambda:me-south-1:559955524753:layer:AWS-AppConfig-Extension-Arm64:82
イスラエル (テルアビブ)	arn:aws:lambda:il-central-1:895787185223:layer:AWS-AppConfig-Extension-Arm64:64
中国 (北京)	arn:aws-cn:lambda:cn-north-1:615057806174:layer:AWS-AppConfig-Extension-Arm64:55
中国 (寧夏)	arn:aws-cn:lambda:cn-northwest-1:615084187847:layer:AWS-AppConfig-Extension-Arm64:53

リージョン	ARN
AWS GovCloud (米国東部)	`arn:aws-us-gov:lambda:us-gov-east-1:946561847325:layer:AWS-AppConfig-Extension-Arm64:56`
AWS GovCloud (米国西部)	`arn:aws-us-gov:lambda:us-gov-west-1:946746059096:layer:AWS-AppConfig-Extension-Arm64:55`

古い拡張機能バージョン

このセクションでは、 AWS リージョン 古いバージョンの AWS AppConfig Lambda 拡張機能の ARNsと を一覧表示します。このリストには、以前のバージョンの AWS AppConfig エージェント Lambda拡張機能のすべての情報は含まれていませんが、新しいバージョンがリリースされると更新されます。

トピック

- 古い拡張バージョン (x86-64 プラットフォーム)
- 古い拡張バージョン (ARM64 プラットフォーム)

古い拡張バージョン (x86-64 プラットフォーム)

次の表ARNs と を示します。 AWS リージョン AWS AppConfig

新しい拡張子に置き換えられた日付: 05/20/2025

バージョン 2.0.1079

リージョン	ARN
米国東部 (バージニア北部)	`arn:aws:lambda:us-east-1:027255383542:layer:AWS-AppConfig-Extension:174`
米国東部 (オハイオ)	`arn:aws:lambda:us-east-2:728743619870:layer:AWS-AppConfig-Extension:133`

リージョン	ARN
米国西部 (北カリフォルニア)	arn:aws:lambda:us-west-1:958113053741:layer:AWS-AppConfig-Extension:223
米国西部 (オレゴン)	arn:aws:lambda:us-west-2:359756378197:layer:AWS-AppConfig-Extension:230
カナダ (中部)	arn:aws:lambda:ca-central-1:039592058896:layer:AWS-AppConfig-Extension:123
カナダ西部 (カルガリー)	arn:aws:lambda:ca-west-1:436199621743:layer:AWS-AppConfig-Extension:27
欧州 (フランクフルト)	arn:aws:lambda:eu-central-1:066940009817:layer:AWS-AppConfig-Extension:159
欧州 (チューリッヒ)	arn:aws:lambda:eu-central-2:758369105281:layer:AWS-AppConfig-Extension:77
欧州 (アイルランド)	arn:aws:lambda:eu-west-1:434848589818:layer:AWS-AppConfig-Extension:160
欧州 (ロンドン)	arn:aws:lambda:eu-west-2:282860088358:layer:AWS-AppConfig-Extension:121
欧州 (パリ)	arn:aws:lambda:eu-west-3:493207061005:layer:AWS-AppConfig-Extension:133

リージョン	ARN
欧州 (ストックホルム)	arn:aws:lambda:eu-north-1:646970417810:layer:AWS-AppConfig-Extension:225
欧州 (ミラノ)	arn:aws:lambda:eu-south-1:203683718741:layer:AWS-AppConfig-Extension:111
欧州 (スペイン)	arn:aws:lambda:eu-south-2:586093569114:layer:AWS-AppConfig-Extension:74
中国 (北京)	arn:aws-cn:lambda:cn-north-1:615057806174:layer:AWS-AppConfig-Extension:106
中国 (寧夏)	arn:aws-cn:lambda:cn-northwest-1:615084187847:layer:AWS-AppConfig-Extension:104
アジアパシフィック (香港)	arn:aws:lambda:ap-east-1:630222743974:layer:AWS-AppConfig-Extension:113
アジアパシフィック (東京)	arn:aws:lambda:ap-northeast-1:980059726660:layer:AWS-AppConfig-Extension:126
アジアパシフィック (ソウル)	arn:aws:lambda:ap-northeast-2:826293736237:layer:AWS-AppConfig-Extension:136
アジアパシフィック (大阪)	arn:aws:lambda:ap-northeast-3:706869817123:layer:AWS-AppConfig-Extension:130

リージョン	ARN
アジアパシフィック (シンガポール)	arn:aws:lambda:ap-southeast-1:421114256042:layer:AWS-AppConfig-Extension:134
アジアパシフィック (シドニー)	arn:aws:lambda:ap-southeast-2:080788657173:layer:AWS-AppConfig-Extension:165
アジアパシフィック (ジャカルタ)	arn:aws:lambda:ap-southeast-3:418787028745:layer:AWS-AppConfig-Extension:121
アジアパシフィック (メルボルン)	arn:aws:lambda:ap-southeast-4:307021474294:layer:AWS-AppConfig-Extension:49
アジアパシフィック (マレーシア)	arn:aws:lambda:ap-southeast-5:631746059939:layer:AWS-AppConfig-Extension:26
アジアパシフィック (ムンバイ)	arn:aws:lambda:ap-south-1:554480029851:layer:AWS-AppConfig-Extension:146
アジアパシフィック (ハイデラバード)	arn:aws:lambda:ap-south-2:489524808438:layer:AWS-AppConfig-Extension:75
南米 (サンパウロ)	arn:aws:lambda:sa-east-1:000010852771:layer:AWS-AppConfig-Extension:179
アフリカ (ケープタウン)	arn:aws:lambda:af-south-1:574348263942:layer:AWS-AppConfig-Extension:123

リージョン	ARN
イスラエル (テルアビブ)	arn:aws:lambda:il-central-1:895787185223:layer:AWS-AppConfig-Extension:52
中東 (アラブ首長国連邦)	arn:aws:lambda:me-central-1:662846165436:layer:AWS-AppConfig-Extension:91
中東 (バーレーン)	arn:aws:lambda:me-south-1:559955524753:layer:AWS-AppConfig-Extension:125
AWS GovCloud (米国東部)	arn:aws-us-gov:lambda:us-gov-east-1:946561847325:layer:AWS-AppConfig-Extension:80
AWS GovCloud (米国西部)	arn:aws-us-gov:lambda:us-gov-west-1:946746059096:layer:AWS-AppConfig-Extension:80

新しい拡張子に置き換えられた日付: 12/12/2024

バージョン 2.0.719

リージョン	ARN
米国東部 (バージニア北部)	arn:aws:lambda:us-east-1:027255383542:layer:AWS-AppConfig-Extension:173
米国東部 (オハイオ)	arn:aws:lambda:us-east-2:728743619870:layer:AWS-AppConfig-Extension:132

リージョン	ARN
米国西部 (北カリフォルニア)	arn:aws:lambda:us-west-1:958113053741:layer:AWS-AppConfig-Extension:221
米国西部 (オレゴン)	arn:aws:lambda:us-west-2:359756378197:layer:AWS-AppConfig-Extension:229
カナダ (中部)	arn:aws:lambda:ca-central-1:039592058896:layer:AWS-AppConfig-Extension:121
カナダ西部 (カルガリー)	arn:aws:lambda:ca-west-1:436199621743:layer:AWS-AppConfig-Extension:27
欧州 (フランクフルト)	arn:aws:lambda:eu-central-1:066940009817:layer:AWS-AppConfig-Extension:158
欧州 (チューリッヒ)	arn:aws:lambda:eu-central-2:758369105281:layer:AWS-AppConfig-Extension:75
欧州 (アイルランド)	arn:aws:lambda:eu-west-1:434848589818:layer:AWS-AppConfig-Extension:159
欧州 (ロンドン)	arn:aws:lambda:eu-west-2:282860088358:layer:AWS-AppConfig-Extension:120
欧州 (パリ)	arn:aws:lambda:eu-west-3:493207061005:layer:AWS-AppConfig-Extension:132

リージョン	ARN
欧州 (ストックホルム)	arn:aws:lambda:eu-north-1:646970417810:layer:AWS-AppConfig-Extension:224
欧州 (ミラノ)	arn:aws:lambda:eu-south-1:203683718741:layer:AWS-AppConfig-Extension:110
欧州 (スペイン)	arn:aws:lambda:eu-south-2:586093569114:layer:AWS-AppConfig-Extension:72
中国 (北京)	arn:aws-cn:lambda:cn-north-1:615057806174:layer:AWS-AppConfig-Extension:104
中国 (寧夏)	arn:aws-cn:lambda:cn-northwest-1:615084187847:layer:AWS-AppConfig-Extension:102
アジアパシフィック (香港)	arn:aws:lambda:ap-east-1:630222743974:layer:AWS-AppConfig-Extension:112
アジアパシフィック (東京)	arn:aws:lambda:ap-northeast-1:980059726660:layer:AWS-AppConfig-Extension:125
アジアパシフィック (ソウル)	arn:aws:lambda:ap-northeast-2:826293736237:layer:AWS-AppConfig-Extension:135
アジアパシフィック (大阪)	arn:aws:lambda:ap-northeast-3:706869817123:layer:AWS-AppConfig-Extension:129

リージョン	ARN
アジアパシフィック (シンガポール)	arn:aws:lambda:ap-southeast-1:421114256042:layer:AWS-AppConfig-Extension:132
アジアパシフィック (シドニー)	arn:aws:lambda:ap-southeast-2:080788657173:layer:AWS-AppConfig-Extension:164
アジアパシフィック (ジャカルタ)	arn:aws:lambda:ap-southeast-3:418787028745:layer:AWS-AppConfig-Extension:120
アジアパシフィック (メルボルン)	arn:aws:lambda:ap-southeast-4:307021474294:layer:AWS-AppConfig-Extension:48
アジアパシフィック (マレーシア)	arn:aws:lambda:ap-southeast-5:631746059939:layer:AWS-AppConfig-Extension:25
アジアパシフィック (ムンバイ)	arn:aws:lambda:ap-south-1:554480029851:layer:AWS-AppConfig-Extension:145
アジアパシフィック (ハイデラバード)	arn:aws:lambda:ap-south-2:489524808438:layer:AWS-AppConfig-Extension:74
南米 (サンパウロ)	arn:aws:lambda:sa-east-1:000010852771:layer:AWS-AppConfig-Extension:178
アフリカ (ケープタウン)	arn:aws:lambda:af-south-1:574348263942:layer:AWS-AppConfig-Extension:122

リージョン	ARN
イスラエル (テルアビブ)	arn:aws:lambda:il-central-1:895787185223:layer:AWS-AppConfig-Extension:50
中東 (アラブ首長国連邦)	arn:aws:lambda:me-central-1:662846165436:layer:AWS-AppConfig-Extension:90
中東 (バーレーン)	arn:aws:lambda:me-south-1:559955524753:layer:AWS-AppConfig-Extension:124
AWS GovCloud (米国東部)	arn:aws-us-gov:lambda:us-gov-east-1:946561847325:layer:AWS-AppConfig-Extension:79
AWS GovCloud (米国西部)	arn:aws-us-gov:lambda:us-gov-west-1:946746059096:layer:AWS-AppConfig-Extension:79

新しい拡張子に置き換えられた日付: 2024 年 8 月 8 日

バージョン 2.0.678

リージョン	ARN
米国東部 (バージニア北部)	arn:aws:lambda:us-east-1:027255383542:layer:AWS-AppConfig-Extension:167
米国東部 (オハイオ)	arn:aws:lambda:us-east-2:728743619870:layer:AWS-AppConfig-Extension:126

リージョン	ARN
米国西部 (北カリフォルニア)	arn:aws:lambda:us-west-1:958113053741:layer:AWS-AppConfig-Extension:213
米国西部 (オレゴン)	arn:aws:lambda:us-west-2:359756378197:layer:AWS-AppConfig-Extension:223
カナダ (中部)	arn:aws:lambda:ca-central-1:039592058896:layer:AWS-AppConfig-Extension:116
カナダ西部 (カルガリー)	arn:aws:lambda:ca-west-1:436199621743:layer:AWS-AppConfig-Extension:21
欧州 (フランクフルト)	arn:aws:lambda:eu-central-1:066940009817:layer:AWS-AppConfig-Extension:152
欧州 (チューリッヒ)	arn:aws:lambda:eu-central-2:758369105281:layer:AWS-AppConfig-Extension:70
欧州 (アイルランド)	arn:aws:lambda:eu-west-1:434848589818:layer:AWS-AppConfig-Extension:153
欧州 (ロンドン)	arn:aws:lambda:eu-west-2:282860088358:layer:AWS-AppConfig-Extension:114
欧州 (パリ)	arn:aws:lambda:eu-west-3:493207061005:layer:AWS-AppConfig-Extension:126

リージョン	ARN
欧州 (ストックホルム)	arn:aws:lambda:eu-north-1:646970417810:layer:AWS-AppConfig-Extension:218
欧州 (ミラノ)	arn:aws:lambda:eu-south-1:203683718741:layer:AWS-AppConfig-Extension:104
欧州 (スペイン)	arn:aws:lambda:eu-south-2:586093569114:layer:AWS-AppConfig-Extension:67
中国 (北京)	arn:aws-cn:lambda:cn-north-1:615057806174:layer:AWS-AppConfig-Extension:99
中国 (寧夏)	arn:aws-cn:lambda:cn-northwest-1:615084187847:layer:AWS-AppConfig-Extension:97
アジアパシフィック (香港)	arn:aws:lambda:ap-east-1:630222743974:layer:AWS-AppConfig-Extension:106
アジアパシフィック (東京)	arn:aws:lambda:ap-northeast-1:980059726660:layer:AWS-AppConfig-Extension:119
アジアパシフィック (ソウル)	arn:aws:lambda:ap-northeast-2:826293736237:layer:AWS-AppConfig-Extension:129
アジアパシフィック (大阪)	arn:aws:lambda:ap-northeast-3:706869817123:layer:AWS-AppConfig-Extension:123

リージョン	ARN
アジアパシフィック (シンガポール)	arn:aws:lambda:ap-southeast-1:421114256042:layer:AWS-AppConfig-Extension:127
アジアパシフィック (シドニー)	arn:aws:lambda:ap-southeast-2:080788657173:layer:AWS-AppConfig-Extension:158
アジアパシフィック (ジャカルタ)	arn:aws:lambda:ap-southeast-3:418787028745:layer:AWS-AppConfig-Extension:114
アジアパシフィック (メルボルン)	arn:aws:lambda:ap-southeast-4:307021474294:layer:AWS-AppConfig-Extension:42
アジアパシフィック (ムンバイ)	arn:aws:lambda:ap-south-1:554480029851:layer:AWS-AppConfig-Extension:139
アジアパシフィック (ハイデラバード)	arn:aws:lambda:ap-south-2:489524808438:layer:AWS-AppConfig-Extension:68
南米 (サンパウロ)	arn:aws:lambda:sa-east-1:000010852771:layer:AWS-AppConfig-Extension:172
アフリカ (ケープタウン)	arn:aws:lambda:af-south-1:574348263942:layer:AWS-AppConfig-Extension:116
イスラエル (テルアビブ)	arn:aws:lambda:il-central-1:895787185223:layer:AWS-AppConfig-Extension:45

リージョン	ARN
中東 (アラブ首長国連邦)	arn:aws:lambda:me-central-1:662846165436:layer:AWS-AppConfig-Extension:84
中東 (バーレーン)	arn:aws:lambda:me-south-1:559955524753:layer:AWS-AppConfig-Extension:118
AWS GovCloud (米国東部)	arn:aws-us-gov:lambda:us-gov-east-1:946561847325:layer:AWS-AppConfig-Extension:73
AWS GovCloud (米国西部)	arn:aws-us-gov:lambda:us-gov-west-1:946746059096:layer:AWS-AppConfig-Extension:73

新しい拡張子に置き換えられた日付: 2024 年 7 月 23 日

バージョン 2.0.501

リージョン	ARN
米国東部 (バージニア北部)	arn:aws:lambda:us-east-1:027255383542:layer:AWS-AppConfig-Extension:153
米国東部 (オハイオ)	arn:aws:lambda:us-east-2:728743619870:layer:AWS-AppConfig-Extension:112
米国西部 (北カリフォルニア)	arn:aws:lambda:us-west-1:958113053741:layer:AWS-AppConfig-Extension:195

リージョン	ARN
米国西部 (オレゴン)	arn:aws:lambda:us-west-2:359756378197:layer:AWS-AppConfig-Extension:210
カナダ (中部)	arn:aws:lambda:ca-central-1:039592058896:layer:AWS-AppConfig-Extension:101
欧州 (フランクフルト)	arn:aws:lambda:eu-central-1:066940009817:layer:AWS-AppConfig-Extension:136
欧州 (チューリッヒ)	arn:aws:lambda:eu-central-2:758369105281:layer:AWS-AppConfig-Extension:53
欧州 (アイルランド)	arn:aws:lambda:eu-west-1:434848589818:layer:AWS-AppConfig-Extension:144
欧州 (ロンドン)	arn:aws:lambda:eu-west-2:282860088358:layer:AWS-AppConfig-Extension:99
欧州 (パリ)	arn:aws:lambda:eu-west-3:493207061005:layer:AWS-AppConfig-Extension:111
欧州 (ストックホルム)	arn:aws:lambda:eu-north-1:646970417810:layer:AWS-AppConfig-Extension:201
欧州 (ミラノ)	arn:aws:lambda:eu-south-1:203683718741:layer:AWS-AppConfig-Extension:89

リージョン	ARN
欧州 (スペイン)	arn:aws:lambda:eu-south-2:586093569114:layer:AWS-AppConfig-Extension:50
中国 (北京)	arn:aws-cn:lambda:cn-north-1:615057806174:layer:AWS-AppConfig-Extension:85
中国 (寧夏)	arn:aws-cn:lambda:cn-northwest-1:615084187847:layer:AWS-AppConfig-Extension:83
アジアパシフィック (香港)	arn:aws:lambda:ap-east-1:630222743974:layer:AWS-AppConfig-Extension:91
アジアパシフィック (東京)	arn:aws:lambda:ap-northeast-1:980059726660:layer:AWS-AppConfig-Extension:104
アジアパシフィック (ソウル)	arn:aws:lambda:ap-northeast-2:826293736237:layer:AWS-AppConfig-Extension:114
アジアパシフィック (大阪)	arn:aws:lambda:ap-northeast-3:706869817123:layer:AWS-AppConfig-Extension:107
アジアパシフィック (シンガポール)	arn:aws:lambda:ap-southeast-1:421114256042:layer:AWS-AppConfig-Extension:112
アジアパシフィック (シドニー)	arn:aws:lambda:ap-southeast-2:080788657173:layer:AWS-AppConfig-Extension:142

リージョン	ARN
アジアパシフィック (ジャカルタ)	arn:aws:lambda:ap-southeast-3:418787028745:layer:AWS-AppConfig-Extension:98
アジアパシフィック (メルボルン)	arn:aws:lambda:ap-southeast-4:307021474294:layer:AWS-AppConfig-Extension:26
アジアパシフィック (ムンバイ)	arn:aws:lambda:ap-south-1:554480029851:layer:AWS-AppConfig-Extension:125
アジアパシフィック (ハイデラバード)	arn:aws:lambda:ap-south-2:489524808438:layer:AWS-AppConfig-Extension:53
南米 (サンパウロ)	arn:aws:lambda:sa-east-1:000010852771:layer:AWS-AppConfig-Extension:155
アフリカ (ケープタウン)	arn:aws:lambda:af-south-1:574348263942:layer:AWS-AppConfig-Extension:102
イスラエル (テルアビブ)	arn:aws:lambda:il-central-1:895787185223:layer:AWS-AppConfig-Extension:28
中東 (アラブ首長国連邦)	arn:aws:lambda:me-central-1:662846165436:layer:AWS-AppConfig-Extension:68
中東 (バーレーン)	arn:aws:lambda:me-south-1:559955524753:layer:AWS-AppConfig-Extension:103

リージョン	ARN
AWS GovCloud (米国東部)	arn:aws-us-gov:lambda:us-gov-east-1:946561847325:layer:AWS-AppConfig-Extension:59
AWS GovCloud (米国西部)	arn:aws-us-gov:lambda:us-gov-west-1:946746059096:layer:AWS-AppConfig-Extension:59

新しい拡張子に置き換えられた日付: 2024 年 7 月 1 日

バージョン 2.0.358

リージョン	ARN
米国東部 (バージニア北部)	arn:aws:lambda:us-east-1:027255383542:layer:AWS-AppConfig-Extension:128
米国東部 (オハイオ)	arn:aws:lambda:us-east-2:728743619870:layer:AWS-AppConfig-Extension:93
米国西部 (北カリフォルニア)	arn:aws:lambda:us-west-1:958113053741:layer:AWS-AppConfig-Extension:141
米国西部 (オレゴン)	arn:aws:lambda:us-west-2:359756378197:layer:AWS-AppConfig-Extension:161
カナダ (中部)	arn:aws:lambda:ca-central-1:039592058896:layer:AWS-AppConfig-Extension:93

リージョン	ARN
欧州 (フランクフルト)	arn:aws:lambda:eu-central-1:066940009817:layer:AWS-AppConfig-Extension:106
欧州 (チューリッヒ)	arn:aws:lambda:eu-central-2:758369105281:layer:AWS-AppConfig-Extension:47
欧州 (アイルランド)	arn:aws:lambda:eu-west-1:434848589818:layer:AWS-AppConfig-Extension:125
欧州 (ロンドン)	arn:aws:lambda:eu-west-2:282860088358:layer:AWS-AppConfig-Extension:93
欧州 (パリ)	arn:aws:lambda:eu-west-3:493207061005:layer:AWS-AppConfig-Extension:98
欧州 (ストックホルム)	arn:aws:lambda:eu-north-1:646970417810:layer:AWS-AppConfig-Extension:159
欧州 (ミラノ)	arn:aws:lambda:eu-south-1:203683718741:layer:AWS-AppConfig-Extension:83
欧州 (スペイン)	arn:aws:lambda:eu-south-2:586093569114:layer:AWS-AppConfig-Extension:44
中国 (北京)	arn:aws-cn:lambda:cn-north-1:615057806174:layer:AWS-AppConfig-Extension:76

リージョン	ARN
中国 (寧夏)	arn:aws-cn:lambda:cn-northwest-1:615084187847:layer:AWS-AppConfig-Extension:76
アジアパシフィック (香港)	arn:aws:lambda:ap-east-1:630222743974:layer:AWS-AppConfig-Extension:83
アジアパシフィック (東京)	arn:aws:lambda:ap-northeast-1:980059726660:layer:AWS-AppConfig-Extension:98
アジアパシフィック (ソウル)	arn:aws:lambda:ap-northeast-2:826293736237:layer:AWS-AppConfig-Extension:108
アジアパシフィック (大阪)	arn:aws:lambda:ap-northeast-3:706869817123:layer:AWS-AppConfig-Extension:101
アジアパシフィック (シンガポール)	arn:aws:lambda:ap-southeast-1:421114256042:layer:AWS-AppConfig-Extension:106
アジアパシフィック (シドニー)	arn:aws:lambda:ap-southeast-2:080788657173:layer:AWS-AppConfig-Extension:106
アジアパシフィック (ジャカルタ)	arn:aws:lambda:ap-southeast-3:418787028745:layer:AWS-AppConfig-Extension:79
アジアパシフィック (メルボルン)	arn:aws:lambda:ap-southeast-4:307021474294:layer:AWS-AppConfig-Extension:20

リージョン	ARN
アジアパシフィック (ムンバイ)	arn:aws:lambda:ap-south-1:554480029851:layer:AWS-AppConfig-Extension:107
アジアパシフィック (ハイデラバード)	arn:aws:lambda:ap-south-2:489524808438:layer:AWS-AppConfig-Extension:47
南米 (サンパウロ)	arn:aws:lambda:sa-east-1:000010852771:layer:AWS-AppConfig-Extension:128
アフリカ (ケープタウン)	arn:aws:lambda:af-south-1:574348263942:layer:AWS-AppConfig-Extension:83
イスラエル (テルアビブ)	arn:aws:lambda:il-central-1:895787185223:layer:AWS-AppConfig-Extension:22
中東 (アラブ首長国連邦)	arn:aws:lambda:me-central-1:662846165436:layer:AWS-AppConfig-Extension:49
中東 (バーレーン)	arn:aws:lambda:me-south-1:559955524753:layer:AWS-AppConfig-Extension:85
AWS GovCloud (米国東部)	arn:aws-us-gov:lambda:us-gov-east-1:946561847325:layer:AWS-AppConfig-Extension:54
AWS GovCloud (米国西部)	arn:aws-us-gov:lambda:us-gov-west-1:946746059096:layer:AWS-AppConfig-Extension:54

新しい拡張子に置き換えられた日付: 2023 年 12 月 1 日

バージョン 2.0.181

リージョン	ARN
米国東部 (バージニア北部)	arn:aws:lambda:us-east-1:027255383542:layer:AWS-AppConfig-Extension:113
米国東部 (オハイオ)	arn:aws:lambda:us-east-2:728743619870:layer:AWS-AppConfig-Extension:81
米国西部 (北カリフォルニア)	arn:aws:lambda:us-west-1:958113053741:layer:AWS-AppConfig-Extension:124
米国西部 (オレゴン)	arn:aws:lambda:us-west-2:359756378197:layer:AWS-AppConfig-Extension:146
カナダ (中部)	arn:aws:lambda:ca-central-1:039592058896:layer:AWS-AppConfig-Extension:81
欧州 (フランクフルト)	arn:aws:lambda:eu-central-1:066940009817:layer:AWS-AppConfig-Extension:93
欧州 (チューリッヒ)	arn:aws:lambda:eu-central-2:758369105281:layer:AWS-AppConfig-Extension:32
欧州 (アイルランド)	arn:aws:lambda:eu-west-1:434848589818:layer:AWS-AppConfig-Extension:110

リージョン	ARN
欧州 (ロンドン)	arn:aws:lambda:eu-west-2:282860088358:layer:AWS-AppConfig-Extension:81
欧州 (パリ)	arn:aws:lambda:eu-west-3:493207061005:layer:AWS-AppConfig-Extension:82
欧州 (ストックホルム)	arn:aws:lambda:eu-north-1:646970417810:layer:AWS-AppConfig-Extension:142
欧州 (ミラノ)	arn:aws:lambda:eu-south-1:203683718741:layer:AWS-AppConfig-Extension:73
欧州 (スペイン)	arn:aws:lambda:eu-south-2:586093569114:layer:AWS-AppConfig-Extension:29
中国 (北京)	arn:aws-cn:lambda:cn-north-1:615057806174:layer:AWS-AppConfig-Extension:68
中国 (寧夏)	arn:aws-cn:lambda:cn-northwest-1:615084187847:layer:AWS-AppConfig-Extension:68
アジアパシフィック (香港)	arn:aws:lambda:ap-east-1:630222743974:layer:AWS-AppConfig-Extension:73
アジアパシフィック (東京)	arn:aws:lambda:ap-northeast-1:980059726660:layer:AWS-AppConfig-Extension:84

リージョン	ARN
アジアパシフィック (ソウル)	arn:aws:lambda:ap-northeast-2:826293736237:layer:AWS-AppConfig-Extension:93
アジアパシフィック (大阪)	arn:aws:lambda:ap-northeast-3:706869817123:layer:AWS-AppConfig-Extension:86
アジアパシフィック (シンガポール)	arn:aws:lambda:ap-southeast-1:421114256042:layer:AWS-AppConfig-Extension:91
アジアパシフィック (シドニー)	arn:aws:lambda:ap-southeast-2:080788657173:layer:AWS-AppConfig-Extension:93
アジアパシフィック (ジャカルタ)	arn:aws:lambda:ap-southeast-3:418787028745:layer:AWS-AppConfig-Extension:64
アジアパシフィック (メルボルン)	arn:aws:lambda:ap-southeast-4:307021474294:layer:AWS-AppConfig-Extension:5
アジアパシフィック (ムンバイ)	arn:aws:lambda:ap-south-1:554480029851:layer:AWS-AppConfig-Extension:94
アジアパシフィック (ハイデラバード)	arn:aws:lambda:ap-south-2:489524808438:layer:AWS-AppConfig-Extension:32
南米 (サンパウロ)	arn:aws:lambda:sa-east-1:000010852771:layer:AWS-AppConfig-Extension:113

リージョン	ARN
アフリカ (ケープタウン)	`arn:aws:lambda:af-south-1:574348263942:layer:AWS-AppConfig-Extension:73`
イスラエル (テルアビブ)	`arn:aws:lambda:il-central-1:895787185223:layer:AWS-AppConfig-Extension:7`
中東 (アラブ首長国連邦)	`arn:aws:lambda:me-central-1:662846165436:layer:AWS-AppConfig-Extension:34`
中東 (バーレーン)	`arn:aws:lambda:me-south-1:559955524753:layer:AWS-AppConfig-Extension:73`
AWS GovCloud (米国東部)	`arn:aws-us-gov:lambda:us-gov-east-1:946561847325:layer:AWS-AppConfig-Extension:46`
AWS GovCloud (米国西部)	`arn:aws-us-gov:lambda:us-gov-west-1:946746059096:layer:AWS-AppConfig-Extension:46`

新しい拡張子に置き換えられた日付: 2023 年 8 月 14 日

バージョン 2.0.165

リージョン	ARN
米国東部 (バージニア北部)	`arn:aws:lambda:us-east-1:027255383542:layer:AWS-AppConfig-Extension:110`

リージョン	ARN
米国東部 (オハイオ)	arn:aws:lambda:us-east-2:728743619870:layer:AWS-AppConfig-Extension:79
米国西部 (北カリフォルニア)	arn:aws:lambda:us-west-1:958113053741:layer:AWS-AppConfig-Extension:121
米国西部 (オレゴン)	arn:aws:lambda:us-west-2:359756378197:layer:AWS-AppConfig-Extension:143
カナダ (中部)	arn:aws:lambda:ca-central-1:039592058896:layer:AWS-AppConfig-Extension:79
欧州 (フランクフルト)	arn:aws:lambda:eu-central-1:066940009817:layer:AWS-AppConfig-Extension:91
欧州 (チューリッヒ)	arn:aws:lambda:eu-central-2:758369105281:layer:AWS-AppConfig-Extension:29
欧州 (アイルランド)	arn:aws:lambda:eu-west-1:434848589818:layer:AWS-AppConfig-Extension:108
欧州 (ロンドン)	arn:aws:lambda:eu-west-2:282860088358:layer:AWS-AppConfig-Extension:79
欧州 (パリ)	arn:aws:lambda:eu-west-3:493207061005:layer:AWS-AppConfig-Extension:80

リージョン	ARN
欧州 (ストックホルム)	arn:aws:lambda:eu-north-1:646970417810:layer:AWS-AppConfig-Extension:139
欧州 (ミラノ)	arn:aws:lambda:eu-south-1:203683718741:layer:AWS-AppConfig-Extension:71
欧州 (スペイン)	arn:aws:lambda:eu-south-2:586093569114:layer:AWS-AppConfig-Extension:26
中国 (北京)	arn:aws-cn:lambda:cn-north-1:615057806174:layer:AWS-AppConfig-Extension:66
中国 (寧夏)	arn:aws-cn:lambda:cn-northwest-1:615084187847:layer:AWS-AppConfig-Extension:66
アジアパシフィック (香港)	arn:aws:lambda:ap-east-1:630222743974:layer:AWS-AppConfig-Extension:71
アジアパシフィック (東京)	arn:aws:lambda:ap-northeast-1:980059726660:layer:AWS-AppConfig-Extension:82
アジアパシフィック (ソウル)	arn:aws:lambda:ap-northeast-2:826293736237:layer:AWS-AppConfig-Extension:91
アジアパシフィック (大阪)	arn:aws:lambda:ap-northeast-3:706869817123:layer:AWS-AppConfig-Extension:84

リージョン	ARN
アジアパシフィック (シンガポール)	arn:aws:lambda:ap-southeast-1:421114256042:layer:AWS-AppConfig-Extension:89
アジアパシフィック (シドニー)	arn:aws:lambda:ap-southeast-2:080788657173:layer:AWS-AppConfig-Extension:91
アジアパシフィック (ジャカルタ)	arn:aws:lambda:ap-southeast-3:418787028745:layer:AWS-AppConfig-Extension:60
アジアパシフィック (メルボルン)	arn:aws:lambda:ap-southeast-4:307021474294:layer:AWS-AppConfig-Extension:2
アジアパシフィック (ムンバイ)	arn:aws:lambda:ap-south-1:554480029851:layer:AWS-AppConfig-Extension:92
アジアパシフィック (ハイデラバード)	arn:aws:lambda:ap-south-2:489524808438:layer:AWS-AppConfig-Extension:29
南米 (サンパウロ)	arn:aws:lambda:sa-east-1:000010852771:layer:AWS-AppConfig-Extension:110
アフリカ (ケープタウン)	arn:aws:lambda:af-south-1:574348263942:layer:AWS-AppConfig-Extension:71
中東 (アラブ首長国連邦)	arn:aws:lambda:me-central-1:662846165436:layer:AWS-AppConfig-Extension:31

リージョン	ARN
中東 (バーレーン)	arn:aws:lambda:me-south-1:559955524753:layer:AWS-AppConfig-Extension:71
AWS GovCloud (米国東部)	arn:aws-us-gov:lambda:us-gov-east-1:946561847325:layer:AWS-AppConfig-Extension:44
AWS GovCloud (米国西部)	arn:aws-us-gov:lambda:us-gov-west-1:946746059096:layer:AWS-AppConfig-Extension:44

新しい拡張子に置き換えられた日付: 2023 年 2 月 21 日

バージョン 2.0.122

リージョン	ARN
米国東部 (バージニア北部)	arn:aws:lambda:us-east-1:027255383542:layer:AWS-AppConfig-Extension:82
米国東部 (オハイオ)	arn:aws:lambda:us-east-2:728743619870:layer:AWS-AppConfig-Extension:59
米国西部 (北カリフォルニア)	arn:aws:lambda:us-west-1:958113053741:layer:AWS-AppConfig-Extension:93
米国西部 (オレゴン)	arn:aws:lambda:us-west-2:359756378197:layer:AWS-AppConfig-Extension:114

リージョン	ARN
カナダ (中部)	arn:aws:lambda:ca-central-1:039592058896:layer:AWS-AppConfig-Extension:59
欧州 (フランクフルト)	arn:aws:lambda:eu-central-1:066940009817:layer:AWS-AppConfig-Extension:70
欧州 (アイルランド)	arn:aws:lambda:eu-west-1:434848589818:layer:AWS-AppConfig-Extension:82
欧州 (ロンドン)	arn:aws:lambda:eu-west-2:282860088358:layer:AWS-AppConfig-Extension:59
欧州 (パリ)	arn:aws:lambda:eu-west-3:493207061005:layer:AWS-AppConfig-Extension:60
欧州 (ストックホルム)	arn:aws:lambda:eu-north-1:646970417810:layer:AWS-AppConfig-Extension:111
欧州 (ミラノ)	arn:aws:lambda:eu-south-1:203683718741:layer:AWS-AppConfig-Extension:54
中国 (北京)	arn:aws-cn:lambda:cn-north-1:615057806174:layer:AWS-AppConfig-Extension:52
中国 (寧夏)	arn:aws-cn:lambda:cn-northwest-1:615084187847:layer:AWS-AppConfig-Extension:52

リージョン	ARN
アジアパシフィック (香港)	arn:aws:lambda:ap-east-1:630222743974:layer:AWS-AppConfig-Extension:54
アジアパシフィック (東京)	arn:aws:lambda:ap-northeast-1:980059726660:layer:AWS-AppConfig-Extension:62
アジアパシフィック (ソウル)	arn:aws:lambda:ap-northeast-2:826293736237:layer:AWS-AppConfig-Extension:70
アジアパシフィック (大阪)	arn:aws:lambda:ap-northeast-3:706869817123:layer:AWS-AppConfig-Extension:59
アジアパシフィック (シンガポール)	arn:aws:lambda:ap-southeast-1:421114256042:layer:AWS-AppConfig-Extension:64
アジアパシフィック (シドニー)	arn:aws:lambda:ap-southeast-2:080788657173:layer:AWS-AppConfig-Extension:70
アジアパシフィック (ジャカルタ)	arn:aws:lambda:ap-southeast-3:418787028745:layer:AWS-AppConfig-Extension:37
アジアパシフィック (ムンバイ)	arn:aws:lambda:ap-south-1:554480029851:layer:AWS-AppConfig-Extension:71
南米 (サンパウロ)	arn:aws:lambda:sa-east-1:000010852771:layer:AWS-AppConfig-Extension:82

リージョン	ARN
アフリカ (ケープタウン)	`arn:aws:lambda:af-south-1:574348263942:layer:AWS-AppConfig-Extension:54`
中東 (バーレーン)	`arn:aws:lambda:me-south-1:559955524753:layer:AWS-AppConfig-Extension:54`
AWS GovCloud (米国東部)	`arn:aws-us-gov:lambda:us-gov-east-1:946561847325:layer:AWS-AppConfig-Extension:29`
AWS GovCloud (米国西部)	`arn:aws-us-gov:lambda:us-gov-west-1:946746059096:layer:AWS-AppConfig-Extension:29`

新しい拡張子に置き換えられた日付: 2022 年 8 月 23 日

バージョン 2.0.58

リージョン	ARN
米国東部 (バージニア北部)	`arn:aws:lambda:us-east-1:027255383542:layer:AWS-AppConfig-Extension:69`
米国東部 (オハイオ)	`arn:aws:lambda:us-east-2:728743619870:layer:AWS-AppConfig-Extension:50`
米国西部 (北カリフォルニア)	`arn:aws:lambda:us-west-1:958113053741:layer:AWS-AppConfig-Extension:78`

リージョン	ARN
米国西部 (オレゴン)	arn:aws:lambda:us-west-2:359756378197:layer:AWS-AppConfig-Extension:101
カナダ (中部)	arn:aws:lambda:ca-central-1:039592058896:layer:AWS-AppConfig-Extension:50
欧州 (フランクフルト)	arn:aws:lambda:eu-central-1:066940009817:layer:AWS-AppConfig-Extension:59
欧州 (アイルランド)	arn:aws:lambda:eu-west-1:434848589818:layer:AWS-AppConfig-Extension:69
欧州 (ロンドン)	arn:aws:lambda:eu-west-2:282860088358:layer:AWS-AppConfig-Extension:50
欧州 (パリ)	arn:aws:lambda:eu-west-3:493207061005:layer:AWS-AppConfig-Extension:51
欧州 (ストックホルム)	arn:aws:lambda:eu-north-1:646970417810:layer:AWS-AppConfig-Extension:98
欧州 (ミラノ)	arn:aws:lambda:eu-south-1:203683718741:layer:AWS-AppConfig-Extension:47
中国 (北京)	arn:aws-cn:lambda:cn-north-1:615057806174:layer:AWS-AppConfig-Extension:46

リージョン	ARN
中国 (寧夏)	arn:aws-cn:lambda:cn-northwest-1:615084187847:layer:AWS-AppConfig-Extension:46
アジアパシフィック (香港)	arn:aws:lambda:ap-east-1:630222743974:layer:AWS-AppConfig-Extension:47
アジアパシフィック (東京)	arn:aws:lambda:ap-northeast-1:980059726660:layer:AWS-AppConfig-Extension:49
アジアパシフィック (ソウル)	arn:aws:lambda:ap-northeast-2:826293736237:layer:AWS-AppConfig-Extension:59
アジアパシフィック (大阪)	arn:aws:lambda:ap-northeast-3:706869817123:layer:AWS-AppConfig-Extension:46
アジアパシフィック (シンガポール)	arn:aws:lambda:ap-southeast-1:421114256042:layer:AWS-AppConfig-Extension:51
アジアパシフィック (シドニー)	arn:aws:lambda:ap-southeast-2:080788657173:layer:AWS-AppConfig-Extension:59
アジアパシフィック (ジャカルタ)	arn:aws:lambda:ap-southeast-3:418787028745:layer:AWS-AppConfig-Extension:24
アジアパシフィック (ムンバイ)	arn:aws:lambda:ap-south-1:554480029851:layer:AWS-AppConfig-Extension:60

リージョン	ARN
南米 (サンパウロ)	arn:aws:lambda:sa-east-1:000010852771:layer:AWS-AppConfig-Extension:69
アフリカ (ケープタウン)	arn:aws:lambda:af-south-1:574348263942:layer:AWS-AppConfig-Extension:47
中東 (バーレーン)	arn:aws:lambda:me-south-1:559955524753:layer:AWS-AppConfig-Extension:47
AWS GovCloud (米国東部)	arn:aws-us-gov:lambda:us-gov-east-1:946561847325:layer:AWS-AppConfig-Extension:23
AWS GovCloud (米国西部)	arn:aws-us-gov:lambda:us-gov-west-1:946746059096:layer:AWS-AppConfig-Extension:23

新しい拡張子に置き換えられた日付: 2022 年 4 月 21 日

バージョン 2.0.45

リージョン	ARN
米国東部 (バージニア北部)	arn:aws:lambda:us-east-1:027255383542:layer:AWS-AppConfig-Extension:68
米国東部 (オハイオ)	arn:aws:lambda:us-east-2:728743619870:layer:AWS-AppConfig-Extension:49

