aws

ユーザーガイド

AWS AppConfig

AWS AppConfig: ユーザーガイド

Table of Contents

AWS AppConfig の概要

AWS AppConfig 機能フラグと動的構成により、ソフトウェア開発者はコードをフルデプロイしなくても本番環境でアプリケーションの動作を迅速かつ安全に調整できます。AWS AppConfig ソフトウェアのリリース頻度を短縮し、アプリケーションの耐障害性を向上させ、緊急の問題に迅速に対処できるようにします。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展開する前に、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。運用フラグと動的構成を使用すると、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 AWS Systems Manager の一機能です。

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、アプリケーション障害の原因となる可能性のある機能フラグを有効にしたり、設定データを更新することを避けるのに役立つ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は AWS AppConfig と統合して、アプリケーションの変更を監視します。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) と統合して、サービスへのきめ細かな役割ベースのアクセスを提供します。 AWS AppConfig はまた、暗号化と監査のために AWS Key Management Service (AWS KMS) と AWS CloudTrail 統合されています。外部の顧客にリリースされる前は、すべての AWS AppConfig 安全制御は当初、このサービスを大規模に利用する社内の顧客と共同で開発され、検証されていました。

AWS AppConfig ユースケース

アプリケーション構成の内容はアプリケーションごとに大きく異なる場合がありますが、お客様の幅広いニーズに対応する AWS AppConfig は以下のユースケースをサポートしています。

- 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。
- 一元化された構成ストレージ — 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。AWS AppConfig を使用して、AWS AppConfig ホストされた設定ストア、システム・マネージャパラメータストア、 AWS Secrets Manager 、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。

AWS AppConfig を使用する利点

AWS AppConfig は、組織に次のような利点を提供します。

- 顧客の予期しないダウンタイムを削減します

  AWS AppConfig を使用すると設定を検証するルールを作成できるため、アプリケーションのダウンタイムを短縮できます。有効でない設定はデプロイできません。AWS AppConfig には設定を検証するための以下の 2 つのオプションがあります。

  - 構文検証には、JSON スキーマを使用することができます。AWS AppConfig は JSON スキーマを使用して設定を検証し、設定の変更がアプリケーションの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では、AWS AppConfig が所有している AWS Lambda 関数を呼び出して、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、中央の場所から構成変更を展開することで、大規模なアプリケーションの管理を簡素化します。AWS AppConfigは、 AWS AppConfigホストされたコンフィグレーション・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に保存されたコンフィグレーションをサポートしています。AWS AppConfig は、EC2インスタンス、 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使用できます。

  ターゲットは、他のシステム・マネージャ機能に必要な システム・マネージャSSM エージェント または (IAM) インスタンスプロファイルを使用して設定する必要はありません。つまり、AWS AppConfig はアンマネージド型インスタンスで動作します。
- 中断することなくアプリケーションを更新する

  AWS AppConfig は、重いビルドプロセスを実行したりターゲットをサービスから外したりせずに、実行時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

  構成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。

AWS AppConfig の仕組み

このセクションでは、その仕組みと AWS AppConfig 使用方法の大まかな説明を提供します。

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、AWS AppConfig を使用して動的に管理したいコード内の設定データを特定することをお勧めします。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します

   アプリケーションの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する必要があります。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。環境とは、 `Beta` または `Production` 環境内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的グループです。`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、構成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、システムは構成をロールバックします。

4. 構成プロファイルの作成

   設定プロファイルにはURIとプロファイルタイプがあります。AWS AppConfig は自分の設定データとプロファイルタイプを割り当て、AWS AppConfig では、機能フラグとフリーフォーム構成の2種類の構成プロファイルがサポートされています。機能フラグ設定プロファイルは、AWS AppConfig ホストにされた設定ストアにデータを格納しますが、URI `hosted` はシンプルです。フリーフォームの設定プロファイルの場合は、AWS AppConfig で説明されているように、AWS AppConfig ホストされた設定ストアまたは AWS と統合される任意の [でのフリーフォーム設定プロファイルの作成 AWS AppConfig] サービスにデータを保存できます。

   設定プロファイルにオプションのバリデータを含めて、設定データが構文的にもセマンティック的にも正しいことを確認することもできます。AWS AppConfig は、デプロイの開始時にバリデータを使用してチェックを実行します。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

   新しいデプロイメントを作成する際には、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、以下の AWS AppConfig タスクが実行されます。

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セマンティック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をAWS AppConfigポーリングさせます。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ローカルホストサーバーに HTTP 呼び出しを行います。AWS AppConfig「簡略化された検索方法」で説明されているように、エージェントはさまざまなユースケースをサポートしています。

   ご使用のユースケースで AWS AppConfig エージェント がサポートされていない場合は、「StartConfigurationSession」 と 「GetLatestConfiguration」 API アクションを直接呼び出して、AWS AppConfigの設定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、AWS AppConfig を直接使用する場合に役立ちます。

「Amazon Web Services YouTube Channel で AWS の詳細のビデオをご覧ください。

以下のブログは、AWS AppConfig とその機能を詳しく説明しています。

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

SDK

AWS AppConfigの言語固有の SDK については、次の関連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

AWS AppConfig の料金

AWS AppConfig の料金は、設定データと機能フラグの取得に基づく従量制料金です。コストを最適化するために AWS AppConfig エージェントの使用をお勧めします。詳細については、「AWS Systems Manager の料金」を参照してください。

AWS AppConfig のクォータ

AWS AppConfig エンドポイントと Service Quotas および他のシステム・マネージャのクォータについては、Amazon Web Services 全般のリファレンス で説明されています。

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

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、通話呼び出しを受け取り、電話キーパッドで検証コードを入力するように求められます。

   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。セキュリティのベストプラクティスとして、ユーザーに管理アクセスを割り当て、ルートユーザーのみを使用してルートユーザーアクセスが必要なタスクを実行してください。

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] を選んで、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、 を保護し AWS アカウントのルートユーザー、 を有効にして AWS IAM Identity Center、日常的なタスクにルートユーザーを使用しないように管理ユーザーを作成します。

のセキュリティ保護 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 「ルートユーザーの仮想 MFA デバイスを有効にする (コンソール)」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、「 ユーザーガイド」の「デフォルト でユーザーアクセス IAM アイデンティティセンターディレクトリを設定するAWS IAM Identity Center 」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「グループの参加」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、 に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 Identity Center 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 SDKs AWS CLI、または AWS APIs。	使用するインターフェイス用の手引きに従ってください。 • については AWS CLI、「 ユーザーガイド」の AWS CLI 「 を使用するための の設定 AWS IAM Identity CenterAWS Command Line Interface 」を参照してください。 • AWS SDKs 、ツール、AWS APIs「 SDK とツールのリファレンスガイド」の「IAM Identity Center 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 SDKs AWS CLI、または AWS APIs。	「IAM ユーザーガイド」の「 AWS リソースでの一時的な認証情報の使用」の手順に従います。

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
IAM	(非推奨) 長期認証情報を使用して、、AWS SDKs AWS CLI、または AWS APIs。	使用するインターフェイス用の手引きに従ってください。 • 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 とツールのリファレンスガイド」の「長期的な認証情報を使用した認証」を参照してください。 AWS SDKs • AWS APIs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。

(オプション) CloudWatch アラームに基づいて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 アラームに応答して、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ときは、 AWS Identity and Access Management (IAM) role を指定します。 は CloudWatch 、アラームをモニタリングできるようにこのロール AWS AppConfig を必要とします。

> **Note**
>
> 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。別のアカウントのメトリクスに応じてデプロイをロールバック AWS AppConfig するように を設定する場合は、クロスアカウントアラームを設定する必要があります。詳細については、

「Amazon ユーザーガイド」の「クロスアカウントクロスリージョン CloudWatch コンソール」を参照してください。 CloudWatch

以下の手順に従って、 が CloudWatch アラームに基づいてロールバックできるようにする IAM AWS AppConfig 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のアクセス許可ポリシーを作成する

API `DescribeAlarms`アクションを呼び出す AWS AppConfig アクセス許可を付与する IAM ポリシーを作成するには、次の手順に従います。

アラームに基づいて CloudWatch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、次に示すように API DescribeAlarms オペレーションに * アクセス許可を割り当てる必要があります。の範囲が狭い場合`DescribeAlarms`、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. https://console.aws.amazon.com/iam/IAMコンソールを開きます。
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. 添付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、 で以下のタスクを完了するのに役立ちます AWS AppConfig。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 ターゲットの論理的な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。

   問題のある設定変更を自動的にロールバックするように、環境ごとに Amazon CloudWatch 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。

3. 設定プロファイルの作成

   設定プロファイルには、 が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。 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 AWS AppConfig をサポートします。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 と統合されている別の Systems Manager 機能または AWS サービスにデータを保存できますでのフリー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、デプロイの開始時にバリデータを使用して AWS AppConfig チェックを実行します。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**
>
> Amazon Simple Storage Service (Amazon S3) にシークレットを保存 AWS Secrets Manager したり、データを管理したりするための特別なニーズがない限り、ホスト AWS AppConfig 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。これは、最も機能と機能強化を提供するためです。