リージョン	ARN
米国西部 (北カリフォルニア)	arn:aws:lambda:us-west-1:958113053741:layer:AWS-AppConfig-Extension:77
米国西部 (オレゴン)	arn:aws:lambda:us-west-2:359756378197:layer:AWS-AppConfig-Extension:100
カナダ (中部)	arn:aws:lambda:ca-central-1:039592058896:layer:AWS-AppConfig-Extension:49
欧州 (フランクフルト)	arn:aws:lambda:eu-central-1:066940009817:layer:AWS-AppConfig-Extension:58
欧州 (アイルランド)	arn:aws:lambda:eu-west-1:434848589818:layer:AWS-AppConfig-Extension:68
欧州 (ロンドン)	arn:aws:lambda:eu-west-2:282860088358:layer:AWS-AppConfig-Extension:49
欧州 (パリ)	arn:aws:lambda:eu-west-3:493207061005:layer:AWS-AppConfig-Extension:50
欧州 (ストックホルム)	arn:aws:lambda:eu-north-1:646970417810:layer:AWS-AppConfig-Extension:97
欧州 (ミラノ)	arn:aws:lambda:eu-south-1:203683718741:layer:AWS-AppConfig-Extension:46

リージョン	ARN
中国 (北京)	arn:aws-cn:lambda:cn-north-1:615057806174:layer:AWS-AppConfig-Extension:45
中国 (寧夏)	arn:aws-cn:lambda:cn-northwest-1:615084187847:layer:AWS-AppConfig-Extension:45
アジアパシフィック (香港)	arn:aws:lambda:ap-east-1:630222743974:layer:AWS-AppConfig-Extension:46
アジアパシフィック (東京)	arn:aws:lambda:ap-northeast-1:980059726660:layer:AWS-AppConfig-Extension:48
アジアパシフィック (ソウル)	arn:aws:lambda:ap-northeast-2:826293736237:layer:AWS-AppConfig-Extension:58
アジアパシフィック (大阪)	arn:aws:lambda:ap-northeast-3:706869817123:layer:AWS-AppConfig-Extension:45
アジアパシフィック (シンガポール)	arn:aws:lambda:ap-southeast-1:421114256042:layer:AWS-AppConfig-Extension:50
アジアパシフィック (シドニー)	arn:aws:lambda:ap-southeast-2:080788657173:layer:AWS-AppConfig-Extension:58
アジアパシフィック (ジャカルタ)	arn:aws:lambda:ap-southeast-3:418787028745:layer:AWS-AppConfig-Extension:23

リージョン	ARN
アジアパシフィック (ムンバイ)	`arn:aws:lambda:ap-south-1:554480029851:layer:AWS-AppConfig-Extension:59`
南米 (サンパウロ)	`arn:aws:lambda:sa-east-1:000010852771:layer:AWS-AppConfig-Extension:68`
アフリカ (ケープタウン)	`arn:aws:lambda:af-south-1:574348263942:layer:AWS-AppConfig-Extension:46`
中東 (バーレーン)	`arn:aws:lambda:me-south-1:559955524753:layer:AWS-AppConfig-Extension:46`
AWS GovCloud (米国東部)	`arn:aws-us-gov:lambda:us-gov-east-1:946561847325:layer:AWS-AppConfig-Extension:22`
AWS GovCloud (米国西部)	`arn:aws-us-gov:lambda:us-gov-west-1:946746059096:layer:AWS-AppConfig-Extension:22`

新しい拡張子に置き換えられた日付: 2022 年 3 月 15 日

バージョン 2.0.30

リージョン	ARN
米国東部 (バージニア北部)	`arn:aws:lambda:us-east-1:027255383542:layer:AWS-AppConfig-Extension:61`

リージョン	ARN
米国東部 (オハイオ)	arn:aws:lambda:us-east-2:728743619870:layer:AWS-AppConfig-Extension:47
米国西部 (北カリフォルニア)	arn:aws:lambda:us-west-1:958113053741:layer:AWS-AppConfig-Extension:61
米国西部 (オレゴン)	arn:aws:lambda:us-west-2:359756378197:layer:AWS-AppConfig-Extension:89
カナダ (中部)	arn:aws:lambda:ca-central-1:039592058896:layer:AWS-AppConfig-Extension:47
欧州 (フランクフルト)	arn:aws:lambda:eu-central-1:066940009817:layer:AWS-AppConfig-Extension:54
欧州 (アイルランド)	arn:aws:lambda:eu-west-1:434848589818:layer:AWS-AppConfig-Extension:59
欧州 (ロンドン)	arn:aws:lambda:eu-west-2:282860088358:layer:AWS-AppConfig-Extension:47
欧州 (パリ)	arn:aws:lambda:eu-west-3:493207061005:layer:AWS-AppConfig-Extension:48
欧州 (ストックホルム)	arn:aws:lambda:eu-north-1:646970417810:layer:AWS-AppConfig-Extension:86

リージョン	ARN
欧州 (ミラノ)	arn:aws:lambda:eu-south-1:203683718741:layer:AWS-AppConfig-Extension:44
中国 (北京)	arn:aws-cn:lambda:cn-north-1:615057806174:layer:AWS-AppConfig-Extension:43
中国 (寧夏)	arn:aws-cn:lambda:cn-northwest-1:615084187847:layer:AWS-AppConfig-Extension:43
アジアパシフィック (香港)	arn:aws:lambda:ap-east-1:630222743974:layer:AWS-AppConfig-Extension:44
アジアパシフィック (東京)	arn:aws:lambda:ap-northeast-1:980059726660:layer:AWS-AppConfig-Extension:45
アジアパシフィック (大阪)	arn:aws:lambda:ap-northeast-3:706869817123:layer:AWS-AppConfig-Extension:42
アジアパシフィック (ソウル)	arn:aws:lambda:ap-northeast-2:826293736237:layer:AWS-AppConfig-Extension:54
アジアパシフィック (シンガポール)	arn:aws:lambda:ap-southeast-1:421114256042:layer:AWS-AppConfig-Extension:45
アジアパシフィック (シドニー)	arn:aws:lambda:ap-southeast-2:080788657173:layer:AWS-AppConfig-Extension:54

リージョン	ARN
アジアパシフィック (ジャカルタ)	`arn:aws:lambda:ap-southeast-3:418787028745:layer:AWS-AppConfig-Extension:13`
アジアパシフィック (ムンバイ)	`arn:aws:lambda:ap-south-1:554480029851:layer:AWS-AppConfig-Extension:55`
南米 (サンパウロ)	`arn:aws:lambda:sa-east-1:000010852771:layer:AWS-AppConfig-Extension:61`
アフリカ (ケープタウン)	`arn:aws:lambda:af-south-1:574348263942:layer:AWS-AppConfig-Extension:44`
中東 (バーレーン)	`arn:aws:lambda:me-south-1:559955524753:layer:AWS-AppConfig-Extension:44`
AWS GovCloud (米国東部)	`arn:aws-us-gov:lambda:us-gov-east-1:946561847325:layer:AWS-AppConfig-Extension:20`
AWS GovCloud (米国西部)	`arn:aws-us-gov:lambda:us-gov-west-1:946746059096:layer:AWS-AppConfig-Extension:20`

古い拡張バージョン (ARM64 プラットフォーム)

次の表ARNs と を示します。 AWS リージョン AWS AppConfig ARM64

新しい拡張子に置き換えられた日付: 05/20/2025

バージョン 2.0.1079

リージョン	ARN
米国東部 (バージニア北部)	arn:aws:lambda:us-east-1:027255383542:layer:AWS-AppConfig-Extension-Arm64:107
米国東部 (オハイオ)	arn:aws:lambda:us-east-2:728743619870:layer:AWS-AppConfig-Extension-Arm64:85
米国西部 (北カリフォルニア)	arn:aws:lambda:us-west-1:958113053741:layer:AWS-AppConfig-Extension-Arm64:100
米国西部 (オレゴン)	arn:aws:lambda:us-west-2:359756378197:layer:AWS-AppConfig-Extension-Arm64:132
カナダ (中部)	arn:aws:lambda:ca-central-1:039592058896:layer:AWS-AppConfig-Extension-Arm64:43
カナダ西部 (カルガリー)	arn:aws:lambda:ca-west-1:436199621743:layer:AWS-AppConfig-Extension-Arm64:18
欧州 (フランクフルト)	arn:aws:lambda:eu-central-1:066940009817:layer:AWS-AppConfig-Extension-Arm64:102
欧州 (チューリッヒ)	arn:aws:lambda:eu-central-2:758369105281:layer:AWS-AppConfig-Extension-Arm64:35
欧州 (アイルランド)	arn:aws:lambda:eu-west-1:434848589818:layer:AWS-AppConfig-Extension-Arm64:98

リージョン	ARN
欧州 (ロンドン)	arn:aws:lambda:eu-west-2:282860088358:layer:AWS-AppConfig-Extension-Arm64:73
欧州 (パリ)	arn:aws:lambda:eu-west-3:493207061005:layer:AWS-AppConfig-Extension-Arm64:52
欧州 (ストックホルム)	arn:aws:lambda:eu-north-1:646970417810:layer:AWS-AppConfig-Extension-Arm64:84
欧州 (ミラノ)	arn:aws:lambda:eu-south-1:203683718741:layer:AWS-AppConfig-Extension-Arm64:39
欧州 (スペイン)	arn:aws:lambda:eu-south-2:586093569114:layer:AWS-AppConfig-Extension-Arm64:35
アジアパシフィック (香港)	arn:aws:lambda:ap-east-1:630222743974:layer:AWS-AppConfig-Extension-Arm64:41
アジアパシフィック (東京)	arn:aws:lambda:ap-northeast-1:980059726660:layer:AWS-AppConfig-Extension-Arm64:79
アジアパシフィック (ソウル)	arn:aws:lambda:ap-northeast-2:826293736237:layer:AWS-AppConfig-Extension-Arm64:44
アジアパシフィック (大阪)	arn:aws:lambda:ap-northeast-3:706869817123:layer:AWS-AppConfig-Extension-Arm64:45

リージョン	ARN
アジアパシフィック (シンガポール)	arn:aws:lambda:ap-southeast-1:421114256042:layer:AWS-AppConfig-Extension-Arm64:86
アジアパシフィック (シドニー)	arn:aws:lambda:ap-southeast-2:080788657173:layer:AWS-AppConfig-Extension-Arm64:108
アジアパシフィック (ジャカルタ)	arn:aws:lambda:ap-southeast-3:418787028745:layer:AWS-AppConfig-Extension-Arm64:58
アジアパシフィック (メルボルン)	arn:aws:lambda:ap-southeast-4:307021474294:layer:AWS-AppConfig-Extension-Arm64:34
アジアパシフィック (マレーシア)	arn:aws:lambda:ap-southeast-5:631746059939:layer:AWS-AppConfig-Extension-Arm64:1
アジアパシフィック (ムンバイ)	arn:aws:lambda:ap-south-1:554480029851:layer:AWS-AppConfig-Extension-Arm64:88
アジアパシフィック (ハイデラバード)	arn:aws:lambda:ap-south-2:489524808438:layer:AWS-AppConfig-Extension-Arm64:33
南米 (サンパウロ)	arn:aws:lambda:sa-east-1:000010852771:layer:AWS-AppConfig-Extension-Arm64:67
アフリカ (ケープタウン)	arn:aws:lambda:af-south-1:574348263942:layer:AWS-AppConfig-Extension-Arm64:51

リージョン	ARN
中東 (アラブ首長国連邦)	`arn:aws:lambda:me-central-1:662846165436:layer:AWS-AppConfig-Extension-Arm64:47`
中東 (バーレーン)	`arn:aws:lambda:me-south-1:559955524753:layer:AWS-AppConfig-Extension-Arm64:53`
イスラエル (テルアビブ)	`arn:aws:lambda:il-central-1:895787185223:layer:AWS-AppConfig-Extension-Arm64:35`
中国 (北京)	`arn:aws-cn:lambda:cn-north-1:615057806174:layer:AWS-AppConfig-Extension-Arm64:28`
中国 (寧夏)	`arn:aws-cn:lambda:cn-northwest-1:615084187847:layer:AWS-AppConfig-Extension-Arm64:26`
AWS GovCloud (米国東部)	`arn:aws-us-gov:lambda:us-gov-east-1:946561847325:layer:AWS-AppConfig-Extension-Arm64:26`
AWS GovCloud (米国西部)	`arn:aws-us-gov:lambda:us-gov-west-1:946746059096:layer:AWS-AppConfig-Extension-Arm64:26`

新しい拡張子に置き換えられた日付: 12/12/2024

バージョン 2.0.678

リージョン	ARN
米国東部 (バージニア北部)	arn:aws:lambda:us-east-1:027255383542:layer:AWS-AppConfig-Extension-Arm64:106
米国東部 (オハイオ)	arn:aws:lambda:us-east-2:728743619870:layer:AWS-AppConfig-Extension-Arm64:84
米国西部 (北カリフォルニア)	arn:aws:lambda:us-west-1:958113053741:layer:AWS-AppConfig-Extension-Arm64:98
米国西部 (オレゴン)	arn:aws:lambda:us-west-2:359756378197:layer:AWS-AppConfig-Extension-Arm64:131
カナダ (中部)	arn:aws:lambda:ca-central-1:039592058896:layer:AWS-AppConfig-Extension-Arm64:41
カナダ西部 (カルガリー)	arn:aws:lambda:ca-west-1:436199621743:layer:AWS-AppConfig-Extension-Arm64:17
欧州 (フランクフルト)	arn:aws:lambda:eu-central-1:066940009817:layer:AWS-AppConfig-Extension-Arm64:101
欧州 (チューリッヒ)	arn:aws:lambda:eu-central-2:758369105281:layer:AWS-AppConfig-Extension-Arm64:33
欧州 (アイルランド)	arn:aws:lambda:eu-west-1:434848589818:layer:AWS-AppConfig-Extension-Arm64:97

リージョン	ARN
欧州 (ロンドン)	arn:aws:lambda:eu-west-2:282860088358:layer:AWS-AppConfig-Extension-Arm64:72
欧州 (パリ)	arn:aws:lambda:eu-west-3:493207061005:layer:AWS-AppConfig-Extension-Arm64:51
欧州 (ストックホルム)	arn:aws:lambda:eu-north-1:646970417810:layer:AWS-AppConfig-Extension-Arm64:83
欧州 (ミラノ)	arn:aws:lambda:eu-south-1:203683718741:layer:AWS-AppConfig-Extension-Arm64:38
欧州 (スペイン)	arn:aws:lambda:eu-south-2:586093569114:layer:AWS-AppConfig-Extension-Arm64:33
アジアパシフィック (香港)	arn:aws:lambda:ap-east-1:630222743974:layer:AWS-AppConfig-Extension-Arm64:40
アジアパシフィック (東京)	arn:aws:lambda:ap-northeast-1:980059726660:layer:AWS-AppConfig-Extension-Arm64:78
アジアパシフィック (ソウル)	arn:aws:lambda:ap-northeast-2:826293736237:layer:AWS-AppConfig-Extension-Arm64:43
アジアパシフィック (大阪)	arn:aws:lambda:ap-northeast-3:706869817123:layer:AWS-AppConfig-Extension-Arm64:44

リージョン	ARN
アジアパシフィック (シンガポール)	arn:aws:lambda:ap-southeast-1:421114256042:layer:AWS-AppConfig-Extension-Arm64:84
アジアパシフィック (シドニー)	arn:aws:lambda:ap-southeast-2:080788657173:layer:AWS-AppConfig-Extension-Arm64:107
アジアパシフィック (ジャカルタ)	arn:aws:lambda:ap-southeast-3:418787028745:layer:AWS-AppConfig-Extension-Arm64:57
アジアパシフィック (メルボルン)	arn:aws:lambda:ap-southeast-4:307021474294:layer:AWS-AppConfig-Extension-Arm64:33
アジアパシフィック (マレーシア)	arn:aws:lambda:ap-southeast-5:631746059939:layer:AWS-AppConfig-Extension-Arm64:1
アジアパシフィック (ムンバイ)	arn:aws:lambda:ap-south-1:554480029851:layer:AWS-AppConfig-Extension-Arm64:87
アジアパシフィック (ハイデラバード)	arn:aws:lambda:ap-south-2:489524808438:layer:AWS-AppConfig-Extension-Arm64:32
南米 (サンパウロ)	arn:aws:lambda:sa-east-1:000010852771:layer:AWS-AppConfig-Extension-Arm64:66
アフリカ (ケープタウン)	arn:aws:lambda:af-south-1:574348263942:layer:AWS-AppConfig-Extension-Arm64:50

リージョン	ARN
中東 (アラブ首長国連邦)	`arn:aws:lambda:me-central-1:662846165436:layer:AWS-AppConfig-Extension-Arm64:46`
中東 (バーレーン)	`arn:aws:lambda:me-south-1:559955524753:layer:AWS-AppConfig-Extension-Arm64:52`
イスラエル (テルアビブ)	`arn:aws:lambda:il-central-1:895787185223:layer:AWS-AppConfig-Extension-Arm64:33`
中国 (北京)	`arn:aws-cn:lambda:cn-north-1:615057806174:layer:AWS-AppConfig-Extension-Arm64:26`
中国 (寧夏)	`arn:aws-cn:lambda:cn-northwest-1:615084187847:layer:AWS-AppConfig-Extension-Arm64:24`
AWS GovCloud (米国東部)	`arn:aws-us-gov:lambda:us-gov-east-1:946561847325:layer:AWS-AppConfig-Extension-Arm64:25`
AWS GovCloud (米国西部)	`arn:aws-us-gov:lambda:us-gov-west-1:946746059096:layer:AWS-AppConfig-Extension-Arm64:25`

新しい拡張子に置き換えられた日付: 2024 年 8 月 8 日

バージョン 2.0.678

リージョン	ARN
米国東部 (バージニア北部)	arn:aws:lambda:us-east-1:027255383542:layer:AWS-AppConfig-Extension-Arm64:100
米国東部 (オハイオ)	arn:aws:lambda:us-east-2:728743619870:layer:AWS-AppConfig-Extension-Arm64:78
米国西部 (北カリフォルニア)	arn:aws:lambda:us-west-1:958113053741:layer:AWS-AppConfig-Extension-Arm64:90
米国西部 (オレゴン)	arn:aws:lambda:us-west-2:359756378197:layer:AWS-AppConfig-Extension-Arm64:125
カナダ西部 (カルガリー)	arn:aws:lambda:ca-west-1:436199621743:layer:AWS-AppConfig-Extension:11
カナダ (中部)	arn:aws:lambda:ca-central-1:039592058896:layer:AWS-AppConfig-Extension-Arm64:36
欧州 (フランクフルト)	arn:aws:lambda:eu-central-1:066940009817:layer:AWS-AppConfig-Extension-Arm64:95
欧州 (チューリッヒ)	arn:aws:lambda:eu-central-2:758369105281:layer:AWS-AppConfig-Extension-Arm64:28
欧州 (アイルランド)	arn:aws:lambda:eu-west-1:434848589818:layer:AWS-AppConfig-Extension-Arm64:91

リージョン	ARN
欧州 (ロンドン)	arn:aws:lambda:eu-west-2:282860088358:layer:AWS-AppConfig-Extension-Arm64:66
欧州 (パリ)	arn:aws:lambda:eu-west-3:493207061005:layer:AWS-AppConfig-Extension-Arm64:45
欧州 (ストックホルム)	arn:aws:lambda:eu-north-1:646970417810:layer:AWS-AppConfig-Extension-Arm64:77
欧州 (ミラノ)	arn:aws:lambda:eu-south-1:203683718741:layer:AWS-AppConfig-Extension-Arm64:32
欧州 (スペイン)	arn:aws:lambda:eu-south-2:586093569114:layer:AWS-AppConfig-Extension-Arm64:28
アジアパシフィック (香港)	arn:aws:lambda:ap-east-1:630222743974:layer:AWS-AppConfig-Extension-Arm64:34
アジアパシフィック (東京)	arn:aws:lambda:ap-northeast-1:980059726660:layer:AWS-AppConfig-Extension-Arm64:72
アジアパシフィック (ソウル)	arn:aws:lambda:ap-northeast-2:826293736237:layer:AWS-AppConfig-Extension-Arm64:37
アジアパシフィック (大阪)	arn:aws:lambda:ap-northeast-3:706869817123:layer:AWS-AppConfig-Extension-Arm64:38

リージョン	ARN
アジアパシフィック (シンガポール)	arn:aws:lambda:ap-southeast-1:421114256042:layer:AWS-AppConfig-Extension-Arm64:79
アジアパシフィック (シドニー)	arn:aws:lambda:ap-southeast-2:080788657173:layer:AWS-AppConfig-Extension-Arm64:101
アジアパシフィック (ジャカルタ)	arn:aws:lambda:ap-southeast-3:418787028745:layer:AWS-AppConfig-Extension-Arm64:51
アジアパシフィック (メルボルン)	arn:aws:lambda:ap-southeast-4:307021474294:layer:AWS-AppConfig-Extension-Arm64:27
アジアパシフィック (ムンバイ)	arn:aws:lambda:ap-south-1:554480029851:layer:AWS-AppConfig-Extension-Arm64:81
アジアパシフィック (ハイデラバード)	arn:aws:lambda:ap-south-2:489524808438:layer:AWS-AppConfig-Extension-Arm64:26
南米 (サンパウロ)	arn:aws:lambda:sa-east-1:000010852771:layer:AWS-AppConfig-Extension-Arm64:60
アフリカ (ケープタウン)	arn:aws:lambda:af-south-1:574348263942:layer:AWS-AppConfig-Extension-Arm64:44
中東 (アラブ首長国連邦)	arn:aws:lambda:me-central-1:662846165436:layer:AWS-AppConfig-Extension-Arm64:40

リージョン	ARN
中東 (バーレーン)	arn:aws:lambda:me-south-1:559955524753:layer:AWS-AppConfig-Extension-Arm64:46
イスラエル (テルアビブ)	arn:aws:lambda:il-central-1:895787185223:layer:AWS-AppConfig-Extension-Arm64:28
中国 (北京)	arn:aws-cn:lambda:cn-north-1:615057806174:layer:AWS-AppConfig-Extension-Arm64:21
中国 (寧夏)	arn:aws-cn:lambda:cn-northwest-1:615084187847:layer:AWS-AppConfig-Extension-Arm64:19
AWS GovCloud (米国東部)	arn:aws-us-gov:lambda:us-gov-east-1:946561847325:layer:AWS-AppConfig-Extension-Arm64:19
AWS GovCloud (米国西部)	arn:aws-us-gov:lambda:us-gov-west-1:946746059096:layer:AWS-AppConfig-Extension-Arm64:19

新しい拡張子に置き換えられた日付: 2024 年 7 月 23 日

バージョン 2.0.501

リージョン	ARN
米国東部 (バージニア北部)	arn:aws:lambda:us-east-1:027255383542:layer:AWS-AppConfig-Extension-Arm64:86

リージョン	ARN
米国東部 (オハイオ)	arn:aws:lambda:us-east-2:728743619870:layer:AWS-AppConfig-Extension-Arm64:64
米国西部 (北カリフォルニア)	arn:aws:lambda:us-west-1:958113053741:layer:AWS-AppConfig-Extension-Arm64:72
米国西部 (オレゴン)	arn:aws:lambda:us-west-2:359756378197:layer:AWS-AppConfig-Extension-Arm64:112
カナダ西部 (カルガリー)	arn:aws:lambda:ca-west-1:436199621743:layer:AWS-AppConfig-Extension:1
カナダ (中部)	arn:aws:lambda:ca-central-1:039592058896:layer:AWS-AppConfig-Extension-Arm64:21
欧州 (フランクフルト)	arn:aws:lambda:eu-central-1:066940009817:layer:AWS-AppConfig-Extension-Arm64:79
欧州 (チューリッヒ)	arn:aws:lambda:eu-central-2:758369105281:layer:AWS-AppConfig-Extension-Arm64:11
欧州 (アイルランド)	arn:aws:lambda:eu-west-1:434848589818:layer:AWS-AppConfig-Extension-Arm64:82
欧州 (ロンドン)	arn:aws:lambda:eu-west-2:282860088358:layer:AWS-AppConfig-Extension-Arm64:51

リージョン	ARN
欧州 (パリ)	arn:aws:lambda:eu-west-3:493207061005:layer:AWS-AppConfig-Extension-Arm64:30
欧州 (ストックホルム)	arn:aws:lambda:eu-north-1:646970417810:layer:AWS-AppConfig-Extension-Arm64:60
欧州 (ミラノ)	arn:aws:lambda:eu-south-1:203683718741:layer:AWS-AppConfig-Extension-Arm64:17
欧州 (スペイン)	arn:aws:lambda:eu-south-2:586093569114:layer:AWS-AppConfig-Extension-Arm64:11
アジアパシフィック (香港)	arn:aws:lambda:ap-east-1:630222743974:layer:AWS-AppConfig-Extension-Arm64:19
アジアパシフィック (東京)	arn:aws:lambda:ap-northeast-1:980059726660:layer:AWS-AppConfig-Extension-Arm64:57
アジアパシフィック (ソウル)	arn:aws:lambda:ap-northeast-2:826293736237:layer:AWS-AppConfig-Extension-Arm64:22
アジアパシフィック (大阪)	arn:aws:lambda:ap-northeast-3:706869817123:layer:AWS-AppConfig-Extension-Arm64:22
アジアパシフィック (シンガポール)	arn:aws:lambda:ap-southeast-1:421114256042:layer:AWS-AppConfig-Extension-Arm64:64

リージョン	ARN
アジアパシフィック (シドニー)	arn:aws:lambda:ap-southeast-2:080788657173:layer:AWS-AppConfig-Extension-Arm64:85
アジアパシフィック (ジャカルタ)	arn:aws:lambda:ap-southeast-3:418787028745:layer:AWS-AppConfig-Extension-Arm64:35
アジアパシフィック (メルボルン)	arn:aws:lambda:ap-southeast-4:307021474294:layer:AWS-AppConfig-Extension-Arm64:11
アジアパシフィック (ムンバイ)	arn:aws:lambda:ap-south-1:554480029851:layer:AWS-AppConfig-Extension-Arm64:67
アジアパシフィック (ハイデラバード)	arn:aws:lambda:ap-south-2:489524808438:layer:AWS-AppConfig-Extension-Arm64:11
南米 (サンパウロ)	arn:aws:lambda:sa-east-1:000010852771:layer:AWS-AppConfig-Extension-Arm64:43
アフリカ (ケープタウン)	arn:aws:lambda:af-south-1:574348263942:layer:AWS-AppConfig-Extension-Arm64:30
中東 (アラブ首長国連邦)	arn:aws:lambda:me-central-1:662846165436:layer:AWS-AppConfig-Extension-Arm64:24
中東 (バーレーン)	arn:aws:lambda:me-south-1:559955524753:layer:AWS-AppConfig-Extension-Arm64:31

リージョン	ARN
イスラエル (テルアビブ)	arn:aws:lambda:il-central-1:895787185223:layer:AWS-AppConfig-Extension-Arm64:11
中国 (北京)	arn:aws-cn:lambda:cn-north-1:615057806174:layer:AWS-AppConfig-Extension-Arm64:7
中国 (寧夏)	arn:aws-cn:lambda:cn-northwest-1:615084187847:layer:AWS-AppConfig-Extension-Arm64:5
AWS GovCloud (米国東部)	arn:aws-us-gov:lambda:us-gov-east-1:946561847325:layer:AWS-AppConfig-Extension-Arm64:5
AWS GovCloud (米国西部)	arn:aws-us-gov:lambda:us-gov-west-1:946746059096:layer:AWS-AppConfig-Extension-Arm64:5

新しい拡張子に置き換えられた日付: 2024 年 7 月 1 日

バージョン 2.0.358

リージョン	ARN
米国東部 (バージニア北部)	arn:aws:lambda:us-east-1:027255383542:layer:AWS-AppConfig-Extension-Arm64:61
米国東部 (オハイオ)	arn:aws:lambda:us-east-2:728743619870:layer:AWS-AppConfig-Extension-Arm64:45

リージョン	ARN
米国西部 (北カリフォルニア)	arn:aws:lambda:us-west-1:958113053741:layer:AWS-AppConfig-Extension-Arm64:18
米国西部 (オレゴン)	arn:aws:lambda:us-west-2:359756378197:layer:AWS-AppConfig-Extension-Arm64:63
カナダ (中部)	arn:aws:lambda:ca-central-1:039592058896:layer:AWS-AppConfig-Extension-Arm64:13
欧州 (フランクフルト)	arn:aws:lambda:eu-central-1:066940009817:layer:AWS-AppConfig-Extension-Arm64:49
欧州 (チューリッヒ)	arn:aws:lambda:eu-central-2:758369105281:layer:AWS-AppConfig-Extension-Arm64:5
欧州 (アイルランド)	arn:aws:lambda:eu-west-1:434848589818:layer:AWS-AppConfig-Extension-Arm64:63
欧州 (ロンドン)	arn:aws:lambda:eu-west-2:282860088358:layer:AWS-AppConfig-Extension-Arm64:45
欧州 (パリ)	arn:aws:lambda:eu-west-3:493207061005:layer:AWS-AppConfig-Extension-Arm64:17
欧州 (ストックホルム)	arn:aws:lambda:eu-north-1:646970417810:layer:AWS-AppConfig-Extension-Arm64:18

リージョン	ARN
欧州 (ミラノ)	arn:aws:lambda:eu-south-1:203683718741:layer:AWS-AppConfig-Extension-Arm64:11
欧州 (スペイン)	arn:aws:lambda:eu-south-2:586093569114:layer:AWS-AppConfig-Extension-Arm64:5
アジアパシフィック (香港)	arn:aws:lambda:ap-east-1:630222743974:layer:AWS-AppConfig-Extension-Arm64:11
アジアパシフィック (東京)	arn:aws:lambda:ap-northeast-1:980059726660:layer:AWS-AppConfig-Extension-Arm64:51
アジアパシフィック (ソウル)	arn:aws:lambda:ap-northeast-2:826293736237:layer:AWS-AppConfig-Extension-Arm64:16
アジアパシフィック (大阪)	arn:aws:lambda:ap-northeast-3:706869817123:layer:AWS-AppConfig-Extension-Arm64:16
アジアパシフィック (シンガポール)	arn:aws:lambda:ap-southeast-1:421114256042:layer:AWS-AppConfig-Extension-Arm64:58
アジアパシフィック (シドニー)	arn:aws:lambda:ap-southeast-2:080788657173:layer:AWS-AppConfig-Extension-Arm64:49
アジアパシフィック (ジャカルタ)	arn:aws:lambda:ap-southeast-3:418787028745:layer:AWS-AppConfig-Extension-Arm64:16

リージョン	ARN
アジアパシフィック (メルボルン)	arn:aws:lambda:ap-southeast-4:307021474294:layer:AWS-AppConfig-Extension-Arm64:5
アジアパシフィック (ムンバイ)	arn:aws:lambda:ap-south-1:554480029851:layer:AWS-AppConfig-Extension-Arm64:49
アジアパシフィック (ハイデラバード)	arn:aws:lambda:ap-south-2:489524808438:layer:AWS-AppConfig-Extension-Arm64:5
南米 (サンパウロ)	arn:aws:lambda:sa-east-1:000010852771:layer:AWS-AppConfig-Extension-Arm64:16
アフリカ (ケープタウン)	arn:aws:lambda:af-south-1:574348263942:layer:AWS-AppConfig-Extension-Arm64:11
中東 (アラブ首長国連邦)	arn:aws:lambda:me-central-1:662846165436:layer:AWS-AppConfig-Extension-Arm64:5
中東 (バーレーン)	arn:aws:lambda:me-south-1:559955524753:layer:AWS-AppConfig-Extension-Arm64:13
イスラエル (テルアビブ)	arn:aws:lambda:il-central-1:895787185223:layer:AWS-AppConfig-Extension-Arm64:5

新しい拡張子に置き換えられた日付: 2023 年 12 月 1 日

バージョン 2.0.181

リージョン	ARN
米国東部 (バージニア北部)	arn:aws:lambda:us-east-1:027255383542:layer:AWS-AppConfig-Extension-Arm64:46
米国東部 (オハイオ)	arn:aws:lambda:us-east-2:728743619870:layer:AWS-AppConfig-Extension-Arm64:33
米国西部 (北カリフォルニア)	arn:aws:lambda:us-west-1:958113053741:layer:AWS-AppConfig-Extension-Arm64:1
米国西部 (オレゴン)	arn:aws:lambda:us-west-2:359756378197:layer:AWS-AppConfig-Extension-Arm64:48
カナダ (中部)	arn:aws:lambda:ca-central-1:039592058896:layer:AWS-AppConfig-Extension-Arm64:1
欧州 (フランクフルト)	arn:aws:lambda:eu-central-1:066940009817:layer:AWS-AppConfig-Extension-Arm64:36
欧州 (アイルランド)	arn:aws:lambda:eu-west-1:434848589818:layer:AWS-AppConfig-Extension-Arm64:48
欧州 (ロンドン)	arn:aws:lambda:eu-west-2:282860088358:layer:AWS-AppConfig-Extension-Arm64:33
欧州 (パリ)	arn:aws:lambda:eu-west-3:493207061005:layer:AWS-AppConfig-Extension-Arm64:1

リージョン	ARN
欧州 (ストックホルム)	arn:aws:lambda:eu-north-1:646970417810:layer:AWS-AppConfig-Extension-Arm64:1
欧州 (ミラノ)	arn:aws:lambda:eu-south-1:203683718741:layer:AWS-AppConfig-Extension-Arm64:1
アジアパシフィック (香港)	arn:aws:lambda:ap-east-1:630222743974:layer:AWS-AppConfig-Extension-Arm64:1
アジアパシフィック (東京)	arn:aws:lambda:ap-northeast-1:980059726660:layer:AWS-AppConfig-Extension-Arm64:37
アジアパシフィック (ソウル)	arn:aws:lambda:ap-northeast-2:826293736237:layer:AWS-AppConfig-Extension-Arm64:1
アジアパシフィック (大阪)	arn:aws:lambda:ap-northeast-3:706869817123:layer:AWS-AppConfig-Extension-Arm64:1
アジアパシフィック (シンガポール)	arn:aws:lambda:ap-southeast-1:421114256042:layer:AWS-AppConfig-Extension-Arm64:43
アジアパシフィック (シドニー)	arn:aws:lambda:ap-southeast-2:080788657173:layer:AWS-AppConfig-Extension-Arm64:36
アジアパシフィック (ジャカルタ)	arn:aws:lambda:ap-southeast-3:418787028745:layer:AWS-AppConfig-Extension-Arm64:1

リージョン	ARN
アジアパシフィック (ムンバイ)	`arn:aws:lambda:ap-south-1:554480029851:layer:AWS-AppConfig-Extension-Arm64:36`
南米 (サンパウロ)	`arn:aws:lambda:sa-east-1:000010852771:layer:AWS-AppConfig-Extension-Arm64:1`
アフリカ (ケープタウン)	`arn:aws:lambda:af-south-1:574348263942:layer:AWS-AppConfig-Extension-Arm64:1`
中東 (バーレーン)	`arn:aws:lambda:me-south-1:559955524753:layer:AWS-AppConfig-Extension-Arm64:1`

新しい拡張子に置き換えられた日付: 2023 年 3 月 30 日

バージョン 2.0.165

リージョン	ARN
米国東部 (バージニア北部)	`arn:aws:lambda:us-east-1:027255383542:layer:AWS-AppConfig-Extension-Arm64:43`
米国東部 (オハイオ)	`arn:aws:lambda:us-east-2:728743619870:layer:AWS-AppConfig-Extension-Arm64:31`
米国西部 (オレゴン)	`arn:aws:lambda:us-west-2:359756378197:layer:AWS-AppConfig-Extension-Arm64:45`

リージョン	ARN
欧州 (フランクフルト)	arn:aws:lambda:eu-central-1:066940009817:layer:AWS-AppConfig-Extension-Arm64:34
欧州 (アイルランド)	arn:aws:lambda:eu-west-1:434848589818:layer:AWS-AppConfig-Extension-Arm64:46
欧州 (ロンドン)	arn:aws:lambda:eu-west-2:282860088358:layer:AWS-AppConfig-Extension-Arm64:31
アジアパシフィック (東京)	arn:aws:lambda:ap-northeast-1:980059726660:layer:AWS-AppConfig-Extension-Arm64:35
アジアパシフィック (シンガポール)	arn:aws:lambda:ap-southeast-1:421114256042:layer:AWS-AppConfig-Extension-Arm64:41
アジアパシフィック (シドニー)	arn:aws:lambda:ap-southeast-2:080788657173:layer:AWS-AppConfig-Extension-Arm64:34
アジアパシフィック (ムンバイ)	arn:aws:lambda:ap-south-1:554480029851:layer:AWS-AppConfig-Extension-Arm64:34

新しい拡張子に置き換えられた日付: 2023 年 2 月 21 日

バージョン 2.0.122

リージョン	ARN
米国東部 (バージニア北部)	arn:aws:lambda:us-east-1:027255383542:layer:AWS-AppConfig-Extension-Arm64:15
米国東部 (オハイオ)	arn:aws:lambda:us-east-2:728743619870:layer:AWS-AppConfig-Extension-Arm64:11
米国西部 (オレゴン)	arn:aws:lambda:us-west-2:359756378197:layer:AWS-AppConfig-Extension-Arm64:16
欧州 (フランクフルト)	arn:aws:lambda:eu-central-1:066940009817:layer:AWS-AppConfig-Extension-Arm64:13
欧州 (アイルランド)	arn:aws:lambda:eu-west-1:434848589818:layer:AWS-AppConfig-Extension-Arm64:20
欧州 (ロンドン)	arn:aws:lambda:eu-west-2:282860088358:layer:AWS-AppConfig-Extension-Arm64:11
アジアパシフィック (東京)	arn:aws:lambda:ap-northeast-1:980059726660:layer:AWS-AppConfig-Extension-Arm64:15
アジアパシフィック (シンガポール)	arn:aws:lambda:ap-southeast-1:421114256042:layer:AWS-AppConfig-Extension-Arm64:16
アジアパシフィック (シドニー)	arn:aws:lambda:ap-southeast-2:080788657173:layer:AWS-AppConfig-Extension-Arm64:13

リージョン	ARN
アジアパシフィック (ムンバイ)	arn:aws:lambda:ap-south-1:554480029851:layer:AWS-AppConfig-Extension-Arm64:13

新しい拡張子に置き換えられた日付: 2022 年 8 月 23 日

バージョン 2.0.58

リージョン	ARN
米国東部 (バージニア北部)	arn:aws:lambda:us-east-1:027255383542:layer:AWS-AppConfig-Extension-Arm64:2
米国東部 (オハイオ)	arn:aws:lambda:us-east-2:728743619870:layer:AWS-AppConfig-Extension-Arm64:2
米国西部 (オレゴン)	arn:aws:lambda:us-west-2:359756378197:layer:AWS-AppConfig-Extension-Arm64:3
欧州 (フランクフルト)	arn:aws:lambda:eu-central-1:066940009817:layer:AWS-AppConfig-Extension-Arm64:2
欧州 (アイルランド)	arn:aws:lambda:eu-west-1:434848589818:layer:AWS-AppConfig-Extension-Arm64:7
欧州 (ロンドン)	arn:aws:lambda:eu-west-2:282860088358:layer:AWS-AppConfig-Extension-Arm64:2

リージョン	ARN
アジアパシフィック (東京)	arn:aws:lambda:ap-northeast-1:980059726660:layer:AWS-AppConfig-Extension-Arm64:2
アジアパシフィック (シンガポール)	arn:aws:lambda:ap-southeast-1:421114256042:layer:AWS-AppConfig-Extension-Arm64:3
アジアパシフィック (シドニー)	arn:aws:lambda:ap-southeast-2:080788657173:layer:AWS-AppConfig-Extension-Arm64:2
アジアパシフィック (ムンバイ)	arn:aws:lambda:ap-south-1:554480029851:layer:AWS-AppConfig-Extension-Arm64:2

新しい拡張子に置き換えられた日付: 2022 年 4 月 21 日

バージョン 2.0.45

リージョン	ARN
米国東部 (バージニア北部)	arn:aws:lambda:us-east-1:027255383542:layer:AWS-AppConfig-Extension-Arm64:1
米国東部 (オハイオ)	arn:aws:lambda:us-east-2:728743619870:layer:AWS-AppConfig-Extension-Arm64:1
米国西部 (オレゴン)	arn:aws:lambda:us-west-2:359756378197:layer:AWS-AppConfig-Extension-Arm64:2

リージョン	ARN
欧州 (フランクフルト)	`arn:aws:lambda:eu-central-1:066940009817:layer:AWS-AppConfig-Extension-Arm64:1`
欧州 (アイルランド)	`arn:aws:lambda:eu-west-1:434848589818:layer:AWS-AppConfig-Extension-Arm64:6`
欧州 (ロンドン)	`arn:aws:lambda:eu-west-2:282860088358:layer:AWS-AppConfig-Extension-Arm64:1`
アジアパシフィック (東京)	`arn:aws:lambda:ap-northeast-1:980059726660:layer:AWS-AppConfig-Extension-Arm64:1`
アジアパシフィック (シンガポール)	`arn:aws:lambda:ap-southeast-1:421114256042:layer:AWS-AppConfig-Extension-Arm64:2`
アジアパシフィック (シドニー)	`arn:aws:lambda:ap-southeast-2:080788657173:layer:AWS-AppConfig-Extension-Arm64:1`
アジアパシフィック (ムンバイ)	`arn:aws:lambda:ap-south-1:554480029851:layer:AWS-AppConfig-Extension-Arm64:1`

Amazon EC2 およびオンプレミスマシンでの AWS AppConfig エージェントの使用

AWS AppConfig エージェントを使用して、Amazon Elastic Compute Cloud (Amazon EC2) Linux インスタンスで実行されているアプリケーション AWS AppConfig と統合できます。エージェントは、次の方法でアプリケーションの処理と管理を強化します。

- エージェントは AWS AppConfig 、 AWS Identity and Access Management (IAM) ロールを使用して設定データのローカルキャッシュを管理することで、ユーザーに代わって を呼び出します。ローカルキャッシュから設定データを引き出すことで、アプリケーションが設定データを管理するために必要となるコードの更新が少なくなり、設定データをミリ秒単位で取得でき、そのようなデータの呼び出しを妨げるネットワークの問題による影響を受けなくなります。*
- エージェントは、 AWS AppConfig 機能フラグを取得して解決するためのネイティブエクスペリエンスを提供します。
- エージェントはすぐにキャッシュ戦略、ポーリング間隔、ローカル設定データの可用性に関するベストプラクティスを提供すると同時に、以降のサービスコールに必要な設定トークンを追跡します。
- バックグラウンドで実行中、エージェントは設定 AWS AppConfig データの更新のためにデータプレーンを定期的にポーリングします。アプリケーションは、ポート 2772 (カスタマイズ可能なデフォルトポート値) で localhost に接続し、HTTP GET を呼び出してデータを取得できます。

*AWS AppConfig エージェントは、サービスが設定データを初めて取得したときにデータをキャッシュします。このため、データを取得する最初の呼び出しは、それ以降の呼び出しよりも時間がかかります。

トピック

- ステップ 1: (必須) リソースの作成と権限の設定
- ステップ 2: (必須) Amazon EC2 インスタンスに AWS AppConfig エージェントをインストールして起動する
- ステップ 3 : CloudWatch ログへのログファイルの送信(オプション、推奨)
- ステップ 4: (オプション) 環境変数を使用して Amazon EC2 の AWS AppConfig エージェントを設定する
- ステップ 5: (必須) 設定データの取得
- ステップ 6 (オプション、ただし推奨): AWS AppConfig エージェントの更新を自動化する

ステップ 1: (必須) リソースの作成と権限の設定

Amazon EC2 インスタンスで実行されているアプリケーション AWS AppConfig と統合するには、機能フラグやフリーフォーム設定データを含む AWS AppConfig アーティファクトと設定データを作成する必要があります。詳細については、「で機能フラグとフリーフォーム設定データを作成する AWS AppConfig」を参照してください。

によってホストされる設定データを取得するには AWS AppConfig、アプリケーションが AWS AppConfig データプレーンへのアクセスで設定されている必要があります。アプリケーションにアクセス権を付与するには、Amazon EC2 インスタンスロールに割り当てられている IAM アクセス権限ポリシーを更新してください。具体的には、 `appconfig:StartConfigurationSession` と `appconfig:GetLatestConfiguration` ポリシーにとアクションを追加する必要があります。以下がその例です。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "appconfig:StartConfigurationSession",
                "appconfig:GetLatestConfiguration"
            ],
            "Resource": "*"
        }
    ]
}
```

ポリシーに権限を追加する方法の詳細については、IAM ユーザーガイドのIAM ID 権限の追加と削除を参照してください。

ステップ 2: (必須) Amazon EC2 インスタンスに AWS AppConfig エージェントをインストールして起動する

AWS AppConfig エージェントは、 が管理する Amazon Simple Storage Service (Amazon S3) バケットでホストされます AWS。Linuxインスタンスで最新バージョンの エージェントをダウンロードしてインストールするには、次の手順を使用します。アプリケーションが複数のインスタンスに分散されている場合は、アプリケーションをホストする各インスタンスでこの手順を実行する必要があります。

> **Note**
>
> 以下の情報に注意してください。
>
> - AWS AppConfig エージェントは、カーネルバージョン 4.15 以降を実行している Linux オペレーティングシステムで使用できます。Ubuntu などの Debian ベースのシステムはサポートされていません。

- このエージェントは x86_64 および ARM64 アーキテクチャをサポートしています。
- 分散アプリケーションでは、インストールコマンドと起動コマンドを Auto Scaling グループの Amazon EC2 ユーザーデータに追加することをお勧めします。追加すると、各インスタンスがコマンドを自動的に実行します。詳細については、「Amazon EC2 ユーザーガイド」の「起動時に Linux インスタンスでコマンドを実行する」を参照してください。さらに、チュートリアル：Amazon EC2 Auto Scalingユーザーガイドのインスタンスメタデータを通してターゲットライフサイクルステートを取得するためのユーザーデータの設定を参照してください。
- このトピックの手順では、インスタンスにログインしてコマンドを実行することでエージェントをインストールするなどのアクションを実行する方法について説明します。ローカルクライアントマシンからコマンドを実行し、ツールである Run Command を使用して 1 つ以上のインスタンスをターゲットにできます AWS Systems Manager。詳細については、AWS Systems Manager ユーザーガイドの「AWS Systems Manager 実行コマンド」を参照してください。
- AWS AppConfig Amazon EC2 Linux インスタンスのエージェントは `systemd`サービスです。

インスタンスに AWS AppConfig エージェントをインストールして起動するには

1. Linux インスタンスにログインします。
2. ターミナルを開き、管理者権限で以下のコマンドを実行します。

   x86_64

   ```
   sudo yum install https://s3.amazonaws.com/aws-appconfig-downloads/aws-appconfig-agent/linux/x86_64/latest/aws-appconfig-agent.rpm
   ```

   ARM64

   ```
   sudo yum install https://s3.amazonaws.com/aws-appconfig-downloads/aws-appconfig-agent/linux/arm64/latest/aws-appconfig-agent.rpm
   ```

   特定のバージョンの AWS AppConfig エージェントをインストールする場合は、URL `latest`のを特定のバージョン番号に置き換えます。次に x86_64 の例を示します。

```
sudo yum install https://s3.amazonaws.com/aws-appconfig-downloads/aws-appconfig-
agent/linux/x86_64/2.0.2/aws-appconfig-agent.rpm
```

3. エージェントを開始するには、次のコマンドを実行します。

```
sudo systemctl start aws-appconfig-agent
```

4. エージェントが実行されていることを確認するには、次のコマンドを実行します。

```
sudo systemctl status aws-appconfig-agent
```

成功した場合、このコマンドは次のような情報を返します。

```
aws-appconfig-agent.service - aws-appconfig-agent
  ...
  Active: active (running) since Mon 2023-07-26 00:00:00 UTC; 0s ago
  ...
```

> **Note**
>
> エージェントを停止するには、次のコマンドを実行します。
>
> ```
> sudo systemctl stop aws-appconfig-agent
> ```

ステップ 3：CloudWatch ログへのログファイルの送信(オプション、推奨)

デフォルトでは、 AWS AppConfig エージェントは STDERR にログを発行します。Systemd は Linux インスタンスで実行されているすべてのサービスの STDOUT と STDERR を systemd ジャーナルにリダイレクトします。 AWS AppConfig エージェントを 1 つまたは 2 つのインスタンスでのみ実行している場合は、systemd ジャーナルのログデータを表示および管理できます。より優れたソリューションとして、分散アプリケーションのソリューションを強く推奨します。それは、ログファイルをディスクに書き込み、Amazon CloudWatch エージェントを使って AWS クラウドにログデータをアップロードすることです。さらに、インスタンスから古いログファイルを削除するように CloudWatch エージェントを設定でき、インスタンスのディスク容量が不足するのを防ぐことができます。

ディスクへのロギングを有効にするには、 `LOG_PATH` で説明されているように「ステップ 4: (オプション) 環境変数を使用して Amazon EC2 の AWS AppConfig エージェントを設定する」環境変数を設定する必要があります。

CloudWatch エージェントを使い始めるには、Amazon CloudWatch ユーザーガイドの「CloudWatch エージェントを使用した Amazon EC2 インスタンスとオンプレミスサーバーからのメトリクスとログの収集」を参照してください。 Systems Manager のツールである高速セットアップを使用すると、CloudWatch エージェントをすばやくインストールできます。詳細については、AWS Systems Manager ユーザーガイドの Quick Setup Host Management を参照してください。

> **Warning**
>
> CloudWatch エージェントを使用せずにログファイルをディスクに書き込む場合は、古いログファイルを削除する必要があります。 AWS AppConfig Agent は 1 時間ごとにログファイルを自動的にローテーションします。古いログファイルを削除しないと、インスタンスのディスク容量が不足する可能性があります。

インスタンスに CloudWatch エージェントをインストールした後、CloudWatch エージェント設定ファイルを作成します。設定ファイルは、エージェントのログファイルの操作方法を CloudWatch AWS AppConfig エージェントに指示します。CloudWatch エージェント設定ファイルの作成に関する詳細については、「CloudWatch エージェント設定ファイルを作成」を参照してください。

インスタンスの CloudWatch `logs` エージェント設定ファイルに次のセクションを追加し、変更を保存します。

```
"logs": {
  "logs_collected": {
    "files": {
      "collect_list": [
        {
          "file_path": "/path_you_specified_for_logging",
          "log_group_name": "${YOUR_LOG_GROUP_NAME}/aws-appconfig-agent.log",
          "auto_removal": true
        },
        ...
      ]
    },
    ...
  },
```

```
  ...
}
```

の値が auto_removalの場合true、CloudWatch エージェントはローテーションされた AWS AppConfig エージェントのログファイルを自動的に削除します。

ステップ 4: (オプション) 環境変数を使用して Amazon EC2 の AWS AppConfig エージェントを設定する

環境変数を使用してAmazon EC2 の AWS AppConfig エージェントを設定できます。systemd サービスの環境変数を設定するには、ドロップインユニットファイルを作成します。次の例は、ドロップインユニットファイルを作成して AWS AppConfig エージェントのログ記録レベルを に設定する方法を示していますDEBUG。

環境変数用のドロップインセルファイルの作成例

1. Linux インスタンスにログインします。
2. ターミナルを開き、管理者権限で以下のコマンドを実行します。このコマンドで設定ディレクトリを作成します。

   ```
   sudo mkdir /etc/systemd/system/aws-appconfig-agent.service.d
   ```

3. 次のコマンドを実行して、 ドロップインユニットファイルを作成します。*file_name* をファイルの名前に置き換えます。拡張子は .conf でなければなりません。

   ```
   sudo touch /etc/systemd/system/aws-appconfig-agent.service.d/file_name.conf
   ```

4. ドロップインユニットファイルに情報を入力します。次の例では、 Service 環境変数を定義するセクションを追加します。この例では、 AWS AppConfig エージェント ログレベルを DEBUG に設定しています。

   ```
   [Service]
   Environment=LOG_LEVEL=DEBUG
   ```

5. システム設定を再読み込むには、次のコマンドを実行します。

   ```
   sudo systemctl daemon-reload
   ```

6. AWS AppConfig エージェントを再起動するには、次のコマンドを実行します。

```
sudo systemctl restart aws-appconfig-agent
```

ドロップインユニットファイルで次の環境変数を指定することで、Amazon EC2 の AWS AppConfig エージェントを設定できます。

> **Note**
>
> 次の表に、サンプル値の列を示します。モニターの解像度によっては、テーブルの下部までスクロールし、右側までスクロールして列を表示する必要がある場合があります。

環境変数	詳細	デフォルト値	サンプル値 (複数可)

ACCESS_TOKEN	この環境変数は、エージェント HTTP サーバーに設定データをリクエストするときに指定する必要があるトークンを定義します。トークンの値は、認可タイプ Bearerの HTTP リクエスト認可ヘッダーに設定する必要があります。以下はその例です。
	`GET /applications/my_app/...`
	`Host: localhost:2772`
	`Authorization: Bearer <token value>`
		なし	MyAccessToken

環境変数	詳細	デフォルト値	サンプル値 (複数可)
BACKUP_DIRECTORY	この環境変数により、 AWS AppConfig エージェントは取得した各設定のバックアップを指定されたディレクトリに保存できます。 **Important** ディスクにバックアップされた設定は暗号化されません。設定に機密データが含まれている場合は、ファイルシステムのアクセス許可で最小特権の原則を実践 AWS AppConfig することをお勧めします。詳細については、「のセキュリティ AWS AppConfig」を参照してください。	なし	/path/to/backups

環境変数	詳細	デフォルト値	サンプル値 (複数可)
HTTP_PORT	この環境変数は、エージェントの HTTP サーバーが実行されるポートを指定します。	2772	2772
LOG_LEVEL	この環境変数は、エージェントがログに記録する詳細レベルを指定します。各レベルには、現在レベルとそれより上位のすべてのレベルが含まれます。値は大文字と小文字を区別しません。詳細度が最も低いログレベルは、trace、、debug、i よび ですnone。trace ログには、エージェントに関するタイミング情報を含む詳細情報が含まれます。	情報	trace デバッグ 情報 warn エラー fatal なし
LOG_PATH	ログが書き込まれるディスクがある場所。指定しない場合、ログは stderr に書き込まれます。	なし	/path/to/logs/agent.log

環境変数	詳細	デフォルト値	サンプル値 (複数可)
MANIFEST	この環境変数は、マルチアカウントの取得やディスクへの設定の保存など、設定ごとに追加の機能を利用するように AWS AppConfig エージェントを設定します。これらの機能の詳細については、「マニフェストを使用して追加の取得機能を有効にする」を参照してください。	なし	マニフェストとして AWS AppConfig 設定を使用する場合: MyApp:MyEnv:MyManifestConfig 。 ディスクからマニフェストをロードする場合: file:/path/to/manifest.json
MAX_CONNECTIONS	この環境変数は、エージェントが AWS AppConfigから設定を取得するために使用する最大接続数を設定します。	3	3

環境変数	詳細	デフォルト値	サンプル値 (複数可)
POLL_INTERVAL	この環境変数は、エージェントが AWS AppConfig 更新された設定データをポーリングする頻度を制御します。間隔の秒数を指定できます。時間単位で数値を指定することもできます。s は秒、m は分、h は時間です。単位が指定されなかった場合、エージェントはデフォルトで秒になります。たとえば、60 、60 秒、1m の場合、ポーリング間隔は同じになります。	45 秒	45 45 秒 5m 1 時間
PREFETCH_LIST	この環境変数は、エージェントが起動すると AWS AppConfig すぐにリクエストする設定データを指定します。複数の設定識別子をカンマ区切りリストで指定できます。	なし	MyApp:MyEnv:MyConfig abcd123:efgh456:ijkl789 MyApp:MyEnv:Config1、MyApp:MyEnv:Config2

環境変数	詳細	デフォルト値	サンプル値 (複数可)
PRELOAD_BACKUPS	に設定するとtrue、AWS AppConfig Agent は で見つかった設定バックアップをメモリBACKUP_DIRECTORY にロードし、サービスから新しいバージョンが存在するかどうかをすぐに確認します。false に設定すると、 AWS AppConfig エージェントは、ネットワークに問題があるなど、サービスから設定データを取得できない場合にのみ、設定バックアップからコンテンツをロードします。	真	真 false
PROXY_HEADERS	この環境変数は、PROXY_URL 環境変数で参照されるプロキシに必要なヘッダーを指定します。値は、コンマで区切られたヘッダーの一覧です。	なし	ヘッダー: 値 h1: v1、h2: v2

環境変数	詳細	デフォルト値	サンプル値 (複数可)
PROXY_URL	この環境変数は、エージェントから への接続に使用するプロキシ URL を指定します AWS AppConfig 。 AWS のサービスHTTPSおよび HTTP URLsがサポートされています。	なし	http://localhost:7474 https://my-proxy.example.com

環境変数	詳細	デフォルト値	サンプル値 (複数可)
REQUEST_TIMEOUT	この環境変数は、エージェントがレスポンスを待機する時間を制御します AWS AppConfig。サービスが応答しない場合、リクエストは失敗となります。 リクエストが初期データ取得に関するものである場合、エージェントはアプリケーションにエラーを返します。 更新データのバックグラウンドチェック中にタイムアウトが発生した場合、エージェントはエラーを記録し、しばらくしてから再試行します。 タイムアウトのミリ秒数を指定できます。時間単位で数値を指定することもできます。msはミリ秒、sは秒を表します。単位が指定されなかった場合、エージェントはデフォルトでミリ秒になります。た	3000 ミリ秒	3000 3000 ミリ秒 5 秒

環境変数	詳細	デフォルト値	サンプル値 (複数可)
	とえば、5000 、5000 ミリ秒、5 秒の場合、リクエストのタイムアウト値は同じになります。		
`ROLE_ARN`	この環境変数は、IAM ロールの Amazon リソースネーム (ARN) を指定します。 AWS AppConfig エージェントはこのロールを引き受けて設定データを取得します。	なし	arn:aws:iam::123456789012:role/MyRole
`ROLE_EXTERNAL_ID`	この環境変数は、引き受けたロール ARN で使用する外部 ID を指定します。	なし	MyExternalId
`ROLE_SESSION_NAME`	この環境変数は、引き受ける IAM ロールの認証情報に関連付けるセッション名を指定します。	なし	AWSAppConfigAgentSession

環境変数	詳細	デフォルト値	サンプル値 (複数可)
SERVICE_REGION	この環境変数は、AWS リージョン エージェントが AWS AppConfig サービスを呼び出すために使用する代替 AWS AppConfig 手段を指定します。未定義のままにすると、エージェントは現在のリージョンを特定しようとします。確認できない場合、エージェントは起動に失敗します。	なし	us-east-1 eu-west-1
WAIT_ON_MANIFEST	この環境変数は、起動が完了する前にマニフェストが処理されるまで待機するように AWS AppConfig エージェントを設定します。	真	真 false

ステップ 5: (必須) 設定データの取得

エージェントから設定データを取得するには、HTTP localhost 呼び出し AWS AppConfig を使用します。以下の例は HTTP クライアントで `curl` を使用しています。アプリケーション言語または AWS SDK を含む使用可能なライブラリでサポートされている任意の HTTP クライアントを使用してエージェントを呼び出すことができます。

デプロイされた設定内容をすべて取得するには

```
$ curl "http://localhost:2772/applications/application_name/
environments/environment_name/configurations/configuration_name"
```

AWS AppConfig タイプの **Feature Flag** 設定から個々のフラグとその属性を取得します。

```
$ curl "http://localhost:2772/applications/application_name/
environments/environment_name/configurations/configuration_name?flag=flag_name"
```

AWS AppConfig タイプの **Feature Flag** 設定から複数のフラグとその属性にアクセスします。

```
$ curl "http://localhost:2772/applications/application_name/
environments/environment_name/configurations/configuration_name?
flag=flag_name_one&flag=flag_name_two"
```

ステップ 6 (オプション、ただし推奨): AWS AppConfig エージェントの更新を自動化する

AWS AppConfig エージェントは定期的に更新されます。インスタンスで最新バージョンの AWS AppConfig エージェント を実行していることを確認するために、Amazon EC2 ユーザーデータに次のコマンドを追加することをお勧めします。コマンドは、インスタンスまたは EC2 Auto Scaling グループのユーザーデータに追加できます。このスクリプトは、インスタンスが起動または再起動するたびに、最新バージョンのエージェントをインストールして起動します。

```
#!/bin/bash
# install the latest version of the agent
yum install -y https://s3.amazonaws.com/aws-appconfig-downloads/aws-appconfig-agent/
linux/x86_64/latest/aws-appconfig-agent.rpm
# optional: configure the agent
mkdir /etc/systemd/system/aws-appconfig-agent.service.d
echo "${MY_AGENT_CONFIG}" > /etc/systemd/system/aws-appconfig-agent.service.d/
overrides.conf
systemctl daemon-reload
# start the agent
systemctl start aws-appconfig-agent
```

Amazon ECS および Amazon EKS での AWS AppConfig エージェントの使用

AWS AppConfig エージェントを使用して、Amazon Elastic Container Service (Amazon ECS) および Amazon Elastic Kubernetes Service (Amazon EKS) AWS AppConfig と統合できます。エージェントは Amazon ECS および Amazon EKS コンテナアプリケーションと並行して実行されるサイドカー

コンテナとして機能します。エージェントは、次の方法でコンテナ化されたアプリケーションの処理と管理を強化します。

- エージェントは AWS AppConfig 、 AWS Identity and Access Management (IAM) ロールを使用して設定データのローカルキャッシュを管理することで、ユーザーに代わって を呼び出します。ローカルキャッシュから設定データを引き出すことで、アプリケーションが設定データを管理するために必要となるコードの更新が少なくなり、設定データをミリ秒単位で取得でき、そのようなデータの呼び出しを妨げるネットワークの問題による影響を受けなくなります。*
- エージェントは、 AWS AppConfig 機能フラグを取得して解決するためのネイティブエクスペリエンスを提供します。
- すぐに使用できるエージェントで、キャッシュ戦略、ポーリング間隔、ローカル設定データの可用性に関するベストプラクティスを提供すると同時に、以降のサービスコールに必要な設定トークンを追跡します。
- バックグラウンドで実行中、エージェントは設定 AWS AppConfig データの更新のためにデータプレーンを定期的にポーリングします。コンテナ化されたアプリケーションは、ポート 2772 (カスタマイズ可能なデフォルトポート値) で localhost に接続し、HTTP GET を呼び出してデータを取得できます。
- AWS AppConfig エージェントは、コンテナを再起動またはリサイクルすることなく、コンテナの設定データを更新します。

*AWS AppConfig エージェントは、サービスが設定データを初めて取得したときにデータをキャッシュします。このため、データを取得する最初の呼び出しは、それ以降の呼び出しよりも時間がかかります。

[開始する前に]

コンテナアプリケーション AWS AppConfig と統合するには、機能フラグやフリーフォーム設定データを含む AWS AppConfig アーティファクトと設定データを作成する必要があります。詳細については、「で機能フラグとフリーフォーム設定データを作成する AWS AppConfig」を参照してください。

によってホストされる設定データを取得するには AWS AppConfig、コンテナアプリケーションが AWS AppConfig データプレーンにアクセスできるように設定されている必要があります。アプリケーションにアクセス権を付与するには、コンテナサービスの IAM ロールが使用する IAM アクセス権限ポリシーを更新します。具体的には、`appconfig:StartConfigurationSession` と `appconfig:GetLatestConfiguration` ポリシーにとアクションを追加する必要があります。コンテナサービスの IAM ロールには以下が含まれます。

- Amazon ECS タスクロール。
- Amazon EKS ノードロール
- ポッド実行ロール (Amazon EKS AWS Fargate コンテナがコンピューティング処理に Fargate を使用している場合)

ポリシーに権限を追加する方法の詳細については、IAM ユーザーガイドのIAM ID 権限の追加と削除を参照してください。

トピック

- Amazon ECS インテグレーション用 AWS AppConfig エージェントを起動します。
- Amazon EKS インテグレーション AWS AppConfig エージェントを起動します。
- (オプション) Amazon EKS で AWS AppConfig を DaemonSet として実行する
- (オプション) 環境変数を使用して Amazon ECS と Amazon EKS の AWS AppConfig エージェントを設定する
- Amazon ECS および Amazon EKS で実行されているアプリケーションの設定データの取得

Amazon ECS インテグレーション用 AWS AppConfig エージェントを起動します。

AWS AppConfig エージェントサイドカーコンテナは、Amazon ECS 環境で自動的に使用できます。これを使用するには、次の手順で説明されているように起動する必要があります。

Amazon ECS (コンソール) を起動するには

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. ナビゲーションペインで、タスクの定義 を選択します。
3. アプリケーションのタスク定義を選択し、最新のリビジョンを選択します。
4. 新しいリビジョンの作成 、JSON で新しいリビジョンを作成の順に選択します。
5. コンテナをさらに追加 を選択します。
6. 名前 に、 AWS AppConfig エージェントコンテナの一意の名前を入力します。
7. イメージ URI には、**`public.ecr.aws/aws-appconfig/aws-appconfig-agent:2.x`**を入力します。
8. エッセンシャルコンテナ で はい を選択します。
9. ポートマッピングセクションで、ポートマッピングを追加を選択します。
10. コンテナパス に、「**2772**」と入力します。

> **Note**
>
> AWS AppConfig エージェントは、デフォルトでポート 2772 で実行されます。または、別のポートを指定することもできます。

11. 作成 を選択します。Amazon ECS は新しいコンテナリビジョンを作成し、詳細を表示します。
12. ナビゲーションペインで、アプリケーション を選択し、リストからアプリケーションの名前を選択します。
13. サービス タブで、アプリケーションのサービスを選択します。
14. 更新 を選択します。
15. デプロイ設定 の リビジョン で、最新のリビジョンを選択します。
16. 更新 を選択します。Amazon ECS は最新のタスク定義をデプロイします。
17. デプロイが完了したら、設定とタスクタブで AWS AppConfig エージェントが実行されていることを確認できます。タスク タブで、実行中のタスクを選択します。
18. 「コンテナ」セクションで、 AWS AppConfig エージェントコンテナがリストされていることを確認します。
19. AWS AppConfig エージェントが起動したことを確認するには、ログタブを選択します。 AWS AppConfig エージェントコンテナの次のようなステートメントを見つけます。 `[appconfig agent] 1970/01/01 00:00:00 INFO serving on localhost:2772`

> **Note**
>
> 以下の情報に注意してください。
>
> - AWS AppConfig エージェントは長時間実行されるプロセスです。Amazon ECS コンテナのベストプラクティスとして、コンテナのヘルスチェックを設定し、特にコンテナの依存関係を HEALTHY 条件に設定します。詳細については、「Amazon Elastic Container Service API リファレンス」のContainerDependency」を参照してください。
> - 環境変数を入力または変更することで、 AWS AppConfig エージェントのデフォルトの動作を調整できます。利用可能な環境変数の詳細については、「(オプション) 環境変数を使用して Amazon ECS と Amazon EKS の AWS AppConfig エージェントを設定する」を参照してください。Amazon ECS の環境変数を変更する方法については、『Amazon

Elastic Container Service 開発者ガイド』の「環境変数をコンテナに渡す」を参照してください。

Amazon EKS インテグレーション AWS AppConfig エージェントを起動します。

AWS AppConfig エージェントサイドカーコンテナは、Amazon EKS 環境で自動的に使用できます。これを使用するには、開始する必要があります。次の手順では、Amazon EKS `kubectl` コマンドラインツールを使用してエージェントを起動する方法について説明します。

> **Note**
>
> 続行する前に、`kubeconfig` ファイルが最新であることを確認します。`kubeconfig` ファイルの作成または編集の詳細については、Amazon EKS ユーザーガイドの「Amazon EKS クラスター用の kubeconfig ファイルの作成または更新」を参照してください。

AWS AppConfig エージェントを開始するには (kubectl コマンドラインツール)

1. アプリケーションのマニフェストを開き、Amazon EKS アプリケーションが単一コンテナデプロイとして実行されていることを確認します。ファイルのコンテンツは以下のようになっているはずです。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app
  namespace: my-namespace
  labels:
    app: my-application-label
spec:
  replicas: 1
  selector:
    matchLabels:
      app: my-application-label
  template:
    metadata:
      labels:
        app: my-application-label
    spec:
      containers:
```

```
      - name: my-app
        image: my-repo/my-image
        imagePullPolicy: IfNotPresent
```

2. AWS AppConfig エージェントコンテナ定義の詳細をデプロイマニフェストに追加します。

```
- name: appconfig-agent
        image: public.ecr.aws/aws-appconfig/aws-appconfig-agent:2.x
        ports:
        - name: http
          containerPort: 2772
          protocol: TCP
        env:
        - name: SERVICE_REGION
          value: AWS #####
        imagePullPolicy: IfNotPresent
```

> **Note**
>
> 以下の情報に注意してください。
>
> - AWS AppConfig エージェントは、デフォルトでポート 2772 で実行されます。または、別のポートを指定することもできます。
> - 環境変数を入力して、 AWS AppConfig エージェントのデフォルトの動作を調整できます。詳細については、「(オプション) 環境変数を使用して Amazon ECS と Amazon EKS の AWS AppConfig エージェントを設定する」を参照してください。
> - には*AWS #####*、 AWS AppConfig エージェントが設定データを取得する AWS リージョン コード (などus-west-1) を指定します。

3. 次の kubectl コマンドを実行して、クラスターに変更を適用します。*my-deployment* をデプロイメントマニフェストの名前に置き換えます。

```
kubectl apply -f my-deployment.yml
```

4. デプロイが完了したら、 AWS AppConfig エージェントが実行されていることを確認します。アプリケーション Pod のログファイルを表示するには、次のコマンドを使用します。

```
kubectl logs -n my-namespace -c appconfig-agent my-pod
```

AWS AppConfig エージェントコンテナの次のようなステートメントを見つけます。

```
[appconfig agent] 1970/01/01 00:00:00 INFO serving on localhost:2772
```

> **Note**
>
> 環境変数を入力または変更することで、 AWS AppConfig エージェントのデフォルトの動作を調整できます。利用可能な環境変数の詳細については、「(オプション) 環境変数を使用して Amazon ECS と Amazon EKS の AWS AppConfig エージェントを設定する」を参照してください。

(オプション) Amazon EKS で AWS AppConfig を DaemonSet として実行する

Amazon EKS では、 AWS AppConfig エージェントをサイドカーとして実行できるため、アプリケーションポッドごとに 1 つのエージェントコンテナになります。または、必要に応じて、 AWS AppConfig エージェントを DaemonSet として実行できます。これにより、クラスター内のノードごとに 1 つのエージェントコンテナになります。

> **Note**
>
> AWS AppConfig エージェントを DaemonSet として実行すると、エージェントは別のポッドで実行されます。つまり、 への呼び出しではエージェントにアクセスできませんlocalhost。エージェントポッドを呼び出すには、エージェントポッドの IP アドレスを挿入するか、検出する必要があります。

AWS AppConfig エージェントを DaemonSet として実行するには、次の内容のマニフェストファイルを作成します。*#######*テキストをアプリケーションと環境の詳細に置き換えます。*AWS ####*として、 AWS リージョン コード (us-west-1 など) を指定します。

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: aws-appconfig-agent
  namespace: my_namespace
  labels:
    app: my_application_label
```

```
spec:
   selector:
    matchLabels:
      app: my_application_label
  template:
    metadata:
      labels:
        app: my_application_label
    spec:
      containers:
      - name: aws-appconfig-agent
        image: public.ecr.aws/aws-appconfig/aws-appconfig-agent:2.x
        ports:
        - name: http
          containerPort: 2772
          protocol: TCP
        env:
        - name: SERVICE_REGION
          value: AWS #####
        imagePullPolicy: IfNotPresent
      # set a high priority class to ensure the agent is running on every node
      priorityClassName: system-node-critical
```

次のコマンドを実行して、 AWS AppConfig エージェント DaemonSet をクラスターに適用します。*aws_appconfig_agent_daemonset* を DaemonSet マニフェストの名前に置き換えます。

```
kubectl apply -f aws_appconfig_agent_daemonset.yml
```

(オプション) 環境変数を使用して Amazon ECS と Amazon EKS の AWS AppConfig エージェントを設定する

AWS AppConfig エージェントコンテナの次の環境変数を変更することで、エージェントを設定できます。

> **Note**
>
> 次の表に、サンプル値の列を示します。モニターの解像度によっては、テーブルの下部までスクロールし、右側までスクロールして列を表示する必要がある場合があります。

環境変数	詳細	デフォルト値	サンプル値 (複数可)

ACCESS_TOKEN	この環境変数は、エージェント HTTP サーバーに設定データをリクエストするときに指定する必要があるトークンを定義します。トークンの値は、認可タイプ BearerのHTTP リクエスト認可ヘッダーに設定する必要があります。以下はその例です。
	`GET /applications/my_app/...`
	`Host: localhost:2772`
	`Authorization: Bearer <token value>`
		なし	MyAccessToken

BACKUP_DIRECTORY	この環境変数により、 AWS AppConfig エージェントは取得した各設定のバックアップを指定されたディレクトリに保存できます。 **Important** ディスクにバックアップ	なし	/path/to/backups

環境変数	詳細	デフォルト値	サンプル値 (複数可)
	された設定は暗号化されません。設定に機密データが含まれている場合は、ファイルシステムのアクセス許可で最小特権の原則を実践 AWS AppConfig することをお勧めします。詳細については、「のセキュリティ AWS AppConfig」を参照してください。		
HTTP_PORT	この環境変数は、エージェントの HTTP サーバーが実行されるポートを指定します。	2772	2772

環境変数	詳細	デフォルト値	サンプル値 (複数可)
LOG_LEVEL	この環境変数は、エージェントがログに記録する詳細レベルを指定します。各レベルには、現在レベルとそれより上位のすべてのレベルが含まれます。値は大文字と小文字を区別しません。詳細度が最も低いログレベルは、trace、、debug、i よび で すnone。trace ログには、エージェントに関するタイミング情報を含む詳細情報が含まれます。	情報	trace デバッグ 情報 warn エラー fatal なし
LOG_PATH	ログが書き込まれるディスクがある場所。指定しない場合、ログは stderr に書き込まれます。	なし	/path/to/logs/agent.log

環境変数	詳細	デフォルト値	サンプル値 (複数可)
MANIFEST	この環境変数は、マルチアカウントの取得やディスクへの設定の保存など、設定ごとに追加の機能を利用するように AWS AppConfig エージェントを設定します。これらの機能の詳細については、「マニフェストを使用して追加の取得機能を有効にする」を参照してください。	なし	マニフェストとして AWS AppConfig 設定を使用する場合: MyApp:MyEnv:MyManifestConfig 。 ディスクからマニフェストをロードする場合: file:/path/to/manifest.json
MAX_CONNECTIONS	この環境変数は、エージェントが AWS AppConfigから設定を取得するために使用する最大接続数を設定します。	3	3

環境変数	詳細	デフォルト値	サンプル値 (複数可)
POLL_INTERVAL	この環境変数は、エージェントが AWS AppConfig 更新された設定データをポーリングする頻度を制御します。間隔の秒数を指定できます。時間単位で数値を指定することもできます。s は秒、m は分、h は時間です。単位が指定されなかった場合、エージェントはデフォルトで秒になります。たとえば、60 、60 秒、1m の場合、ポーリング間隔は同じになります。	45 秒	45 45 秒 5m 1 時間
PREFETCH_LIST	この環境変数は、エージェントが起動すると AWS AppConfig すぐにリクエストする設定データを指定します。複数の設定識別子をカンマ区切りリストで指定できます。	なし	MyApp:MyEnv:MyConfig abcd123:efgh456:ijkl789 MyApp:MyEnv:Config1、MyApp:MyEnv:Config2

環境変数	詳細	デフォルト値	サンプル値 (複数可)
PRELOAD_BACKUPS	に設定するとtrue、AWS AppConfig Agent は で見つかった設定バックアップをメモリBACKUP_DIRECTORY にロードし、サービスから新しいバージョンが存在するかどうかをすぐに確認します。false に設定すると、 AWS AppConfig エージェントは、ネットワークに問題があるなど、サービスから設定データを取得できない場合にのみ、設定バックアップからコンテンツをロードします。	真	真 false
PROXY_HEADERS	この環境変数は、PROXY_URL 環境変数で参照されるプロキシに必要なヘッダーを指定します。値は、コンマで区切られたヘッダーの一覧です。	なし	ヘッダー: 値 h1: v1、h2: v2

環境変数	詳細	デフォルト値	サンプル値 (複数可)
PROXY_URL	この環境変数は、エージェントから への接続に使用するプロキシ URL を指定します AWS AppConfig 。 AWS のサービスHTTPSおよび HTTP URLsがサポートされています。	なし	http://localhost:7474 https://my-proxy.example.com

環境変数	詳細	デフォルト値	サンプル値 (複数可)
REQUEST_TIMEOUT	この環境変数は、エージェントがレスポンスを待機する時間を制御します AWS AppConfig。サービスが応答しない場合、リクエストは失敗となります。 リクエストが初期データ取得に関するものである場合、エージェントはアプリケーションにエラーを返します。 更新データのバックグラウンドチェック中にタイムアウトが発生した場合、エージェントはエラーを記録し、しばらくしてから再試行します。 タイムアウトのミリ秒数を指定できます。時間単位で数値を指定することもできます。msはミリ秒、sは秒を表します。単位が指定されなかった場合、エージェントはデフォルトでミリ秒になります。た	3000 ミリ秒	3000 3000 ミリ秒 5 秒

環境変数	詳細	デフォルト値	サンプル値 (複数可)
	とえば、5000 、5000 ミリ秒、5 秒の場合、リクエストのタイムアウト値は同じになります。		
`ROLE_ARN`	この環境変数は、IAM ロールの Amazon リソースネーム (ARN) を指定します。 AWS AppConfig エージェントはこのロールを引き受けて設定データを取得します。	なし	arn:aws:iam::123456789012:role/MyRole
`ROLE_EXTERNAL_ID`	この環境変数は、引き受けたロール ARN で使用する外部 ID を指定します。	なし	MyExternalId
`ROLE_SESSION_NAME`	この環境変数は、引き受ける IAM ロールの認証情報に関連付けるセッション名を指定します。	なし	AWSAppConfigAgentSession

環境変数	詳細	デフォルト値	サンプル値 (複数可)
SERVICE_REGION	この環境変数は、AWS リージョン エージェントが AWS AppConfig サービスを呼び出すために使用する代替 AWS AppConfig 手段を指定します。未定義のままにすると、エージェントは現在のリージョンを特定しようとします。確認できない場合、エージェントは起動に失敗します。	なし	us-east-1 eu-west-1
WAIT_ON_MANIFEST	この環境変数は、起動を完了する前にマニフェストが処理されるまで待機するように AWS AppConfig エージェントを設定します。	真	真 false

Amazon ECS および Amazon EKS で実行されているアプリケーションの設定データの取得

HTTP localhost 呼び出しを使用して、Amazon ECS および Amazon EKS で実行されているアプリケーションの AWS AppConfig エージェントから設定データを取得できます。以下の例は HTTP クライアントで `curl` を使用しています。アプリケーション言語または使用可能なライブラリでサポートされている任意の HTTP クライアントを使用してエージェントを呼び出すことができます。

> **Note**
> アプリケーションがスラッシュを使用している場合 (「test-backend/test-service」など)、設定データを取得するには、URL エンコーディングを使用する必要があります。

デプロイされた設定の完全なコンテンツを取得するには

```
$ curl "http://localhost:2772/applications/application_name/
environments/environment_name/configurations/configuration_name"
```

AWS AppConfig タイプの **Feature Flag** 設定から個々のフラグとその属性を取得します。

```
$ curl "http://localhost:2772/applications/application_name/
environments/environment_name/configurations/configuration_name?flag=flag_name"
```

AWS AppConfig タイプの **Feature Flag** 設定から複数のフラグとその属性にアクセスします。

```
$ curl "http://localhost:2772/applications/application_name/
environments/environment_name/configurations/configuration_name?
flag=flag_name_one&flag=flag_name_two"
```

呼び出しは、設定バージョン、コンテンツタイプ、設定バージョンラベル (該当する場合) を含む設定メタデータを HTTP ヘッダーで返します。エージェントレスポンスの本文には、設定コンテンツが含まれています。以下がその例です。

```
HTTP/1.1 200 OK
Configuration-Version: 1
Content-Type: application/json
Date: Tue, 18 Feb 2025 20:20:16 GMT
Content-Length: 31

My test config
```

基本およびマルチバリアント機能フラグの取得

機能フラグ設定 (タイプ の設定`AWS.AppConfig.FeatureFlags`) の場合、 AWS AppConfig エージェントを使用すると、設定内の単一のフラグまたはフラグのサブセットを取得できます。設定プロ

ファイルからいくつかのフラグのみを使用する必要があるユースケースの場合、1 つまたは 2 つのフラグを取得すると便利です。次の例では、cURL を使用しています。

> **Note**
>
> 設定で単一の機能フラグまたはフラグのサブセットを呼び出す機能は、 AWS AppConfig エージェントバージョン 2.0.45 以降でのみ使用できます。

ローカル HTTP エンドポイントから AWS AppConfig 設定データを取得できます。特定のフラグまたはフラグのリストにアクセスするには、`?flag=FLAG_KEY` 設定プロファイルの AWS AppConfig クエリパラメータを使用します。

1 つのフラグとその属性を取得するには

```
curl "http://localhost:2772/applications/APPLICATION_NAME/
environments/ENVIRONMENT_NAME/configurations/CONFIGURATION_NAME?flag=FLAG_KEY"
```

複数のフラグとその属性を取得するには