トピック

設定例

の一機能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。設定は、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。次に例を示します。

機能フラグ設定

次の機能フラグ設定は、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

次のフリーフォームの設定では、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

以下のアクセス制御リストのフリーフォーム設定では、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
```

設定プロファイルの IAM ロールについて

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 AWS AppConfig。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、システムはその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、 というアプリケーションを作成して MyMobileApp 、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設定データを整理および管理できます。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 AWS AppConfig 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。

> **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)

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

アプリケーションを作成するには

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、コンテナなどの AppConfig ターゲットの論理的な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。 環境ごとに Amazon CloudWatch 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

AWS AppConfig アラームに応答して が設定をロールバックできるようにする場合は、 が CloudWatch アラーム AWS AppConfig に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 CloudWatch 。このロールは、次の手順で選択します。詳細については、「(オプション) CloudWatch アラームに基づいて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、アプリケーションの名前を選択して詳細ページを開きます。
3. 環境 タブを選択し、環境の作成 を選択します。
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、アラームがトリガーされた場合に設定をロールバックするアクセス許可を持つ IAM ロールを選択します。
7. CloudWatch アラームリストで、モニタリングするアラームを 1 つ以上選択します。これらのアラームのいずれかがアラーム状態になった場合、 は設定デプロイを AWS AppConfig ロールバックします。
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 の場合) 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。

設定プロファイルには、 が保存場所と設定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。 は、機能フラグとフリーフォーム設定の 2 種類の設定プロファイル AWS AppConfig をサポートしています。機能フラグ設定は AWS AppConfig ホストされた設定ストアにデータを保存し、URI は単に ですhosted。フリーフォーム設定では、 AWS AppConfig ホストされた設定ストア、さまざまな Systems Manager 機能、または と統合する AWS サービスにデータを保存できます AWS AppConfig。詳細については、「でのフリー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。

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 AWS AppConfig チェックを実行します。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、最も多くの機能と機能強化を提供します。

トピック

- バリデータについて
- で機能フラグ設定プロファイルを作成する AWS AppConfig
- でのフリーフォーム設定プロファイルの作成 AWS AppConfig

バリデータについて

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バリデータを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 : 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 関数バリデータ

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。

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、機能フラグ設定を、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 ホスト設定ストアに AWS AppConfig 保存します。 AWS AppConfig ホスト設定ストアの詳細については、AWS AppConfig ホストされた設定ストアについて「」セクションを参照してください。

トピック

- 機能フラグ設定プロファイルの作成 (コンソール)
- 機能フラグと機能フラグ設定プロファイル (コマンドライン)の作成
- AWS.AppConfig.FeatureFlags リファレンスを入力します。

開始する前に

次の手順では、オプションの暗号化セクションで、 AWS Key Management Service （ AWS KMS) キーを選択できます。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、「 でAWS AppConfig カスタマーマネージャーキーをサポート」を参照してくださいAWS AppConfig のセキュリティ。

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も提供します。拡張機能を使用すると、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が強化されます。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。

最後に、機能フラグ属性セクションで、新しい機能フラグの属性の詳細を入力するときに、制約を指定できます。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なります。 は、次のタイプのフラグ属性とそれに対応する制約 AWS AppConfig 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
数値	最小値	属性の最小数値

タイプ	制約事項	説明
	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

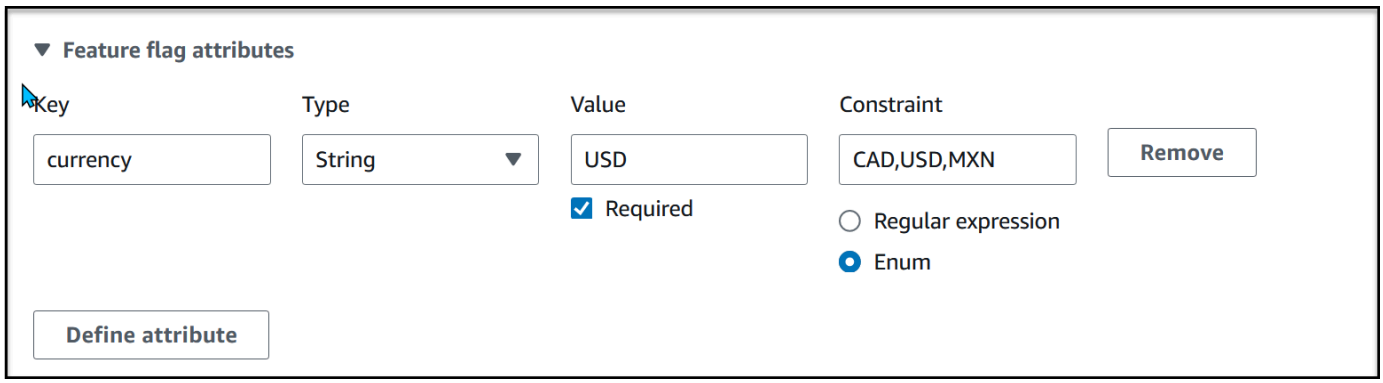
1. https://console.aws.amazon.com/systems-manager/appconfig/ で AWS Systems Manager コンソールを開きます。
2. ナビゲーションペインで、アプリケーション を選択し、 で作成したアプリケーションを選択しますAWS AppConfigでアプリケーションの名前空間を作成します。。
3. 設定プロファイルと機能フラグ タブを選択し、設定の作成 を選択します。
4. 「設定オプション」セクションで、「機能」フラグを選択します。
5. 下にスクロールします。「設定プロファイル」セクションの「設定プロファイル名」に名前を入力します。
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。

    b. 「拡張の関連付け」セクションで、リストから拡張を選択します。

c. 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] をクリックします。
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は、同じキーを持つことはできません。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. 選択 これは短期フラグで、必要に応じてフラグを無効化または削除する日付を選択します。AWS AppConfig は フラグを無効にしないことに注意してください。
13. フラグ属性 セクションで、属性の定義 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。
14. キー で、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。オプションで、属性値を指定した制約と比較して検証することができます。次のイメージは例を示しています。



属性の定義 を選択して、属性を追加します。

> **Note**
>
> 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。
> - 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。

15. 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。必要に応じて、指定した廃止日に達したときにフラグを無効にするには、同じトグルを使用します。
16. [次へ] をクリックします。
17. 確認と保存ページで、 フラグの詳細を確認し、保存して のデプロイを続行します。

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。

機能フラグと機能フラグ設定プロファイル (コマンドライン)の作成

次の手順では、 AWS Command Line Interface (Linux または Windows の場合) または Tools for Windows を使用して AWS AppConfig 機能フラグ設定プロファイル PowerShell 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、次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。

機能フラグの設定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. 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 The_application_ID \
     --name A_name_for_the_configuration_profile \
     --location-uri hosted \
     --type AWS.AppConfig.FeatureFlags
   ```

   Windows

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

PowerShell

```
New-APPCConfigurationProfile `
  -Name A_name_for_the_configuration_profile `
  -ApplicationId The_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、「AWS.AppConfig.FeatureFlags リファレンスを入力します。」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 The_application_ID \
  --configuration-profile-id The_configuration_profile_id \
  --content-type "application/json" \
  --content file://path/to/feature_flag_configuration_data \
  file_name_for_system_to_store_configuration_data
```

Windows

```
aws appconfig create-hosted-configuration-version ^
  --application-id The_application_ID ^
  --configuration-profile-id The_configuration_profile_id ^
  --content-type "application/json" ^
  --content file://path/to/feature_flag_configuration_data ^
  file_name_for_system_to_store_configuration_data
```

PowerShell

```
New-APPCHostedConfigurationVersion `
  -ApplicationId The_application_ID `
  -ConfigurationProfileId The_configuration_profile_id `
  -ContentType "application/json" `
```

```
  -Content file://path/to/feature_flag_configuration_data `
  file_name_for_system_to_store_configuration_data
```

Linux のサンプルコマンドを次に示します。

```
aws appconfig create-hosted-configuration-version \
  --application-id 1a2b3cTestApp \
  --configuration-profile-id 4d5e6fTestConfigProfile \
  --content-type "application/json" \
  --content Base64Content
```

この content パラメータは、以下の base64 エンコードデータを使用します。

```
{
  "flags": {
    "flagkey": {
      "name": "WinterSpecialBanner"
    }
  },
  "values": {
    "flagkey": {
      "enabled": 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"          : "1a2b3cTestApp",
   "ConfigurationProfileId" : "4d5e6fTestConfigProfile",
   "VersionNumber"          : "1",
   "ContentType"            : "application/json"
}
```

Windows

```
{
   "ApplicationId"          : "1a2b3cTestApp",
   "ConfigurationProfileId" : "4d5e6fTestConfigProfile",
   "VersionNumber"          : "1",
   "ContentType"            : "application/json"
}
```