```
curl "http://localhost:2772/applications/APPLICATION_NAME/
environments/ENVIRONMENT_NAME/configurations/CONFIGURATION_NAME?
flag=FLAG_KEY_ONE&flag=FLAG_KEY_TWO"
```

発信者コンテキストに基づいて機能フラグバリアントを取得するには

次の Python の例は、発信者コンテキストに基づいて機能フラグバリアントを取得する方法を示しています。これらの呼び出しを行う方法を分かりやすく説明するために、このセクションでは、顧客が次のバリアントを作成したシナリオに基づいてサンプル呼び出しを使用します。

Feature flag variants Info

Reorder variant up | Reorder variant down | Edit | Create variant

Name	Enabled value	Attribute values	Rule
beta testers	ON	-	(or (eq $userId "Alice") (eq $userId "123456789012"))
EU demographic	ON	-	(and (ends_with $email "@example.com") (eq $continent "EU"))
QA testing	ON	-	(and (matches pattern::".*@example\ \.com" in::$email) (contains $roles "Engineer") (gt $tenure 5))
default	ON	-	-

Variant order is used for evaluation logic
Variants are evaluated as an ordered list based on the order shown and any specified rules. The variant at the top of the list is evaluated first. If no rules match the supplied context, AWS AppConfig returns the default variant.

> **Note**
>
> フラグバリアントを取得するには、コンピューティング環境で最新バージョンの AWS AppConfig エージェントを使用する必要があります。詳細については、以下の各コンピューティング環境のエージェントを更新、インストール、または追加する方法を説明する以下のトピックを参照してください。
>
> - Lambda コンピューティング環境の場合: AWS AppConfig エージェント Lambda 拡張機能の追加
> - Amazon EC2 コンピューティング環境の場合: ステップ 2: (必須) Amazon EC2 インスタンスに AWS AppConfig エージェントをインストールして起動する
> - Amazon ECS コンピューティング環境の場合: Amazon ECS インテグレーション用 AWS AppConfig エージェントを起動します。
> - Amazon EKS コンピューティング環境の場合: Amazon EKS インテグレーション AWS AppConfig エージェントを起動します。

jane_doe@example.org (ベータプログラムにオプトインしていないユーザー) の発信者コンテキストを使用してフラグデータを取得するには:

```
curl http://localhost:2772/applications/UIRefresh/environments/Production/
configurations/Features \
-H "Context: email=jane_doe@example.org" \
-H "Context: opted_in_to_beta=false"
{
  "ui_refresh": {"_variant":"QA","dark_mode_support":true,"enabled":true}
```

```
}
```

jane_doe@example.org (ベータプログラムにオプトインしているユーザー) の発信者コンテキストを使用してフラグデータを取得するには:

```
curl http://localhost:2772/applications/UIRefresh/environments/Production/
configurations/Features \
-H "Context: email=jane_doe@example.org" \
-H "Context: opted_in_to_beta=true"
{
  "ui_refresh": {"_variant":"QA","dark_mode_support":true,"enabled":true}
}
```

jane_doe@qa-testers.example.org (Example Organization の品質保証テスター) の発信者コンテキストを使用してフラグデータを取得するには:

```
curl http://localhost:2772/applications/UIRefresh/environments/Production/
configurations/Features \
-H "Context: email=jane_doe@qa-testers.example.org"
{
  "ui_refresh": {"_variant":"QA","dark_mode_support":true,"enabled":true}
}
```

発信者コンテキストなしでフラグデータを取得するには (デフォルトバリアントを返します)

```
curl http://localhost:2772/applications/UIRefresh/environments/Production/
configurations/Features
{
"ui_refresh": {"_variant":"Default Variant","enabled":false}
}
```

トラフィック分割シナリオのフラグデータを取得して、ランダムに選ばれた 10 人の発信者のうち 1 人が「サンプルポピュレーション」バリアントを受信しているかどうかを判断するには

```
for i in {0..9} do ; \
curl http://localhost:2772/applications/UIRefresh/environments/Production/
configurations/Features \
-H "Context: email=$i@example.org"
{
  "ui_refresh": {"_variant":"Default Variant","enabled":false}
}
```

```
{
  "ui_refresh": {"_variant":"Default Variant","enabled":false}
}
{
  "ui_refresh": {"_variant":"Default Variant","enabled":false}
}
{
  "ui_refresh": {"_variant":"Default Variant","enabled":false}
}
{
  "ui_refresh": {"_variant":"Sample
 Population","dark_mode_support":false,"enabled":true}
}
{
  "ui_refresh": {"_variant":"Default Variant","enabled":false}
}
{
  "ui_refresh": {"_variant":"Default Variant","enabled":false}
}
{
  "ui_refresh": {"_variant":"Default Variant","enabled":false}
}
{
  "ui_refresh": {"_variant":"Default Variant","enabled":false}
}
{
  "ui_refresh": {"_variant":"Default Variant","enabled":false}
}
{
  "ui_refresh": {"_variant":"Default Variant","enabled":false}
}
```

マニフェストを使用して追加の取得機能を有効にする

AWS AppConfig エージェントには、アプリケーションの設定を取得するのに役立つ以下の追加機能があります。

- 複数のアカウントから設定を取得するように AWS AppConfig エージェントを設定する: プライマリまたは取得 AWS アカウント から AWS AppConfig エージェントを使用して、複数のベンダーアカウントから設定データを取得します。
- ディスクに設定のコピーを書き込むように AWS AppConfig エージェントを設定する: AWS AppConfig エージェントを使用して設定データをディスクに書き込みます。この機能を使用す

ると、ディスクから設定データを読み取るアプリケーションを使用しているお客様は、 AWS AppConfigと統合できます。

エージェントマニフェストについて

これらの AWS AppConfig エージェント機能を有効にするには、マニフェストを作成します。マニフェストは、エージェントが実行できるアクションを制御するために提供する設定データのセットです。マニフェストは JSON で記述されます。これには、 を使用してデプロイしたさまざまな設定に対応する一連の最上位キーが含まれています AWS AppConfig。

マニフェストには複数の設定を含めることができます。さらに、マニフェスト内の各設定は、指定された設定に使用する 1 つ以上のエージェント機能を識別できます。マニフェストのコンテンツは、次の形式を使用します。

```
{
    "application_name:environment_name:configuration_name": {
        "agent_feature_to_enable_1": {
            "feature-setting-key": "feature-setting-value"
        },
        "agent_feature_to_enable_2": {
            "feature-setting-key": "feature-setting-value"
        }
    }
}
```

以下は、2 つの設定を持つマニフェストの JSON の例です。最初の設定 (*MyApp*) では、 AWS AppConfig エージェント機能は使用されません。2 番目の設定 (*My2ndApp*) では、ディスクへの書き込み設定コピーとマルチアカウント取得機能を使用します。

```
{
        "MyApp:Test:MyAllowListConfiguration": {},

        "My2ndApp:Beta:MyEnableMobilePaymentsFeatureFlagConfiguration": {
            "credentials": {
                "roleArn": "arn:us-west-1:iam::123456789012:role/MyTestRole",
                "roleExternalId": "00b148e2-4ea4-46a1-ab0f-c422b54d0aac",
                "roleSessionName": "AwsAppConfigAgent",
                "credentialsDuration": "2h"
            },
            "writeTo": {
```

```
                "path": "/tmp/aws-appconfig/my-2nd-app/beta/my-enable-payments-feature-
flag-configuration.json"
            }
        }
    }
```

エージェントマニフェストを提供する方法

マニフェストは、 AWS AppConfig エージェントが読み取れる場所にファイルとして保存できます。または、マニフェストを設定として保存 AWS AppConfig し、エージェントに指示することもできます。エージェントマニフェストを指定するには、`MANIFEST` 環境変数を次のいずれかの値で設定する必要があります。

マニフェストの場所	環境変数の値	ユースケース
ファイル	file:/path/to/agent-manifest.json	マニフェストが頻繁に変更されない場合は、この方法を使用します。
AWS AppConfig 設定	*application-name*:*environment-name*:*configuration-name*	動的な更新には、この方法を使用します。他の設定を保存するのと同じ方法で AWS AppConfig 、設定 AWS AppConfig として に保存されているマニフェストを更新してデプロイできます。
環境変数	マニフェストコンテンツ (JSON)	マニフェストが頻繁に変更されない場合は、この方法を使用します。この方法は、ファイルを公開するよりも環境変数を設定する方が簡単なコンテナ環境に役立ちます。

AWS AppConfig エージェントの変数の設定の詳細については、ユースケースに関連するトピックを参照してください。

- AWS AppConfig エージェント Lambda 拡張機能の設定

- Amazon EC2 での AWS AppConfig エージェントの使用
- Amazon ECS および Amazon EKS での AWS AppConfig エージェントの使用

複数のアカウントから設定を取得するように AWS AppConfig エージェントを設定する

AWS AppConfig エージェントマニフェストに認証情報の上書きを入力することで、複数の AWS アカウント から設定を取得するように AWS AppConfig エージェントを設定できます。認証情報の上書きには、 AWS Identity and Access Management (IAM) ロールの Amazon リソースネーム (ARN)、ロール ID、セッション名、エージェントがロールを引き受けることができる期間が含まれます。

これらの詳細は、マニフェストの [認証情報] セクションに入力します。[認証情報] セクションでは、次の形式を使用します。

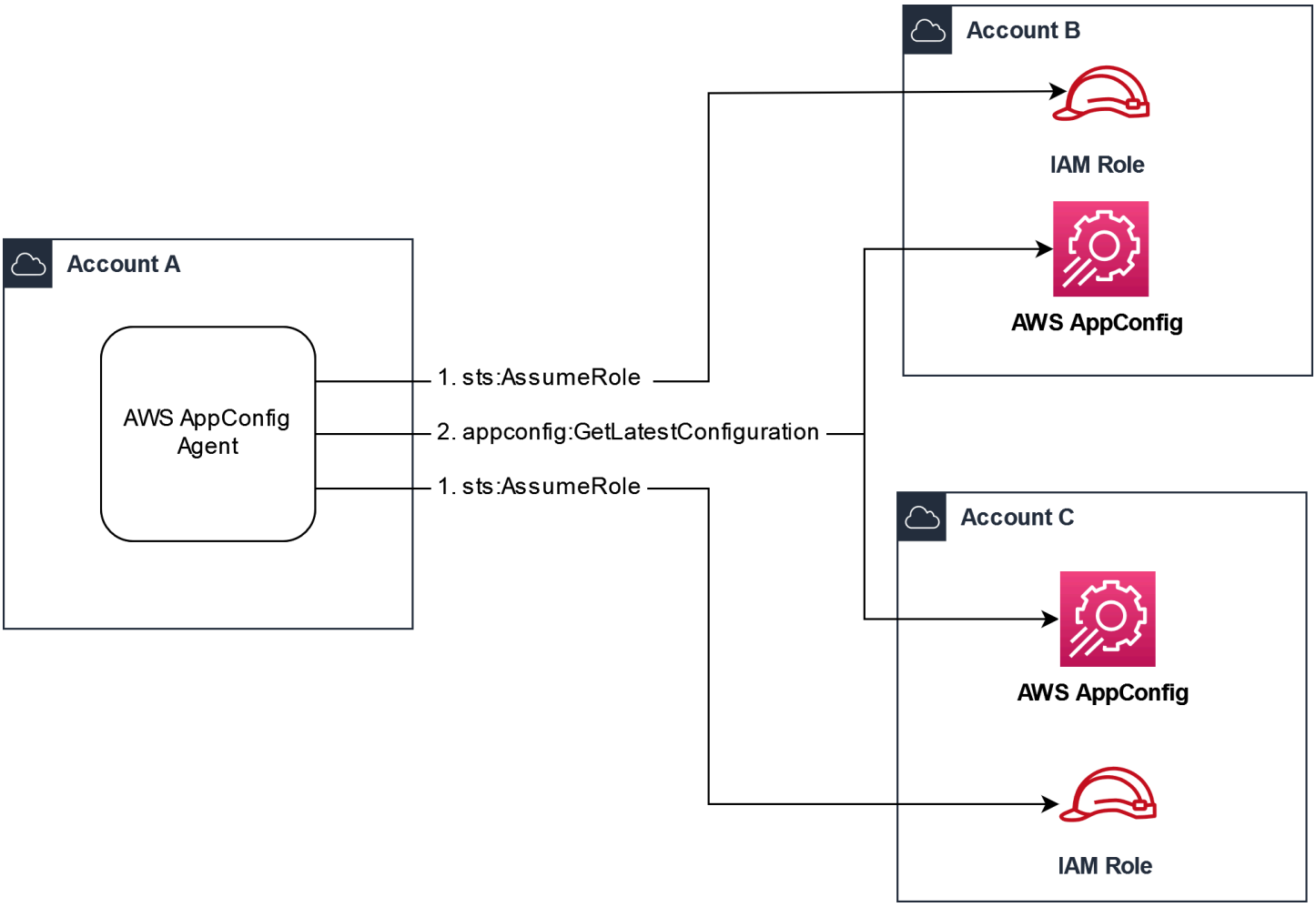
```
{
    "application_name:environment_name:configuration_name": {
        "credentials": {
            "roleArn": "arn:partition:iam::account_ID:role/roleName",
            "roleExternalId": "string",
            "roleSessionName": "string",
            "credentialsDuration": "time_in_hours"
        }
    }
}
```

以下がその例です。

```
{
    "My2ndApp:Beta:MyEnableMobilePaymentsFeatureFlagConfiguration": {
        "credentials": {
            "roleArn": "arn:us-west-1:iam::123456789012:role/MyTestRole",
            "roleExternalId": "00b148e2-4ea4-46a1-ab0f-c422b54d0aac",
            "roleSessionName": "AWSAppConfigAgent",
            "credentialsDuration": "2h"
        }
    }
}
```

設定を取得する前に、エージェントはマニフェストから設定の認証情報の詳細を読み取り、その設定に指定された IAM ロールを引き受けます。1 つのマニフェストで、異なる設定に対して異なる認証

情報オーバーライドのセットを指定できます。次の図は、 AWS AppConfig エージェントがアカウント A (取得アカウント) で実行されているときに、アカウント B と C (ベンダーアカウント) に指定された個別のロールを引き受け、GetLatestConfiguration API オペレーションを呼び出して、それらのアカウントで実行されている設定データ AWS AppConfig を取得する方法を示しています。



ベンダーアカウントから設定データを取得するアクセス許可を設定する

AWS AppConfig 取得アカウントで実行されているエージェントには、ベンダーアカウントから設定データを取得するアクセス許可が必要です。エージェントにアクセス許可を付与するには、各ベンダーアカウントに AWS Identity and Access Management (IAM) ロールを作成します。取得アカウントの AWS AppConfig エージェントはこのロールを引き受けて、ベンダーアカウントからデータを取得します。このセクションの手順を実行して、IAM アクセス許可ポリシーと IAM ロールを作成し、マニフェストにエージェントオーバーライドを追加します。

[開始する前に]

IAM でアクセス許可ポリシーとロールを作成する前に、以下の情報を収集します。

- それぞれの IDs AWS アカウント。取得アカウントは、設定データのために他のアカウントを呼び出すアカウントです。ベンダーアカウントは、設定データを取得アカウントに供給するアカウントです。
- 取得アカウント AWS AppConfig で が使用する IAM ロールの名前。以下に示しているのは AWS AppConfig、デフォルトで で使用されるロールのリストです。
  - Amazon Elastic Compute Cloud (Amazon EC2) の場合、 はインスタンスロール AWS AppConfig を使用します。
  - の場合 AWS Lambda、Lambda 実行ロール AWS AppConfig を使用します。
  - Amazon Elastic Container Service (Amazon ECS) および Amazon Elastic Kubernetes Service (Amazon EKS) の場合、 はコンテナロール AWS AppConfig を使用します。

  `ROLE_ARN` 環境変数を指定して別の IAM ロールを使用するように AWS AppConfig エージェントを設定した場合は、その名前を書き留めます。

アクセス許可ポリシーを作成する

IAM コンソールを使用してアクセス許可ポリシーを作成するには、次の手順に従います。取得アカウントの設定データ AWS アカウント を提供する各 の手順を完了します。

IAM ポリシーを作成するには

1. ベンダーアカウントの AWS Management Console にサインインします。
2. https://console.aws.amazon.com/iam/ で IAM コンソール を開きます。
3. ナビゲーションペインで ポリシーを選択してから ポリシーの作成を選択します。
4. [JSON] オプションを選択します。
5. [ポリシーエディタ] を選択し、デフォルトの JSON を次の JSON ポリシーに置き換えます。*##############*をベンダーアカウントの詳細で更新します。

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": [
            "appconfig:StartConfigurationSession",
            "appconfig:GetLatestConfiguration"
        ],
        "Resource":
 "arn:partition:appconfig:region:vendor_account_ID:application/
```

```
vendor_application_ID/environment/vendor_environment_ID/
configuration/vendor_configuration_ID"
    }
   ]
}
```

例を示します。

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": [
            "appconfig:StartConfigurationSession",
            "appconfig:GetLatestConfiguration"
        ],
        "Resource": "arn:aws:appconfig:us-east-2:111122223333:application/abc123/
environment/def456/configuration/hij789"
    }
   ]
}
```

6. [次へ] を選択します。
7. [ポリシー名] フィールドに名前を入力します。
8. (オプション) [タグを追加] では、1 つ以上のタグキーと値のペアを追加して、このポリシーのアクセスを整理、追跡、または制御できます。
9. [Create policy] (ポリシーの作成) を選択します。システムによってポリシーページに戻ります。
10. 取得アカウントの設定データ AWS アカウント を提供する各 でこの手順を繰り返します。

IAM ロールの作成

IAM コンソールを使用して IAM ロールを作成するには、次の手順に従います。取得アカウントの設定データ AWS アカウント を提供する各 の手順を完了します。

IAM ロールを作成するには

1. ベンダーアカウントの AWS Management Console にサインインします。
2. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
3. ナビゲーションペインで [ロール] を選択し、[ポリシーを作成] を選択します。

4. 信頼できるエンティティタイプ で、AWS アカウント を選択します。
5. [AWS アカウント] セクションで、[別の AWS アカウント] を選択します。
6. [アカウント ID] フィールドに、取得アカウント ID を入力します。
7. (オプション) このロールを引き受ける場合のセキュリティのベストプラクティスとして、[外部 ID が必要] を選択し、文字列を入力します。
8. [次へ] を選択します。
9. [許可を追加] ページで [検索] フィールドを使用し、前の手順で作成したポリシーを見つけます。名前の横にあるチェックボックスを選択します。
10. [次へ] を選択します。
11. [Role name] (ロール名) に名前を入力します。
12. (オプション) [説明] に説明を入力します。
13. [ステップ 1: 信頼されたエンティティを選択] で、[編集] を選択します。デフォルトの JSON 信頼ポリシーを次のポリシーに置き換えます。各################を、取得アカウントの情報で更新します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS":
 "arn:aws:iam::retrieval_account_ID:role/appconfig_role_in_retrieval_account"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

14. (オプション) [Tags] (タグ)で、タグとキーの値のペアを 1 つまたは複数追加して、このロールのアクセスを整理、追跡、または制御します。
15. [Create role] (ロールの作成) を選択します。ロールページが再度表示されます。
16. 作成したロールを検索します。これを選択します。[ARN] セクションで、ARN をコピーします。次の手順でこの情報を指定します。

マニフェストに認証情報オーバーライドを追加する

ベンダーアカウントに IAM ロールを作成したら、取得アカウントのマニフェストを更新します。具体的には、ベンダーアカウントから設定データを取得するための認証情報ブロックと IAM ロール ARN を追加します。JSON 形式は次のとおりです。

```
{
    "vendor_application_name:vendor_environment_name:vendor_configuration_name": {
        "credentials": {
            "roleArn":
 "arn:partition:iam::vendor_account_ID:role/name_of_role_created_in_vendor_account",
            "roleExternalId": "string",
            "roleSessionName": "string",
            "credentialsDuration": "time_in_hours"
        }
    }
}
```

以下がその例です。

```
{
    "My2ndApp:Beta:MyEnableMobilePaymentsFeatureFlagConfiguration": {
        "credentials": {
            "roleArn": "arn:us-west-1:iam::123456789012:role/MyTestRole",
            "roleExternalId": "00b148e2-4ea4-46a1-ab0f-c422b54d0aac",
            "roleSessionName": "AwsAppConfigAgent",
            "credentialsDuration": "2h"
        }
    }
}
```

マルチアカウント取得が機能していることを確認します

エージェント AWS AppConfig ログを確認することで、エージェントが複数のアカウントから設定データを取得できることを検証できます。「YourApplicationName:YourEnvironmentName:YourConfigurationName」の取得された初期データの INFO レベルのログは、取得が成功したことを示す最適な指標です。取得に失敗した場合、失敗の理由を示す ERROR レベルのログが表示されます。ベンダーアカウントからの正常な取得の例を次に示します。

```
[appconfig agent] 2023/11/13 11:33:27 INFO AppConfig Agent 2.0.x
[appconfig agent] 2023/11/13 11:33:28 INFO serving on localhost:2772
```

```
[appconfig agent] 2023/11/13 11:33:28 INFO retrieved initial data for
 'MyTestApplication:MyTestEnvironment:MyDenyListConfiguration' in XX.Xms
```

ディスクに設定のコピーを書き込むように AWS AppConfig エージェントを設定する

エージェントを設定 AWS AppConfig して、設定のコピーをプレーンテキストでディスクに自動的に保存できます。この機能を使用すると、ディスクから設定データを読み取るアプリケーションを使用しているお客様は、 AWS AppConfigと統合できます。

この機能は、設定バックアップ機能として使用するように設計されていません。 AWS AppConfig エージェントは、ディスクにコピーされた設定ファイルから読み取りません。設定をディスクにバックアップする場合は、Amazon EC2 で AWS AppConfig エージェントを使用するか、Amazon ECS `BACKUP_DIRECTORY`と Amazon EKS で エージェントを使用するための および `PRELOAD_BACKUP`環境変数を参照してください。 AWS AppConfig

> **Warning**
>
> この機能に関する次の重要事項に留意してください。
>
> - ディスクに保存されている設定はプレーンテキストで保存され、人間が読み取ることができます。機密データを含む設定では、この機能を有効にしないでください。
> - この機能はローカルディスクに書き込みます。ファイルシステムのアクセス許可には、最小特権の原則を使用します。詳細については、「最小特権アクセスの実装」を参照してください。

ディスクへの設定コピーの書き込みを有効にするには

1. マニフェストを編集します。
2. ディスクに AWS AppConfig 書き込む設定を選択し、 `writeTo` 要素を追加します。以下がその例です。

```
{
    "application_name:environment_name:configuration_name": {
        "writeTo": {
            "path": "path_to_configuration_file"
        }
    }
}
```

以下がその例です。

```
{
    "MyTestApp:MyTestEnvironment:MyNewConfiguration": {
        "writeTo": {
            "path": "/tmp/aws-appconfig/mobile-app/beta/enable-mobile-payments"
        }
    }
}
```

3. 変更内容を保存します。configuration.json ファイルは、新しい設定データがデプロイされるたびに更新されます。

ディスクへの設定コピーの書き込みが機能していることを検証する

エージェント AWS AppConfig ログを確認することで、設定のコピーがディスクに書き込まれていることを確認できます。「INFO wrote configuration '*application* : *environment* : *configuration*' to *file_path*」というフレーズを含むINFOログエントリは、 AWS AppConfig エージェントが設定コピーをディスクに書き込むことを示します。

以下がその例です。

```
[appconfig agent] 2023/11/13 11:33:27 INFO AppConfig Agent 2.0.x
[appconfig agent] 2023/11/13 11:33:28 INFO serving on localhost:2772
[appconfig agent] 2023/11/13 11:33:28 INFO retrieved initial data for
 'MobileApp:Beta:EnableMobilePayments' in XX.Xms
[appconfig agent] 2023/11/13 17:05:49 INFO wrote configuration
 'MobileApp:Beta:EnableMobilePayments' to /tmp/configs/your-app/your-env/your-
config.json
```

OpenAPI 仕様を使用したクライアントの生成

OpenAPI の次の YAML 仕様を使用して、OpenAPI Generator のようなツールを使って SDK を作成できます。この仕様を更新して、アプリケーション、環境、または設定用のハードコーディングされた値を含めることができます。また、パスを追加して (複数の設定タイプがある場合)、設定スキーマを含めて、SDK クライアント用の設定固有の型付けされたモデルを生成することもできます。OpenAPI (Swagger とも呼ばれる) の詳細については、「OpenAPI の仕様」を参照してください。

```
openapi: 3.0.0
```

```
info:
  version: 1.0.0
  title: AWS AppConfig Agent API
  description: An API model for AWS AppConfig Agent.
servers:
  - url: http://localhost:{port}/
    variables:
      port:
        default:
          '2772'
paths:
  /applications/{Application}/environments/{Environment}/configurations/
{Configuration}:
    get:
      operationId: getConfiguration
      tags:
        - configuration
      parameters:
        - in: path
          name: Application
          description: The application for the configuration to get. Specify either the
 application name or the application ID.
          required: true
          schema:
            type: string
        - in: path
          name: Environment
          description: The environment for the configuration to get. Specify either the
 environment name or the environment ID.
          required: true
          schema:
            type: string
        - in: path
          name: Configuration
          description: The configuration to get. Specify either the configuration name
 or the configuration ID.
          required: true
          schema:
            type: string
        - in: query
          name: flag
          description: The key(s) of the feature flag(s) to retrieve. If not provided,
 all flags are returned.
          required: false
```

```
          schema:
            type: array
            items:
              type: string
        - in: header
          name: context
          description: Request context used to evaluate multi-variant feature flags.
          required: false
          schema:
            type: array
            items:
              type: string
              pattern: '^\w+=\w+$'
      responses:
        200:
          headers:
            ConfigurationVersion:
              schema:
                type: string
          content:
            application/octet-stream:
              schema:
                type: string
                format: binary
          description: successful config retrieval
        400:
          description: BadRequestException
          content:
            application/text:
              schema:
                $ref: '#/components/schemas/Error'
        404:
          description: ResourceNotFoundException
          content:
            application/text:
              schema:
                $ref: '#/components/schemas/Error'
        500:
          description: InternalServerException
          content:
            application/text:
              schema:
                $ref: '#/components/schemas/Error'
        502:
```

```
          description: BadGatewayException
          content:
            application/text:
              schema:
                $ref: '#/components/schemas/Error'
        504:
          description: GatewayTimeoutException
          content:
            application/text:
              schema:
                $ref: '#/components/schemas/Error'

components:
  schemas:
    Error:
      type: string
      description: The response error
```

AWS AppConfig エージェントローカル開発モードの使用

AWS AppConfig エージェントはローカル開発モードをサポートしています。ローカル開発モードを有効にすると、エージェントはディスク上の指定されたディレクトリから設定データを読み取ります。設定データは取得されません AWS AppConfig。指定されたディレクトリ内のファイルを更新することで、設定のデプロイをシミュレートできます。次のユースケースでは、ローカル開発モードをお勧めします。

- AWS AppConfigを使用して、デプロイする前にさまざまな設定バージョンをテストする。
- コードリポジトリに変更をコミットする前に、新機能のさまざまな設定オプションをテストする。
- さまざまな設定シナリオをテストして、期待どおりに動作することを確認する。

> **Warning**
>
> 本番稼働環境では、ローカル開発モードを使用しないでください。このモードは、デプロイの検証や自動ロールバックなどの重要な AWS AppConfig 安全機能をサポートしていません。

エージェントをローカル開発モードに設定する AWS AppConfig には、次の手順に従います。

AWS AppConfig エージェントをローカル開発モードに設定するには

1. コンピューティング環境で説明されている方法を使用して エージェントをインストールします。 AWS AppConfig エージェントは以下を使用します AWS のサービス。
   - AWS Lambda
   - Amazon EC2
   - Amazon ECS と Amazon EKS
2. エージェントが実行中の場合は停止します。
3. 環境変数のリストに `LOCAL_DEVELOPMENT_DIRECTORY` を追加します。エージェントに読み取りアクセス許可を提供するファイルシステム上のディレクトリを指定します。例えば、`/tmp/local_configs`。
4. ディレクトリ内にファイルを作成します。ファイル名は以下のフォーマットを使用する必要があります。

   ```
   application_name:environment_name:configuration_profile_name
   ```

   以下がその例です。

   ```
   Mobile:Development:EnableMobilePaymentsFeatureFlagConfiguration
   ```

   > **Note**
   >
   > - `LOCAL_DEVELOPMENT_DIRECTORY` ディレクトリ内のファイルに追加できる機能フラグのサンプルを表示するには、「」を参照してくださいAWS AppConfig エージェントローカル開発モードの機能フラグのサンプル。
   > - (オプション) ファイルで指定した拡張子に基づいて、エージェントが設定データに対して返すコンテンツタイプを制御できます。例えば、ファイルに .json 拡張子を付けると、アプリケーションがリクエストしたときにエージェントは `application/json` コンテンツタイプを返します。拡張子を省略すると、エージェントはコンテンツタイプに `application/octet-stream` を使用します。正確な制御が必要な場合は、`.type%subtype` 形式で拡張子を指定できます。エージェントは `.type/subtype` のコンテンツタイプを返します。
5. 次のコマンドを実行してエージェントを再起動し、設定データをリクエストします。

```
curl http://localhost:2772/applications/application_name/
environments/environment_name/configurations/configuration_name
```

エージェントは、エージェントに指定されたポーリング間隔でローカルファイルへの変更をチェックします。ポーリング間隔が指定されていない場合、エージェントはデフォルトの間隔である 45 秒を使用します。ポーリング間隔でのこのチェックにより、エージェントはローカル開発環境で、 AWS AppConfig サービスとやり取りするように設定されたときと同じように動作します。

> **Note**
>
> ローカル開発設定ファイルの新しいバージョンをデプロイするには、新しいデータでファイルを更新します。

AWS AppConfig エージェントローカル開発モードの機能フラグのサンプル

このセクションには、ローカル開発モードで AWS AppConfig エージェントで使用できる機能フラグのサンプルが含まれています。ローカル開発モードでは、データの取得時刻形式の機能フラグデータが想定されます。取得時刻形式は、 フラグが GetLatestConfiguration API から取得されたときに返される形式です。これには、 フラグの値のみが含まれます。取得時刻形式には、フラグの完全な定義は含まれません (CreateHostedConfigurationVersion API に渡されます) 。フラグの完全な定義には、属性名と値、制約、フラグの有効状態などの情報も含まれています。

トピック

- 基本的な機能フラグのサンプル
- マルチバリアント機能フラグのサンプル

基本的な機能フラグのサンプル

ローカル開発モードで AWS AppConfig エージェントで次の基本機能フラグサンプルを使用します。

> **Note**
>
> エージェントがローカル機能フラグデータのコンテンツタイプを `application/json` (ローカル開発モードではない環境で からフラグデータを取得する場合と同様に) としてレポート

する場合は、ローカル機能フラグファイル AWS AppConfig で .json 拡張機能を使用する必要があります。例えば、`Local:MyFeatureFlags:SampleB1.json`。

サンプル 1: UI 更新を表す単一のフラグ。

```
{
    "ui_refresh": {
        "enabled": true,
        "new_styleguide_colors": true
    }
}
```

サンプル 2: 運用機能フラグを表す複数のフラグ。

```
{
   "background_worker": {
        "enabled": true,
        "num_threads": 4,
        "queue_name": "MyWorkQueue"
   },
   "emergency_shutoff_switch": {
        "enabled": false
   },
   "logger_settings": {
        "enabled": true,
        "level": "INFO"
   }
}
```

マルチバリアント機能フラグのサンプル

少なくとも 1 つのマルチバリアント機能フラグを含む機能フラグ設定の取得時刻形式は、JSON データではなく Amazon Ion データとして表されます。この形式では、マルチバリアントフラグは注釈付きリストとして表され、基本フラグは注釈付き文字列として表されます。マルチバリアントフラグのリスト要素は、単一のバリアントを表すタプル (長さが 2 のリスト) またはデフォルトのバリアントを表す文字列です。バリアントタプル内では、最初の要素はバリアントのルールを表す s 式で、2 番目の要素はバリアントのコンテンツを表す文字列です。

エージェントがこれらのファイルを適切に解釈するには、ローカル機能フラグファイルで拡張子 を使用する必要があります`application%ion%type=AWS.AppConfig.FeatureFlags`。

例えば、`Local:MyFeatureFlags:SampleMV1.application%ion%type=AWS.AppConfig.FeatureFlags`。

サンプル 1: 新機能の階層型リリースを表す多変量フラグ。

```
'tiered_release'::[
  [
    (or (and (eq $group "Tier1") (split by::$userId pct::1 seed::"2025.01.01")) (and
 (eq $group "Tier2") (split by::$userId pct::7 seed::"2025.01.01"))),
    '''{"_variant": "ShowFeature", "enabled": true}'''
  ],
  '''{"_variant": "HideFeature", "enabled": false}'''
]
```

サンプル 2: ユーザーの ID に基づいて異なる UX を表す複数のフラグが表示されます。最初の 2 つのフラグはマルチバリアントで、最後のフラグは基本です。

```
'colorway'::[
  [
    (contains $userId "beta"),
    '''{"_variant": "BetaTesters", "enabled": true, "background": "blue", "foreground":
 "red"}''',
  ],
  [
    (split by::$userId pct::10),
    '''{"_variant": "SplitRollOutRedAndBlue", "enabled": true, "background": "blue",
 "foreground": "red"}''',
  ],
  '''{"_variant": "default", "enabled": true, "background": "green", "foreground":
 "green"}''',
]

'simple_feature'::[
  [
    (contains $userId "beta"),
    '''{"_variant": "BetaTesters", "enabled": true}'''
  ],
  '''{"_variant": "default", "enabled": false}'''
]

'button_color'::'''{"enabled": true, "color": "orange"}'''
```

AWS AppConfig ブラウザとモバイルの使用に関する考慮事項

機能フラグを使用すると、アプリケーションストアリリースのオーバーヘッド、リスク、または剛性なしに、ウェブページとモバイルアプリケーションのエクスペリエンスをその場で更新できます。機能フラグを使用すると、選択した時点でユーザーベースに変更を徐々にリリースできます。エラーが発生した場合は、ユーザーが新しいソフトウェアバージョンにアップグレードすることなく、変更をすぐにロールバックできます。つまり、機能フラグにより、アプリケーションに変更をデプロイする際の制御と柔軟性が向上します。

以下のセクションでは、ウェブページとモバイルデバイスで AWS AppConfig 機能フラグを使用する際の重要な考慮事項について説明します。

トピック

- 設定データとフラグの取得
- 認証と Amazon Cognito
- キャッシュ
- セグメンテーション
- 帯域幅 (モバイルユースケース)
- その他のフラグのユースケース

設定データとフラグの取得

ブラウザとモバイルのユースケースでは、多くのお客様がウェブまたはモバイルアプリケーションと の間にプロキシレイヤーを採用することを選択します AWS AppConfig。これにより、 AWS AppConfig 通話量がユーザーベースのサイズから切り離されるため、コストが削減されます。また、AWS AppConfig エージェントを活用してフラグの取得パフォーマンスを最適化し、マルチバリアント flags などの機能をサポートすることもできます。 を使用してプロキシ AWS Lambda を作成する AWS AppConfig ことをお勧めします。フラグを直接取得する代わりに AWS AppConfig、AWS AppConfig Lambda 関数内の機能フラグを取得するように Lambda 拡張機能を設定します。イベントリクエストから AWS AppConfig の取得パラメータを受け入れ、対応する設定データを Lambda レスポンスに返す関数を記述します。Lambda 関数 URLs。

プロキシを設定したら、データを取得する頻度を考慮します。モバイルユースケースでは、通常、高頻度ポーリング間隔は必要ありません。アプリケーションがプロキシから更新するよりも AWS AppConfig 頻繁にデータを更新するように AWS AppConfig エージェントを設定します。

認証と Amazon Cognito

Lambda 関数 URLs、 `AWS_IAM`と の 2 つのアクセスコントロール形式をサポートしていますNONE。Lambda 関数に独自の認証と認可を実装NONEする場合は、 を使用します。 NONEは、ユースケースでエンドポイントを公開することを許可し、設定データに機密データが含まれていない場合にも推奨されるオプションです。その他のすべてのユースケースでは、 を使用しますAWS_IAM。

> **Important**
>
> 認証なしでエンドポイントをインターネットに公開する場合は、個人を特定できる情報 (PII)、ユーザー IDs、未公開の機能名などの機密データが設定データに漏れないようにしてください。

を使用する場合はAWS_IAM、Amazon Cognito で認証情報を管理する必要があります。Amazon Cognito の使用を開始するには、ID プールを作成します。ID プールを使用すると、認証されたユーザーまたはゲストユーザーのために、短期認証情報をアプリケーションに供給できます。ユーザーが Lambda 関数InvokeFunctionUrlに を使用できるようにするロールを ID プールに追加する必要があります。これにより、アプリケーションのインスタンスは設定データを取得するために必要な認証情報にアクセスできます。

アプリケーションで Amazon Cognito を使用する場合は、 の使用を検討してくださいAWS Amplify。Amplify は、 とのモバイル/ウェブアプリケーションのやり取りを簡素化 AWS し、Amazon Cognito の組み込みサポートを提供します。

キャッシュ

を使用する場合は AWS AppConfig、常に設定データをデバイスまたはブラウザでローカルにキャッシュする必要があります。キャッシュには以下の利点があります。

- レイテンシーとバッテリードレインを削減してパフォーマンスを向上
- ネットワークアクセスへの依存関係を排除することで安定性を実現
- データ取得頻度を減らすことでコストを削減

モバイルユースケースでは、インメモリキャッシュと永続オンデバイスキャッシュを実装することをお勧めします。インメモリキャッシュから必要な設定を取得しようとし、必要に応じてプロキシからの取得にフォールバックするようにアプリケーションを設定します。プロキシから正常に取得された

ら、インメモリキャッシュを更新し、設定をデバイスに保持します。バックグラウンドプロセスを使用してキャッシュを反復処理し、各設定を更新します。アプリケーション起動後に初めて設定を取得する場合、取得に失敗した場合は、永続設定に延期します (それを使用してインメモリキャッシュをシードします) 。

セグメンテーション

機能フラグを使用する場合、機能フラグ付けエクスペリエンスを顧客ベース全体でセグメント化できます。そのためには、フラグの取得呼び出しにコンテキストを指定し、提供されたコンテキストに基づいて機能フラグのさまざまなバリアントを返すようにルールを設定します。たとえば、iOS 18.X ユーザー用の機能フラグバリアント、iOS 17.X ユーザー用のバリアント、および他のすべてのバージョンの iOS 用のデフォルトフラグがあるとします。バリアントを使用すると、同じ環境で同じ設定をターゲットにするようにアプリケーションのすべての iOS バージョンを設定できますが、取得呼び出しで指定されたコンテキスト (「バージョン」 : 「iOS18.1」など) に基づいて、デバイスは設定の適切なバリアントを受け取ります。

> **Note**
>
> モバイルユースケースで AWS AppConfig 機能フラグバリアントを使用している場合は、AWS AppConfig エージェントとプロキシを使用して機能フラグを取得する必要があります。

AWS AppConfig エージェントを使用して機能フラグを取得しない場合は、環境を活用して AWS AppConfig シンプルで低カーディナリティのセグメンテーションを行うことができます。環境は、ターゲットの論理デプロイグループです。設定を開発、テスト、本番環境に分割するだけでなく、デバイスタイプ (タブレットと電話) や OS メジャーバージョンなどのモバイル固有の環境を作成することで、顧客ベースを分割できます。個別の環境では、同じまたは異なる設定データのセットをデプロイして、顧客ベースの特定の要件を満たすことができます。

帯域幅 (モバイルユースケース)

一般的に、各フラグセットのサイズを小さくすることを目指します。モバイルユースケースには、低帯域幅の制約が伴う傾向があります。データのサイズを最小限に抑えることで、ユーザーベース全体で一貫したエクスペリエンスを維持できます。また、モバイルデバイスは多くの場合、低帯域幅環境と無帯域幅環境の間で動作するため、デバイス上のキャッシュが重要です。設定データを取得できない場合に正常に失敗するアプリケーションコードも重要です。

その他のフラグのユースケース

機能フラグの機能は、機能リリースの利便性を超えて拡張されます。長期運用フラグを使用して、アプリケーションの運用体制を改善できます。たとえば、イベント中に追加のメトリクスを発行し、データをデバッグするパフォーマンスモニタリングトグルを作成できます。または、顧客ベースのセグメントのアプリケーション更新レートを維持および調整することもできます。

AWS AppConfig エージェントを使用せずに設定データを取得する

から設定データを取得する推奨方法は、Amazon が開発および管理する AWS AppConfig エージェントを使用すること AWS AppConfig です。エージェントを使用すると、設定データをローカルにキャッシュし、データ AWS AppConfig プレーンサービスを更新のために非同期的にポーリングできます。このキャッシュ/ポーリングプロセスにより、レイテンシーとコストを最小限に抑えながら、アプリケーションで設定データを常に利用できるようになります。エージェントを使用しない場合は、 AWS AppConfig データプレーンサービスから直接パブリック APIsを呼び出すことができます。

データプレーンサービスは、StartConfigurationSession と GetLatestConfiguration という 2 つの API アクションを使用します。データプレーンサービスは、 AWS AppConfig コントロールプレーンとは別のエンドポイントも使用します。

> **Note**
>
> データプレーンサービスは、`GetConfiguration` API アクションを使用して設定データを取得する以前のプロセスに代わるものです。`GetConfiguration` API は非推奨です。

仕組み

データプレーンサービスを使用して AWS AppConfig APIs を直接呼び出すプロセスは次のとおりです。

アプリケーションは、最初に StartConfigurationSession API オペレーションを使用して設定セッションを確立確立することで、設定データを取得します。次に、セッションのクライアントは定期的に GetLatestConfiguration を呼び出し、最新の利用可能なデータを確認して取得します。

`StartConfigurationSession` が呼び出されると、コードは次の情報を送信します。

- セッションが追跡する AWS AppConfig アプリケーション、環境、および設定プロファイルの識別子 (ID または名前) 。

- （オプショナル）セッションクライアントが GetLatestConfiguration を呼び出すまでに待機しなければならない最短時間

レスポンスとして、 はセッションのクライアントに渡す InitialConfigurationToken AWS AppConfig を提供し、そのセッションGetLatestConfigurationを初めて呼び出すときに を使用します。

> **Important**
>
> このトークンは、GetLatestConfiguration の最初の呼び出し時に1回だけ使用する必要があります。以降の GetLatestConfiguration への呼び出しでは、GetLatestConfiguration 応答 NextPollConfigurationToken で新しいトークンを使用する必要があります。トークンは、ロングポーリングのユースケースをサポートするため、最大 24 時間有効です。GetLatestConfiguration コールで期限切れのトークンが使用されると、システムから返されます BadRequestException。

GetLatestConfiguration を呼び出すと、クライアントコードは最新の ConfigurationToken 値を送信し、応答を受信します。

- NextPollConfigurationToken :次回のGetLatestConfigurationへの次の呼び出しで使用するConfigurationToken値。
- NextPollIntervalInSeconds：クライアントが次に GetLatestConfiguration を呼び出すまで待機する時間。
- 設定:セッション用の最新データ。クライアントにすでに最新バージョンの設定がある場合は、この値は空になる可能性があります。

> **Important**
>
> 次の重要な情報に注意してください。
>
> - StartConfigurationSession API は、サービスとのセッションを確立するために、アプリケーション、環境、設定プロファイル、およびクライアントごとに 1 回のみ呼び出す必要があります。これは、通常、アプリケーションの起動時または設定の初回取得の直前に行われます。

- KmsKeyIdentifier を使用して設定をデプロイする場合、設定を受け取るリクエストには kms:Decrypt レスポンス権限が含まれている必要があります。詳細については、AWS Key Management Service API リファレンスの「復号化」を参照してください。
- 以前は設定データを取得するために使用されていた GetConfiguration API オペレーションは廃止されました。GetConfiguration API オペレーションは暗号化された設定をサポートしません。

(例) AWS AppConfig APIs を呼び出して設定を取得する

次の AWS CLI 例は、 AWS AppConfig Data StartConfigurationSessionおよび GetLatestConfiguration API オペレーションを使用して設定データを取得する方法を示しています。最初のコマンドは設定セッションを開始します。この呼び出しには、 AWS AppConfig アプリケーション、環境、および設定プロファイルの IDs (または名前) が含まれます。API は、設定データをフェッチするために使用される InitialConfigurationToken を返します。