PowerShell

```
ApplicationId          : 1a2b3cTestApp
ConfigurationProfileId : 4d5e6fTestConfigProfile
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、 はデータが JSON `AWS.AppConfig.FeatureFlags` スキーマに準拠していること AWS AppConfig を検証します。 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。

AWS.AppConfig.FeatureFlags リファレンスを入力します。

`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": {
            "$ref": "#/definitions/flagValues"
          }
        },
        "required": ["version", "flags"],
        "additionalProperties": false
      },
      "flagDefinitions": {
        "type": "object",
        "patternProperties": {
          "^[a-z][a-zA-Z\\d-]{0,63}$": {
            "$ref": "#/definitions/flagDefinition"
          }
        },
        "maxProperties": 100,
        "additionalProperties": false
      },
      "flagDefinition": {
        "type": "object",
        "properties": {
          "name": {
            "$ref": "#/definitions/customerDefinedName"
          },
          "description": {
            "$ref": "#/definitions/customerDefinedDescription"
          },
          "_createdAt": {
            "type": "string"
          },
          "_updatedAt": {
            "type": "string"
          },
          "_deprecation": {
            "type": "object",
            "properties": {
              "status": {
                "type": "string",
                "enum": ["planned"]
              }
```

```
            },
            "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"
          }
        },
        "maxProperties": 100,
```

```
      "additionalProperties": false
    },
    "flagValue": {
      "type": "object",
      "properties": {
        "enabled": {
          "type": "boolean"
        },
        "_createdAt": {
          "type": "string"
        },
        "_updatedAt": {
          "type": "string"
        }
      },
      "patternProperties": {
        "^[a-z][a-zA-Z\\d-_]{0,63}$": {
          "$ref": "#/definitions/attributeValue",
          "maxProperties": 25
        }
      },
      "required": ["enabled"],
      "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" }
          ]
        },
        "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GetLatest「設定」を参照してください。

アプリケーションが GetLatestConfiguration を呼び出し、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
    }
  }
}
```

でのフリー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、 が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。 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 AWS AppConfig 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 は単に ですhosted。フリーフォーム設定プロファイルの場合、 AWS AppConfig ホストされた設定ストア、または以下の AWS サービスや Systems Manager 機能のいずれかにデータを保存できます。

ロケーション	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。API アクションを使用して追加された場合の AWS AppConfig CreateHostedConfigurationVersion任意のファイルタイプ。
Amazon Simple Storage Service (Amazon S3)	すべて
AWS CodePipeline	パイプライン (サービスによって定義される)

ロケーション	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、デプロイを開始するときにバリデータを使用して AWS AppConfig チェックを実行します。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

- 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 に保存された設定について
- フリーフォーム設定および設定プロファイルの作成

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	2 MB のデフォルト、最大 4 MB	2 MB S3 AWS AppConfig ではなく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 ではなく AWS AppConfig によって強制される CodePipeline
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	いいえ	はい
AWS CloudForm	はい	データの作成または更	はい	はい	いいえ	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
ation のサポート		新用ではありません				
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に比べて次のような利点があります。

- 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ための 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 に保存された設定について

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。また、オブジェクトを取得するアクセス許可を付与 AWS AppConfig する (IAM) ロールの Amazon リソースネーム AWS Identity and Access Management (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、、、、STANDARD、INTELLIGENT_TIERING 、STANDARD_IA および の S3 REDUCED_REDUNDANCY ストレージクラスをサポートします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:GetObjectバージョン

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: GetBucket口ケーション

- s3:バーGetBucketジョニング
- s3:ListBucket
- s3:ListBucketバージョン

S3 オブジェクトに保存されている設定を AWS AppConfig が取得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

S3 オブジェクトに保存されている設定を AWS AppConfig が取得できるようにする IAM ポリシーを作成するには、次の手順に従います。

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。##########独自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:::DOC-EXAMPLE-BUCKET/my-configurations/my-
configuration.json"
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetBucketLocation",
        "s3:GetBucketVersioning",
        "s3:ListBucketVersions",
        "s3:ListBucket"
```

```
      ],
      "Resource": [
        "arn:aws:s3:::DOC-EXAMPLE-BUCKET"
      ]
    },
    {
      "Effect": "Allow",
      "Action": "s3:ListAllMyBuckets",
      "Resource": "*"
    }
  ]
}
```

5. レビューポリシー 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

S3 オブジェクトに保存されている設定を AWS AppConfig が取得できるようにする IAM ロールを作成するには、次の手順に従います。

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. https://console.aws.amazon.com/iam/IAMコンソールを開きます。
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。

フリーフォーム設定および設定プロファイルの作成

このセクションでは、フリーフォーム設定および設定プロファイルを作成する方法について説明します。開始する前に、以下の情報に注意してください。

- 次の手順では、選択した設定ストアの設定データに AWS AppConfig がアクセスできるように、IAM サービスロールを指定する必要があります。 AWS AppConfig ホストされた設定ストアを使用する場合、このロールは必要ありません。S3、Parameter Store、または Systems Manager のドキュメントストアを選択した場合は、既存の IAM ロールを選択するか、システムによってロールを自動的に作成するオプションを選択する必要があります。詳細については、「設定プロファイルの IAM ロールについて」を参照してください。

- 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も提供します。拡張機能を使用すると、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が強化されます。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。
- S3 に保存されている設定用の設定プロファイルを作成する場合は、アクセス許可を設定する必要があります。S3 を設定ストアとして使用するためのアクセス許可およびその他の要件の詳細については、「Amazon S3 に保存された設定について」を参照してください。
- バリデータを使用する場合は、バリデータを使用するための詳細と要件を確認してください。詳細については、「バリデータについて」を参照してください。

トピック

- 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)

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、 で作成したアプリケーションを選択しますAWS AppConfigでアプリケーションの名前空間を作成します。。
3. 設定プロファイルと機能フラグ タブを選択し、設定の作成 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. 「拡張の関連付け」セクションで、リストから拡張を選択します。

   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] をクリックします。
9. 「設定データを指定」ページの「設定定義」セクションで、オプションを選択します。

10. 次の表に示すように、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すると、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。

選択したオプション	詳細
AWS Systems Manager document	1. リストからドキュメントを選択するか、新しいドキュメントの作成 を選択します。 2. 新しいドキュメントの作成 を選択した場合は、ドキュメント名 に名前を入力します。オプションで、バージョン名を展開し、ド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 では、リストから JSON スキーマを選択するか、スキーマの作成 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、「レプリケーション設定スキーマの作成」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、検証の追加ページで、JSON スキーマまたは を選択しますAWS Lambda。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] をクリックします。
14. 確認と保存ページで、保存を選択して のデプロイを続行します。

> **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として CodePipeline を指定するパイプライン AWS AppConfig を に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、「API を直接呼び出して設定を取得します。」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、「AWS CodePipeline ユーザーガイド」の「チュートリアル: をデプロイプロバイダー AWS AppConfig として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 の場合) 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、ロケーション 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 The_application_ID \
  --name A_name_for_the_configuration_profile \
  --description A_description_of_the_configuration_profile \
  --location-uri A_URI_to_locate_the_configuration or hosted \
  --retrieval-role-
arn The_ARN_of_the_IAM_role_with_permission_to_access_the_configuration_at_the_specified
 \
  --tags User_defined_key_value_pair_metadata_of_the_configuration_profile \
  --validators "Content=JSON_Schema_content_or_the_ARN_of_an_AWS
 Lambda_function,Type=JSON_SCHEMA or LAMBDA"
```

Windows

```
aws appconfig create-configuration-profile ^
  --application-id The_application_ID ^
  --name A_name_for_the_configuration_profile ^
  --description A_description_of_the_configuration_profile ^
  --location-uri A_URI_to_locate_the_configuration or hosted  ^
  --retrieval-role-
arn The_ARN_of_the_IAM_role_with_permission_to_access_the_configuration_at_the_specified
 ^
  --tags User_defined_key_value_pair_metadata_of_the_configuration_profile ^
  --validators "Content=JSON_Schema_content_or_the_ARN_of_an_AWS
 Lambda_function,Type=JSON_SCHEMA or LAMBDA"
```