```
aws appconfigdata start-configuration-session \
    --application-identifier application_name_or_ID \
    --environment-identifier environment_name_or_ID \
    --configuration-profile-identifier configuration_profile_name_or_ID
```

システムから以下の形式の情報で応答します。

```
{
   "InitialConfigurationToken": initial configuration token
}
```

セッションを開始したら、設定データをフェッチするために、InitialConfigurationToken を使用して GetLatestConfiguration を呼び出します。設定データは、mydata.json ファイルに保存されます。

```
aws appconfigdata get-latest-configuration \
    --configuration-token initial configuration token mydata.json
```

GetLatestConfiguration への初回呼び出しでは、StartConfigurationSession から取得された ConfigurationToken を使用します。次の情報が返されます。

```
{
    "NextPollConfigurationToken" : next configuration token,
```

```
    "ContentType" : content type of configuration,
    "NextPollIntervalInSeconds" : 60
}
```

それ以降の GetLatestConfiguration への呼び出しでは、前の応答から NextPollConfigurationToken を提供する必要があります。

```
aws appconfigdata get-latest-configuration \
    --configuration-token next configuration token mydata.json
```

> **Important**
>
> GetLatestConfiguration API オペレーションに関する以下の重要な詳細に留意してください。
>
> - GetLatestConfiguration 応答には、設定データを示す Configuration セクションが含まれています。Configuration セクションは、新規または更新された設定データをシステムが検出した場合にのみ表示されます。新規または更新された設定データをシステムが検出しない場合、Configuration データは空です。
> - GetLatestConfiguration からの応答があるたびに新しい ConfigurationToken を受け取ります。
> - 予算、予想頻度、設定のターゲット数に基づいて、GetLatestConfiguration API コールのポーリング頻度を調整することをお勧めします。

拡張機能を使用した AWS AppConfig ワークフローの拡張

拡張機能は、設定を作成またはデプロイする AWS AppConfig ワークフローのさまざまな時点でロジックまたは動作を挿入する機能を強化します。例えば、拡張機能で以下タイプのタスク (いくつか例を挙げる) を実行できます。

- 設定プロファイルがデプロイされると、Amazon Simple Notification Service (Amazon SNS) トピックに通知を送信します。
- デプロイを開始する前に、設定プロファイルの内容を調べて機密データを洗い出します。
- 機能フラグに変更が加えられるたびに Atlassian Jira の問題を作成または更新します。
- デプロイの開始時に、サービスまたはデータソースのコンテンツを設定データにマージします。
- 設定をデプロイするたびに、Amazon Simple Storage Service (Amazon S3) のバケットに設定をバックアップします。

これらのタイプのタスクは、 AWS AppConfig アプリケーション、環境、および設定プロファイルに関連付けることができます。

内容

- AWS AppConfig 拡張機能について
- AWS が作成した拡張機能での作業
- チュートリアル: カスタム AWS AppConfig 拡張機能の作成

AWS AppConfig 拡張機能について

このトピックでは、 AWS AppConfig 拡張機能の概念と用語について説明します。この情報は、AWS AppConfig 拡張機能の設定と使用に必要な各ステップのコンテキストで説明されています。

トピック

- ステップ 1: 拡張機能を使用する操作を決定します
- ステップ 2: 拡張機能をいつ実行するかを決める
- ステップ 3: 拡張機能の作成
- ステップ 4: 設定をデプロイし、拡張機能のアクションが実行されたことを確認する

ステップ 1: 拡張機能を使用する操作を決定します

AWS AppConfig デプロイが完了するたびに Slack にメッセージを送信するウェブフックに通知を受け取りますか？ 設定をデプロイする前に、設定プロファイルを Amazon Simple Storage Service (Amazon S3) バケットにバックアップしますか? 設定をデプロイする前に、設定データから機密情報を削除したいですか？ 拡張機能を使用すると、これらのタイプのタスクだけでなく、その他のタスクも実行できます。カスタム拡張機能を作成することも、 に含まれている AWS 作成済みの拡張機能を使用することもできます AWS AppConfig。

> **Note**
>
> ほとんどのユースケースでは、カスタム拡張機能を作成するには、拡張機能で定義された計算と処理を実行する AWS Lambda 関数を作成する必要があります。詳細については、「チュートリアル: カスタム AWS AppConfig 拡張機能の作成」を参照してください。

次の AWS 作成された拡張機能は、設定デプロイを他の サービスとすばやく統合するのに役立ちます。これらの拡張機能は、 AWS AppConfig コンソールで、または、 AWS CLI AWS Tools for PowerShell、 SDK から直接拡張機能 API アクションを呼び出すことで使用できます。

拡張機能	説明
Amazon CloudWatch Evidently A/B テスト中	この拡張機能により、アプリケーションは、EvaluateFeature オペレーションを呼び出す代わりに、ローカルでユーザーセッションにバリエーションを割り当てることができます。詳細については、「Amazon CloudWatch Evidently エクステンションの使用」を参照してください。
AWS AppConfig EventBridge へのイベントのデプロイ	設定がデプロイされると、この拡張機能は EventBridge のデフォルトのイベントバスにイベントを送信します。
AWS AppConfig Amazon Simple Notification Service (Amazon SNS) へのデプロイイベント	この拡張機能は、設定のデプロイ時に指定した Amazon SNS トピックにメッセージを送信します。

拡張機能	説明
AWS AppConfig Amazon Simple Queue Service (Amazon SQS) へのデプロイイベント	この拡張機能は、設定がデプロイされるとメッセージを Amazon SQS キューにエンキューします。
「インテグレーションエクステンション — アトラシアン Jira」	この拡張機能を使用すると AWS AppConfig 、機能フラグを変更するたびに問題を作成および更新できます。

ステップ 2: 拡張機能をいつ実行するかを決める

拡張機能は、 AWS AppConfig ワークフロー中に実行する 1 つ以上のアクションを定義します。たとえば、 AWS 作成されたAWS AppConfig deployment events to Amazon SNS拡張機能には、Amazon SNS トピックに通知を送信するアクションが含まれています。各アクションは、 を操作するとき、 AWS AppConfig または AWS AppConfig がユーザーに代わってプロセスを実行するときに呼び出されます。これらはアクションポイントと呼ばれます。 AWS AppConfig 拡張機能は次のアクションポイントをサポートします。

PRE_* アクションポイント: PRE_* アクションポイントに設定された拡張アクションは、リクエストの検証後に適用されますが、 AWS AppConfig はアクションポイント名に対応するアクティビティを実行します。これらのアクション呼び出しはリクエストと同時に処理されます。複数のリクエストが行われた場合、アクション呼び出しは順番に実行されます。また、PRE_* アクションポイントは設定を受け取り、設定の内容を変更できることにも注意してください。PRE_* アクションポイントはエラーに応答してアクションが起こらないようにすることもできます。

- PRE_CREATE_HOSTED_CONFIGURATION_VERSION
- PRE_START_DEPLOYMENT

ON_* アクションポイント: 拡張機能は、ON_*アクションポイントを使用して AWS AppConfig ワークフローと並行して実行することもできます。 ON_* アクションポイントは非同期的に呼び出されます。 ON_*アクションポイントは設定の内容を受信しません。ON_* アクションポイント中に拡張機能でエラーが発生した場合、サービスはそのエラーを無視してワークフローを続行します。

- ON_DEPLOYMENT_START
- ON_DEPLOYMENT_STEP

- `ON_DEPLOYMENT_BAKING`
- `ON_DEPLOYMENT_COMPLETE`
- `ON_DEPLOYMENT_ROLLED_BACK`

`AT_*` アクションポイント: `AT_*` アクションポイントに設定された拡張機能アクションは、ワークフローと同期的かつ並行して AWS AppConfig 呼び出されます。`AT_*` アクションポイント中に拡張機能にエラーが発生した場合、サービスはワークフローを停止し、デプロイをロールバックします。

- `AT_DEPLOYMENT_TICK`

ステップ 3: 拡張機能の作成

拡張機能を作成したり、 AWS 作成した拡張機能を設定したりするには、特定の AWS AppConfig リソースが使用されているときに拡張機能を呼び出すアクションポイントを定義します。たとえば、特定のアプリケーションの設定デプロイが開始されるたびに、`AWS AppConfig deployment events to Amazon SNS` 拡張機能を実行して Amazon SNS トピックに関する通知を受信するように選択できます。特定の AWS AppConfig リソースの拡張機能を呼び出すアクションポイントの定義は、拡張機能の関連付けと呼ばれます。拡張機能の関連付けは、アプリケーションや設定プロファイルなど、拡張機能と AWS AppConfig リソースの間の指定された関係です。

1 つの AWS AppConfig アプリケーションに複数の環境と設定プロファイルを含めることができます。拡張機能をアプリケーションまたは環境に関連付けると、 はアプリケーションまたは環境リソースに関連するワークフローの拡張機能を AWS AppConfig 呼び出します。

たとえば、AccessList という設定プロファイルを含む MobileApps という AWS AppConfig アプリケーションがあるとします。また、MobileApps アプリケーションにはベータ、インテグレーション、プロダクション環境が含まれているとします。 AWS 作成した Amazon SNS 通知拡張機能の拡張機能関連付けを作成し、その拡張機能を MobileApps アプリケーションに関連付けます。Amazon SNS 通知拡張は、3 つの環境のいずれかにアプリケーションの設定がデプロイされるたびに呼び出されます。

> **Note**
>
> AWS オーサリングされた拡張機能を使用するには拡張機能を作成する必要はありませんが、拡張機能の関連付けを作成する必要があります。

ステップ 4: 設定をデプロイし、拡張機能のアクションが実行されたことを確認する

関連付けを作成した後、ホスト設定が作成されるか、設定がデプロイされると、 は拡張機能を AWS AppConfig 呼び出し、指定されたアクションを実行します。拡張機能が呼び出されると、PRE-*アクションポイント中にシステムにエラーが発生した場合、 はそのエラーに関する情報 AWS AppConfig を返します。

AWS が作成した拡張機能での作業

AWS AppConfig には、次の AWS 作成された拡張機能が含まれています。これらの拡張機能は、AWS AppConfig ワークフローを他の サービスと統合するのに役立ちます。これらの拡張機能は、、AWS Management Console 、または SDK から直接拡張機能 API アクション AWS CLI AWS Tools for PowerShellを呼び出すことで使用できます。

拡張機能	説明
Amazon CloudWatch Evidently A/B テスト中	この拡張機能により、アプリケーションは、EvaluateFeature オペレーションを呼び出す代わりに、ローカルでユーザーセッションにバリエーションを割り当てることができます。詳細については、「Amazon CloudWatch Evidently エクステンションの使用」を参照してください。
AWS AppConfig EventBridge へのイベントのデプロイ	設定がデプロイされると、この拡張機能は EventBridge のデフォルトのイベントバスにイベントを送信します。
AWS AppConfig Amazon Simple Notification Service (Amazon SNS) への デプロイイベント	この拡張機能は、設定のデプロイ時に指定した Amazon SNS トピックにメッセージを送信します。
AWS AppConfig Amazon Simple Queue Service (Amazon SQS) へのデプロイイベント	この拡張機能は、設定がデプロイされるとメッセージを Amazon SQS キューにエンキューします。

拡張機能	説明
「インテグレーションエクステンション — アトラシアン Jira」	この拡張機能を使用すると AWS AppConfig 、機能フラグを変更するたびに問題を作成および更新できます。

Amazon CloudWatch Evidently エクステンションの使用

Amazon CloudWatch Evidently を使用すると、機能のロールアウト中に、新しい機能を指定した割合のユーザーに提供することによって安全に検証できます。新機能のパフォーマンスをモニターリングすると、ユーザーへのトラフィックを増やしていくタイミングを決定するのに役立ちます。こうすれば、機能を完全に起動する前に、リスクを軽減し、意図しない結果を明確化できます。また、A/B 実験を実行して、証拠とデータに基づいて機能の設計を決定することもできます。

CloudWatch Evidently の AWS AppConfig 拡張機能を使用すると、アプリケーションは EvaluateFeature オペレーションを呼び出す代わりに、ローカルでユーザーセッションにバリエーションを割り当てることができます。ローカルセッションにより、API コールに伴うレイテンシーと可用性のリスクが軽減されます。エクステンションの設定方法および使用方法については、Amazon CloudWatch CloudWatch ユーザーガイドの「CloudWatch による起動と A/B 実験を明確に行う」を参照してください。

Amazon EventBridge 拡張機能への AWS AppConfig デプロイイベントの使用

`AWS AppConfig deployment events to Amazon EventBridge` 拡張機能は、 AWS AppConfig 設定デプロイワークフローのモニタリングと対応に役立つ AWS 作成済みの拡張機能です。設定がデプロイされるたびに、この拡張機能は EventBridge のデフォルトのイベントバスにイベント通知を送信します。拡張機能をアプリケーション、環境、または設定プロファイルのいずれかに関連付けると、 は設定デプロイの開始、終了、ロールバックのたびにイベント通知をイベントバス AWS AppConfig に送信します AWS AppConfig 。

どのアクションポイントが EventBridge 通知を送信するかをより細かく制御したい場合は、カスタムエクステンションを作成し、URI フィールドに EventBridge のデフォルトイベントバス Amazon リソースネーム (ARN) を入力できます。拡張機能の作成の詳細については、「チュートリアル: カスタム AWS AppConfig 拡張機能の作成」を参照してください。

> **Important**
>
> この拡張モジュールは EventBridge のデフォルトイベントバスのみをサポートします。

拡張機能の使用

`AWS AppConfig deployment events to Amazon EventBridge` 拡張機能を使用するには、まず拡張機能の関連付けを作成して、拡張機能を AWS AppConfig リソースの 1 つにアタッチします。コンソール AWS AppConfig または CreateExtensionAssociation API アクションを使用して関連付けを作成します。関連付けを作成するときは、 AWS AppConfig アプリケーション、環境、または設定プロファイルの ARN を指定します。エクステンションをアプリケーションまたは環境に関連付けると、指定したアプリケーションまたは環境に含まれるすべての設定プロファイルに対してイベント通知が送信されます。

関連付けを作成した後、指定された AWS AppConfig リソースの設定がデプロイされると、 は拡張機能を AWS AppConfig 呼び出し、拡張機能で指定されたアクションポイントに従って通知を送信します。

> **Note**
>
> このエクステンションは、以下のアクションポイントによって呼び出されます。
>
> - `ON_DEPLOYMENT_START`
> - `ON_DEPLOYMENT_COMPLETE`
> - `ON_DEPLOYMENT_ROLLED_BACK`
>
> この拡張機能の使用、カスタマイズはできません。さまざまなアクションポイントを呼び出すための、独自のエクステンションを作成できます。詳細については、「チュートリアル: カスタム AWS AppConfig 拡張機能の作成」を参照してください。

AWS Systems Manager コンソールまたは を使用して AWS AppConfig 拡張機能の関連付けを作成するには、次の手順に従います AWS CLI。

拡張機能の関連付けを作成します (コンソール)

1. https://console.aws.amazon.com/systems-manager/appconfig/ で AWS Systems Manager コンソールを開きます。
2. ナビゲーションペインで、AWS AppConfig を選択します。
3. エクステンションタブでリソースに追加を選択します。
4. 「拡張リソースの詳細」セクションの「リソースタイプ」で、 AWS AppConfig 「リソースタイプ」を選択します。選択したリソースに応じて、 は他のリソースを選択するように AWS AppConfig 促します。
5. リソースとの関連付けを作成 を選択します。

拡張機能が呼び出されたときに EventBridge に送信されるサンプルイベントを次に示します。

```
{
   "version":"0",
   "id":"c53dbd72-c1a0-2302-9ed6-c076e9128277",
   "detail-type":"On Deployment Complete",
   "source":"aws.appconfig",
   "account":"111122223333",
   "time":"2022-07-09T01:44:15Z",
   "region":"us-east-1",
   "resources":[
      "arn:aws:appconfig:us-east-1:111122223333:extensionassociation/z763ff5"
   ],
   "detail":{
      "InvocationId":"5tfjcig",
       "Parameters":{

      },
      "Type":"OnDeploymentComplete",
      "Application":{
         "Id":"ba8toh7",
         "Name":"MyApp"
      },
      "Environment":{
         "Id":"pgil2o7",
         "Name":"MyEnv"
      },
      "ConfigurationProfile":{
         "Id":"ga3tqep",
```

```
            "Name":"MyConfigProfile"
        },
        "DeploymentNumber":1,
        "ConfigurationVersion":"1"
    }
}
```

Amazon SNS 拡張機能への AWS AppConfig デプロイイベントの使用

`AWS AppConfig deployment events to Amazon SNS` 拡張機能は、 AWS AppConfig 設定デプロイワークフローのモニタリングと対応に役立つ AWS 作成済みの拡張機能です。拡張機能は、設定がデプロイされるたびに Amazon SNS トピックにメッセージを発行します。拡張機能を AWS AppConfig アプリケーション、環境、または設定プロファイルのいずれかに関連付けると、 は設定デプロイの開始、終了、ロールバックのたびに、 トピックにメッセージ AWS AppConfig を発行します。

どのアクションポイントがAmazon SNS 通知を送信するかをより細かく制御したい場合は、カスタムエクステンションを作成し、URI フィールドに Amazon SNS トピック のAmazon リソースネーム (ARN) を入力できます。拡張機能の作成の詳細については、「チュートリアル: カスタム AWS AppConfig 拡張機能の作成」を参照してください。

拡張機能の使用

このセクションでは、 `AWS AppConfig deployment events to Amazon SNS` 拡張機能を使用する方法について説明します。

ステップ 1: トピック AWS AppConfig にメッセージを発行するように を設定する

Amazon SNS トピックにアクセスコントロールポリシーを追加して AWS AppConfig (`appconfig.amazonaws.com`) パブリッシュアクセス権限 (`sns:Publish`) を付与します。詳細については、「Amazon SNS アクセスコントロールのケース例」を参照してください。

ステップ 2: 拡張機能の作成

拡張機能の関連付けを作成して、いずれかの AWS AppConfig リソースに拡張機能をアタッチします。コンソール AWS AppConfig または CreateExtensionAssociation API アクションを使用して関連付けを作成します。関連付けを作成するときは、 AWS AppConfig アプリケーション、環境、または設定プロファイルの ARN を指定します。エクステンションをアプリケーションまたは環境に関連付けると、指定したアプリケーションまたは環境に含まれるすべての設定プロファイルに通知が送信されます。関連付けを作成するときは、使用する Amazon SNS トピックの ARN を含む `topicArn` パラメータの値を入力する必要があります。

関連付けを作成した後、指定された AWS AppConfig リソースの設定がデプロイされると、 は拡張機能を AWS AppConfig 呼び出し、拡張機能で指定されたアクションポイントに従って通知を送信します。

> **Note**
>
> このエクステンションは、以下のアクションポイントによって呼び出されます。
>
> - `ON_DEPLOYMENT_START`
> - `ON_DEPLOYMENT_COMPLETE`
> - `ON_DEPLOYMENT_ROLLED_BACK`
>
> この拡張機能の使用、カスタマイズはできません。さまざまなアクションポイントを呼び出すための、独自のエクステンションを作成できます。詳細については、「チュートリアル: カスタム AWS AppConfig 拡張機能の作成」を参照してください。

AWS Systems Manager コンソールまたは を使用して AWS AppConfig 拡張機能の関連付けを作成するには、次の手順に従います AWS CLI。

拡張機能の関連付けを作成します (コンソール)

1. https://console.aws.amazon.com/systems-manager/appconfig/ で AWS Systems Manager コンソールを開きます。
2. ナビゲーションペインで、AWS AppConfig を選択します。
3. エクステンションタブでリソースに追加を選択します。
4. 「拡張リソースの詳細」セクションの「リソースタイプ」で、 AWS AppConfig 「リソースタイプ」を選択します。選択したリソースに応じて、 は他のリソースを選択するように AWS AppConfig 促します。
5. リソースとの関連付けを作成 を選択します。

拡張機能が呼び出されたときに Amazon SNS トピックに送信されるメッセージのサンプルを次に示します。

```
{
    "Type": "Notification",
```

```
    "MessageId": "ae9d702f-9a66-51b3-8586-2b17932a9f28",
    "TopicArn": "arn:aws:sns:us-east-1:111122223333:MySNSTopic",
    "Message": {
        "InvocationId": "7itcaxp",
        "Parameters": {
            "topicArn": "arn:aws:sns:us-east-1:111122223333:MySNSTopic"
        },
        "Application": {
            "Id": "1a2b3c4d",
            "Name": MyApp
        },
        "Environment": {
            "Id": "1a2b3c4d",
            "Name": MyEnv
        },
        "ConfigurationProfile": {
            "Id": "1a2b3c4d",
            "Name": "MyConfigProfile"
        },
        "Description": null,
        "DeploymentNumber": "3",
        "ConfigurationVersion": "1",
        "Type": "OnDeploymentComplete"
    },
    "Timestamp": "2022-06-30T20:26:52.067Z",
    "SignatureVersion": "1",
    "Signature": "<...>",
    "SigningCertURL": "<...>",
    "UnsubscribeURL": "<...>",
    "MessageAttributes": {
        "MessageType": {
            "Type": "String",
            "Value": "OnDeploymentStart"
        }
    }
}
```

Amazon SQS 拡張機能への AWS AppConfig デプロイイベントの使用

`AWS AppConfig deployment events to Amazon SQS` 拡張機能は、 AWS AppConfig 設定デプロイワークフローのモニタリングと対応に役立つ AWS 作成済みの拡張機能です。設定がデプロイされるたびに、拡張機能は Amazon Simple Queue Service (Amazon SQS) キューにメッセージをエンキューします。拡張機能を AWS AppConfig アプリケーション、環境、または設定プロファイ

ルのいずれかに関連付けると、 は設定デプロイの開始、終了、ロールバックのたびにメッセージをキューに AWS AppConfig キューに入れます。

Amazon SQS 通知を送信するアクションポイントをより細かく制御したい場合は、カスタムエクステンションを作成し、URI フィールドに Amazon SQS キューの Amazon リソースネーム (ARN) を入力できます。拡張機能の作成の詳細については、「チュートリアル: カスタム AWS AppConfig 拡張機能の作成」を参照してください。

拡張機能の使用

このセクションでは、 `AWS AppConfig deployment events to Amazon SQS` 拡張機能を使用する方法について説明します。

ステップ 1: メッセージをキューに入れる AWS AppConfig ように を設定する

Amazon SQS キューに Amazon SQS ポリシーを追加して、 AWS AppConfig (`appconfig.amazonaws.com`) メッセージの送信権限 (`sqs:SendMessage`) を付与します。詳細については、「Basic examples of Amazon SQS policies」を参照してください。

ステップ 2: 拡張機能の作成

拡張機能の関連付けを作成して、いずれかの AWS AppConfig リソースに拡張機能をアタッチします。コンソール AWS AppConfig または CreateExtensionAssociation API アクションを使用して関連付けを作成します。関連付けを作成するときは、 AWS AppConfig アプリケーション、環境、または設定プロファイルの ARN を指定します。エクステンションをアプリケーションまたは環境に関連付けると、指定したアプリケーションまたは環境に含まれるすべての設定プロファイルに通知が送信されます。関連付けを作成するときは、使用する Amazon SQS キューの ARN を含む `Here` パラメータを入力する必要があります。

関連付けを作成した後、指定された AWS AppConfig リソースの設定が作成またはデプロイされると、 は拡張機能を AWS AppConfig 呼び出し、拡張機能で指定されたアクションポイントに従って通知を送信します。

> **Note**
>
> このエクステンションは、以下のアクションポイントによって呼び出されます。
>
> - `ON_DEPLOYMENT_START`
> - `ON_DEPLOYMENT_COMPLETE`
> - `ON_DEPLOYMENT_ROLLED_BACK`

この拡張機能の使用、カスタマイズはできません。さまざまなアクションポイントを呼び出すための、独自のエクステンションを作成できます。詳細については、「チュートリアル: カスタム AWS AppConfig 拡張機能の作成」を参照してください。

AWS Systems Manager コンソールまたは を使用して AWS AppConfig 拡張機能の関連付けを作成するには、次の手順に従います AWS CLI。

拡張機能の関連付けを作成します (コンソール)

1. https://console.aws.amazon.com/systems-manager/appconfig/ で AWS Systems Manager コンソールを開きます。
2. ナビゲーションペインで、AWS AppConfig を選択します。
3. エクステンションタブでリソースに追加を選択します。
4. 「拡張リソースの詳細」セクションの「リソースタイプ」で、 AWS AppConfig リソースタイプを選択します。選択したリソースに応じて、 は他のリソースを選択するように AWS AppConfig 促します。
5. リソースとの関連付けを作成 を選択します。

エクステンションが呼び出されたときに Amazon SQS キューに送信されるメッセージの例を次に示します。

```
{
   "InvocationId":"7itcaxp",
   "Parameters":{
      "queueArn":"arn:aws:sqs:us-east-1:111122223333:MySQSQueue"
   },
   "Application":{
      "Id":"1a2b3c4d",
      "Name":MyApp
   },
   "Environment":{
      "Id":"1a2b3c4d",
      "Name":MyEnv
   },
   "ConfigurationProfile":{
      "Id":"1a2b3c4d",
```

```
        "Name":"MyConfigProfile"
    },
    "Description":null,
    "DeploymentNumber":"3",
    "ConfigurationVersion":"1",
    "Type":"OnDeploymentComplete"
}
```

の Atlassian Jira 拡張機能の使用 AWS AppConfig

Atlassian Jira と統合することで、指定した の の機能フラグを変更するたびに、 AWS アカウント は Atlassian コンソールで問題を作成および更新 AWS AppConfig できます AWS リージョン。Jira の各課題には、フラグ名、アプリケーション ID、設定プロファイル ID、フラグ値が含まれます。フラグの変更を更新、保存、 およびデプロイすると、Jira は変更の詳細をもとに既存の問題を更新します。

> **Note**
>
> Jira は機能フラグを作成または更新するたびに課題を更新します。Jira は、親レベルのフラグから子レベルのフラグ属性を削除したときにも課題を更新します。Jira は親レベルのフラグを削除しても情報を記録しません。

統合を設定するには、以下を実行する必要があります。

- AWS AppConfig Jira 統合のアクセス許可の設定
- AWS AppConfig Jira 統合アプリケーションの設定

AWS AppConfig Jira 統合のアクセス許可の設定

Jira と AWS AppConfig の統合を設定するときは、ユーザーの認証情報を指定します。具体的には、AWS AppConfig for Jira アプリケーションにユーザーのアクセスキー ID とシークレットキーを入力します。このユーザーは、 と通信するアクセス許可を Jira に付与します AWS AppConfig。は、これらの認証情報を 1 回 AWS AppConfig 使用して、 と Jira の AWS AppConfig 関連付けを確立します。認証情報は保存されません。 AWS AppConfig for Jira アプリケーションをアンインストールすることで、関連付けを削除できます。

ユーザーアカウントには、以下のアクションを含む権限ポリシーが必要です。

- `appconfig:CreateExtensionAssociation`
- `appconfig:GetConfigurationProfile`
- `appconfig:ListApplications`
- `appconfig:ListConfigurationProfiles`
- `appconfig:ListExtensionAssociations`
- `sts:GetCallerIdentity`

IAM 権限ポリシーと Jira インテグレーション用の AWS AppConfig ユーザーを作成するには、以下のタスクを実行します。

タスク

- タスク 1: AWS AppConfig と Jira 統合の IAM アクセス許可ポリシーを作成する
- タスク 2: AWS AppConfig と Jira 統合のユーザーを作成する

タスク 1: AWS AppConfig と Jira 統合の IAM アクセス許可ポリシーを作成する

Atlassian Jira との通信を許可する IAM アクセス許可ポリシーを作成するには、次の手順に従います AWS AppConfig。新しいポリシーを作成し、このポリシーを新しい IAM ロールにアタッチすることをお勧めします。必要なアクセス権限を既存の IAM ポリシーとロールに追加することは、最小特権の原則に反するため、お勧めしません。

AWS AppConfig と Jira 統合の IAM ポリシーを作成するには

1. https://console.aws.amazon.com/iam/ で IAM コンソール を開きます。
2. ナビゲーションペインで ポリシーを選択してから ポリシーの作成 を選択します。
3. ポリシーの作成 ページの JSON タブを選択し、デフォルトの内容を次の JSON ポリシーに置き換えます。次のポリシーでは、*#####*、*##### ID*、*######## ID*、*##_profile_ID* を、ご使用の AWS AppConfig 機能フラグ環境からの情報に置き換えます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                        "appconfig:CreateExtensionAssociation",
```

```
                        "appconfig:ListExtensionAssociations",
                        "appconfig:GetConfigurationProfile"
             ],
            "Resource": [

 "arn:aws:appconfig:Region:account_ID:application/application_ID",

 "arn:aws:appconfig:Region:account_ID:application/application_ID/
configurationprofile/configuration_profile_ID"
             ]
        },
       {
            "Effect": "Allow",
            "Action": [
                        "appconfig:ListApplications"

             ],
            "Resource": [
                        "arn:aws:appconfig:Region:account_ID:*"
             ]
        },
         {
            "Effect": "Allow",
            "Action": [
                        "appconfig:ListConfigurationProfiles"
             ],
            "Resource": [

 "arn:aws:appconfig:Region:account_ID:application/application_ID"
             ]
        },
        {
            "Effect": "Allow",
            "Action": "sts:GetCallerIdentity",
            "Resource": "*"
        }
    ]
}
```

4. 次へ: タグ を選択します。
5. (オプショナル) 1 つ以上のタグ/値ペアを追加して、このポリシーのアクセスを整理、追跡、または制御し、次へ: 確認 を選択します。

6. ポリシーの確認 ページの 名前ボックスに **AppConfigJiraPolicy** などの名前を入力し、説明を入力します。
7. [Create policy] (ポリシーの作成) を選択します。

タスク 2: AWS AppConfig と Jira 統合のユーザーを作成する

AWS AppConfig および Atlassian Jira 統合のユーザーを作成するには、次の手順に従います。ユーザーを作成したら、統合を完了するときに指定するアクセスキー ID とシークレットキーをコピーできます。

AWS AppConfig と Jira 統合のユーザーを作成するには

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションペインで ユーザー 、ユーザーを追加 の順に選択します。
3. ユーザー名 フィールドに、**AppConfigJiraUser** などの名前を入力します。
4. AWS 認証情報タイプの選択 で、アクセスキー - プログラムによるアクセスを選択します。
5. [Next: Permissions] (次のステップ: 許可) を選択します。
6. アクセス許可の設定で、既存のポリシーを直接アタッチを選択します。作成したポリシー を検索し、チェックボックスをオンにして、タスク 1: AWS AppConfig と Jira 統合の IAM アクセス許可ポリシーを作成する次へ: タグを選択します。
7. タグの追加 (オプショナル) ページで、このユーザーのアクセスを整理、追跡、または制御するために、1つまたは複数のタグとキーの値のペアを追加します。次へ: レビュー を選択します。
8. 確認 ページで、スタックの詳細を確認します。
9. ユーザーの作成 を選択します。システムは、ユーザーのアクセスキー ID とシークレットキーを表示します。.csv ファイルをダウンロードするか、これらの認証情報を別の場所にコピーします。これらの認証情報は、統合を設定するときに指定します。

AWS AppConfig Jira 統合アプリケーションの設定

AWS AppConfig for Jira アプリケーションで必要なオプションを設定するには、次の手順に従います。この手順を完了すると、Jira は指定された の各機能フラグ AWS アカウント に新しい問題を作成します AWS リージョン。で機能フラグを変更すると AWS AppConfig、Jira は既存の問題に詳細を記録します。

> **Note**
>
> AWS AppConfig 機能フラグには、複数の子レベルのフラグ属性を含めることができます。Jira は親レベルの機能フラグごとに 1 つの問題を作成します。子レベルの属性を変更すると、親レベルのフラグのJira 問題でその変更の詳細を確認できます。

統合を設定するには

1. 「アトラシアン Marketplace」 にログインします。
2. 検索フィールドに **`AWS AppConfig`** と入力して入力キーを押します。
3. アプリケーションを Jira インスタンスにインストールします。
4. Atlassian コンソールで アプリの管理 を選択し、AWS AppConfig for Jira を選択します。
5. Configure (設定) を選択します。
6. 設定の詳細 で Jira プロジェクト を選択し、機能フラグに関連付けるプロジェクトを選択します。 AWS AppConfig
7. AWS リージョン を選択し、 AWS AppConfig 機能フラグが置かれている地域を選択します。
8. アプリケーション ID フィールドに、 AWS AppConfig 機能フラグを含むアプリケーションの名前を入力します。
9. 設定プロファイル ID フィールドに、 AWS AppConfig 機能フラグの設定プロファイルの名前を入力します。
10. アクセスキー ID と シークレットキー フィールドに、コピーした認証情報を入力しますタスク 2: AWS AppConfig と Jira 統合のユーザーを作成する。オプションで、セッショントークンを指定することもできます。
11. [Submit] を選択してください。
12. Atlassian コンソールで、プロジェクトを選択し、 AWS AppConfig 統合用に選択したプロジェクトを選択します。問題ページには、指定された AWS アカウント と の各機能フラグの問題が表示されます AWS リージョン。

AWS AppConfig for Jira アプリケーションとデータを削除する

AWS AppConfig 機能フラグで Jira 統合を使用しない場合は、Atlassian コンソールで AWS AppConfig for Jira アプリケーションを削除できます。アプリケーション統合 SDKページで、次を実行します。

- Jira インスタンスと の関連付けを削除します。 AWS AppConfig
- から Jira インスタンスの詳細を削除します。 AWS AppConfig

AWS AppConfig for Jira アプリケーションを削除するには

1. Atlassian コンソールで アプリの管理 を選択します。
2. AWS AppConfig for Jira を選択します。
3. アンインストール を選択します。

チュートリアル: カスタム AWS AppConfig 拡張機能の作成

カスタム AWS AppConfig 拡張機能を作成するには、次のタスクを実行します。各タスクについては、後のトピックで詳しく説明します。

> **Note**
>
> GitHub でカスタム AWS AppConfig 拡張機能のサンプルを表示できます。
>
> - Systems Manager Change Calendar を使用して `blocked day` モラトリアムカレンダーによるデプロイを防止するサンプル拡張機能
> - git-secrets を使用してシークレットが設定データに漏洩するのを防ぐサンプル拡張機能
> - Amazon Comprehend を使用して、個人を特定できる情報 (PII) が設定データに漏洩するのを防ぐサンプル拡張機能

1. AWS Lambda 関数を作成する

   ほとんどのユースケースでは、カスタム拡張機能を作成するには、拡張機能で定義された計算と処理を実行する AWS Lambda 関数を作成する必要があります。この規則の例外は、この規則の例外は、アクションポイントを追加または削除するために、AWS 「オーサリング通知拡張機能」のカスタムバージョンを作成する場合です。この例外の詳細については、「ステップ 3: カスタム AWS AppConfig 拡張機能を作成する」を参照してください。

2. カスタム拡張機能のアクセス許可を設定する

   カスタム拡張機能の使用:次のいずれかを実行します。

   - アクセス`InvokeFunction`許可を含む AWS Identity and Access Management (IAM) サービスロールを作成します。

- Lambda AddPermission API アクションを使用してリソースポリシーを作成します。

このウォークスルーでは、IAM サービスロールを作成する方法について説明します。

3. 拡張機能を作成する

   拡張機能を作成するには、 AWS AppConfig コンソールを使用するか AWS CLI AWS Tools for PowerShell、 または SDK から CreateExtension API アクションを呼び出します。このチュートリアルではコンソールを使用します。

4. 拡張機能の関連付けを作成する

   拡張機能の関連付けを作成するには、 AWS AppConfig コンソールを使用するか AWS CLI AWS Tools for PowerShell、 または SDK から CreateExtensionAssociation API アクションを呼び出します。このチュートリアルではコンソールを使用します。

5. 拡張機能を呼び出すアクションを実行します。

   関連付けを作成すると、そのリソースに対して拡張機能で定義されたアクションポイントが発生すると、 は拡張機能を AWS AppConfig 呼び出します。たとえば、`PRE_CREATE_HOSTED_CONFIGURATION_VERSION` アクションを含むエクステンションを関連付けると、新しいホスト設定バージョンを作成するたびにエクステンションが呼び出されます。

このセクションのトピックでは、 AWS AppConfig カスタム拡張機能の作成に関連する各タスクについて説明します。各タスクは、お客様が Amazon Simple Storage Service (Amazon S3) バケットに設定を自動的にバックアップする拡張機能を作成したいというユースケースのコンテキストで説明されています。この拡張機能は、ホスト設定を作成 `PRE_CREATE_HOSTED_CONFIGURATION_VERSION` またはデプロイ `PRE_START_DEPLOYMENT` するたびに実行されます。

トピック

- ステップ 1: カスタム AWS AppConfig 拡張機能の Lambda 関数を作成する
- ステップ 2: カスタム AWS AppConfig 拡張機能のアクセス許可を設定する
- ステップ 3: カスタム AWS AppConfig 拡張機能を作成する
- ステップ 4: カスタム拡張機能の AWS AppConfig 拡張機能関連付けを作成する

ステップ 1: カスタム AWS AppConfig 拡張機能の Lambda 関数を作成する

ほとんどのユースケースでは、カスタム拡張機能を作成するには、拡張機能で定義された計算と処理を実行する AWS Lambda 関数を作成する必要があります。このセクションでは、カスタム AWS AppConfig 拡張機能の Lambda 関数サンプルコードについて説明します。このセクションには、ペイロードリクエストとレスポンスリファレンスの詳細も含まれています。Lambda 関数の作成の詳細については、AWS Lambda デベロッパーガイドの「Lambda の開始方法」を参照してください。

「サンプルコード」

次の Lambda 関数のサンプルコードは、呼び出されると、 AWS AppConfig 設定を Amazon S3 バケットに自動的にバックアップします。新しい設定が作成またはデプロイされるたびに、設定はバックアップされます。このサンプルでは拡張パラメータを使用しているため、バケット名を Lambda 関数にハードコーディングする必要はありません。拡張パラメータを使用することで、ユーザーは拡張を複数のアプリケーションにアタッチし、設定を異なるバケットにバックアップできます。コードサンプルには、この機能をさらに説明するコメントが含まれています。

AWS AppConfig 拡張機能のサンプル Lambda 関数

```
from datetime import datetime
import base64
import json

import boto3


def lambda_handler(event, context):
    print(event)

    # Extensions that use the PRE_CREATE_HOSTED_CONFIGURATION_VERSION and
 PRE_START_DEPLOYMENT
    # action points receive the contents of AWS AppConfig configurations in Lambda
 event parameters.
    # Configuration contents are received as a base64-encoded string, which the lambda
 needs to decode
    # in order to get the configuration data as bytes. For other action points, the
 content
    # of the configuration isn't present, so the code below will fail.
    config_data_bytes = base64.b64decode(event["Content"])

    # You can specify parameters for extensions. The CreateExtension API action lets
 you define
```

```
    # which parameters an extension supports. You supply the values for those
 parameters when you
    # create an extension association by calling the CreateExtensionAssociation API
 action.
    # The following code uses a parameter called S3_BUCKET to obtain the value
 specified in the
    # extension association. You can specify this parameter when you create the
 extension
    # later in this walkthrough.
    extension_association_params = event.get('Parameters', {})
    bucket_name = extension_association_params['S3_BUCKET']
    write_backup_to_s3(bucket_name, config_data_bytes)

    # The PRE_CREATE_HOSTED_CONFIGURATION_VERSION and PRE_START_DEPLOYMENT action
 points can
    # modify the contents of a configuration. The following code makes a minor change
    # for the purposes of a demonstration.
    old_config_data_string = config_data_bytes.decode('utf-8')
    new_config_data_string = old_config_data_string.replace('hello', 'hello!')
    new_config_data_bytes = new_config_data_string.encode('utf-8')

    # The lambda initially received the configuration data as a base64-encoded string
    # and must return it in the same format.
    new_config_data_base64string =
 base64.b64encode(new_config_data_bytes).decode('ascii')

    return {
        'statusCode': 200,
        # If you want to modify the contents of the configuration, you must include the
 new contents in the
        # Lambda response. If you don't want to modify the contents, you can omit the
 'Content' field shown here.
        'Content': new_config_data_base64string
    }


def write_backup_to_s3(bucket_name, config_data_bytes):
    s3 = boto3.resource('s3')
    new_object = s3.Object(bucket_name,
 f"config_backup_{datetime.now().isoformat()}.txt")
    new_object.put(Body=config_data_bytes)
```

このウォークスルーでこのサンプルを使用する場合は、**MyS3ConfigurationBackUpExtension** 名前を付けて保存し、関数の Amazon リソースネーム (ARN) をコピーします。次のセクションで AWS Identity and Access Management (IAM) 継承ロールを作成するときに ARN を指定します。拡張機能の作成時にARN と名前を指定します。

ペイロードリファレンス

このセクションには、カスタム AWS AppConfig 拡張機能を操作するためのペイロードリクエストとレスポンスリファレンスの詳細が含まれています。

リクエスト構造

AtDeploymentTick

```
{
    'InvocationId': 'o2xbtm7',
    'Parameters': {
        'ParameterOne': 'ValueOne',
        'ParameterTwo': 'ValueTwo'
    },
    'Type': 'OnDeploymentStart',
    'Application': {
        'Id': 'abcd123'
    },
    'Environment': {
        'Id': 'efgh456'
    },
    'ConfigurationProfile': {
        'Id': 'ijkl789',
        'Name': 'ConfigurationName'
    },
    'DeploymentNumber': 2,
    'Description': 'Deployment description',
    'ConfigurationVersion': '2',
    'DeploymentState': 'DEPLOYING',
    'PercentageComplete': '0.0'
}
```

リクエスト構造

事前作成済みのホスト設定バージョン

```
{
```

```
    'InvocationId': 'vlns753', // id for specific invocation
    'Parameters': {
        'ParameterOne': 'ValueOne',
        'ParameterTwo': 'ValueTwo'
    },
    'ContentType': 'text/plain',
    'ContentVersion': '2',
    'Content': 'SGVsbG8gZWFydGgh', // Base64 encoded content
    'Application': {
        'Id': 'abcd123',
        'Name': 'ApplicationName'
    },
    'ConfigurationProfile': {
        'Id': 'ijkl789',
        'Name': 'ConfigurationName'
    },
    'Description': '',
    'Type': 'PreCreateHostedConfigurationVersion',
    'PreviousContent': {
        'ContentType': 'text/plain',
        'ContentVersion': '1',
        'Content': 'SGVsbG8gd29ybGQh'
    }
}
```

デプロイ開始前

```
{
    'InvocationId': '765ahdm',
    'Parameters': {
        'ParameterOne': 'ValueOne',
        'ParameterTwo': 'ValueTwo'
    },
    'ContentType': 'text/plain',
    'ContentVersion': '2',
    'Content': 'SGVsbG8gZWFydGgh',
    'Application': {
        'Id': 'abcd123',
        'Name': 'ApplicationName'
    },
    'Environment': {
        'Id': 'ibpnqlq',
        'Name': 'EnvironmentName'
```

```
    },
    'ConfigurationProfile': {
        'Id': 'ijkl789',
        'Name': 'ConfigurationName'
    },
    'DeploymentNumber': 2,
    'Description': 'Deployment description',
    'Type': 'PreStartDeployment'
}
```

非同期イベント

デプロイ開始時、デプロイステップ時、デプロイ中

```
{
    'InvocationId': 'o2xbtm7',
    'Parameters': {
        'ParameterOne': 'ValueOne',
        'ParameterTwo': 'ValueTwo'
    },
    'Type': 'OnDeploymentStart',
    'Application': {
        'Id': 'abcd123'
    },
    'Environment': {
        'Id': 'efgh456'
    },
    'ConfigurationProfile': {
        'Id': 'ijkl789',
        'Name': 'ConfigurationName'
    },
    'DeploymentNumber': 2,
    'Description': 'Deployment description',
    'ConfigurationVersion': '2'
}
```

応答の構造

次の例は、カスタム AWS AppConfig 拡張機能からのリクエストに応答して Lambda 関数が返す内容を示しています。

PRE_* 同期イベント - 正常なレスポンス

内容を変換したい場合は、次の手順を実行します。

```
"Content": "SomeBase64EncodedByteArray"
```

AT_* 同期イベント - 正常なレスポンス

デプロイの次のステップを制御する (デプロイを継続するかロールバックする) 場合、レスポンスの Directiveと Description 属性。

```
"Directive": "ROLL_BACK"
"Description" "Deployment event log description"
```

Directive は、 CONTINUEまたは の 2 つの値をサポートしますROLL_BACK。ペイロードレスポンスでこれらの列挙型を使用して、デプロイの次のステップを制御します。

同期イベント-応答成功

内容を変換したい場合は、次の手順を実行します。

```
"Content": "SomeBase64EncodedByteArray"
```

コンテンツを変換したくない場合は、何もリターンしないでください。

非同期イベント-応答成功

戻り値なし。

すべてのエラーイベント

```
{
        "Error": "BadRequestError",
        "Message": "There was malformed stuff in here",
        "Details": [{
            "Type": "Malformed",
            "Name": "S3 pointer",
            "Reason": "S3 bucket did not exist"
        }]
    }
```

ステップ 2: カスタム AWS AppConfig 拡張機能のアクセス許可を設定する

次の手順を使用して、 AWS Identity and Access Management (IAM) サービスロールを作成および設定します (またはロールを引き受けます) 。 はこのロール AWS AppConfig を使用して Lambda 関数を呼び出します。

IAM サービスロールを作成し、 AWS AppConfig に引き受けさせるには

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションペインで ロール を選択してから、ロールを作成する を選択します。
3. 信頼されたエンティティの選択 で、カスタム信頼ポリシー を選択します。
4. 以下のポリシーを ポリシー フィールドに貼り付けます。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "appconfig.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

   次へ をクリックします。
5. アクセス許可を追加ページで、ポリシーの作成を選択します。新しいタブで ポリシーの作成 ページが開きます。
6. JSON タブを選択して、次のカスタムポリシーを JSON エディタに貼り付けます。`lambda:InvokeFunction` アクションは `PRE_*` アクションポイントに使用されます。`lambda:InvokeAsync` アクションは `ON_*` アクションポイントに使用されます。*`Lambda ARN #`* Lambda Amazon リソースネーム (ARN) に置き換えます。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
```

```
      "Effect": "Allow",
      "Action": [
        "lambda:InvokeFunction",
        "lambda:InvokeAsync"
      ],
      "Resource": "Your Lambda ARN"
    }
  ]
}
```

7. 次へ: タグ を選択します。
8. タグの追加 (オプショナル) ページで 1 つ以上のキーと値のペアを追加し、次へ:レビュー を選択します。
9. レビューポリシーページでポリシー名と説明を入力し、ポリシーの作成を選択します。
10. カスタム信頼ポリシーのブラウザータブで 更新 アイコンを選択し、作成した権限ポリシーを検索します。
11. ポリシーのチェックボックスを選択にし、次のステップ を選択します。
12. 確認 ページの ロール名 ボックスに名前を入力し、続いて説明を入力します。
13. ロールの作成 を選択します。ロールページが再度表示されます。バナーの ロールを表示 を選択します。
14. ARN をコピーします。拡張機能の作成時にARN を指定します。

ステップ 3: カスタム AWS AppConfig 拡張機能を作成する

拡張機能は、 AWS AppConfig ワークフロー中に実行する 1 つ以上のアクションを定義します。たとえば、 AWS 作成されたAWS AppConfig deployment events to Amazon SNS拡張機能には、Amazon SNS トピックに通知を送信するアクションが含まれています。各アクションは、 を操作するとき、 AWS AppConfig または AWS AppConfig がユーザーに代わってプロセスを実行するときに呼び出されます。これらはアクションポイントと呼ばれます。 AWS AppConfig 拡張機能は次のアクションポイントをサポートします。

PRE_* アクションポイント: PRE_* アクションポイントに設定された拡張アクションは、リクエストの検証後、 がアクションポイント名に対応するアクティビティを実行する前に AWS AppConfig 適用されます。これらのアクション呼び出しはリクエストと同時に処理されます。複数のリクエストが行われた場合、アクション呼び出しは順番に実行されます。また、PRE_* アクションポイントは設定を受け取り、設定の内容を変更できることにも注意してください。PRE_* アクションポイントはエラーに応答してアクションが起こらないようにすることもできます。

- PRE_CREATE_HOSTED_CONFIGURATION_VERSION
- PRE_START_DEPLOYMENT

ON_* アクションポイント: 拡張機能は、ON_*アクションポイントを使用して AWS AppConfig ワークフローと並行して実行することもできます。 ON_* アクションポイントは非同期的に呼び出されます。 ON_*アクションポイントは設定の内容を受信しません。ON_* アクションポイント中に拡張機能でエラーが発生した場合、サービスはそのエラーを無視してワークフローを続行します。

- ON_DEPLOYMENT_START
- ON_DEPLOYMENT_STEP
- ON_DEPLOYMENT_BAKING
- ON_DEPLOYMENT_COMPLETE
- ON_DEPLOYMENT_ROLLED_BACK

AT_* アクションポイント: AT_* アクションポイントに設定された拡張機能アクションは、ワークフローと同期的かつ並行して AWS AppConfig 呼び出されます。AT_* アクションポイント中に拡張機能にエラーが発生した場合、サービスはワークフローを停止し、デプロイをロールバックします。

- AT_DEPLOYMENT_TICK

> **Note**
>
> AT_DEPLOYMENT_TICK アクションポイントは、サードパーティーのモニタリング統合をサポートします。 AT_DEPLOYMENT_TICKは、設定デプロイ処理オーケストレーション中に呼び出されます。サードパーティーのモニタリングソリューション (Datadog など) を使用する場合は、AT_DEPLOYMENT_TICKアクションポイントでアラームをチェックする AWS AppConfig 拡張機能を作成し、アラームをトリガーした場合、安全ガードレールとしてデプロイをロールバックできます。AT_DEPLOYMENT_TICK アクションポイントを使用して Datadog と統合する AWS AppConfig 拡張機能のコードサンプルを表示するには、GitHub の「aws-samples / aws-appconfig-tick-extn-for-datadog」を参照してください。

拡張機能の例

次の例では、PRE_CREATE_HOSTED_CONFIGURATION_VERSION アクションポイントを呼び出すアクションを定義します。Uri フィールドでは、このウォークスルーで先に作成した

`MyS3ConfigurationBackUpExtension` Lambda 関数の Amazon リソースネーム (ARN) を指定します。このアクションでは、このチュートリアルの前半で作成した AWS Identity and Access Management (IAM) 継承ロール ARN も指定します。

サンプル AWS AppConfig 拡張機能

```
{
    "Name": "MySampleExtension",
    "Description": "A sample extension that backs up configurations to an S3 bucket.",
    "Actions": {
        "PRE_CREATE_HOSTED_CONFIGURATION_VERSION": [
            {
                "Name": "PreCreateHostedConfigVersionActionForS3Backup",
                "Uri": "arn:aws:lambda:aws-
region:111122223333:function:MyS3ConfigurationBackUpExtension",
                "RoleArn": "arn:aws:iam::111122223333:role/ExtensionsTestRole"
            }
        ]
    },
    "Parameters" : {
        "S3_BUCKET": {
            "Required": false
        }
    }
}
```

> **Note**
>
> 拡張機能を作成する際のリクエスト構文とフィールドの説明を確認するには、AWS AppConfig API リファレンスの CreateExtension トピックを参照してください。

拡張機能 (コンソール)を作成するには

1. https://console.aws.amazon.com/systems-manager/appconfig/ で AWS Systems Manager コンソールを開きます。
2. ナビゲーションペインで、AWS AppConfig を選択します。
3. 拡張機能 タブで 拡張機能の作成 を選択します。

4. 名前 に一意の名前を入力します。このチュートリアルでは、**MyS3ConfigurationBackUpExtension** と入力します。必要に応じて説明に説明を入力します。
5. アクション セクションで、アクションの追加 を選択します。
6. バケット名に、一意の名前を入力します。このチュートリアルでは、**PreCreateHostedConfigVersionActionForS3Backup** と入力します。この名前は、アクションが使用するアクションポイントと拡張の目的を表しています。
7. アクションポイント リストで PRE_CREATE_HOSTED_CONFIGURATION_VERSION を選択します。
8. Uri の場合は、Lambda 関数 を選択し、Lambda 関数 リストで関数を選択します。関数が表示されない場合は、関数を作成した AWS リージョン のと同じ にあることを確認します。
9. IAM ロール に、このチュートリアルで先ほど作成したロールを選択します。
10. 拡張パラメータ (オプショナル) セクションで、新規パラメータを追加 を選択します。
11. パラメータ名に名前を入力します。このチュートリアルでは、**S3_BUCKET** と入力します。
12. ステップ 5 ~ 11 を繰り返して、 PRE_START_DEPLOYMENT アクションポイントに 2 つ目のアクションを作成します。
13. 拡張機能の作成 を選択します。

AWS オーサリングされた通知拡張機能のカスタマイズ

AWS オーサリングの通知拡張機能を使用するために、Lambdaまたは拡張機能を作成する必要はありません。エクステンションの関連付けを作成して、サポートされているアクションポイントのいずれかを呼び出す操作を実行するだけで済みます。デフォルトでは、 AWS 作成された通知拡張機能は次のアクションポイントをサポートします。

- ON_DEPLOYMENT_START
- ON_DEPLOYMENT_COMPLETE
- ON_DEPLOYMENT_ROLLED_BACK

AWS AppConfig deployment events to Amazon SNS 拡張機能や AWS AppConfig deployment events to Amazon SQS 拡張機能のカスタムバージョンを作成する場合は、通知を受け取るアクションポイントを指定できます。

> **Note**
>
> `AWS AppConfig deployment events to EventBridge` エクステンションは `PRE_*` アクションポイントをサポートしていません。 AWS 作成されたバージョンに割り当てられたデフォルトのアクションポイントの一部を削除する場合は、カスタムバージョンを作成できます。

AWS オーサリング通知拡張機能のカスタムバージョンを作成する場合は、Lambda関数を作成する必要はありません。新しい拡張機能のフィールドで、Amazon リソースネーム (ARN) を`Uri` で指定するだけで、新しい拡張機能の使用できます。

- カスタムEventBridge 通知拡張の場合は、イベントブリッジのデフォルトイベントの ARN を `Uri` フィールドに入力します。
- Amazon SNS の通知拡張機能の使用するには、`Uri` フィールドに Amazon SNS トピックの ARN を入力します。
- Amazon SQS の通知拡張機能の使用するには、`Uri` フィールドに Amazon SQS メッセージキューの ARN を入力します。

ステップ 4: カスタム拡張機能の AWS AppConfig 拡張機能関連付けを作成する

拡張機能を作成したり、 AWS 作成した拡張機能を設定したりするには、特定の AWS AppConfig リソースが使用されているときに拡張機能を呼び出すアクションポイントを定義します。たとえば、特定のアプリケーションの設定デプロイが開始されるたびに、`AWS AppConfig deployment events to Amazon SNS` 拡張機能を実行して Amazon SNS トピックに関する通知を受信するように選択できます。特定の AWS AppConfig リソースの拡張機能を呼び出すアクションポイントの定義は、拡張機能の関連付けと呼ばれます。拡張機能の関連付けは、アプリケーションや設定プロファイルなど、拡張機能と AWS AppConfig リソースの間の指定された関係です。

1 つの AWS AppConfig アプリケーションに複数の環境と設定プロファイルを含めることができます。拡張機能をアプリケーションまたは環境に関連付けると、 は、該当する場合、アプリケーションまたは環境リソースに関連するワークフローの拡張機能を AWS AppConfig 呼び出します。

たとえば、AccessList という設定プロファイルを含む MobileApps という AWS AppConfig アプリケーションがあるとします。また、MobileApps アプリケーションにはベータ、インテグレーション、プロダクション環境が含まれているとします。 AWS 作成した Amazon SNS 通知拡張機能の拡

張機能関連付けを作成し、その拡張機能を MobileApps アプリケーションに関連付けます。Amazon SNS 通知拡張は、3 つの環境のいずれかにアプリケーションの設定がデプロイされるたびに呼び出されます。

AWS AppConfig コンソールを使用して AWS AppConfig 拡張機能の関連付けを作成するには、次の手順に従います。

拡張機能の関連付けを作成します (コンソール)

1. https://console.aws.amazon.com/systems-manager/appconfig/ で AWS Systems Manager コンソールを開きます。
2. ナビゲーションペインで、AWS AppConfig を選択します。
3. 拡張機能 タブでExtensionsのオプションボタンを選択し、リソースに追加 を選択します。このチュートリアルでは、MyS3 設定バックアップ拡張機能を選択してください。
4. 「拡張リソースの詳細」セクションの「リソースタイプ」で、 AWS AppConfig 「リソースタイプ」を選択します。選択したリソースに応じて、 は他のリソースを選択するように AWS AppConfig 促します。このチュートリアルでは、アプリケーション を選択します。
5. リストからアプリケーションを選択します。
6. パラメータセクションで、キー フィールドに S3_BUCKET が表示されていることを確認します。値フィールドに、Lambda 拡張機能の ARN を貼り付けます。例: `arn:aws:lambda:`*`aws-region`*`:111122223333:function:MyS3ConfigurationBackUpExtension`。
7. リソースとの関連付けを作成 を選択します。

関連付けを作成したら、`MyS3ConfigurationBackUpExtension` の `hosted` を指定する新しい設定プロファイルを作成してを作成することで `SourceUri` 拡張機能を呼び出すことができます。新しい設定を作成するワークフローの一部として、 AWS AppConfig はPRE_CREATE_HOSTED_CONFIGURATION_VERSIONアクションポイントを検出します。このアクションポイントに遭遇するとMyS3ConfigurationBackUpExtension拡張機能が呼び出され、新しく作成された設定が拡張機能に関連付けられているParameterセクションで指定されているS3バケットに自動的にバックアップされます。

コードサンプルを使用して一般的な AWS AppConfig タスクを実行する

このセクションでは、一般的な AWS AppConfig アクションをプログラムで実行するためのコードサンプルについて説明します。これらのサンプルを、Java、Python、および JavaScript SDK で使用して、テスト環境でアクションを実行することをお勧めします。このセクションでは、完了後にテスト環境をクリーンアップするためのコードサンプルも含まれています。

トピック

- ホスト設定ストアに保存されているフリーフォーム設定の作成または更新
- Secrets Manager に保存されているシークレットの設定プロファイルの作成
- 設定プロファイルのデプロイ
- AWS AppConfig エージェントを使用してフリーフォーム設定プロファイルを読み取る
- AWS AppConfig エージェントを使用して特定の機能フラグを読み取る
- AWS AppConfig エージェントを使用してバリアントを含む機能フラグを取得する
- GetLatestConfiguration API アクションを使用してフリーフォーム設定プロファイルを読み取る
- 環境のクリーンアップ

ホスト設定ストアに保存されているフリーフォーム設定の作成または更新

次の各サンプルには、コードによって実行されるアクションに関するコメントが含まれています。このセクションのサンプルでは、次の API を呼び出します。

- CreateApplication
- CreateConfigurationProfile
- CreateHostedConfigurationVersion

Java

```
public CreateHostedConfigurationVersionResponse createHostedConfigVersion() {
        AppConfigClient appconfig = AppConfigClient.create();
```

```
        // Create an application
        CreateApplicationResponse app = appconfig.createApplication(req ->
 req.name("MyDemoApp"));

        // Create a hosted, freeform configuration profile
        CreateConfigurationProfileResponse configProfile =
 appconfig.createConfigurationProfile(req -> req
            .applicationId(app.id())
            .name("MyConfigProfile")
            .locationUri("hosted")
            .type("AWS.Freeform"));

        // Create a hosted configuration version
        CreateHostedConfigurationVersionResponse hcv =
 appconfig.createHostedConfigurationVersion(req -> req
            .applicationId(app.id())
            .configurationProfileId(configProfile.id())
            .contentType("text/plain; charset=utf-8")
            .content(SdkBytes.fromUtf8String("my config data")));

        return hcv;
    }
```

Python

```
import boto3

appconfig = boto3.client('appconfig')

# create an application
application = appconfig.create_application(Name='MyDemoApp')

# create a hosted, freeform configuration profile
config_profile = appconfig.create_configuration_profile(
    ApplicationId=application['Id'],
    Name='MyConfigProfile',
    LocationUri='hosted',
    Type='AWS.Freeform')

# create a hosted configuration version
hcv = appconfig.create_hosted_configuration_version(
    ApplicationId=application['Id'],
    ConfigurationProfileId=config_profile['Id'],
```

```
    Content=b'my config data',
    ContentType='text/plain')
```

JavaScript

```
import {
  AppConfigClient,
  CreateApplicationCommand,
  CreateConfigurationProfileCommand,
  CreateHostedConfigurationVersionCommand,
} from "@aws-sdk/client-appconfig";

const appconfig = new AppConfigClient();

// create an application
const application = await appconfig.send(
  new CreateApplicationCommand({ Name: "MyDemoApp" })
);

// create a hosted, freeform configuration profile
const profile = await appconfig.send(
  new CreateConfigurationProfileCommand({
    ApplicationId: application.Id,
    Name: "MyConfigProfile",
    LocationUri: "hosted",
    Type: "AWS.Freeform",
  })
);

// create a hosted configuration version
await appconfig.send(
  new CreateHostedConfigurationVersionCommand({
    ApplicationId: application.Id,
    ConfigurationProfileId: profile.Id,
    ContentType: "text/plain",
    Content: "my config data",
  })
);
```

Secrets Manager に保存されているシークレットの設定プロファイルの作成

次の各サンプルには、コードによって実行されるアクションに関するコメントが含まれています。このセクションのサンプルでは、次の API を呼び出します。

- CreateApplication
- CreateConfigurationProfile

Java

```
private void createSecretsManagerConfigProfile() {
        AppConfigClient appconfig = AppConfigClient.create();

        // Create an application
        CreateApplicationResponse app = appconfig.createApplication(req ->
 req.name("MyDemoApp"));

        // Create a configuration profile for Secrets Manager Secret
        CreateConfigurationProfileResponse configProfile =
 appconfig.createConfigurationProfile(req -> req
            .applicationId(app.id())
            .name("MyConfigProfile")
            .locationUri("secretsmanager://MySecret")
            .retrievalRoleArn("arn:aws:iam::000000000000:role/
RoleTrustedByAppConfigThatCanRetrieveSecret")
            .type("AWS.Freeform"));
    }
```

Python

```
import boto3

appconfig = boto3.client('appconfig')

# create an application
application = appconfig.create_application(Name='MyDemoApp')

# create a configuration profile for Secrets Manager Secret
config_profile = appconfig.create_configuration_profile(
```

```
    ApplicationId=application['Id'],
    Name='MyConfigProfile',
    LocationUri='secretsmanager://MySecret',
    RetrievalRoleArn='arn:aws:iam::000000000000:role/
RoleTrustedByAppConfigThatCanRetrieveSecret',
    Type='AWS.Freeform')
```

JavaScript

```
import {
  AppConfigClient,
  CreateConfigurationProfileCommand,
} from "@aws-sdk/client-appconfig";

const appconfig = new AppConfigClient();

// create an application
const application = await appconfig.send(
  new CreateApplicationCommand({ Name: "MyDemoApp" })
);

// create a configuration profile for Secrets Manager Secret
await appconfig.send(
  new CreateConfigurationProfileCommand({
    ApplicationId: application.Id,
    Name: "MyConfigProfile",
    LocationUri: "secretsmanager://MySecret",
    RetrievalRoleArn: "arn:aws:iam::000000000000:role/
RoleTrustedByAppConfigThatCanRetrieveSecret",
    Type: "AWS.Freeform",
  })
);
```

設定プロファイルのデプロイ

次の各サンプルには、コードによって実行されるアクションに関するコメントが含まれています。このセクションのサンプルでは、次の API を呼び出します。

- CreateApplication
- CreateConfigurationProfile
- CreateHostedConfigurationVersion

- CreateEnvironment
- StartDeployment
- GetDeployment

Java

```
private void createDeployment() throws InterruptedException {
        AppConfigClient appconfig = AppConfigClient.create();

        // Create an application
        CreateApplicationResponse app = appconfig.createApplication(req ->
 req.name("MyDemoApp"));

        // Create a hosted, freeform configuration profile
        CreateConfigurationProfileResponse configProfile =
 appconfig.createConfigurationProfile(req -> req
            .applicationId(app.id())
            .name("MyConfigProfile")
            .locationUri("hosted")
            .type("AWS.Freeform"));

        // Create a hosted configuration version
        CreateHostedConfigurationVersionResponse hcv =
 appconfig.createHostedConfigurationVersion(req -> req
            .applicationId(app.id())
            .configurationProfileId(configProfile.id())
            .contentType("text/plain; charset=utf-8")
            .content(SdkBytes.fromUtf8String("my config data")));


        // Create an environment
        CreateEnvironmentResponse env = appconfig.createEnvironment(req -> req
            .applicationId(app.id())
            .name("Beta")
            // If you have CloudWatch alarms that monitor the health of your
 service, you can add them here and they
            // will trigger a rollback if they fire during an appconfig deployment
            //.monitors(Monitor.builder().alarmArn("arn:aws:cloudwatch:us-
east-1:520900602629:alarm:MyAlarm")
            //
  .alarmRoleArn("arn:aws:iam::520900602629:role/MyAppConfigAlarmRole").build())
        );
```

```
        // Start a deployment
        StartDeploymentResponse deploymentResponse = appconfig.startDeployment(req -
> req
            .applicationId(app.id())
            .configurationProfileId(configProfile.id())
            .environmentId(env.id())
            .configurationVersion(hcv.versionNumber().toString())
            .deploymentStrategyId("AppConfig.Linear50PercentEvery30Seconds")
        );

        // Wait for deployment to complete
        List<DeploymentState> nonFinalDeploymentStates = Arrays.asList(
            DeploymentState.DEPLOYING,
            DeploymentState.BAKING,
            DeploymentState.ROLLING_BACK,
            DeploymentState.VALIDATING);
        GetDeploymentRequest getDeploymentRequest =
 GetDeploymentRequest.builder().applicationId(app.id())

 .environmentId(env.id())

 .deploymentNumber(deploymentResponse.deploymentNumber()).build();
        GetDeploymentResponse deployment =
 appconfig.getDeployment(getDeploymentRequest);
        while (nonFinalDeploymentStates.contains(deployment.state())) {
            System.out.println("Waiting for deployment to complete: " + deployment);
            Thread.sleep(1000L);
            deployment = appconfig.getDeployment(getDeploymentRequest);
        }

        System.out.println("Deployment complete: " + deployment);
    }
```

Python

```
import boto3

appconfig = boto3.client('appconfig')

# create an application
application = appconfig.create_application(Name='MyDemoApp')
```

```
# create an environment
environment = appconfig.create_environment(
    ApplicationId=application['Id'],
    Name='MyEnvironment')

# create a configuration profile
config_profile = appconfig.create_configuration_profile(
    ApplicationId=application['Id'],
    Name='MyConfigProfile',
    LocationUri='hosted',
    Type='AWS.Freeform')

# create a hosted configuration version
hcv = appconfig.create_hosted_configuration_version(
    ApplicationId=application['Id'],
    ConfigurationProfileId=config_profile['Id'],
    Content=b'my config data',
    ContentType='text/plain')

# start a deployment
deployment = appconfig.start_deployment(
    ApplicationId=application['Id'],
    EnvironmentId=environment['Id'],
    ConfigurationProfileId=config_profile['Id'],
    ConfigurationVersion=str(hcv['VersionNumber']),
    DeploymentStrategyId='AppConfig.Linear20PercentEvery6Minutes')
```

JavaScript

```
import {
  AppConfigClient,
  CreateApplicationCommand,
  CreateEnvironmentCommand,
  CreateConfigurationProfileCommand,
  CreateHostedConfigurationVersionCommand,
  StartDeploymentCommand,
} from "@aws-sdk/client-appconfig";

const appconfig = new AppConfigClient();

// create an application
const application = await appconfig.send(
  new CreateApplicationCommand({ Name: "MyDemoApp" })
```

```
);

// create an environment
const environment = await appconfig.send(
  new CreateEnvironmentCommand({
    ApplicationId: application.Id,
    Name: "MyEnvironment",
  })
);

// create a configuration profile
const config_profile = await appconfig.send(
  new CreateConfigurationProfileCommand({
    ApplicationId: application.Id,
    Name: "MyConfigProfile",
    LocationUri: "hosted",
    Type: "AWS.Freeform",
  })
);

// create a hosted configuration version
const hcv = await appconfig.send(
  new CreateHostedConfigurationVersionCommand({
    ApplicationId: application.Id,
    ConfigurationProfileId: config_profile.Id,
    Content: "my config data",
    ContentType: "text/plain",
  })
);

// start a deployment
await appconfig.send(
  new StartDeploymentCommand({
    ApplicationId: application.Id,
    EnvironmentId: environment.Id,
    ConfigurationProfileId: config_profile.Id,
    ConfigurationVersion: hcv.VersionNumber.toString(),
    DeploymentStrategyId: "AppConfig.Linear20PercentEvery6Minutes",
  })
);
```

AWS AppConfig エージェントを使用してフリーフォーム設定プロファイルを読み取る

次の各サンプルには、コードによって実行されるアクションに関するコメントが含まれています。

Java

```
public void retrieveConfigFromAgent() throws Exception {
        /*
        In this sample, we will retrieve configuration data from the AWS AppConfig
 Agent.
        The agent is a sidecar process that handles retrieving configuration data
 from AppConfig
        for you in a way that implements best practices like configuration caching.

        For more information about the agent, see How to use AWS AppConfig Agent
        */

        // The agent runs a local HTTP server that serves configuration data
        // Make a GET request to the agent's local server to retrieve the
 configuration data
        URL url = new URL("http://localhost:2772/applications/MyDemoApp/
environments/Beta/configurations/MyConfigProfile");
        HttpURLConnection con = (HttpURLConnection) url.openConnection();
        con.setRequestMethod("GET");
        StringBuilder content;
        try (BufferedReader in = new BufferedReader(new
 InputStreamReader(con.getInputStream()))) {
            content = new StringBuilder();
            int ch;
            while ((ch = in.read()) != -1) {
                content.append((char) ch);
            }
        }
        con.disconnect();
        System.out.println("Configuration from agent via HTTP: " + content);
    }
```

Python

```
# in this sample, we will retrieve configuration data from the AWS AppConfig Agent.
```

```
# the agent is a sidecar process that handles retrieving configuration data from AWS
 AppConfig
# for you in a way that implements best practices like configuration caching.
#
# for more information about the agent, see
# How to use AWS AppConfig Agent
#

import requests

application_name = 'MyDemoApp'
environment_name = 'MyEnvironment'
config_profile_name = 'MyConfigProfile'

# the agent runs a local HTTP server that serves configuration data
# make a GET request to the agent's local server to retrieve the configuration data
response = requests.get(f"http://localhost:2772/applications/{application_name}/
environments/{environment_name}/configurations/{config_profile_name}")
config = response.content
```

JavaScript

```
// in this sample, we will retrieve configuration data from the AWS AppConfig Agent.
// the agent is a sidecar process that handles retrieving configuration data from
 AppConfig
// for you in a way that implements best practices like configuration caching.

// for more information about the agent, see
// How to use AWS AppConfig Agent

const application_name = "MyDemoApp";
const environment_name = "MyEnvironment";
const config_profile_name = "MyConfigProfile";

// the agent runs a local HTTP server that serves configuration data
// make a GET request to the agent's local server to retrieve the configuration data
const url = `http://localhost:2772/applications/${application_name}/environments/
${environment_name}/configurations/${config_profile_name}`;
const response = await fetch(url);
const config = await response.text(); // (use `await response.json()` if your config
 is json)
```

AWS AppConfig エージェントを使用して特定の機能フラグを読み取る

次の各サンプルには、コードによって実行されるアクションに関するコメントが含まれています。

Java

```
public void retrieveSingleFlagFromAgent() throws Exception {
        /*
          You can retrieve a single flag's data from the agent by providing the
 "flag" query string parameter.
          Note: the configuration's type must be AWS.AppConfig.FeatureFlags
        */

        URL url = new URL("http://localhost:2772/applications/MyDemoApp/
environments/Beta/configurations/MyFlagsProfile?flag=myFlagKey");
        HttpURLConnection con = (HttpURLConnection) url.openConnection();
        con.setRequestMethod("GET");
        StringBuilder content;
        try (BufferedReader in = new BufferedReader(new
 InputStreamReader(con.getInputStream()))) {
            content = new StringBuilder();
            int ch;
            while ((ch = in.read()) != -1) {
                content.append((char) ch);
            }
        }
        con.disconnect();
        System.out.println("MyFlagName from agent: " + content);
    }
```

Python

```
import requests

application_name = 'MyDemoApp'
environment_name = 'MyEnvironment'
config_profile_name = 'MyConfigProfile'
flag_key = 'MyFlag'

# retrieve a single flag's data by providing the "flag" query string parameter
# note: the configuration's type must be AWS.AppConfig.FeatureFlags
```

```
response = requests.get(f"http://localhost:2772/applications/{application_name}/
environments/{environment_name}/configurations/{config_profile_name}?
flag={flag_key}")
config = response.content
```

JavaScript

```
const application_name = "MyDemoApp";
const environment_name = "MyEnvironment";
const config_profile_name = "MyConfigProfile";
const flag_name = "MyFlag";

// retrieve a single flag's data by providing the "flag" query string parameter
// note: the configuration's type must be AWS.AppConfig.FeatureFlags
const url = `http://localhost:2772/applications/${application_name}/environments/
${environment_name}/configurations/${config_profile_name}?flag=${flag_name}`;
const response = await fetch(url);
const flag = await response.json(); // { "enabled": true/false }
```

AWS AppConfig エージェントを使用してバリアントを含む機能フラグを取得する

次の各サンプルには、コードによって実行されるアクションに関するコメントが含まれています。

Java

```
public static void retrieveConfigFromAgentWithVariants() throws Exception {
    /*
    This sample retrieves feature flag configuration data
    containing variants from AWS AppConfig Agent.

    For more information about the agent, see How to use AWS AppConfig Agent
    */

    // Make a GET request to the agent's local server to retrieve the configuration
 data
    URL url = new URL("http://localhost:2772/applications/MyDemoApp/environments/
Beta/configurations/MyConfigProfile");
    HttpURLConnection con = (HttpURLConnection) url.openConnection();

    // Provide context in the 'Context' header
```

```
    // In the header value, use '=' to separate context key from context value
    // Note: Multiple context values may be passed either across
    // multiple headers or as comma-separated values in a single header
    con.setRequestProperty("Context", "country=US");

    StringBuilder content;
    try (BufferedReader in = new BufferedReader(new
 InputStreamReader(con.getInputStream()))) {
        content = new StringBuilder();
        int ch;
        while ((ch = in.read()) != -1) {
            content.append((char) ch);
        }
    }
    con.disconnect();
    System.out.println("Configuration from agent via HTTP: " + content);
}
```

Python

```
# This sample retrieve features flag configuration data
# containing variants from AWS AppConfig Agent.

# For more information about the agent, see How to use AWS AppConfig Agent

import requests

application_name = 'MyDemoApp'
environment_name = 'Beta'
config_profile_name = 'MyConfigProfile'

# make a GET request to the agent's local server to retrieve the configuration data
response = requests.get(f"http://localhost:2772/applications/{application_name}/
environments/{environment_name}/configurations/{configuration_profile_name}",
                        headers = {
                            "Context": "country=US" # Provide context in the
 'Context' header
                                                    # In the header value, use '='
 to separate context key from context value
                                                    # Note: Multiple context values
 may be passed either across
                                                    # multiple headers or as comma-
separated values in a single header
```

```
                        }
)
print("Configuration from agent via HTTP: ", response.json())
```

JavaScript

```
// This sample retrieves feature flag configuration data
// containing variants from AWS AppConfig Agent.

// For more information about the agent, see How to use AWS AppConfig Agent

const application_name = "MyDemoApp";
const environment_name = "Beta";
const config_profile_name = "MyConfigProfile";

const url = `http://localhost:2772/applications/${application_name}/environments/
${environment_name}/configurations/${configuration_profile_name}`;

// make a GET request to the agent's local server to retrieve the configuration data
const response = await fetch(url, {
    method: 'GET',
    headers: {
        'Context': 'country=US' // Provide context in the 'Context' header
                                // In the header value, use '=' to separate context
 key from context value
                                // Note: Multiple context values may be passed
 either across
                                // multiple headers or as comma-separated values in
 a single header
    }
});

const config = await response.json();
console.log("Configuration from agent via HTTP: ", config);
```

GetLatestConfiguration API アクションを使用してフリーフォーム設定プロファイルを読み取る

次の各サンプルには、コードによって実行されるアクションに関するコメントが含まれています。このセクションのサンプルでは、次の API を呼び出します。

- GetLatestConfiguration
- StartConfigurationSession

Java

```
/*
The example below uses two AWS AppConfig Data APIs: StartConfigurationSession and
 GetLatestConfiguration.
For more information about these APIs, see AWS AppConfig Data.

This class is meant to be used as a singleton to retrieve the latest configuration
 data from AWS AppConfig.
This class maintains a cache of the latest configuration data in addition to the
 configuration token to be
passed to the next GetLatestConfiguration API call.
*/
class AppConfigApiRetriever {
    /* AWS AppConfig Data SDK client used to interact with the AWS AppConfig Data
 service.
     */
    private AppConfigDataClient appConfigData;

    /*
    The configuration token to be passed to the next GetLatestConfiguration API
 call.
     */
    private String configurationToken;

    /*
    The cached configuration data to be returned when there is no new configuration
 data available.
     */
    private SdkBytes configuration;

    public AppConfigApiRetriever() {
        this.appConfigData = AppConfigDataClient.create();
    }

    /*
    Returns the latest configuration data stored in AWS AppConfig.
     */
    public SdkBytes getConfig() {
```

```
        /*
        If there is no configuration token yet, get one by starting a new session
with the StartConfigurationSession API.
        Note that this API does not return configuration data. Rather, it returns an
initial configuration token that is
        subsequently passed to the GetLatestConfiguration API.
         */
        if (this.configurationToken == null) {
            StartConfigurationSessionResponse session =
appConfigData.startConfigurationSession(req -> req
                    .applicationIdentifier("MyDemoApp")
                    .configurationProfileIdentifier("MyConfig")
                    .environmentIdentifier("Beta"));
            this.configurationToken = session.initialConfigurationToken();
        }

        /*
        Retrieve the configuration from the GetLatestConfiguration API, providing
the current configuration token.
        If this caller does not yet have the latest configuration (e.g. this is the
first call to GetLatestConfiguration
        or new configuration data has been deployed since the first call), the
latest configuration data will be returned.
        Otherwise, the GetLatestConfiguration API will not return any data since the
caller already has the latest.
         */
        GetLatestConfigurationResponse response =
appConfigData.getLatestConfiguration(

GetLatestConfigurationRequest.builder().configurationToken(this.configurationToken).build()

        /*
        Save the returned configuration token so that it can be passed to the next
GetLatestConfiguration API call.
        Warning: Not persisting this token for use in the next
GetLatestConfiguration API call may result in higher
        than expected usage costs.
         */
        this.configurationToken = response.nextPollConfigurationToken();

        /*
        If the GetLatestConfiguration API returned configuration data, update the
cached configuration with the returned data.
```

```
        Otherwise, assume the configuration has not changed, and return the cached
 configuration.
         */
        SdkBytes configFromApi = response.configuration();
        if (configFromApi.asByteArray().length != 0) {
            this.configuration = configFromApi;
            System.out.println("Configuration contents have changed since the last
 GetLatestConfiguration call, new contents = " + this.configuration.asUtf8String());
        } else {
            System.out.println("GetLatestConfiguration returned an empty response
 because we already have the latest configuration");
        }

        return this.configuration;
    }
}
```

Python

```
# The example below uses two AWS AppConfig Data APIs: StartConfigurationSession and
 GetLatestConfiguration.
# For more information about these APIs, see AWS AppConfig Data.
#
# This class is meant to be used as a singleton to retrieve the latest configuration
 data from AWS AppConfig.
# This class maintains a cache of the latest configuration data in addition to the
 configuration token to be
# passed to the next GetLatestConfiguration API call.
class AppConfigApiRetriever:
    def __init__(self):
        # AWS AppConfig Data SDK client used to interact with the AWS AppConfig Data
 service.
        self.appconfigdata = boto3.client('appconfigdata')

        # The configuration token to be passed to the next GetLatestConfiguration
 API call.
        self.configuration_token = None

        # The cached configuration data to be returned when there is no new
 configuration data available.
        self.configuration = None

    # Returns the latest configuration data stored in AWS AppConfig.
```

```
    def get_config(self):
        # If there is no configuration token yet, get one by starting a new session
with the StartConfigurationSession API.
        # Note that this API does not return configuration data. Rather, it returns
an initial configuration token that is
        # subsequently passed to the GetLatestConfiguration API.
        if not self.configuration_token:
            session = self.appconfigdata.start_configuration_session(
                ApplicationIdentifier='MyDemoApp',
                ConfigurationProfileIdentifier='MyConfig',
                EnvironmentIdentifier='Beta'
            )
            self.configuration_token = session['InitialConfigurationToken']

        # Retrieve the configuration from the GetLatestConfiguration API, providing
the current configuration token.
        # If this caller does not yet have the latest configuration (e.g. this is
the first call to GetLatestConfiguration
        # or new configuration data has been deployed since the first call), the
latest configuration data will be returned.
        # Otherwise, the GetLatestConfiguration API will not return any data since
the caller already has the latest.
        response =
self.appconfigdata.get_latest_configuration(ConfigurationToken=self.configuration_token)

        # Save the returned configuration token so that it can be passed to the next
GetLatestConfiguration API call.
        # Warning: Not persisting this token for use in the next
GetLatestConfiguration API call may result in higher
        # than expected usage costs.
        self.configuration_token = response['NextPollConfigurationToken']

        # If the GetLatestConfiguration API returned configuration data, update the
cached configuration with the returned data.
        # Otherwise, assume the configuration has not changed, and return the cached
configuration.
        config_from_api = response['Configuration'].read()
        if config_from_api:
            self.configuration = config_from_api
            print('Configuration contents have changed since the last
GetLatestConfiguration call, new contents = ' + str(self.configuration))
        else:
            print('GetLatestConfiguration returned an empty response because we
already have the latest configuration')
```

```
        return self.configuration
```

JavaScript

```
/*
The example below uses two AWS AppConfig Data APIs: StartConfigurationSession and
 GetLatestConfiguration.
For more information about these APIs, see AWS AppConfig Data.