PowerShell

```
New-APPCConfigurationProfile `
  -Name A_name_for_the_configuration_profile `
  -ApplicationId The_application_ID `
  -Description Description_of_the_configuration_profile `
  -LocationUri A_URI_to_locate_the_configuration or hosted `
  -
RetrievalRoleArn The_ARN_of_the_IAM_role_with_permission_to_access_the_configuration_at_
 `
  -
Tag Hashtable_type_user_defined_key_value_pair_metadata_of_the_configuration_profile
 `
  -Validators "Content=JSON_Schema_content_or_the_ARN_of_an_AWS
 Lambda_function,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として CodePipeline を指定するパイプライン AWS AppConfig を に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、「API を直接呼び出して設定を取得します。」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、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。

設定データのその他のソース

このトピックには、 と統合されている他の AWS サービスに関する情報が含まれています AWS AppConfig。

AWS AppConfig との統合 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「 とは AWS Secrets Manager」を参照してください。Secrets Manager を使用する

設定プロファイルの作成については、「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。

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。

AWS AppConfig は Amazon と統合 CloudWatch してデプロイをモニタリングします。デプロイによって でアラームがトリガーされた場合 CloudWatch、 はデプロイ AWS AppConfig を自動的にロールバックして、アプリケーションユーザーへの影響を最小限に抑えます。

トピック

- デプロイ戦略の使用
- 構成のデプロイ
- AWS AppConfig デプロイと の統合 CodePipeline

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、設定が をデプロイまたはロールアウトする方法を定義します。 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します。<code>G*(2^N)</code>。この式で、<code>G</code> 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、<code>N</code> 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、</td></tr>
</table>

設定	説明

	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するには AWS AppConfig 、 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、「(オプション) CloudWatch アラームに基づいて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
- デプロイ戦略の作成

定義済みのデプロイ戦略

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.Linear20PercentEvery6Minutes	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 分間モニタリングします。この時間内にアラームが受信されなければ、デプ

デプロイ戦略	説明
	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、 AWS ベストプラクティスに沿ったものであり、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 10 分間 CloudWatch アラームをモニタリングします。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 10 分間 CloudWatch アラームをモニタリングします。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 分間モニタリングします。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

事前定義された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. 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、デプロイ戦略 を選択し、デプロイ戦略の作成 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. ベーク時間 には、デプロイの次のステップに進む前、またはデプロイの完了を検討する前に 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として CodePipeline を指定するパイプライン AWS AppConfig を に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、「API を直接呼び出して設定を取得します。」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、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。

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 AWS Systems Manager Parameter Store 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 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モニタリングし、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 設定をデプロイするには、 AWS Systems Manager 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. 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、 で作成したアプリケーションを選択しますAWS AppConfigでアプリケーションの名前空間を作成します。。
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. 追加の暗号化オプション で、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
```

AWS AppConfig デプロイと の統合 CodePipeline

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あり、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確実に行うためにリリースパイプラインを自動化するのに役立ちます。 は、定義したリリースモデルに基づいて、コードが変更されるたびにリリースプロセスのビルド、テスト、デプロイフェーズ CodePipeline 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 を統合すると、次の利点 CodePipeline があります。

- CodePipeline を使用してオーケストレーションを管理するお客様は、コードベース全体をデプロイしなくても、アプリケーションに設定変更をデプロイできる軽量な方法を利用できるようになりました。
- AWS AppConfig を使用して設定のデプロイを管理したいが、 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、追加のオプションが追加されました。 は AWS CodeCommit、 GitHub、および BitBucket （ いくつか) CodePipeline をサポートしています。

> **Note**
>
> AWS AppConfig と の統合 CodePipeline は、 AWS リージョン CodePipeline が利用可能なでのみサポートされます。

統合の仕組み

まず、 をセットアップして設定します CodePipeline。これには、 CodePipelineがサポートするコードストアへの設定の追加が含まれます。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として CodePipeline を指定するパイプラインを に作成 AWS AppConfig します。 その後、設定を変更して CodePipeline コードストアにアップロードできます。新しい設定をアップロードすると、 で新しいデプロイが自動的に開始されま

す CodePipeline。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: をデプロイプロバイダー AWS AppConfig として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、 データ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 AWS AppConfig データを取得します。このセクションで説明する簡略化された取得方法のいずれかを使用する場合、 AWS AppConfig エージェント Lambda 拡張機能または AWS AppConfig エージェントのいずれかが、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。エージェントをローカルホストとして設定 AWS AppConfig し、エージェントに設定の更新 AWS AppConfig をポーリングします。エージェントはStartConfigurationセッション API アクションとGetLatest設定 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簡略化された検索方法。

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。

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。

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。

これに応じて、 `InitialConfigurationToken`はセッション AWS AppConfig のクライアントにを渡して、`GetLatestConfiguration`そのセッションを初めて呼び出すときに使用します。

`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。

このセクションでは、次の情報を紹介します。

内容

- 簡略化された検索方法
- API を直接呼び出して設定を取得します。

AWS AppConfig データプレーンサービスについて

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、StartConfigurationセッション とGetLatest設定 の 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。

2022 年 1 月 28 日より AWS AppConfig 前に の使用を開始した場合、サービスは GetConfiguration API アクションを直接呼び出すか AWS、 AWS AppConfig エージェント Lambda 拡張機能などの が提供するクライアントを使用してこの API アクションを呼び出す可能性があります。GetConfiguration API アクションを直接呼び出す場合は、StartConfigurationSession および GetLatestConfiguration API アクションを使用する手順を実行してください。 AWS AppConfig エージェント Lambda 拡張機能を使用している場合は、このトピックで後述する「この変更が AWS AppConfig エージェント Lambda 拡張機能に与える影響」というタイトルのセクションを参照してください。

新しいデータプレーン API アクションには、現在廃止された GetConfiguration API アクションに比べて次のような利点があります。

1. ClientID パラメータを管理する必要はありません。データプレーンサービスを使用して、ClientID が StartConfigurationSession で作成したセッショントークンを内部で管理します。
2. 設定データのキャッシュバージョン示すために ClientConfigurationVersion を含める必要はなくなりました。データプレーンサービスを使用して、ClientConfigurationVersion が StartConfigurationSession で作成したセッショントークンを内部で管理します。
3. データプレーン API 呼び出し用の新しい専用エンドポイントは、コントロールプレーンとデータプレーンの呼び出しを分離することでコード構造を改善します。
4. 新しいデータプレーンサービスにより、データプレーン運用の将来的拡張性が向上します。 AWS AppConfig チームは、設定データの取得を管理する設定セッションを利用することで、将来的により強力な機能拡張を作成できます。

GetConfiguration から **GetLatestConfiguration** への移行

新しいデータプレーンサービスの使用を開始するには、GetConfiguration API アクションを呼び出すコードを更新する必要があります。StartConfigurationSession API アクションを使用して設定セッションを開始し、GetLatestConfiguration API アクションを呼び出して設定データを取得します。パフォーマンスを向上させるため、設定データをローカルにキャッシュすることをお勧めします。　詳細については、「API を直接呼び出して設定を取得します。」を参照してください。

この変更が AWS AppConfig エージェント Lambda 拡張機能に与える影響

この変更は、 AWS AppConfig エージェント Lambda 拡張機能の動作に直接影響しません。AWS AppConfig エージェント Lambda 拡張機能の古いバージョンは、ユーザーに代わって GetConfiguration API アクションを呼び出しました。新しいバージョンでは、データプレーン API アクションが呼び出されます。 AWS AppConfig Lambda 拡張機能を使用している場合は、エクステンションを最新の Amazon リソースネーム (ARN) に更新し、新しい API コールのアクセス権限を更新することをお勧めします。詳細については、「AWS AppConfig エージェント Lambda 拡張機能を使用した設定データの取得」を参照してください。

簡略化された検索方法

AWS AppConfig には、設定データを取得するための簡略化された方法がいくつか用意されています。 AWS Lambda 関数で AWS AppConfig 機能フラグまたはフリーフォーム設定データを使用する場合は、 AWS AppConfig エージェント Lambda 拡張機能を使用して設定を取得できます。Amazon EC2 インスタンスで実行されているアプリケーションがある場合は、 AWS AppConfig エージェントを使用して設定を取得できます。 AWS AppConfig エージェントは、Amazon Elastic Kubernetes Service (Amazon EKS) または Amazon Elastic Container Service (Amazon ECS) コンテナイメージで実行されているアプリケーションもサポートします。

初期設定が完了すると、これらの設定データを取得する方法は AWS AppConfig APIs を直接呼び出すよりも簡単です。ベストプラクティスが自動的に実装され、設定を取得するための API コールが少なくなるため AWS AppConfig 、 の使用コストを削減できる可能性があります。

トピック

- AWS AppConfig エージェント Lambda 拡張機能を使用した設定データの取得
- Amazon EC2 インスタンスからの設定データの取得
- Amazon ECS および Amazon EKS からの設定データを取得します。
- その他の取り出し機能
- AWS AppConfig エージェントローカル開発

AWS AppConfig エージェント Lambda 拡張機能を使用した設定データの取得

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。 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縮されるため、コストを削減できます。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。