This class is meant to be used as a singleton to retrieve the latest configuration
 data from AWS AppConfig.
This class maintains a cache of the latest configuration data in addition to the
 configuration token to be
passed to the next GetLatestConfiguration API call.
*/
class AppConfigApiRetriever {
    constructor() {
        /* AWS AppConfig Data SDK client used to interact with the AWS AppConfig
 Data service.
         */
        this.appconfigdata = new AppConfigDataClient();

        /*
        The configuration token to be passed to the next GetLatestConfiguration API
 call.
         */
        this.configurationToken = null;

        /*
        The cached configuration data to be returned when there is no new
 configuration data available.
         */
        this.configuration = null;
    }

    /*
    Returns the latest configuration data stored in AWS AppConfig.
     */
    async getConfig() {
        /*
        If there is no configuration token yet, get one by starting a new session
 with the StartConfigurationSession API.
```

```
        Note that this API does not return configuration data. Rather, it returns an
initial configuration token that is
        subsequently passed to the GetLatestConfiguration API.
        */
        if (!this.configurationToken) {
            const session = await this.appconfigdata.send(
                new StartConfigurationSessionCommand({
                    ApplicationIdentifier: "MyDemoApp",
                    ConfigurationProfileIdentifier: "MyConfig",
                    EnvironmentIdentifier: "Beta"
                })
            );
            this.configurationToken = session.InitialConfigurationToken;
        }

        /*
        Retrieve the configuration from the GetLatestConfiguration API, providing
the current configuration token.
        If this caller does not yet have the latest configuration (e.g. this is the
first call to GetLatestConfiguration
        or new configuration data has been deployed since the first call), the
latest configuration data will be returned.
        Otherwise, the GetLatestConfiguration API will not return any data since the
caller already has the latest.
        */
        const response = await this.appconfigdata.send(
            new GetLatestConfigurationCommand({
                ConfigurationToken: this.configurationToken
            })
        );

        /*
        Save the returned configuration token so that it can be passed to the next
GetLatestConfiguration API call.
        Warning: Not persisting this token for use in the next
GetLatestConfiguration API call may result in higher
        than expected usage costs.
        */
        this.configurationToken = response.NextPollConfigurationToken;

        /*
        If the GetLatestConfiguration API returned configuration data, update the
cached configuration with the returned data.
```

```
        Otherwise, assume the configuration has not changed, and return the cached
 configuration.
        */
        const configFromApi = response.Configuration.transformToString();
        if (configFromApi) {
            this.configuration = configFromApi;
            console.log("Configuration contents have changed since the last
 GetLatestConfiguration call, new contents = " + this.configuration);
        } else {
            console.log("GetLatestConfiguration returned an empty response because
 we already have the latest configuration");
        }

        return this.configuration;
    }
}
```

環境のクリーンアップ

このセクションで 1 つ以上のコードサンプルを実行した場合は、次のいずれかのサンプルを使用して、それらのコードサンプルによって作成された AWS AppConfig リソースを見つけて削除することをお勧めします。このセクションのサンプルでは、次の API を呼び出します。

- ListApplications
- DeleteApplication
- ListEnvironments
- DeleteEnvironments
- ListConfigurationProfiles
- DeleteConfigurationProfile
- ListHostedConfigurationVersions
- DeleteHostedConfigurationVersion

Java

```
/*
    This sample provides cleanup code that deletes all the AWS AppConfig resources
 created in the samples above.
```

```
   WARNING: this code will permanently delete the given application and all of its
sub-resources, including
   configuration profiles, hosted configuration versions, and environments. DO NOT
run this code against
   an application that you may need in the future.
   */

   public void cleanUpDemoResources() {
       AppConfigClient appconfig = AppConfigClient.create();

       // The name of the application to delete
       // IMPORTANT: verify this name corresponds to the application you wish to
delete
       String applicationToDelete = "MyDemoApp";


appconfig.listApplicationsPaginator(ListApplicationsRequest.builder().build()).items().forE
-> {
           if (app.name().equals(applicationToDelete)) {
               System.out.println("Deleting App: " + app);
               appconfig.listConfigurationProfilesPaginator(req ->
req.applicationId(app.id())).items().forEach(cp -> {
                   System.out.println("Deleting Profile: " + cp);
                   appconfig
                       .listHostedConfigurationVersionsPaginator(req -> req
                           .applicationId(app.id())
                           .configurationProfileId(cp.id()))
                       .items()
                       .forEach(hcv -> {
                           System.out.println("Deleting HCV: " + hcv);
                           appconfig.deleteHostedConfigurationVersion(req -> req
                               .applicationId(app.id())
                               .configurationProfileId(cp.id())
                               .versionNumber(hcv.versionNumber()));
                       });
                   appconfig.deleteConfigurationProfile(req -> req
                       .applicationId(app.id())
                       .configurationProfileId(cp.id()));
               });

               appconfig.listEnvironmentsPaginator(req-
>req.applicationId(app.id())).items().forEach(env -> {
                   System.out.println("Deleting Environment: " + env);
```

```
                    appconfig.deleteEnvironment(req-
>req.applicationId(app.id()).environmentId(env.id()));
                });

                appconfig.deleteApplication(req -> req.applicationId(app.id()));
            }
        });
    }
```

Python

```
# this sample provides cleanup code that deletes all the AWS AppConfig resources
 created in the samples above.
#
# WARNING: this code will permanently delete the given application and all of its
 sub-resources, including
#   configuration profiles, hosted configuration versions, and environments. DO NOT
 run this code against
#   an application that you may need in the future.
#

import boto3

# the name of the application to delete
# IMPORTANT: verify this name corresponds to the application you wish to delete
application_name = 'MyDemoApp'

# create and iterate over a list paginator such that we end up with a list of pages,
 which are themselves lists of applications
# e.g. [ [{'Name':'MyApp1',...},{'Name':'MyApp2',...}], [{'Name':'MyApp3',...}] ]
list_of_app_lists = [page['Items'] for page in
 appconfig.get_paginator('list_applications').paginate()]
# retrieve the target application from the list of lists
application = [app for apps in list_of_app_lists for app in apps if app['Name'] ==
 application_name][0]
print(f"deleting application {application['Name']} (id={application['Id']})")

# delete all configuration profiles
list_of_config_lists = [page['Items'] for page in
 appconfig.get_paginator('list_configuration_profiles').paginate(ApplicationId=application['
for config_profile in [config for configs in list_of_config_lists for config in
 configs]:
```

```
    print(f"\tdeleting configuration profile {config_profile['Name']}
 (Id={config_profile['Id']})")

    # delete all hosted configuration versions
    list_of_hcv_lists = [page['Items'] for page in
 appconfig.get_paginator('list_hosted_configuration_versions').paginate(ApplicationId=applic
 ConfigurationProfileId=config_profile['Id'])]
    for hcv in [hcv for hcvs in list_of_hcv_lists for hcv in hcvs]:

 appconfig.delete_hosted_configuration_version(ApplicationId=application['Id'],
 ConfigurationProfileId=config_profile['Id'], VersionNumber=hcv['VersionNumber'])
        print(f"\t\tdeleted hosted configuration version {hcv['VersionNumber']}")

    # delete the config profile itself
    appconfig.delete_configuration_profile(ApplicationId=application['Id'],
 ConfigurationProfileId=config_profile['Id'])
    print(f"\tdeleted configuration profile {config_profile['Name']}
 (Id={config_profile['Id']})")

# delete all environments
list_of_env_lists = [page['Items'] for page in
 appconfig.get_paginator('list_environments').paginate(ApplicationId=application['Id'])]
for environment in [env for envs in list_of_env_lists for env in envs]:
    appconfig.delete_environment(ApplicationId=application['Id'],
 EnvironmentId=environment['Id'])
    print(f"\tdeleted environment {environment['Name']} (Id={environment['Id']})")

# delete the application itself
appconfig.delete_application(ApplicationId=application['Id'])
print(f"deleted application {application['Name']} (id={application['Id']})")
```

JavaScript

```
// this sample provides cleanup code that deletes all the AWS AppConfig resources
 created in the samples above.

// WARNING: this code will permanently delete the given application and all of its
 sub-resources, including
//   configuration profiles, hosted configuration versions, and environments. DO NOT
 run this code against
//   an application that you may need in the future.

import {
```

```
  AppConfigClient,
  paginateListApplications,
  DeleteApplicationCommand,
  paginateListConfigurationProfiles,
  DeleteConfigurationProfileCommand,
  paginateListHostedConfigurationVersions,
  DeleteHostedConfigurationVersionCommand,
  paginateListEnvironments,
  DeleteEnvironmentCommand,
} from "@aws-sdk/client-appconfig";

const client = new AppConfigClient();

// the name of the application to delete
// IMPORTANT: verify this name corresponds to the application you wish to delete
const application_name = "MyDemoApp";

// iterate over all applications, deleting ones that have the name defined above
for await (const app_page of paginateListApplications({ client }, {})) {
  for (const application of app_page.Items) {

    // skip applications that dont have the name thats set
    if (application.Name !== application_name) continue;

    console.log( `deleting application ${application.Name} (id=${application.Id})`);

    // delete all configuration profiles
    for await (const config_page of paginateListConfigurationProfiles({ client },
 { ApplicationId: application.Id })) {
      for (const config_profile of config_page.Items) {
        console.log(`\tdeleting configuration profile ${config_profile.Name} (Id=
${config_profile.Id})`);

        // delete all hosted configuration versions
        for await (const hosted_page of
 paginateListHostedConfigurationVersions({ client },
          { ApplicationId: application.Id, ConfigurationProfileId:
 config_profile.Id }
        )) {
          for (const hosted_config_version of hosted_page.Items) {
            await client.send(
              new DeleteHostedConfigurationVersionCommand({
                ApplicationId: application.Id,
                ConfigurationProfileId: config_profile.Id,
```

```
                VersionNumber: hosted_config_version.VersionNumber,
              })
            );
            console.log(`\t\tdeleted hosted configuration version
 ${hosted_config_version.VersionNumber}`);
          }
        }

        // delete the config profile itself
        await client.send(
          new DeleteConfigurationProfileCommand({
            ApplicationId: application.Id,
            ConfigurationProfileId: config_profile.Id,
          })
        );
        console.log(`\tdeleted configuration profile ${config_profile.Name} (Id=
${config_profile.Id})`)
      }

      // delete all environments
      for await (const env_page of paginateListEnvironments({ client },
 { ApplicationId: application.Id })) {
        for (const environment of env_page.Items) {
          await client.send(
            new DeleteEnvironmentCommand({
              ApplicationId: application.Id,
              EnvironmentId: environment.Id,
            })
          );
          console.log(`\tdeleted environment ${environment.Name} (Id=
${environment.Id})`)
        }
      }
    }

    // delete the application itself
    await client.send(
      new DeleteApplicationCommand({ ApplicationId: application.Id })
    );
    console.log(`deleted application ${application.Name} (id=${application.Id})`)
  }
}
```

AWS AppConfig 削除保護の設定

AWS AppConfig は、ユーザーが意図せずにアクティブに使用されている環境と設定プロファイルを削除しないようにするためのアカウント設定を提供します。 は GetLatestConfiguration と GetConfiguration への呼び出し AWS AppConfig を監視し、60 分間隔 (デフォルト設定) でこれらの呼び出しに含まれた設定プロファイルと環境を追跡します。その間隔内にアクセスされた設定プロファイルまたは環境は、アクティブと見なされます。アクティブな設定プロファイルまたは環境を削除しようとすると、 はエラー AWS AppConfig を返します。必要に応じて、`DeletionProtectionCheck`パラメータを使用してこのエラーをバイパスできます。詳細については、「削除保護チェックをバイパスまたは強制する」を参照してください。

コンソールを使用して削除保護を設定する

コンソールを使用して削除保護を設定するには、 AWS Systems Manager 次の手順に従います。

削除保護を設定するには (コンソール)

1. AWS Systems Manager コンソールを https://console.aws.amazon.com/systems-manager/appconfig/://https://https://https://https://https://https
2. ナビゲーションペインで [設定] を選択します。
3. 削除保護を有効または無効にするには、 トグルを使用します。
4. 保護期間では、アクティブなリソースの定義を 15 ~ 1440 分に設定します。
5. [適用] をクリックします。

を使用して削除保護を設定する AWS CLI

を使用して削除保護を設定するには、次の手順に従います AWS CLI。次のコマンドの##、環境で使用する値に置き換えます。

> **Note**
>
> 開始する前に、 の最新バージョンに更新することをお勧めします AWS CLI。詳細については、「AWS Command Line Interface ユーザーガイド」の「AWS CLIの最新バージョンをインストールまたは更新する」を参照してください。

削除保護を設定するには (CLI)

1. 次のコマンドを実行して、現在の削除保護設定を表示します。

```
aws appconfig get-account-settings
```

2. 削除保護を有効または無効にするには、次のコマンドを実行します。削除保護を無効にするには `true`を指定し`false`、有効にするには を指定します。

```
aws appconfig update-account-settings --deletion-protection Enabled=value
```

3. デフォルトの間隔を最大 24 時間まで増やすことができます。次のコマンドを実行して、新しい間隔を指定します。

```
aws appconfig update-account-settings --deletion-protection
 Enabled=true,ProtectionPeriodInMinutes=a number between 15 and 1440
```

削除保護チェックをバイパスまたは強制する

削除保護の管理に役立つように、DeleteEnvironment API と DeleteConfigurationProfile API には `DeletionProtectionCheck` というパラメータが含まれています。このパラメータでは次の値がサポートされます。

- `BYPASS`: 削除保護チェックをバイパスし、削除保護によって妨げられた場合でも設定プロファイルを削除する AWS AppConfig ように に指示します。
- `APPLY`: アカウントレベルで削除保護が無効になっている場合でも、削除保護チェックを実行するように指示します。また、`APPLY` は、通常は削除保護チェックから除外される過去 1 時間以内に作成されたリソースに対しても削除保護チェックを強制的に実行します。
- `ACCOUNT_DEFAULT`: AWS AppConfig に `UpdateAccountSettings` API で指定された削除保護値を実装するように指示するデフォルト設定。

> **Note**
>
> デフォルトでは、`DeletionProtectionCheck` は過去 1 時間に作成された設定プロファイルと環境をスキップします。デフォルト設定は、有効期間の短いリソースを作成するテストやデモに削除保護が干渉するのを防ぐことを目的としています。この動作

は、DeleteEnvironment または DeleteConfigurationProfile を呼び出すときに DeletionProtectionCheck=APPLY を渡すことで上書きできます。

次の CLI チュートリアルでは、サンプルコマンドを使用して DeletionProtectionCheckパラメータの使用方法を示します。次のコマンドの *ID* をアー AWS AppConfig ティファクトの ID に置き換えます。

1. デプロイされた設定で GetLatestConfiguration を呼び出します。

```
aws appconfigdata get-latest-configuration --configuration-token $(aws
 appconfigdata start-configuration-session --application-identifier ID --
environment-identifier ID --configuration-profile-identifier ID --query
 InitialConfigurationToken) outfile.txt
```

2. 設定がアクティブであることを が登録 AWS AppConfig するまで 60 秒待ちます。
3. 次のコマンドを実行して DeleteEnvironment を呼び出し、環境に削除保護を適用します。

```
aws appconfig delete-environment --environment-id ID --application-id ID --
deletion-protection-check APPLY
```

コマンドは次のエラーを返します。

```
An error occurred (BadRequestException) when calling the DeleteEnvironment
 operation: Environment Beta is actively being used in your application and cannot
 be deleted.
```

4. 次のコマンドを実行して、削除保護をバイパスし、環境を削除します。

```
aws appconfig delete-environment --environment-id ID --application-id ID --
deletion-protection-check BYPASS
```

のセキュリティ AWS AppConfig

のクラウドセキュリティが最優先事項 AWS です。お客様は AWS 、セキュリティの影響を受けやすい組織の要件を満たすように構築されたデータセンターとネットワークアーキテクチャからメリットを得られます。

セキュリティは、 AWS とユーザーの間で共有される責任です。責任共有モデルでは、これをクラウドのセキュリティおよびクラウド内のセキュリティと説明しています。

- クラウドのセキュリティ – AWS は、 で AWS サービスを実行するインフラストラクチャを保護する責任があります AWS クラウド。 AWS また、 では、安全に使用できるサービスも提供しています。サードパーティーの監査者は、AWS コンプライアンスプログラムコンプライアンスプログラムの一環として、当社のセキュリティの有効性を定期的にテストおよび検証。が適用されるコンプライアンスプログラムの詳細については AWS Systems Manager、「コンプライアンスAWS プログラムによる対象範囲内のサービスコンプライアンス」を参照してください。
- クラウド内のセキュリティ – お客様の責任は、使用する AWS サービスによって決まります。また、ユーザーは、データの機密性、会社の要件、適用される法律や規制など、その他の要因についても責任を負います。

AWS AppConfig は のツールです AWS Systems Manager。の使用時に責任共有モデルを適用する方法については AWS AppConfig、「 のセキュリティ AWS Systems Manager」を参照してください。ここでは、 AWS AppConfigのセキュリティおよびコンプライアンス目標を達成するために Systems Manager を設定する方法を説明しています。

最小特権アクセスの実装

セキュリティのベストプラクティスとして、ID が特定の条件下で特定のリソースに対して特定のアクションを実行するために必要な最小限のアクセス許可を付与します。 AWS AppConfig エージェントには、エージェントがインスタンスまたはコンテナのファイルシステムにアクセスできるようにする 2 つの機能があります。バックアップとディスクへの書き込みです。これらの機能を有効にする場合は、 AWS AppConfig エージェントのみがファイルシステム上の指定された設定ファイルに書き込むアクセス許可を持っていることを確認します。また、これらの設定ファイルから読み取る必要があるプロセスのみがその権限を持っていることも確認してください。最小特権アクセスの実装は、セキュリティリスクと、エラーや悪意によってもたらされる可能性のある影響の低減における基本です。

最小特権アクセスの実装の詳細については、「AWS Well-Architected Tool ユーザーガイド」の「SEC03-BP02 最小特権アクセスを付与する」を参照してください。このセクションで説明する AWS AppConfig エージェント機能の詳細については、「」を参照してくださいマニフェストを使用して追加の取得機能を有効にする。

AWS AppConfigでの静止時のデータの暗号化

AWS AppConfig は、 を使用して保管中の顧客データを保護するために、デフォルトで暗号化を提供します AWS 所有のキー。

AWS 所有のキー - デフォルトでは、これらのキー AWS AppConfig を使用して、サービスによってデプロイされ、データストアでホストされている AWS AppConfig データを自動的に暗号化します。表示、管理、使用 AWS 所有のキー、またはそれらの使用を監査することはできません。ただし、データを暗号化するキーを保護するために何か行動を起こしたり、プログラムを変更したりする必要はありません。詳細については、「AWS Key Management Service デベロッパーガイド」の 「AWS 所有のキー」を参照してください。

この暗号化レイヤーを無効にしたり、代替の暗号化タイプを選択したりすることはできませんが、AWS AppConfig データストアでホストされている設定データを保存したり、設定データをデプロイしたりするときに使用するカスタマーマネージドキーを指定できます。

カスタマーマネージドキー — 作成、所有、管理する対称カスタマーマネージドキー AWS AppConfig を使用して、既存のキーに 2 番目の暗号化レイヤーを追加します AWS 所有のキー。ユーザーがこの暗号化レイヤーを完全に制御できるため、次のようなタスクを実行できます。

- キーポリシーとグラントの策定と維持
- IAM ポリシーの策定と維持
- キーポリシーの有効化と無効化
- キー暗号化マテリアルのローテーション
- タグの追加
- キーエイリアスの作成
- キー削除のスケジュール設定

詳細については、AWS Key Management Service デベロッパーガイドの「カスタマーマネージドキー」を参照してください。

AWS AppConfig がカスタマーマネージドキーをサポート

AWS AppConfig では、設定データのカスタマーマネージドキー暗号化がサポートされています。 AWS AppConfig ホストされたデータストアに保存されている設定バージョンの場合、お客様は対応する設定プロファイルKmsKeyIdentifierで を設定できます。CreateHostedConfigurationVersion API オペレーションを使用して設定データの新しいバージョンが作成されるたびに、 は から AWS KMS データキー AWS AppConfig を生成KmsKeyIdentifierし、保存する前にデータを暗号化します。GetHostedConfigurationVersion または StartDeployment API オペレーション中に後でデータにアクセスすると、 は生成されたデータキーに関する情報を使用して設定データを AWS AppConfig 復号します。

AWS AppConfig は、デプロイされた設定データのカスタマーマネージドキー暗号化もサポートしています。設定データを暗号化するために、お客様はデプロイKmsKeyIdentifierにを提供できます。 はこれを使用して AWS KMS データキー AWS AppConfig を生成KmsKeyIdentifierし、StartDeploymentAPI オペレーションのデータを暗号化します。

AWS AppConfig 暗号化アクセス

顧客管理キーを作成するときは、次のキーポリシーを使用して、キーが使用可能であることを確認してください。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
        "Sid": "Allow use of the key",
        "Effect": "Allow",
        "Principal": {
            "AWS": "arn:aws:iam::account_ID:role/role_name"
        },
        "Action": [
            "kms:Decrypt",
            "kms:GenerateDataKey"
            ],
        "Resource": "*"
        }
 ]
```

ホストされている設定データをカスタマー管理キーで暗号化するには、ID 呼び出しにユーザー、グループ、CreateHostedConfigurationVersion またはロールに割り当てることができる以下のポリシーステートメントが必要です。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "kms:GenerateDataKey,
            "Resource": "arn:aws:kms:Region:account_ID:key_ID"
        }
    ]
}
```

Secrets Manager のシークレットや、顧客管理キーで暗号化されたその他の設定データを使用している場合は `retrievalRoleArn` 、`kms:Decrypt` データを復号化して取得する必要があります。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "kms:Decrypt",
      "Resource": "arn:aws:kms:Region:account_ID:configuration source/object"
    }
  ]
}
```

AWS AppConfig StartDeployment API オペレーションを呼び出す場合、ID 呼び出しには、ユーザー、グループ、またはロールに割り当てることができる次の IAM ポリシー`StartDeployment`が必要です。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kms:GenerateDataKey*"
      ],
      "Resource": "arn:aws:kms:Region:account_ID:key_ID"
    }
  ]
}
```

AWS AppConfig GetLatestConfiguration API オペレーションを呼び出す場合、ID 呼び出しには、ユーザー、グループ、またはロールに割り当てることができる次のポリシーGetLatestConfigurationが必要です。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "kms:Decrypt,
            "Resource": "arn:aws:kms:Region:account_ID:key_ID"
        }
    ]
}
```

暗号化コンテキスト

暗号化コンテキストは、データに関する追加のコンテキスト情報が含まれたキーバリューペアのオプションのセットです。

AWS KMS は、追加の認証済みデータとして暗号化コンテキストを使用して、認証済み暗号化をサポートします。データを暗号化するリクエストに暗号化コンテキストを含めると、 は暗号化コンテキストを暗号化されたデータに AWS KMS バインドします。データを復号化するには、そのリクエストに (暗号化時と) 同じ暗号化コンテキストを含めます。

AWS AppConfig 暗号化コンテキスト: 暗号化されたホスト設定データとデプロイのすべての AWS KMS 暗号化オペレーションで暗号化コンテキスト AWS AppConfig を使用します。コンテキストには、データのタイプに対応するキーと、特定のデータ項目を識別する値が含まれます。

の暗号化キーのモニタリング AWS

で AWS KMS カスタマーマネージドキーを使用する場合 AWS AppConfig、 AWS CloudTrail または Amazon CloudWatch Logs を使用して、 AWS AppConfig が に送信するリクエストを追跡できます AWS KMS。

次の例は、 がカスタマーマネージドキーによって暗号化されたデータにアクセス AWS AppConfig するために によって呼び出されるオペレーションをモニタリングDecrypt AWS KMS するための CloudTrail イベントです。

```
{
    "eventVersion": "1.08",
```

```
    "userIdentity": {
        "type": "AWSService",
        "invokedBy": "appconfig.amazonaws.com"
    },
    "eventTime": "2023-01-03T02:22:28z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "Decrypt",
    "awsRegion": "Region",
    "sourceIPAddress": "172.12.34.56",
    "userAgent": "ExampleDesktop/1.0 (V1; OS)",
    "requestParameters": {
        "encryptionContext": {
            "aws:appconfig:deployment:arn":
 "arn:aws:appconfig:Region:account_ID:application/application_ID/
environment/environment_ID/deployment/deployment_ID"
        },
        "keyId": "arn:aws:kms:Region:account_ID:key/key_ID",
        "encryptionAlgorithm": "SYMMETRIC_DEFAULT"
    },
    "responseElements": null,
    "requestID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "eventID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "readOnly": true,
    "resources": [
        {
            "accountId": "account_ID",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:Region:account_ID:key_ID"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "eventCategory": "Management",
    "recipientAccountId": "account_ID",
    "sharedEventID": "dc129381-1d94-49bd-b522-f56a3482d088"
}
```

インターフェイスエンドポイント (AWS PrivateLink) AWS AppConfig を使用した へのアクセス

を使用して AWS PrivateLink 、VPC と の間にプライベート接続を作成できます AWS AppConfig。インターネットゲートウェイ、NAT デバイス、VPN 接続、または AWS Direct Connect 接続を使用

せずに、VPC 内にある AWS AppConfig かのように にアクセスできます。VPC内のインスタンスはAWS AppConfigにアクセスするためにパブリックIPアドレスを必要としません。

このプライベート接続を確立するには、 AWS PrivateLinkを利用したインターフェイスエンドポイントを作成します。インターフェイスエンドポイントに対して有効にする各サブネットにエンドポイントネットワークインターフェイスを作成します。これらは、 AWS AppConfig宛てのトラフィックのエントリポイントとして機能するリクエスタ管理型ネットワークインターフェイスです。

詳細については「 AWS PrivateLink Guide (AWS PrivateLink ガイド)」の「Access an AWS のサービス using an interface VPC endpoint (インターフェイス VPC エンドポイントを使用して にアクセスする)」を参照してください。

に関する考慮事項 AWS AppConfig

のインターフェイスエンドポイントを設定する前に AWS AppConfig、「 AWS PrivateLink ガイド」の「考慮事項」を参照してください。

AWS AppConfig は、インターフェイスエンドポイントを介した `appconfig`および `appconfigdata`サービスの呼び出しをサポートしています。

AWS AppConfig用のインターフェイスエンドポイントの作成

Amazon VPC コンソールまたは AWS Command Line Interface () AWS AppConfig を使用して、 のインターフェイスエンドポイントを作成できますAWS CLI。詳細については、「AWS PrivateLink ガイド」の「インターフェイスエンドポイントを作成」を参照してください。

次のサービス名 AWS AppConfig を使用して、 用のインターフェイスエンドポイントを作成します。

```
com.amazonaws.region.appconfig
```

```
com.amazonaws.region.appconfigdata
```

インターフェース・エンドポイントのプライベートDNSを有効にすると、デフォルトの地域DNS名を使用して AWS AppConfig へのAPI要求を行うことができます。例えば、`appconfig.us-east-1.amazonaws.com` と `appconfigdata.us-east-1.amazonaws.com` です。

インターフェイスエンドポイントのエンドポイントポリシーを作成する

エンドポイントポリシーは、インターフェイスエンドポイントにアタッチできる IAM リソースです。デフォルトのエンドポイントポリシーでは、インターフェイスエンドポイント AWS AppConfig を介した へのフルアクセスが許可されます。VPC AWS AppConfig から に許可されるアクセスを制御するには、インターフェイスエンドポイントにカスタムエンドポイントポリシーをアタッチします。

エンドポイントポリシーは以下の情報を指定します。

- アクションを実行できるプリンシパル (AWS アカウント、IAM ユーザー、IAM ロール)。
- 実行可能なアクション。
- このアクションを実行できるリソース。

詳細については、AWS PrivateLink ガイドのControl access to services using endpoint policies (エンドポイントポリシーを使用してサービスへのアクセスをコントロールする)を参照してください。

例: AWS AppConfig アクションの VPC エンドポイントポリシー

以下は、カスタムエンドポイントポリシーの例です。インターフェイスエンドポイントにアタッチされると、このポリシーは、すべてのリソースですべてのプリンシパルに、リストされている AWS AppConfig アクションへのアクセス権を付与します。

```
{
   "Statement": [
      {
         "Principal": "*",
         "Effect": "Allow",
         "Action": [
            "appconfig:CreateApplication",
            "appconfig:CreateEnvironment",
            "appconfig:CreateConfigurationProfile",
            "appconfig:StartDeployment",
            "appconfig:GetLatestConfiguration"
            "appconfig:StartConfigurationSession"
         ],
         "Resource":"*"
      }
   ]
}
```

Secrets Manager のキーローテーション

このセクションでは、Secrets Manager と AWS AppConfig の統合に関する重要なセキュリティ情報について説明します。Secrets Manager の詳細については、 AWS Secrets Manager ユーザーガイドの「 とは AWS Secrets Manager」を参照してください。

によってデプロイされた Secrets Manager シークレットの自動ローテーションの設定 AWS AppConfig

ローテーションとは、Secrets Manager に保存されているシークレットを定期的に更新するためのプロセスのことです。 シークレットのローテーションを行うと、シークレット、ならびに、データベースまたはサービスの認証情報が更新されます。 AWS Lambda 関数を使用してシークレットとデータベースを更新することで、Secrets Manager でシークレットの自動ローテーションを設定できます。詳細については、AWS Secrets Manager ユーザーガイド の「シークレット AWS Secrets Manager のローテーション」を参照してください。

によってデプロイされた Secrets Manager シークレットのキーローテーションを有効にするには AWS AppConfig、ローテーション Lambda 関数を更新し、ローテーションされたシークレットをデプロイします。

> **Note**
>
> シークレットがローテーションされ、新しいバージョンに完全に更新されたら、 AWS AppConfig 設定プロファイルをデプロイします。`VersionStage` のステータスが `AWSPENDING` から `AWSCURRENT` に変更されたためにシークレットがローテーションされたかどうかを判断できます。シークレットローテーションの完了は、Secrets Managerのローテーションテンプレート `finish_secret` 機能内で行われます。

シークレットがローテーションされた後に AWS AppConfig デプロイを開始する 関数の例を次に示します。

```
import time
import boto3
client = boto3.client('appconfig')

def finish_secret(service_client, arn, new_version):
    """Finish the rotation by marking the pending secret as current
```

```
    This method finishes the secret rotation by staging the secret staged AWSPENDING
with the AWSCURRENT stage.
    Args:
        service_client (client): The secrets manager service client
        arn (string): The secret ARN or other identifier
        new_version (string): The new version to be associated with the secret
    """
    # First describe the secret to get the current version
    metadata = service_client.describe_secret(SecretId=arn)
    current_version = None
    for version in metadata["VersionIdsToStages"]:
        if "AWSCURRENT" in metadata["VersionIdsToStages"][version]:
            if version == new_version:
                # The correct version is already marked as current, return
                logger.info("finishSecret: Version %s already marked as AWSCURRENT for
%s" % (version, arn))
                return
            current_version = version
            break

    # Finalize by staging the secret version current
    service_client.update_secret_version_stage(SecretId=arn, VersionStage="AWSCURRENT",
MoveToVersionId=new_version, RemoveFromVersionId=current_version)

    # Deploy rotated secret
    response = client.start_deployment(
            ApplicationId='TestApp',
            EnvironmentId='TestEnvironment',
            DeploymentStrategyId='TestStrategy',
            ConfigurationProfileId='ConfigurationProfileId',
            ConfigurationVersion=new_version,
            KmsKeyIdentifier=key,
            Description='Deploy secret rotated at ' + str(time.time())
        )

    logger.info("finishSecret: Successfully set AWSCURRENT stage to version %s for
secret %s." % (new_version, arn))
```

モニタリング AWS AppConfig

モニタリングは、 およびその他の AWS AppConfig AWS ソリューションの信頼性、可用性、パフォーマンスを維持する上で重要な部分です。 AWS には、監視 AWS AppConfig、問題の発生時の報告、必要に応じて自動アクションを実行するための以下のモニタリングツールが用意されています。

- Amazon CloudWatch は、 AWS リソースと で実行するアプリケーションを AWS リアルタイムでモニタリングします。メトリクスの収集と追跡、カスタマイズしたダッシュボードの作成、および指定したメトリクスが指定したしきい値に達したときに通知またはアクションを実行するアラームの設定を行うことができます。例えば、CloudWatch で Amazon EC2 インスタンスの CPU 使用率などのメトリクスを追跡し、必要に応じて新しいインスタンスを自動的に起動できます。詳細については、「Amazon CloudWatch ユーザーガイド」を参照してください。
- AWS CloudTrail は、 AWS アカウントによって、またはアカウントに代わって行われた API コールおよび関連イベントをキャプチャし、指定した Amazon S3 バケットにログファイルを配信します。が呼び出したユーザーとアカウント AWS、呼び出し元のソース IP アドレス、呼び出しの発生日時を特定できます。詳細については、AWS CloudTrail ユーザーガイドをご参照ください。
- Amazon CloudWatch Logs では、Amazon EC2 インスタンス、CloudTrail、およびその他のソースからのログファイルをモニタリング、保存、およびアクセスできます。CloudWatch Logs は、ログファイル内の情報をモニタリングし、特定のしきい値が満たされたときに通知します。高い耐久性を備えたストレージにログデータをアーカイブすることも可能です。詳細については、「Amazon CloudWatch Logs ユーザーガイド」を参照してください。
- Amazon EventBridge を使用して AWS サービスを自動化し、アプリケーションの可用性の問題やリソースの変更などのシステムイベントに自動的に対応できます。 AWS サービスからのイベントは、ほぼリアルタイムで EventBridge に配信されます。簡単なルールを記述して、注目するイベントと、イベントがルールに一致した場合に自動的に実行するアクションを指定できます。詳細については、「Amazon EventBridge ユーザーガイド」を参照してください。

トピック

- を使用した AWS AppConfig API コールのログ記録 AWS CloudTrail
- AWS AppConfig データプレーン呼び出しのメトリクスのログ記録
- 自動ロールバックのためのデプロイのモニタリング

を使用した AWS AppConfig API コールのログ記録 AWS CloudTrail

AWS AppConfig は、ユーザーAWS CloudTrail、ロール、または によって実行されたアクションを記録するサービスである と統合されています AWS のサービス。CloudTrail は、 のすべての API コールをイベント AWS AppConfig としてキャプチャします。キャプチャされた呼び出しには、 AWS AppConfig コンソールからの呼び出しと AWS AppConfig API オペレーションへのコード呼び出しが含まれます。CloudTrail によって収集された情報を使用して、リクエストの実行元の IP アドレス AWS AppConfig、リクエストの実行日時などの詳細を確認できます。

各イベントまたはログエントリには、誰がリクエストを生成したかという情報が含まれます。アイデンティティ情報は、以下を判別するのに役立ちます。

- ルートユーザーまたはユーザー認証情報のどちらを使用してリクエストが送信されたか。
- リクエストが IAM Identity Center ユーザーに代わって行われたかどうか。
- リクエストがロールまたはフェデレーションユーザーのテンポラリなセキュリティ認証情報を使用して行われたかどうか。
- リクエストが、別の AWS のサービスによって送信されたかどうか。

CloudTrail は、アカウントを作成する AWS アカウント と でアクティブになり、CloudTrail イベント履歴に自動的にアクセスできます。CloudTrail の [イベント履歴] では、 AWS リージョンで過去 90 日間に記録された 管理イベントの表示、検索、およびダウンロードが可能で、変更不可能な記録を確認できます。詳細については、「AWS CloudTrail ユーザーガイド」の「CloudTrail イベント履歴の使用」を参照してください。[イベント履歴] の閲覧には CloudTrail の料金はかかりません。

AWS アカウント 過去 90 日間のイベントの継続的な記録については、証跡または CloudTrail Lake イベントデータストアを作成します。

CloudTrail 証跡

追跡により、CloudTrail はログファイルを Amazon S3 バケットに配信できます。を使用して作成された証跡はすべてマルチリージョン AWS Management Console です。 AWS CLIを使用する際は、単一リージョンまたは複数リージョンの証跡を作成できます。アカウント AWS リージョン 内のすべての でアクティビティをキャプチャするため、マルチリージョン証跡を作成することをお勧めします。単一リージョンの証跡を作成する場合、証跡の AWS リージョンに記録されたイベントのみを表示できます。証跡の詳細については、「AWS CloudTrail ユーザーガイド」の「AWS アカウントの証跡の作成」および「組織の証跡の作成」を参照してください。

証跡を作成すると、進行中の管理イベントのコピーを 1 つ無料で CloudTrail から Amazon S3 バケットに配信できますが、Amazon S3 ストレージには料金がかかります。CloudTrail の料金の詳細については、「AWS CloudTrail の料金」を参照してください。Amazon S3 の料金に関する詳細については、「Amazon S3 の料金」を参照してください。

CloudTrail Lake イベントデータストア

[CloudTrail Lake] を使用すると、イベントに対して SQL ベースのクエリを実行できます。CloudTrail Lake は、行ベースの JSON 形式の既存のイベントを Apache ORC 形式に変換します。ORC は、データを高速に取得するために最適化された単票ストレージ形式です。イベントは、イベントデータストアに集約されます。イベントデータストアは、高度なイベントセレクタを適用することによって選択する条件に基づいた、イベントのイミュータブルなコレクションです。どのイベントが存続し、クエリに使用できるかは、イベントデータストアに適用するセレクタが制御します。CloudTrail Lake の詳細については、 AWS CloudTrail ユーザーガイドのAWS CloudTrail 「Lake の使用」を参照してください。

CloudTrail Lake のイベントデータストアとクエリにはコストがかかります。イベントデータストアを作成する際に、イベントデータストアに使用する料金オプションを選択します。料金オプションによって、イベントの取り込みと保存にかかる料金、および、そのイベントデータストアのデフォルトと最長の保持期間が決まります。CloudTrail の料金の詳細については、「AWS CloudTrail の料金」を参照してください。

AWS AppConfig CloudTrail のデータイベント

データイベントは、リソース上でまたはリソース内で実行されたリソースオペレーションに関する情報を提供します (例えば、GetLatestConfiguration を呼び出して最新のデプロイされた設定を取得するなど)。これらのイベントは、データプレーンオペレーションとも呼ばれます。データイベントは、多くの場合、高ボリュームのアクティビティです。デフォルトでは、CloudTrail はデータイベントをログ記録しません。CloudTrail [イベント履歴] にはデータイベントは記録されません。

追加の変更がイベントデータに適用されます。CloudTrail の料金の詳細については、「AWS CloudTrail の料金」を参照してください。

CloudTrail コンソール、または CloudTrail CloudTrail API オペレーションを使用して AWS CLI、AWS AppConfig リソースタイプのデータイベントをログに記録できます。このセクションの表は、使用可能なリソースタイプを示しています AWS AppConfig。

- CloudTrail コンソールを使用してデータイベントを記録するには、データイベントをログに記録する証跡またはイベントデータストアを作成するか、既存の証跡またはイベントデータストアを更新してデータイベントをログに記録します。

  1. データイベントをログに記録するには、[データイベント] を選択します。
  2. [データイベントタイプ] リストから、[AWS AppConfig] を選択します。
  3. 使用するログセレクタテンプレートを選択します。リソースタイプのすべてのデータイベントをログに記録したり、すべての `readOnly` イベントをログに記録したり、すべての `writeOnly` イベントをログに記録したり、カスタムログセレクタテンプレートを作成して `readOnly`、`eventName`、`resources.ARN` フィールドでフィルタリングしたりできます。
  4. [セレクタ名] に AppConfigDataEvents と入力します。データイベント証跡で Amazon CloudWatch Logs を有効にする方法については、「AWS AppConfig データプレーン呼び出しのメトリクスのログ記録」を参照してください。

- を使用してデータイベントをログに記録するには AWS CLI、 `--advanced-event-selectors` パラメータを設定して、 `eventCategory`フィールドを に、 `Data resources.type`フィールドをリソースタイプ値に等しくします (表を参照) 。条件を追加して、`readOnly`、`eventName` および `resources.ARN` フィールドの値でフィルタリングできます。

  - データイベントをログに記録するように証跡を設定するには、put-event-selectors コマンドを実行します。詳細については、「AWS CLIを使用した証跡へのデータイベントのログ記録」を参照してください。
  - データイベントをログ記録するようにイベントデータストアを設定するには、create-event-data-store コマンドを実行してデータイベントをログ記録する新しいイベントデータストアを作成するか、update-event-data-store コマンドを実行して既存のイベントデータストアを更新します。詳細については、「AWS CLIを使用したイベントデータストアへのデータイベントのログ記録」を参照してください。

次の表に、 AWS AppConfig リソースタイプを示します。データイベントタイプ (コンソール) 列には、CloudTrail コンソールの [データイベントタイプ] リストから選択する値が表示されます。resources.type 値列には、 AWS CLI または CloudTrail APIs を使用して高度なイベントセレクタを設定するときに指定する `resources.type`値が表示されます。CloudTrail に記録されたデータ API 列には、リソース タイプの CloudTrail にログ記録された API コールが表示されます。

データイベントタイプ (コンソール)	resources.type 値	CloudTrail* にログ記録されたデータ API
AWS AppConfig	`AWS::AppConfig::Configuration`	• GetLatestConfiguration • StartConfigurationSession

*`eventName`、`readOnly`、および `resources.ARN` フィールドでフィルタリングして、自分にとって重要なイベントのみをログに記録するように高度なイベントセレクタを設定できます。フィールドの詳細については、「AdvancedFieldSelector」を参照してください。

AWS AppConfig CloudTrail の管理イベント

AWS AppConfig は、すべての AWS AppConfig コントロールプレーンオペレーションを管理イベントとしてログに記録します。CloudTrail に AWS AppConfig ログ記録する AWS AppConfig コントロールプレーンオペレーションのリストについては、 AWS AppConfig API リファレンスを参照してください。

AWS AppConfig イベントの例

各イベントは任意の送信元からの単一のリクエストを表し、リクエストされた API オペレーション、オペレーションの日時、リクエストパラメータなどに関する情報を含みます。CloudTrail ログファイルは、パブリック API コールの順序付けられたスタックトレースではないため、イベントは特定の順序で表示されません。

次の例は、StartConfigurationSession オペレーションを示す CloudTrail イベントを示しています。

```
{
      "eventVersion": "1.09",
      "userIdentity": {
        "type": "IAMUser",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/Administrator",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
          "sessionIssuer": {},
          "attributes": {
            "creationDate": "2024-01-11T14:37:02Z",
```

```
            "mfaAuthenticated": "false"
          }
        }
      },
      "eventTime": "2024-01-11T14:45:15Z",
      "eventSource": "appconfig.amazonaws.com",
      "eventName": "StartConfigurationSession",
      "awsRegion": "us-east-1",
      "sourceIPAddress": "203.0.113.0",
      "userAgent": "Boto3/1.34.11 md/Botocore#1.34.11 ua/2.0 os/macos#22.6.0
 md/arch#x86_64 lang/python#3.11.4 md/pyimpl#CPython cfg/retry-mode#legacy
 Botocore/1.34.11",
      "requestParameters": {
        "applicationIdentifier": "rrfexample",
        "environmentIdentifier": "mexampleqe0",
        "configurationProfileIdentifier": "3eexampleu1"
      },
      "responseElements": null,
      "requestID": "a1b2c3d4-5678-90ab-cdef-aaaaaEXAMPLE",
      "eventID": "a1b2c3d4-5678-90ab-cdef-bbbbbEXAMPLE",
      "readOnly": false,
      "resources": [
        {
          "accountId": "123456789012",
          "type": "AWS::AppConfig::Configuration",
          "ARN": "arn:aws:appconfig:us-east-1:123456789012:application/rrfexample/
environment/mexampleqe0/configuration/3eexampleu1"
        }
      ],
      "eventType": "AwsApiCall",
      "managementEvent": false,
      "recipientAccountId": "123456789012",
      "eventCategory": "Data",
      "tlsDetails": {
        "tlsVersion": "TLSv1.3",
        "cipherSuite": "TLS_AES_128_GCM_SHA256",
        "clientProvidedHostHeader": "appconfigdata.us-east-1.amazonaws.com"
      }
    }
```


CloudTrail レコードの内容については、「AWS CloudTrail ユーザーガイド」の「CloudTrail record contents」を参照してください。

AWS AppConfig データプレーン呼び出しのメトリクスのログ記録

AWS AppConfig データイベントをログ AWS CloudTrail に記録するように を設定した場合は、Amazon CloudWatch Logs を有効にして、 AWS AppConfig データプレーンへの呼び出しのメトリクスを記録できます。その後、1 つまたは複数のメトリクスフィルターを作成して、CloudWatch Logs でログデータを検索およびフィルタリングできます。メトリクスフィルタは CloudWatch Logs に送信されたログデータを検索するための語句とパターンを定義します。CloudWatch Logs は、これらのメトリクスフィルターを使用して、ログデータを数値の CloudWatch メトリクスに変換します。メトリクスをグラフ化したり、アラームを設定したりできます。

[開始する前に]

で AWS AppConfig データイベントのログ記録を有効にします AWS CloudTrail。次の手順では、CloudTrail で既存の AWS AppConfig 証跡のメトリクスログ記録を有効にする方法について説明します。 AWS AppConfig データプラン呼び出しで CloudTrail ログ記録を有効にする方法については、「」を参照してください[AWS AppConfig CloudTrail のデータイベント](#)。

次の手順を使用して、CloudWatch Logs が AWS AppConfig データプレーンへの呼び出しのメトリクスを記録できるようにします。

CloudWatch Logs が AWS AppConfig データプレーンへの呼び出しのメトリクスを記録できるようにするには

1. CloudTrail コンソールの https://console.aws.amazon.com/cloudtrail/ を開いてください。
2. ダッシュボードで、 AWS AppConfig 証跡を選択します。
3. [CloudWatch ログ] セクションで [編集] を選択します。
4. [有効] を選択します。
5. [ロググループ名] については、デフォルト名のままにするか、名前を入力します。名前をメモします。後で CloudWatch Logs コンソールでロググループを選択します。
6. [Role name] (ロール名) に名前を入力します。
7. [Save changes] (変更の保存) をクリックします。

次の手順を使用して、CloudWatch Logs AWS AppConfig で のメトリクスとメトリクスフィルターを作成します。この手順では、`operation` による呼び出しと、(オプションで) `operation` と `Amazon Resource Name (ARN)` による呼び出しのメトリクスフィルターを作成する方法について説明します。

CloudWatch Logs AWS AppConfig で のメトリクスとメトリクスフィルターを作成するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、ログ、ロググループ の順に選択します。
3. AWS AppConfig ロググループの横にあるチェックボックスをオンにします。
4. [アクション]、[メトリクスフィルターの作成] の順に選択します。
5. [フィルター名前] に名前を入力します。
6. [フィルターパターン] で、次のように入力します。

```
{ $.eventSource = "appconfig.amazonaws.com" }
```

7. (オプション) [パターンをテスト] セクションで、[テストするログデータを選択] リストからロググループを選択します。CloudTrail が呼び出しを記録していない場合は、このステップをスキップできます。
8. [Next (次へ)] を選択します。
9. [Metric namespace] (メトリクス名前空間) に **AWS AppConfig** と入力します。
10. [Metric name] (メトリクス名) に、「**Calls**」を入力します。
11. [Metric value] (メトリクス値) に **1** と入力します。
12. [デフォルト値] と [単位] をスキップします。
13. [ディメンション名] には、**operation** と入力します。
14. [ディメンション値] には、**$.eventName** と入力します。

    (オプション) 呼び出しを行う Amazon リソースネーム (ARN) を含む 2 番目のディメンションを入力できます。2 番目のディメンションを追加するには、[ディメンション名] に **resource** と入力します。[ディメンション値] には、**$.resources[0].ARN** と入力します。

    [Next (次へ)] を選択します。
15. フィルターの詳細を確認し、[メトリクスフィルターを作成] を選択します。

(オプション) この手順を繰り返して、AccessDenied などの特定のエラーコードの新しいメトリクスフィルターを作成できます。その場合は、次の詳細を入力します。

1. [フィルター名前] に名前を入力します。
2. [フィルターパターン] で、次のように入力します。

```
{ $.errorCode = "codename" }
```

例

```
{ $.errorCode = "AccessDenied" }
```

3. [Metric namespace] (メトリクス名前空間) に **AWS AppConfig** と入力します。
4. [Metric name] (メトリクス名) に、「**Errors**」を入力します。
5. [Metric value] (メトリクス値) に **1** と入力します。
6. [デフォルト値] には、ゼロ (0) を入力します。
7. [単位]、[ディメンション]、[アラーム] をスキップします。

CloudTrail が API コールをログに記録すると、CloudWatch でメトリクスを表示できます。詳細については、「Amazon CloudWatch ユーザーガイド」の「コンソールでのメトリクスおよびログの表示」を参照してください。作成したメトリクスを検索する方法については、「使用可能なメトリクスを検索する」を参照してください。

> **Note**
>
> ここで説明するように、ディメンションなしでエラーメトリクスを設定した場合、[ディメンションの定められていないメトリクス] ページでこれらのメトリクスを表示できます。

CloudWatch メトリクスのアラームを作成します

メトリクスを作成したら、CloudWatch でメトリクスアラームを作成できます。例えば、前の手順で作成した AWS AppConfig 呼び出しメトリクスのアラームを作成できます。具体的には、しきい値を超える `StartConfigurationSession` API アクションへの AWS AppConfig 呼び出しのアラームを作成できます。メトリクスのアラームを作成する方法については、「Amazon CloudWatch ユーザーガイド」の「静的しきい値に基づいて CloudWatch アラームを作成する」を参照してください。AWS AppConfig データプレーンへの呼び出しのデフォルト制限の詳細については、「」の「データプレーンのデフォルト制限」を参照してくださいAmazon Web Services 全般のリファレンス。

自動ロールバックのためのデプロイのモニタリング

デプロイ中、Amazon CloudWatch アラームに基づく AWS AppConfig デプロイ戦略と自動ロールバックの組み合わせを使用することで、誤った形式または誤った設定データによってアプリケーションにエラーが発生する状況を軽減できます。設定すると、デプロイ中に 1 つ以上の CloudWatch アラームが `ALARM`または `INSUFFICIENT_DATA`状態になると、 AWS AppConfig は自動的に設定データを以前のバージョンにロールバックするため、アプリケーションの停止やエラーを防ぐことができます。また、デプロイが進行中に StopDeployment API オペレーションを呼び出すことで、設定をロールバックすることもできます。

> **Important**
>
> 正常に完了したデプロイの場合、 は、StopDeployment API オペレーションで `AllowRevert`パラメータを使用して設定データを以前のバージョンに戻す AWS AppConfig こともサポートします。一部のお客様では、デプロイが成功した後に以前の設定に戻すと、データはデプロイ前と同じであることが保証されます。また、元に戻すとアラームモニターも無視され、アプリケーションの緊急事態中にロールフォワードが進行するのを防ぐことができます。詳細については、「設定の復元」を参照してください。

自動ロールバックを設定するには、 AWS AppConfig 環境を作成 (または編集) するときに [CloudWatch アラーム] フィールドに 1 つ以上の CloudWatch メトリクスの Amazon リソースネーム (ARN) を指定します。詳細については、「AWS AppConfigでのアプリケーション環境を作成します」を参照してください。

> **Note**
>
> サードパーティーのモニタリングソリューション (Datadog など) を使用している場合は、`AT_DEPLOYMENT_TICK`アクションポイントでアラームをチェックする AWS AppConfig 拡張機能を作成し、アラームをトリガーした場合、安全ガードレールとしてデプロイをロールバックできます。拡張機能の詳細については AWS AppConfig 、「」を参照してください拡張機能を使用した AWS AppConfig ワークフローの拡張。カスタム拡張機能の詳細については、「」を参照してくださいチュートリアル: カスタム AWS AppConfig 拡張機能の作成。`AT_DEPLOYMENT_TICK` アクションポイントを使用して Datadog と統合する AWS AppConfig 拡張機能のコードサンプルを表示するには、GitHub の「aws-samples / aws-appconfig-tick-extn-for-datadog」を参照してください。

自動ロールバックをモニタリングするための推奨メトリクス

モニタリングするメトリクスは、アプリケーションで使用されるハードウェアとソフトウェアによって異なります。 の AWS AppConfig お客様は、多くの場合、次のメトリクスをモニタリングします。でグループ化された推奨メトリクスの完全なリストについては AWS のサービス、Amazon CloudWatch ユーザーガイド」の「推奨アラーム」を参照してください。

モニタリングするメトリクスを特定したら、CloudWatch を使用してアラームを設定します。詳細については、「Using Amazon CloudWatch alarms」(Amazon CloudWatch アラームを使用する) を参照してください。

サービス	メトリクス	詳細
Amazon API Gateway	4XXError	このアラームは、クライアント側エラーの発生率が高いことを検出します。これは、認可パラメータまたはクライアントリクエストパラメータに問題があることを示している可能性があります。また、リソースが削除されたか、存在しないリソースをクライアントがリクエストしている可能性もあります。Amazon CloudWatch Logs を有効にして、4XX エラーの原因となっている可能性のあるエラーがないか確認することを検討してください。さらに、詳細な CloudWatch メトリクスを有効にして、リソースやメソッドごとにこのメトリクスを表示し、エラーの原因を絞り込むことを検討してください。また、設定したスロットリング制限を超えたためにエラーが発生することもあります。

サービス	メトリクス	詳細
Amazon API Gateway	5XXError	このアラームは、サーバー側エラーの発生率が高いことを検出するのに役立ちます。これは、API バックエンド、ネットワーク、または API ゲートウェイとバックエンド API の統合に問題があることを示している可能性があります。
Amazon API Gateway	レイテンシー	このアラームは、ステージ内のレイテンシーが大きいことを検出します。`IntegrationLatency` メトリクス値を調べて API バックエンドのレイテンシーをチェックします。2 つのメトリクスがほぼ一致している場合、API バックエンドがレイテンシー増加の原因となるため、問題がないか調査する必要があります。CloudWatch Logs を有効にして、レイテンシーが大きいことの原因となっている可能性のあるエラーをチェックすることも検討してください。

サービス	メトリクス	詳細
Amazon EC2 Auto Scaling	GroupInServiceCapacity	このアラームは、グループ内の容量がワークロードに必要な希望容量を下回ったことを検出するのに役立ちます。トラブルシューティングを行うには、スケーリングアクティビティの起動に障害がないかどうかをチェックし、必要な容量設定が正しいことを確認します。
Amazon EC2	CPUUtilization	このアラームは EC2 インスタンスの CPU 使用率をモニタリングするのに役立ちます。アプリケーションによっては、使用率が定常的に高いことが普通かもしれません。しかし、パフォーマンスが低下し、アプリケーションがディスク I/O、メモリ、ネットワークリソースの制約を受けていない場合、CPU が上限に達しているということは、リソースのボトルネックやアプリケーションのパフォーマンス上の問題を示している可能性があります。
Amazon ECS	CPUReservation	このアラームは、ECS クラスターの CPU 予約率が高いことを検出するのに役立ちます。CPU 予約率が高い場合は、そのタスク用に登録されている CPU がクラスターに不足している可能性があります。

サービス	メトリクス	詳細
Amazon ECS	HTTPCode_Target_5XX_Count	このアラームは、ECS サービスのサーバー側エラー数が多いことを検出するのに役立ちます。これは、サーバーがリクエストを処理できない原因となるエラーがあることを示している可能性があります。
Container Insights を使用する Amazon EKS	node_cpu_utilization	このアラームは、Amazon EKS クラスターのワーカーノードの CPU 使用率が高いことを検出するのに役立ちます。使用率が定常的に高い場合は、ワーカーノードを CPU の大きいインスタンスに置き換えるか、システムを水平的にスケールする必要があることを示している可能性があります。
Container Insights を使用する Amazon EKS	node_memory_utilization	このアラームは、Amazon EKS クラスターのワーカーノードのメモリ使用率が高いことを検出するのに役立ちます。使用率が定常的に高い場合は、ポッドレプリカの数をスケールするか、アプリケーションを最適化する必要があることを示している可能性があります。

サービス	メトリクス	詳細
Container Insights を使用する Amazon EKS	pod_cpu_utilization_over_pod_limit	このアラームは、Amazon EKS クラスターのポッドの CPU 使用率が高いことを検出するのに役立ちます。使用率が定常的に高い場合は、影響を受けるポッドの CPU 上限を増やす必要があることを示している可能性があります。
Container Insights を使用する Amazon EKS	pod_memory_utilization_over_pod_limit	このアラームは、Amazon EKS クラスターのポッドの CPU 使用率が高いことを検出するのに役立ちます。使用率が定常的に高い場合は、影響を受けるポッドの CPU 上限を増やす必要があることを示している可能性があります。
AWS Lambda	エラー	このアラームは、エラー数が多いことを検出します。エラーには、コードによってスローされた例外と、Lambda ランタイムによってスローされた例外が含まれます。
AWS Lambda	Throttles	このアラームは、スロットリングされた呼び出しリクエストが多いことを検出します。スロットリングは、スケールアップに同時実行が利用できない場合に発生します。

サービス	メトリクス	詳細
Lambda Insights	memory_utilization	このアラームは、Lambda 関数のメモリ使用率が設定された上限に近づいているかどうかを検出するために使用されます。
Amazon S3	4xxErrors	このアラームは、クライアントのリクエストに応答して生成されたエラーステータスコード 4xx の総数を報告するのに役立ちます。例えば、エラーコード 403 は IAM ポリシーが正しくないことを示している可能性があり、エラーコード 404 はクライアントアプリケーションの動作が正しくないことを示している可能性があります。
Amazon S3	5xxErrors	このアラームは、サーバー側エラー数が多いことを検出するのに役立ちます。これらのエラーは、クライアントがリクエストを行ったが、サーバーが完了できなかったことを示しています。これは、S3 が原因でアプリケーションが直面している問題を関連付けることに役立ちます。

AWS AppConfig ユーザーガイドのドキュメント履歴

次の表は、 の前回のリリース以降のドキュメントの重要な変更点を示しています AWS AppConfig。

現在の API バージョン: 2019-10-09

変更	説明	日付
IPv6 サポート	AWS AppConfig APIs IPv4 および IPv6 呼び出しを完全にサポートするようになりました。詳細については、IPv6 サポートについて」を参照してください。	2025 年 4 月 23 日
新しいトピック: 以前の機能フラグバージョンを新しいバージョンに保存する	機能フラグを更新すると、 は変更を新しいバージョン AWS AppConfig に自動的に保存します。以前の機能フラグバージョンを使用する場合は、それをドラフトバージョンにコピーしてから保存する必要があります。新しいバージョンに保存しないと、以前のフラグバージョンへの変更を編集して保存することはできません。詳細については、「以前の機能フラグバージョンを新しいバージョンに保存」を参照してください。	2025 年 4 月 15 日
新しいトピック: AWS AppConfig エージェントローカル開発モードの機能フラグのサンプル	AWS AppConfig エージェントはローカル開発モードをサポートしています。ローカル開発モードを有効にすると、エージェントはディスク上の指定されたディレクトリから	2025 年 2 月 18 日

	設定データを読み取ります。から設定データを取得しません AWS AppConfig。ローカル開発モードの使用方法をよりよく理解できるように、このガイドには機能フラグのサンプルを含むトピックが含まれるようになりました。詳細については、AWS AppConfig「エージェントローカル開発モードの機能フラグのサンプル」を参照してください。	
新しいトピック: 非ネイティブデータソースの設定プロファイルの作成	このトピックでは、 拡張機能を使用して AWS AppConfig 、Amazon RDS や Amazon DynamoDB などの他の AWS サービスや、Git Hub、GitLab、ローカルリポジトリなどのサードパーティーソースなど、ネイティブにサポートされていないソースから設定データを取得するための大まかなプロセスについて説明します。詳細については、「非ネイティブデータソースの設定プロファイルの作成」を参照してください。	2024 年 12 月 19 日

更新されたトピック: 機能フラグタイプのリファレンスの正規表現を修正しました	機能フラグタイプのリファレンスの JSON スキーマは、以前はさまざまな場所で次の正規表現パターンを示していました: `"^[a-z][a-zA-Z\\d-_]{0,63}$"` 。正しい正規表現パターンは です`"^[a-z][a-zA-Z\\d_-]{0,63}$"` 。ハイフンはアンダースコアの後に表示されます。詳細については、「AWS.AppConfig.FeatureFlags の型リファレンスを理解する」を参照してください。	2024 年 12 月 18 日
更新されたトピック: 環境変数のサンプルを追加	次のトピックの環境変数を説明する表が更新され、サンプルが含まれました。 • (オプション) 環境変数を使用して Amazon ECS および Amazon EKS の AWS AppConfig エージェントを設定する • (オプション) 環境変数を使用して Amazon EC2 の AWS AppConfig エージェントを設定する • AWS AppConfig エージェント Lambda 拡張機能の設定	2024 年 12 月 12 日

新しいセクション: 分割演算子について	新しいセクションでは、例を使用して、split演算子がマルチバリアント機能フラグルールに対してどのように機能するかを説明します。詳細については、「マルチバリアント機能フラグルールについて」を参照してください。	2024 年 11 月 22 日

新しい拡張アクションポイント: AT_DEPLOYMENT_TICK	AWS AppConfig は、カスタム拡張機能を作成するユーザー向けに新しいアクションポイントを開始しました。AT_DEPLOYMENT_TICK アクションポイントは、サードパーティーのモニタリング統合をサポートします。 AT_DEPLOYMENT_TICK は、設定デプロイ処理オーケストレーション中に呼び出されます。サードパーティーのモニタリングソリューション (Datadog など) を使用する場合は、AT_DEPLOYMENT_TICK アクションポイントでアラームをチェックする AWS AppConfig 拡張機能を作成し、アラームをトリガーした場合、安全ガードレールとしてデプロイをロールバックできます。拡張機能の詳細については AWS AppConfig 、「拡張機能を使用した AWS AppConfig ワークフローの拡張」を参照してください。カスタム拡張機能の詳細については、「チュートリアル: カスタム AWS AppConfig 拡張機能の作成」を参照してください。AT_DEPLOYMENT_TICK アクションポイントを使用して Datadog と統合する AWS AppConfig 拡張機能のコードサンプルを表示する	2024 年 11 月 22 日

	には、GitHub の「aws-samples / aws-appconfig-tick-extn-for-datadog」を参照してください。	
新しいトピック： AWS AppConfig モバイル使用に関する考慮事項	このガイドの新しいトピックでは、モバイルデバイスで AWS AppConfig 機能フラグを使用する際の重要な考慮事項について説明します。詳細については、AWS AppConfig「モバイル使用に関する考慮事項」を参照してください。	2024 年 11 月 21 日
新機能: AWS AppConfig 削除保護	AWS AppConfig では、アクティブに使用されている環境と設定プロファイルをユーザーが意図せずに削除しないようにするアカウント設定が提供されるようになりました。詳細については、AWS AppConfig 「削除保護の設定」を参照してください。	2024 年 8 月 28 日
AWS AppConfig エージェント Lambda 拡張機能の新しいバージョン	エージェントは、マイナーな機能強化とバグ修正で更新されました。拡張機能の新しい Amazon リソースネーム (ARNs」を参照してください。 AWS AppConfig	2024 年 8 月 9 日
フラグバリアントを取得するための新しいコードサンプル	詳細については、AWS AppConfig 「 エージェントを使用してバリアントを含む機能フラグを取得する」を参照してください。	2024 年 8 月 9 日

AWS AppConfig エージェント Lambda 拡張機能の新しいバージョン	エージェントが更新され、機能フラグのターゲット、バリアント、分割がサポートされました。拡張機能の新しい Amazon リソースネーム (ARNs」を参照してください。 AWS AppConfig	2024 年 7 月 23 日
新機能: マルチバリアント機能フラグ	マルチバリアント機能フラグを使用すると、リクエストに対して返す可能性のあるフラグ値のセットを定義できます。マルチバリアントフラグには、異なるステータス (有効または無効) を設定することもできます。バリアントで設定されたフラグをリクエストする場合、アプリケーションは、 AWS AppConfig がユーザー定義のルールセットに対して評価するコンテキストを提供します。リクエストで指定されたコンテキストとバリアントに定義されたルールに応じて、 は異なるフラグ値をアプリケーションに AWS AppConfig 返します。詳細については、「マルチバリアント機能フラグの作成」を参照してください。	2024 年 7 月 23 日

AWS AppConfig エージェント Lambda 拡張機能の新しいバージョン	エージェントは、マイナーな機能強化とバグ修正で更新されました。拡張機能の新しい Amazon リソースネーム (ARNs」を参照してください。 AWS AppConfig	2024 年 2 月 1 日
AWS AppConfig カスタム拡張機能のサンプル	「チュートリアル: カスタム AWS AppConfig 拡張機能の作成」のトピックに、GitHub の以下のサンプル拡張機能へのリンクが含まれるようになりました。 • Systems Manager Change Calendar を使用して `blocked day` モラトリアムカレンダーによるデプロイを防止するサンプル拡張機能 • git-secrets を使用してシークレットが設定データに漏洩するのを防ぐサンプル拡張機能 • Amazon Comprehend を使用して、個人を特定できる情報 (PII) が設定データに漏洩するのを防ぐサンプル拡張機能	2024 年 2 月 1 日

新しいトピック: を使用した AWS AppConfig API コールのログ記録 AWS CloudTrail	AWS AppConfig は、ユーザー AWS CloudTrail、ロール、または のサービスによって実行されたアクションを記録する AWS サービスである と統合されています AWS AppConfig。CloudTrail は、AWS AppConfig のすべての API コールをイベントとしてキャプチャします。この新しいトピックでは、 AWS Systems Manager ユーザーガイドの対応するコンテンツにリンクするのではなく、AWS AppConfig特定のコンテンツを提供します。詳細については、「 を使用した AWS AppConfig API コールのログ記録 AWS CloudTrail」を参照してください。	2024 年 1 月 18 日

AWS AppConfig が をサポートするようになりました AWS PrivateLink	を使用して AWS PrivateLink 、VPC と の間にプライベート接続を作成できます AWS AppConfig。インターネットゲートウェイ、NAT デバイス、VPN 接続、または AWS Direct Connect 接続を使用せずに、VPC 内にあるかのように にアクセスできます AWS AppConfig 。VPC のインスタンスは、パブリック IP アドレスがなくても AWS AppConfigにアクセスできます。詳細については、「インターフェイスエンドポイント (AWS PrivateLink) AWS AppConfig を使用したアクセス」を参照してください。	2023 年 12 月 6 日

AWS AppConfig エージェント取り出しの追加機能と新しいローカル開発モード	AWS AppConfig エージェントには、アプリケーションの設定を取得するのに役立つ以下の追加機能があります。 追加の取得機能 • マルチアカウントの取得: プライマリまたは取得から AWS AppConfig エージェント AWS アカウント を使用して、複数のベンダーアカウントから設定データを取得します。 • ディスクへの設定コピーの書き込み: AWS AppConfig エージェントを使用して設定データをディスクに書き込みます。この機能を使用すると、ディスクから設定データを読み取るアプリケーションを使用しているお客様は、 AWS AppConfig と統合できます。 Note ディスクへの書き込み設定は、設定バックアップ機能として設計されていません。AWS AppConfig エージェントは、ディスクにコピーされた設定ファイルから読み取りません。設定をディス	2023 年 12 月 1 日

クにバックアップする場合は、Amazon EC2 で AWS AppConfig エージェントを使用するか、Amazon ECS `BACKUP_DIRECTORY` と Amazon EKS で エージェントを使用するための および `PRELOAD_BACKUP` 環境変数を参照してください。AWS AppConfig

ローカル開発モード

AWS AppConfig エージェントはローカル開発モードをサポートしています。ローカル開発モードを有効にすると、エージェントはディスク上の指定されたディレクトリから設定データを読み取ります。から設定データを取得しません AWS AppConfig。指定されたディレクトリ内のファイルを更新することで、設定のデプロイをシミュレートできます。次のユースケースでは、ローカル開発モードをお勧めします。

- AWS AppConfigを使用して、デプロイする前にさまざまな設定バージョンをテストする。

	• コードリポジトリに変更をコミットする前に、新機能のさまざまな設定オプションをテストする。 • さまざまな設定シナリオをテストして、期待どおりに動作することを確認する。	
新しいコードサンプルのトピック	このガイドに新しいコードサンプルトピックを追加しました。このトピックでは、6 つの一般的な AWS AppConfig アクションをプログラムで実行するための Java、Python、JavaScript の例が含まれています。	2023 年 11 月 17 日
「 AWS AppConfig ワークフローをより正確に反映するように目次を改訂しました」	このユーザーガイドの内容は、「ワークフローの作成、展開、取得、および拡張」という見出しの下にグループ化されています。この設定は、AWS AppConfig コンテンツを使用する際のワークフローをよりよく反映し、コンテンツを見つけやすくすることを目指しています。	2023 年 11 月 7 日
ペイロード参照が追加されました	「 AWS AppConfig カスタム拡張用の Lambda 関数の作成」トピックに、リクエストとレスポンスペイロードのリファレンスが含まれるようになりました。	2023 年 11 月 7 日

新しい AWS 事前定義されたデプロイ戦略	AWS AppConfig では、`AppConfig.Linear20PercentEvery6Minutes` 事前定義されたデプロイ戦略が提供され、推奨されるようになりました。詳細については、「定義済みのデプロイ戦略」を参照してください。	2023 年 8 月 11 日
AWS AppConfig Amazon EC2 との統合	AWS AppConfig エージェントを使用して、Amazon Elastic Compute Cloud (Amazon EC2) Linux インスタンスで実行されているアプリケーション AWS AppConfig と統合できます。このエージェントは Amazon EC2 の x86_64 および ARM64 アーキテクチャをサポートしています。詳細については、「AWS AppConfig と Amazon EC2 の統合」を参照してください。	2023 年 7 月 20 日

AWS CloudFormation 新しい AWS AppConfig リソースのサポートと機能フラグの例	AWS CloudFormation は、拡張機能の使用を開始する AWS AppConfig のに役立つ AWS::AppConfig::Extension リソースと AWS::AppConfig::ExtensionAssociation リソースをサポートするようになりました。 「AWS:: AppConfig:: ConfigurationProfile」「AWS:: AppConfig:: HostedConfigurationVersion」リソースに、 AWS AppConfig ホストされた設定ストアに機能フラグ設定プロファイルを作成する例が含まれるようになりました。	2023 年 4 月 12 日

AWS AppConfig との統合 AWS Secrets Manager	AWS AppConfig は と統合されます AWS Secrets Manager。 Secrets Manager を使用して、データベースやその他のサービスの認証情報を安全に暗号化、保存、取得できます。アプリケーション内で認証情報をハードコーディングする代わりに、必要なときにいつでも Secrets Manager を呼び出して認証情報を取得できます。Secrets Manager を使用すると、シークレットアクセスのローテーションと管理が可能になるため、IT リソースとデータへのアクセスを保護できます。 自由形式の設定プロファイルを作成する場合、Secrets Manager を設定データのソースとして選択できます。設定プロファイルを作成する前に、Secrets Manager をオンボーディングしてシークレットを作成する必要があります。Secrets Manager の詳細については、AWS Secrets Manager 「 ユーザーガイド」の「What is AWS Secrets Manager？」を参照してください。設定プロファイルの作成について詳しくは、「フリーフォーム設定プロファイルの作成」を参照してください。	2023 年 2 月 2 日

AWS AppConfig Amazon ECS および Amazon EKS との統合	AWS AppConfig エージェントを使用して、Amazon Elastic Container Service (Amazon ECS) および Amazon Elastic Kubernetes Service (Amazon EKS) AWS AppConfig と統合できます。エージェントは Amazon ECS および Amazon EKS コンテナアプリケーションと並行して実行されるサイドカーコンテナとして機能します。エージェントは、次の方法でコンテナ化されたアプリケーションの処理と管理を強化します。 • エージェントは、 AWS Identity and Access Management (IAM) ロールを使用して設定データのローカルキャッシュを管理することで、 AWS AppConfig ユーザーに代わって を呼び出します。ローカルキャッシュから設定データを引き出すことで、アプリケーションが設定データを管理するために必要となるコードの更新が少なくなり、設定データをミリ秒単位で取得でき、呼び出しを妨げるネットワークの問題による影響を受けなくなります。 • エージェントは、 AWS AppConfig 機能フラグを取	2022 年 12 月 2 日

得して解決するためのネイティブエクスペリエンスを提供します。

- すぐに使用できるエージェントで、キャッシュ戦略、ポーリング間隔、ローカル設定データの可用性に関するベストプラクティスを提供すると同時に、以降のサービスコールに必要な設定トークンを追跡します。
- バックグラウンドで実行中、エージェントは設定 AWS AppConfig データの更新のためにデータプレーンを定期的にポーリングします。コンテナ化されたアプリケーションは、ポート 2772 (カスタマイズ可能なデフォルトポート値) で localhost に接続し、HTTP GET を呼び出してデータを取得できます。
- AWS AppConfig エージェントは、コンテナを再起動またはリサイクルすることなく、コンテナの設定データを更新します。

詳細については、「Amazon ECS と Amazon EKSとAWS AppConfig の統合」を参照してください。

CloudWatch Evidently の新しい拡張機能： AWS AppConfig 拡張機能	Amazon CloudWatch Evidently を使用すると、機能のロールアウト中に、新しい機能を指定した割合のユーザーに提供することによって安全に検証できます。新機能のパフォーマンスをモニターリングすると、ユーザーへのトラフィックを増やしていくタイミングを決定するのに役立ちます。こうすれば、機能を完全に起動する前に、リスクを軽減し、意図しない結果を明確化できます。また、A/B 実験を実行して、証拠とデータに基づいて機能の設計を決定することもできます。 CloudWatch Evidently の AWS AppConfig 拡張機能を使用すると、アプリケーションは EvaluateFeature オペレーションを呼び出す代わりに、ローカルでユーザーセッションにバリエーションを割り当てることができます。ローカルセッションにより、API コールに伴うレイテンシーと可用性のリスクが軽減されます。エクステンションの設定方法および使用方法については、Amazon CloudWatch CloudWatch ユーザーガイドの「CloudWatch による起動と A/B 実験を明確に行う」を参照してください。	2022 年 9 月 13 日

GetConfiguration API アクションの非推奨	2021 年 11 月 18 日、 は新しいデータプレーンサービスを AWS AppConfig リリースしました。このサービスは、GetConfiguration API アクションを使用して設定データを取得する以前のプロセスに代わるものです。データプレーンサービスは、StartConfigurationSession と GetLatestConfiguration という 2 つの新しい API アクションを使用します。データプレーンサービスは新しいエンドポイントも使用します。 詳細については、AWS AppConfig 「データプレーンサービスについて」を参照してください。	2022 年 9 月 13 日
AWS AppConfig エージェント Lambda 拡張機能の新しいバージョン	AWS AppConfig エージェント Lambda 拡張機能のバージョン 2.0.122 が利用可能になりました。新しいエクステンションは、異なる Amazon Resource Name (ARN) を使用します。詳細については、「AWS AppConfig エージェント Lambda 拡張機能リリースノート」を参照してください。	2022 年 8 月 23 日

AWS AppConfig 拡張機能の起動	拡張機能は、設定を作成またはデプロイする AWS AppConfig ワークフローのさまざまな時点でロジックまたは動作を挿入する機能を強化します。 AWS認可された拡張機能を使用することも、独自の拡張機能を作成することもできます。詳細については、AWS AppConfig 「拡張機能の使用」を参照してください。	2022 年 7 月 12 日
AWS AppConfig エージェント Lambda 拡張機能の新しいバージョン	AWS AppConfig エージェント Lambda 拡張機能のバージョン 2.0.58 が利用可能になりました。新しいエクステンションは、異なる Amazon Resource Name (ARN) を使用します。詳細については、AWS AppConfig 「Lambda 拡張機能の利用可能なバージョン」を参照してください。	2022 年 5 月 3 日

AWS AppConfig Atlassian Jira との統合	を Atlassian Jira と統合すると AWS AppConfig 、 で指定された の機能フラグを変更するたびに、 は Atlassian コンソールで問題を作成および更新 AWS アカウント できます AWS リージョン。Jira の各課題には、フラグ名、アプリケーション ID、設定プロファイル ID、フラグ値が含まれます。フラグの変更を更新、保存、 およびデプロイすると、Jira は変更の詳細をもとに既存の問題を更新します。詳細については、「Atlassian Jira とAWS AppConfig の統合」 を参照してください。	2022 年 4 月 7 日

ARM64 (Graviton2) プロセッサーの機能フラグと Lambda 拡張機能サポートの一般提供開始します	AWS AppConfig 機能フラグを使用すると、新機能を開発し、ユーザーから機能を非表示にして本番環境にデプロイできます。まず、設定データ AWS AppConfig として フラグを に追加します。機能をリリースする準備ができたら、コードをデプロイせずにフラグ設定データを更新できます。この機能により、機能をリリースするために新しいコードをデプロイする必要がなくなるため、開発運用環境の安全性が向上します。詳細については、「設定および設定プロファイルの作成」を参照してください。 AWS AppConfig の機能フラグの一般提供には、以下の機能強化が含まれます。 • コンソールには、フラグを短期フラグとして指定するオプションが含まれています。短期フラグののリストをフィルタリングしたり、ソートしたりできます。 • AWS Lambdaで機能フラグを使用しているお客様の場合、新しい Lambda 拡張機能では、HTTP エンドポイントを使用して個々の機能フラグを呼び出すことができます。詳細については、「機能フラグ設定から	2022 年 3 月 15 日

	1 つ以上のフラグを取得する」を参照してください。	
	この更新では、ARM64 (Graviton2) プロセッサ用に開発された拡張機能 AWS Lambda もサポートされます。 詳細については、AWS AppConfig 「Lambda 拡張機能の利用可能なバージョン」を参照してください。	
GetConfiguration API アクションは廃止されました	`GetConfiguration` API アクションは廃止されました。設定データを受信する呼び出しでは、`StartConfigurationSession` と `GetLatestConfiguration` API を使用する必要があります これらの API とその使用方法の詳細については、「設定を取得する」を参照してください。	2022 年 1 月 28 日
AWS AppConfig Lambda 拡張機能の新しいリージョン ARN	AWS AppConfig Lambda 拡張機能は、新しいアジアパシフィック (大阪) リージョンで利用できます。Lambda をリージョンに作成するには、Amazon リソースネーム (ARN) が必要です。アジアパシフィック (大阪) リージョン ARN の詳細については、AWS AppConfig 「Lambda 拡張機能の追加」を参照してください。	2021 年 3 月 4 日

AWS AppConfig Lambda 拡張機能	AWS AppConfig を使用して Lambda 関数の設定を管理する場合は、 AWS AppConfig Lambda 拡張機能を追加することをお勧めします。この拡張機能には、コストを削減 AWS AppConfig しながら の使用を簡素化するベストプラクティスが含まれています。コスト削減は、 AWS AppConfig サービスへの API コールが少なくなり、Lambda 関数の処理時間が短くなることでコストが削減されます。詳細については、「AWS AppConfig と Lambda 拡張機能の統合」を参照してください。	2020 年 10 月 8 日
新規セクション	AWS AppConfigのセットアップの手順を提供する新しいセクションが追加されました。詳細については、「セットアップ AWS AppConfig」を参照してください。	2020 年 9 月 30 日
コマンドラインプロシージャの追加	このユーザーガイドの手順には、 AWS Command Line Interface （ AWS CLI) および Tools for Windows PowerShell のコマンドラインステップが含まれるようになりました。詳細については、「 の使用 AWS AppConfig」を参照してください。	2020 年 9 月 30 日

AWS AppConfig ユーザーガイドの起動	のツール AWS AppConfigである を使用して AWS Systems Manager、アプリケーション設定を作成、管理、迅速にデプロイします。 AWS AppConfig は、あらゆるサイズのアプリケーションへの制御されたデプロイをサポートし、組み込みの検証チェックとモニタリングが含まれています。EC2 インスタンス AWS AppConfig 、、 AWS Lambdaコンテナ、モバイルアプリケーション、または IoT デバイスでホストされているアプリケーションで を使用できます。	2020 年 7 月 31 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。