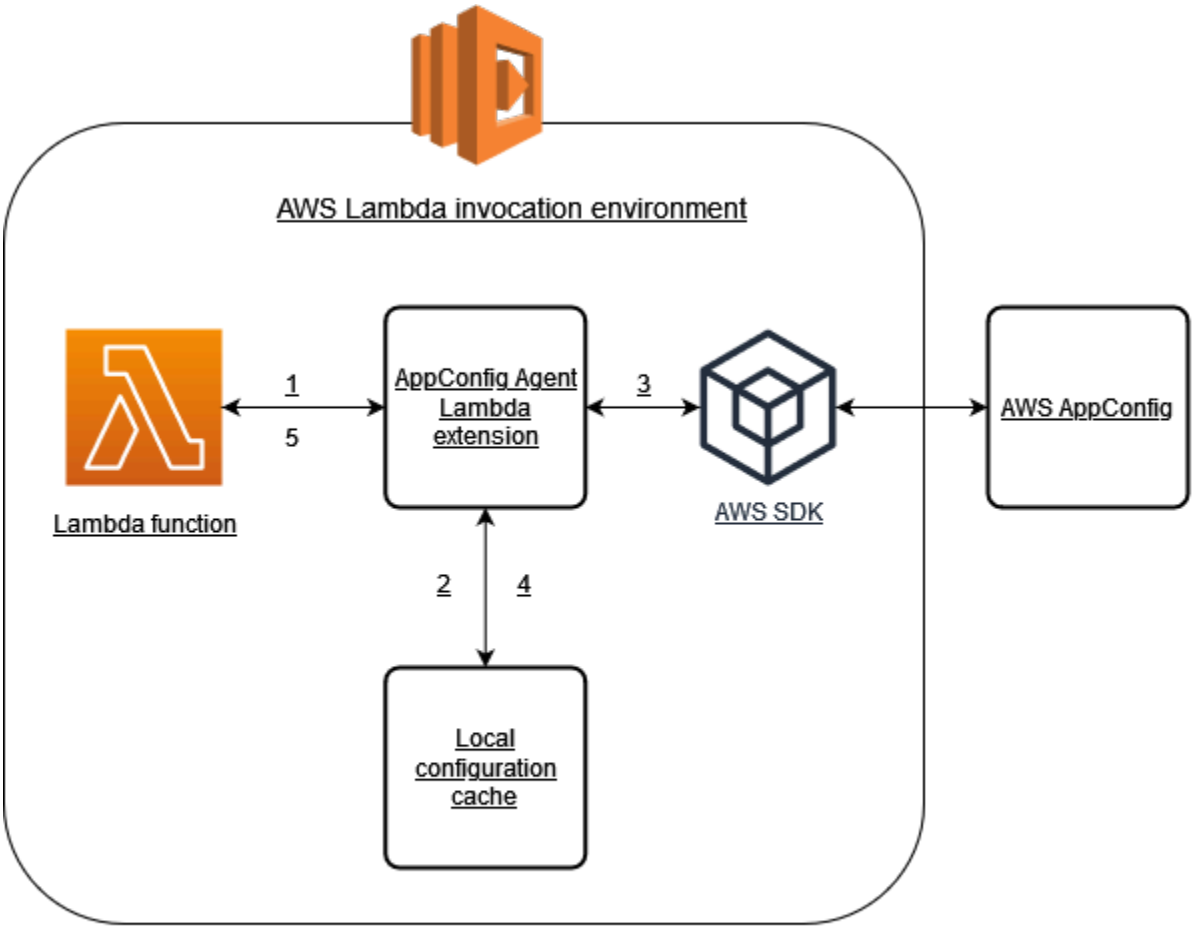
> **Note**
>
> AWS AppConfig の料金は、設定が呼び出されて受信された回数に基づきます。Lambda が複数のコールドスタートを実行し、新しい設定データを頻繁に取得すると、コストが増加します。

このトピックでは、 AWS AppConfig エージェント Lambda 拡張機能に関する情報と、Lambda 関数で動作するように拡張機能を設定する手順について説明します。

仕組み

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、StartConfigurationセッション API アクションと設定 API アクションと統合して、設定の更新を受信するように Lambda GetLatest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 拡張機能は、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 を呼び出し AWS AppConfig 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、変更があった場合に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。
> - から更新された設定をデプロイした後、Lambda 関数に設定変更が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、拡張機能に設定したポーリング間隔によって異なります。

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 関数の実行ロールで使用される `appconfig:GetLatestConfiguration` 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。

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。

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、Lambda AWS AppConfig 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。

AWS AppConfig エージェント Lambda 拡張機能をコンテナイメージとして追加するには、「」を参照してくださいコンテナイメージを使用して AWS AppConfig エージェント Lambda 拡張機能を追加する。

AWS AppConfig エージェント Lambda 拡張機能の設定

拡張機能は、次の AWS Lambda 環境変数を変更することで設定できます。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、指定された設定が から取得されます。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 が設定データの管理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。

環境変数	詳細	デフォルト値
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
AWS_APPCONFIG_EXTENSION_LOG_LEVEL	この環境変数は、関数の Amazon CloudWatch Logs に送信される AWS AppConfig 拡張固有のログを指定します。大文字と小文字を区別しない有効な値は debug、info、warn、error、および none です。デバッグには、タイミング情報など、拡張機能に関する詳細情報が含まれます。	info
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
AWS_APPCONFIG_EXTENSION_POLL_INTERVAL_SECONDS	この環境変数は、拡張機能が更新された設定 AWS AppConfig をポーリングする頻度を秒単位で制御します。	45

環境変数	詳細	デフォルト値
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、 からの応答を 拡張機能が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
AWS_APPCONFIG_EXTENSION_PREFETCH_LIST	この環境変数は、関数が初期化されてハンドラーが実行される前に、拡張機能が取得を開始する設定データを指定します。これにより、機能のコールドスタート時間を大幅に短縮できます。	なし
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。各ヘッダーは次の形式を使用します。 `"header: value"`	なし

環境変数	詳細	デフォルト値
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
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
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
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
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

環境変数	詳細	デフォルト値
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。 次のいずれかの値を入力できます。 • "app:env:manifest-config" • "file:/fully/qualified/path/to/manifest.json" これらの機能の詳細については、「その他の取り出し機能」を参照してください。	true
AWS_APPCONFIG_EXTENSION_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。	true

機能フラグ設定から 1 つ以上のフラグを取得する

機能フラグ設定 (AWS.AppConfig.FeatureFlags タイプの設定) の場合、Lambda 拡張機能を使用すると、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。1 つまたは 2 つのフラグを取得すると、Lambda で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場合に便利です。次の例では Python 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 Lambda 拡張機能バージョン 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_name* 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にアクセスする方法

```
import urllib.request

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

複数のフラグとその属性にアクセスする方法

```
import urllib.request

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

AWS AppConfig エージェント Lambda 拡張機能ログの表示

AWS AppConfig エージェント Lambda 拡張機能の AWS Lambda ログデータは、 ログで表示できます。ログエントリの先頭には が付きますappconfig agent。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 拡張機能の利用可能なバージョン

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。正常に動作するには、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 データプレーンサービスについて」を参照してください。
- AWS AppConfig は、「」に記載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

次の表は、Lambda AWS AppConfig 拡張機能の最新バージョンに加えられた変更点を示しています。

Version	開始日	メモ
2.0.358	12/01/2023	以下の取り出し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 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: エージェントを使用して AWS AppConfig 設定データをディスクに書き込みます。この機能により、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するお客様は、 と統合できます AWS AppConfig。
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。
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

Version	開始日	メモ
		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。

Version	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 エージェント 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、AWS レイヤーリストから AWS-AppConfig-Extension を選択します。
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。エージェント AWS AppConfig Lambda 拡張機能のバージョンは、「実行の詳細」セクションで見つけます。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 2.0.358

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`
欧州 (フランクフルト)	`arn:aws:lambda:eu-central-1:066940009817:layer:AWS-AppConfig-Extension:106`

リージョン	ARN
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
中国 (寧夏)	arn:aws-cn:lambda:cn-northwest-1:615084187847:layer:AWS-AppConfig-Extension:76

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: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
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

リージョン	ARN
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

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 2.0.358

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
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

リージョン	ARN
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
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

リージョン	ARN
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
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

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-Arm64:5
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

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 Lambda 拡張機能の ARNs と AWS AppConfig 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。

古い拡張バージョン (x86-64 プラットフォーム)

次の表に、x86-64 プラットフォーム用に開発された AWS AppConfig エージェント Lambda 拡張機能の ARNs と AWS リージョン 古いバージョンの を示します。

新しい拡張子に置き換えられた日付: 12/01/2023

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

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: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
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

リージョン	ARN
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
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

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: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
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

リージョン	ARN
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
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

リージョン	ARN
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
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

リージョン	ARN
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
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

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: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
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
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

リージョン	ARN
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
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

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: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
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
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

リージョン	ARN
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
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

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: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
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
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

リージョン	ARN
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
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

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: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
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
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

リージョン	ARN
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
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

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: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
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)

次の表に、ARM64 プラットフォーム用に開発された AWS AppConfig エージェント Lambda 拡張機能の ARNs と AWS リージョン 古いバージョンの を示します。

新しい拡張子に置き換えられた日付: 12/01/2023

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

リージョン	ARN
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
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

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-Arm64:36
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
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

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-Arm64:31
米国西部 (オレゴン)	arn:aws:lambda:us-west-2:359756378197:layer:AWS-AppConfig-Extension-Arm64:45
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

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

リージョン	ARN
欧州 (ロンドン)	arn:aws:lambda:eu-west-2:282860088358:layer:AWS-AppConfig-Extension-Arm64:2
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

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-Arm64:2
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

コンテナイメージを使用して AWS AppConfig エージェント Lambda 拡張機能を追加する

AWS AppConfig エージェント Lambda 拡張機能をコンテナイメージとしてパッケージ化して、Amazon Elastic Container Registry (Amazon ECR) でホストされているコンテナレジストリにアップロードできます。

AWS AppConfig エージェント Lambda 拡張機能を Lambda コンテナイメージとして追加するには

1. 以下に示すように、() に AWS リージョン と Amazon リソースネーム AWS Command Line Interface (ARN AWS CLI) を入力します。リージョンと ARN の値をリージョンと一致する ARN に置き換えて、Lambda Layer のコピーを取得します。x86-64 および ARM64 プラットフォーム用の ARNs AWS AppConfig を提供します。

   ```
   aws lambda get-layer-version-by-arn \
     --region AWS ##### \
     --arn extension ARN
   ```

   以下に例を示します。

   ```
   aws lambda get-layer-version-by-arn \
     --region us-east-1 \
     --arn arn:aws:lambda:us-east-1:027255383542:layer:AWS-AppConfig-Extension:128
   ```

   レスポンスは次のようになります。

   ```
   {
     "LayerVersionArn": "arn:aws:lambda:us-east-1:027255383542:layer:AWS-AppConfig-
   Extension:128",
     "Description": "AWS AppConfig extension: Use dynamic configurations deployed via
    AWS AppConfig for your AWS Lambda functions",
     "CreatedDate": "2021-04-01T02:37:55.339+0000",
     "LayerArn": "arn:aws:lambda::layer:AWS-AppConfig-Extension",

     "Content": {
       "CodeSize": 5228073,
       "CodeSha256": "8otOgbLQbexpUm3rKlMhvcE6Q5TvwcLCKrc4Oe+vmMY=",
       "Location" : "S3-Bucket-Location-URL"
     },
   ```

```
  "Version": 30,
  "CompatibleRuntimes": [
    "python3.8",
    "python3.7",
    "nodejs12.x",
    "ruby2.7"
  ],
}
```

2. 上記の応答では、Location に対して返された値は、Lambda 拡張機能を含む Amazon Simple Storage Service (Amazon S3) バケットの URL です。ウェブブラウザに URL を貼り付けて、Lambda 拡張機能の.zip ファイルをダウンロードします。

> **Note**
>
> Amazon S3 バケット URL は 10 分間のみ有効です。

（オプショナル）または、次の curl コマンドを使用して、Lambda 拡張機能をダウンロードすることもできます。

```
curl -o extension.zip "S3-Bucket-Location-URL"
```

（オプショナル）または、ステップ 1。 とステップ 2。 を組み合わせて、ARN の取得および拡張機能の .zip ファイルのダウンロードを一度に済ませることもできます。

```
aws lambda get-layer-version-by-arn \
  --arn extension ARN \
  | jq -r '.Content.Location' \
  | xargs curl -o extension.zip
```

3. 次の行を Dockerfile に追加して、コンテナイメージに拡張機能を追加します。

```
COPY extension.zip extension.zip
RUN yum install -y unzip \
  && unzip extension.zip /opt \
  && rm -f extension.zip
```

4. Lambda 関数の実行ロールに appconfig:GetConfiguration アクセス許可セットがあることを確認します。

例

このセクションでは、コンテナイメージベースの Python Lambda 関数で AWS AppConfig エージェント Lambda 拡張機能を有効にする例を示します。

1. 次のような Dockerfile を作成します。

```
FROM public.ecr.aws/lambda/python:3.8 AS builder
COPY extension.zip extension.zip
RUN yum install -y unzip \
  && unzip extension.zip -d /opt \
  && rm -f extension.zip

FROM public.ecr.aws/lambda/python:3.8
COPY --from=builder /opt /opt
COPY index.py ${LAMBDA_TASK_ROOT}
CMD [ "index.handler" ]
```

2. 拡張機能レイヤーを、Dockerfile と同じディレクトリにダウンロードします。

```
aws lambda get-layer-version-by-arn \
  --arn extension ARN \
  | jq -r '.Content.Location' \
  | xargs curl -o extension.zip
```

3. index.py という名前の Python ファイルを、Dockerfile と同じディレクトリに作成します。

```
import urllib.request

def handler(event, context):
    return {
       # replace parameters here with your application, environment, and
 configuration names
       'configuration': get_configuration('myApp', 'myEnv', 'myConfig'),
    }

def get_configuration(app, env, config):
    url = f'http://localhost:2772/applications/{app}/environments/{env}/
configurations/{config}'
    return urllib.request.urlopen(url).read()
```

4. 次のステップを実行して docker イメージを構築し、それを Amazon ECR にアップロードします。

```
// set environment variables
export ACCOUNT_ID = <YOUR_ACCOUNT_ID>
export REGION = <AWS_REGION>

// create an ECR repository
aws ecr create-repository --repository-name test-repository

// build the docker image
docker build -t test-image .

// sign in to AWS
aws ecr get-login-password \
  | docker login \
  --username AWS \
  --password-stdin "$ACCOUNT_ID.dkr.ecr.$REGION.amazonaws.com"

// tag the image
docker tag test-image:latest "$ACCOUNT_ID.dkr.ecr.$REGION.amazonaws.com/test-
repository:latest"

// push the image to ECR
docker push "$ACCOUNT_ID.dkr.ecr.$REGION.amazonaws.com/test-repository:latest"
```

5. 先ほど作成した Amazon ECR イメージを使用して、Lambda 関数を作成します。コンテナとしての Lambda 関数の詳細については、「コンテナイメージとして定義された Lambda 関数を作成する」を参照してください。
6. Lambda 関数の実行ロールに appconfig:GetConfiguration アクセス許可セットがあることを確認します。

OpenAPI と統合する

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: AppConfig Agent Lambda extension API
  description: An API model for the AppConfig Agent Lambda extension.
servers:
  - url: https://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
      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
```

Amazon EC2 インスタンスからの設定データの取得

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 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、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、エージェントは定期的に AWS AppConfig データプレーンをポーリングして設定データを更新します。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する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

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などの 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、アプリケーションに AWS AppConfig データプレーンへのアクセスを設定する必要があります。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、 によって管理される 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Amazon EC2」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。の一機能である Run Command を使用して、ローカルクライアントマシンからコマンドを実行し、1 つ以上のインスタンスをターゲットにすることができます 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 インスタンス上のエージェントは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、x86_64 アーキテクチャーの管理者権限で以下のコマンドを実行します。

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

ARM64 アーキテクチャーでは、以下のコマンドを実行します。

```
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: (オプションですが推奨) ログファイルを CloudWatch ログに送信する

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、systemd ジャーナルのログデータを表示および管理できます。より良いソリュー

ションである分散アプリケーションに強く推奨されるソリューションは、ログファイルをディスクに書き込み、Amazon CloudWatch エージェントを使用してログデータを AWS クラウドに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ます。これにより、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 エージェントの使用を開始するには、「Amazon ユーザーガイド」の「 エージェントを使用して Amazon EC2 インスタンスとオンプレミスサーバーからメトリクスとログを収集する CloudWatch 」を参照してください。 CloudWatch Systems Manager の一機能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ら、 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、 CloudWatch エージェントログファイルの操作方法を 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の詳細については、 CloudWatch 「エージェント設定ファイルの作成」を参照してください。

インスタンスの CloudWatch エージェント設定ファイルに次のlogs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 の場合true、 CloudWatch エージェントauto_removal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. 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 AWS AppConfig 。

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

AWS AppConfig Agent for Amazon EC2 を設定するには、ドロップインユニットファイルに次の環境変数を指定します。

環境変数	詳細	デフォルト値

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、認証タイプ Bearer のHTTP リクエスト認証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

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。	なし

環境変数	詳細	デフォルト値
	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、「AWS AppConfig のセキュリティ」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
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。変数は、大文字と小文字が区別されます。ログレベルは debug, info, warn, error, none.情報など、エージェントに関する詳細情報が含まれます。Debug には、エージェントに関する時間情報などの詳細情報が含まれます。	info

環境変数	詳細	デフォルト値
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
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。次のいずれかの値を入力できます。 • "app:env:manifest-config" • "file:/fully/qualified/path/to/manifest.json" これらの機能の詳細については、「その他の取り出し機能」を参照してください。	true
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

環境変数	詳細	デフォルト値
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
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。	なし
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。に設定するとfalse、 AWS AppConfig エージェントは、ネットワークに問題がある場合など、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。	true

環境変数	詳細	デフォルト値
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。各ヘッダーは次の形式を使用します。 `"header: value"`	なし
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。 HTTPSおよび HTTP URL AWS のサービスがサポートされています。 URLs	なし

環境変数	詳細	デフォルト値
REQUEST_TIMEOUT	この環境変数は、エージェントが からの応答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。	3000 ミリ秒

環境変数	詳細	デフォルト値
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
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
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
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
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。	true

ステップ 5: (必須) 設定データの取得

HTTP localhost 呼び出しを使用して、 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 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 からの設定データを取得します。

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、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、エージェントは定期的に AWS AppConfig データプレーンをポーリングして設定データを更新します。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する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

- 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。
- 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
- 設定データの取得

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などの 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に AWS AppConfig データプレーンへのアクセスを設定する必要があります。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
- AWS Fargate (Fargate) ポッド実行ロール (Amazon EKS 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。

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。AWS AppConfig エージェントサイドカーコンテナを使用するには、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に対して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、「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」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。AWS AppConfig エージェントサイドカーコンテナを使用するには、起動する必要があります。以下の手順では、Amazon EKS kubectl コマンドラインツールを使用して、コンテナアプリケーションの kubeconfig ファイルを変更する方法を説明します。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. kubeconfig ファイルを開き、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な感じ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 YAML デプロイファイルに追加します。

```
- 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: region
        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、「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」を参照してください。
> - *SERVICE_REGION* には、 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. kubectl ツールで以下のコマンドを実行して、クラスターに変更を適用します。

```
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に対して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、「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」を参照してください。

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

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。

環境変数	詳細	デフォルト値
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、認証タイプ Bearer のHTTP リクエスト認証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
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	なし

環境変数	詳細	デフォルト値
	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、「AWS AppConfig のセキュリティ」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

環境変数	詳細	デフォルト値
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。変数は、大文字と小文字が区別されます。ログレベルは debug, info, warn, error, none.情報など、エージェントに関する詳細情報が含まれます。Debug には、エージェントに関する時間情報などの詳細情報が含まれます。	info
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。次のいずれかの値を入力できます。 • "app:env:manifest-config" • "file:/fully/qualified/path/to/manifest.json" これらの機能の詳細については、「その他の取り出し機能」を参照してください。	true

環境変数	詳細	デフォルト値
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
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
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。	なし

環境変数	詳細	デフォルト値
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。に設定するとfalse、 AWS AppConfig エージェントは、ネットワークに問題がある場合など、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。	true
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。各ヘッダーは次の形式を使用します。 `"header: value"`	なし
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。 HTTPSおよび HTTP URL AWS のサービスがサポートされています。URLs	なし

環境変数	詳細	デフォルト値
REQUEST_TIMEOUT	この環境変数は、エージェントが からの応答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。	3000 ミリ秒

環境変数	詳細	デフォルト値
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
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
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
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
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。	true

設定データの取得

HTTP localhost 呼び出しを使用して、 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「test-backend/test-service」などのスラッシュを使用している場合に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"
```

その他の取り出し機能

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。

- 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 の 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。この機能により、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するお客様は、 と統合できます AWS AppConfig。

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指定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。マニフェストの内容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"
        }
    }
}
```

2 つの設定を持つマニフェストの JSON の例を次に示します。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指定する方法

マニフェストは、 AWS AppConfig エージェントが読み取ることができ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、次のいずれかの値でMANIFEST環境変数を設定する必要があります。

マニフェストの場所	環境変数値	ユースケース
File	ファイル:/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*	動的更新には、このメソッドを使用します。に保存されているマニフェストは、他の AWS AppConfig 設定を保存するのと同じ方法で、設定 AWS AppConfig として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

マルチアカウント取得

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して、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、 (IAM) ロールの Amazon リソースネーム AWS Identity and Access Management (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。

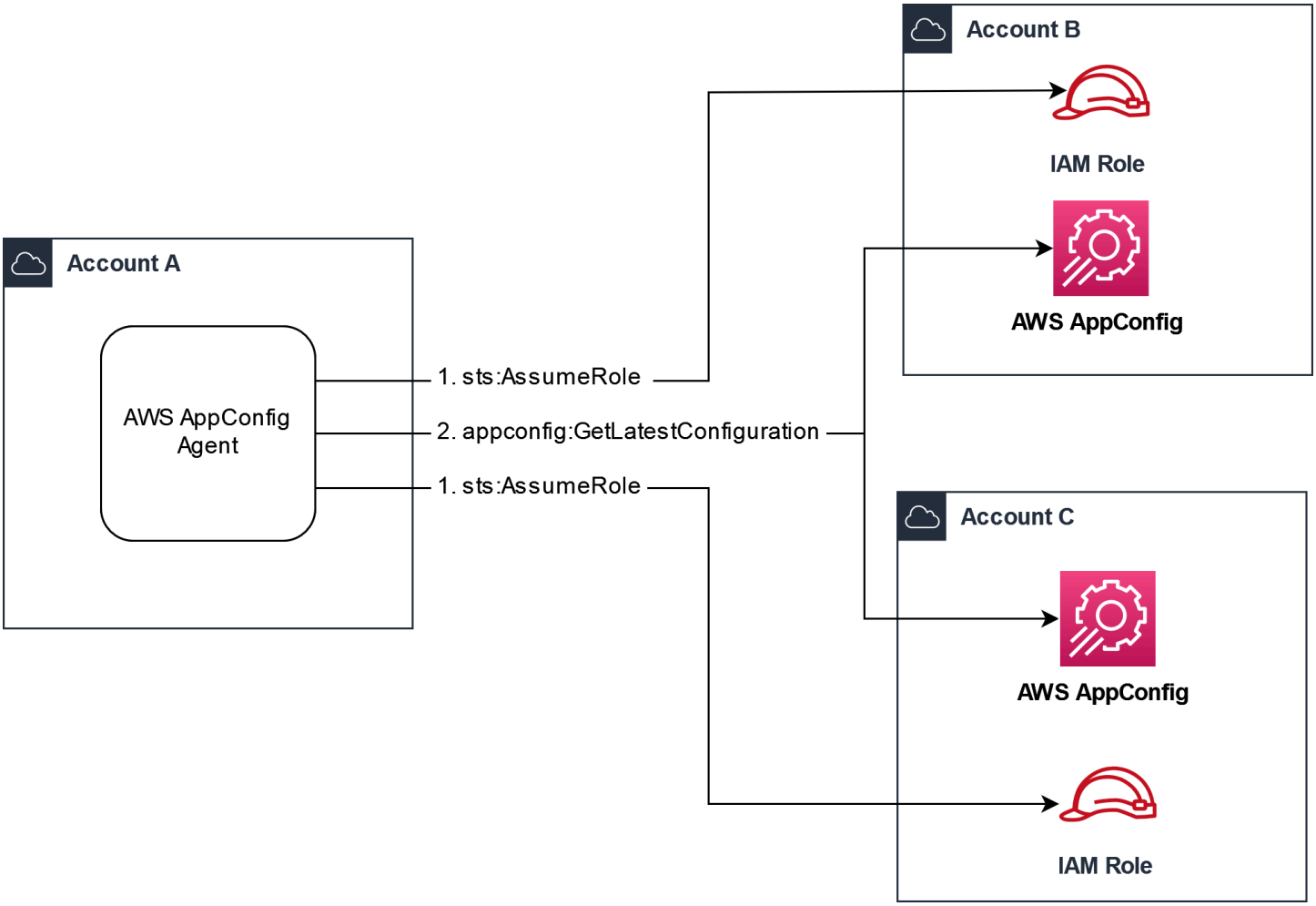
```
{
    "application_name:environment_name:configuration_name": {
        "credentials": {
            "roleArn": "arn:partition:iam::account_ID:role/roleName",
            "roleExternalId": "string",
            "roleSessionName": "string",
            "credentialsDuration": "time_in_hours"
     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

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、設定ごとに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 アカウント A (取得アカウント) で

実行されているときに、エージェントがアカウント 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、GetLatest設定 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 AWS AppConfig 設定データ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ためのアクセス許可が必要です。各ベンダーアカウントに AWS Identity and Access Management (IAM) ロールを作成して、エージェントにアクセス許可を付与します。取得アカウントの AWS AppConfig エージェントは、ベンダーアカウントからデータを取得するためにこのロールを引き受けます。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、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、次の情報を収集します。

- 各 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、設定データを取り出し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、 AWS AppConfigデフォルトで で使用されるロールのリストを示します。
  - 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を提供する各 AWS アカウント 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 で、デフォルトの JSON を次のポリシーステートメントに置き換えます。各*###############*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] をクリックします。
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) タグを追加する で、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、または制御します。
9. 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を提供する各 AWS アカウント でこの手順を繰り返します。

IAM ロールを作成する

IAM コンソールを使用して IAM ロールを作成するには、次の手順を使用します。取得アカウントの設定データを提供する各 AWS アカウント 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、ポリシーの作成 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) この継承ロール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] をクリックします。
9. アクセス許可の追加ページで、検索フィールドを使用して、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] をクリックします。
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: 信頼できるエンティティ を選択する 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*###############*を取得アカウントから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. 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。この情報は、次の手順で指定します。

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で 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検証する

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::' の初期データを取得するためのINFOレベルログYourEnvironmentNameYourConfigurationNameは、取得が成功するための最適な指標です。取り出しが失敗した場合、失敗の理由を示すERRORレベル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
```

ディスクへの設定コピーの書き込み

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するように AWS AppConfig エージェントを設定できます。この機能により、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するお客様は、 と統合できます AWS AppConfig。

この機能は、設定の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」または「Amazon ECS BACKUP_DIRECTORYと Amazon EKS でエージェントを使用する」の環境変数とPRELOAD_BACKUP環境変数を参照してください。 AWS AppConfig

> **Warning**
>
> この機能に関する以下の重要な情報に注意してください。
>
> - ディスクに保存された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。

ディスクへの設定の書き込みコピーを有効にするには

1. マニフェストを編集します。
2. ディスクに書き込む設定 AWS AppConfig 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。

ディスクへの設定のコピーの書き込みが機能していることを検証する

AWS AppConfig エージェント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 が設定 '*application* : *environment* : *configuration* ' to *file_path*」を記述した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
```

AWS AppConfig エージェントローカル開発

AWS AppConfig エージェントはローカル開発モード をサポートします。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。指定し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。

- 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します AWS AppConfig。
- 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します。

- 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します。

> **Warning**
>
> 本番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 エージェントは次の で動作します 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LOCAL_DEVELOPMENT_DIRECTORYに を追加します。読み取りアクセス許可をエージェントに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` です。
4. ディレクトリに ファイルを作成します。ファイル名は次の形式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**
   >
   > (オプション) 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application/jsonしたときにエージェントはコンテンツタイプ を返します。拡張機能を省略すると、エージェントはコンテンツタイ

プapplication/octet-streamに を使用します。正確な制御が必要な場合は、 形式で拡張機能を提供できます.*type*%*subtype*。エージェントはコンテンツタイプ を返します.type/subtype。

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動作をします。

> **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。

API を直接呼び出して設定を取得します。

アプリケーションは、まず Session API StartConfigurationオペレーションを使用して設定セッションを確立することで、設定データを取得します。その後、セッションのクライアントは、GetLatestConfiguration を定期的に呼び出して、利用可能な最新のデータをチェックして取得します。

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

これに応じて、 InitialConfigurationTokenはセッション AWS AppConfig のクライアントに を渡して、GetLatestConfigurationそのセッションを初めて呼び出すときに使用します。

> **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。
>
> - StartConfigurationセッション 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、クライアントごとに 1 回だけ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。

設定例を取得しています。

次の AWS CLI 例は、データStartConfigurationSessionおよび GetLatestConfiguration API オペレーションを使用して設定 AWS AppConfig 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、InitialConfigurationトークンを使用してGetLatest設定を呼び出し、設定データを取得します。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。

拡張機能によるワークフローの拡張

拡張機能を使用すると、設定を作成またはデプロイする AWS AppConfig ワークフロー中に、ロジックや動作をさまざまなポイントに注入できます。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。

コンテンツ

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

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通知をウェブフックに送信しますか？ 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。カスタム拡張を作成することも、 に含まれているオー AWS サリングされた拡張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。

以下のオー AWS サリングされた拡張機能は、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 コンソールで使用できます。または AWS CLI、、 AWS Tools for PowerShell、または SDK から拡張 API アクションを直接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	この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。詳細については、「Amazon CloudWatch Evidently 拡張機能の使用」を参照してください。
AWS AppConfig への デプロイイベント EventBridge	この拡張機能は、設定がデプロイされたときに EventBridge 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」	この拡張機能により 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。例えば、 AWS 作成した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_CREATE_HOSTED_CONFIGURATION_VERSION
- PRE_START_DEPLOYMENT
- ON_DEPLOYMENT_START
- ON_DEPLOYMENT_STEP
- ON_DEPLOYMENT_BAKING
- ON_DEPLOYMENT_COMPLETE
- ON_DEPLOYMENT_ROLLED_BACK

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 AWS AppConfig を実行する前に適用されます。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。

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。

ステップ 3: 拡張機能の作成

拡張機能を作成したり、 AWS オーサリングされた拡張機能を設定したりするには、特定の AWS AppConfig リソースの使用時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の 拡張機能と AWS AppConfig リソース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。

例えば、 という設定プロファイル MobileApps を含む という AWS AppConfig アプリケーションがあるとします AccessList。また、 MobileApps アプリケーションにベータ環境、統合環境、本番稼働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の関連付けを作成し、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、ホストされた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で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、次のオー AWS サリング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	この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。詳細については、「Amazon CloudWatch Evidently 拡張機能の使用」を参照してください。
AWS AppConfig への デプロイイベント EventBridge	この拡張機能は、設定がデプロイされたときに EventBridge 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。
「インテグレーションエクステンション — アトラシアン Jira」	この拡張機能により 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 拡張機能の使用

Amazon CloudWatch Evidently を使用すると、機能のロールアウト中に、指定した割合のユーザーに提供することで、新しい機能を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。拡張機能を設定して使用する方法については、「Amazon ユーザーガイド」の CloudWatch 「Evidently で起動と A/B 実験を実行する」を参照してください。CloudWatch

AWS AppConfig deployment events to Amazon EventBridge 拡張機能との連携

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オー AWS サリングされた拡張機能です。拡張機能は、設定がデプロイされるたびに EventBridge 、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、 は設定のデプロイの開始、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詳細に制御したい場合は、カスタム拡張を作成し、URI フィールドに EventBridge デフォルトのイベントバス 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**
>
> この拡張機能は、 EventBridge デフォルトの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。 AWS AppConfig コンソールまたは 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 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 EventBridge ときに に送信されるイベントの例を次に示します。

```
{
```

```
    "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"
    }
}
```

AWS AppConfig deployment events to Amazon SNS 拡張機能との連携

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 オーサリングされた拡張機能です。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、 は設定のデプロイの開始、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、拡張機能を AWS AppConfig リソースの 1 つにアタッチします。AWS AppConfig コンソールまたは 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 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"
        }
    }
}
```

AWS AppConfig deployment events to Amazon SQS 拡張機能との連携

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 オーサリングされた拡張機能です。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 1 つに関連付けると、 は設定のデプロイの開始、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、拡張機能を AWS AppConfig リソースの 1 つにアタッチします。AWS AppConfig コンソールまたは 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 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、 AWS アカウント 指定した の で機能フラグを変更するたびに、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、Jira に と通信するアクセス許可を付与します AWS AppConfig。これらの認証情報を 1 回 AWS AppConfig 使用して、 と Jira との 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 統合用の 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 統合用の IAM アクセス許可ポリシーを作成する

以下の手順を使用して、Atlassian Jira が と通信できるようにする IAM アクセス許可ポリシーを作成します 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. 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

以下の手順を使用して、 AWS AppConfig と Atlassian Jira 統合のユーザーを作成します。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. 次へ: 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 統合用の 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 について、 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. 送信 を選択します。
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、指定した AWS アカウント および 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 統合が不要になった場合は、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 の間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**
>
> でカスタム AWS AppConfig 拡張機能のサンプルを表示できます GitHub。
>
> - Systems Manager Change Calendar `blocked day` を使用してモレータームカレンダーによるデプロイを禁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、「カスタム AWS AppConfig 拡張機能の作成」を参照してください。

2. カスタム拡張機能の設定

   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. エクステンションを作成します

   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. 拡張機能の作成

   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、 は、拡張機能で定義されたアクションポイントがそのリソースに対して発生したときに、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

- カスタム AWS AppConfig 拡張機能用の Lambda 関数の作成
- カスタム AWS AppConfig 拡張機能のアクセス許可の設定
- カスタム AWS AppConfig 拡張機能の作成
- カスタム拡張機能の AWS AppConfig 拡張機能の関連付けの作成
- カスタム AWS AppConfig 拡張機能を呼び出すアクションを実行する

カスタム AWS AppConfig 拡張機能用の Lambda 関数の作成

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。このセクションには、カスタム AWS

AppConfig 拡張機能の Lambda 関数サンプルコードが含まれています。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」

Lambda 関数の次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。ARN は、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 拡張機能を使用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

PreCreateHostedConfigurationVersion

```
{
    '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'
    }
}
```

PreStartDeployment

```
{
    '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

OnStartDeployment, OnDeploymentStep, OnDeployment

```
{
    '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。

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"
        }]
    }
```

カスタム AWS AppConfig 拡張機能のアクセス許可の設定

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する (またはロールを継承する) 。 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、 がそのロールを引き受け AWS AppConfig ることを許可するには

1. https://console.aws.amazon.com/iam/IAMコンソールを開きます。

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。

カスタム AWS AppConfig 拡張機能の作成

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。例えば、 AWS 作成した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_CREATE_HOSTED_CONFIGURATION_VERSION
- PRE_START_DEPLOYMENT
- ON_DEPLOYMENT_START
- ON_DEPLOYMENT_STEP
- ON_DEPLOYMENT_BAKING
- ON_DEPLOYMENT_COMPLETE
- ON_DEPLOYMENT_ROLLED_BACK

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 AWS AppConfig を実行する前に適用されます。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。

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エクステンション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。

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。アクションは、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**
>
> 拡張機能の作成時にリクエストの構文とフィールドの説明を表示するには、 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 にい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 オーサリング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 作成し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 通知拡張機能の場合は、 Uriフィールドに EventBridge デフォルトのイベントの ARN を入力します。
- 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。

カスタム拡張機能の AWS AppConfig 拡張機能の関連付けの作成

拡張機能を作成したり、 AWS オーサリングされた拡張機能を設定したりするには、特定の AWS AppConfig リソースの使用時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 リソースの拡張を呼び出すアクションポイントの定義は、拡張の関連付け 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の 拡張機能と AWS AppConfig リソース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。

例えば、 という設定プロファイル MobileApps を含む という AWS AppConfig アプリケーションがあるとします AccessList。また、 MobileApps アプリケーションにベータ環境、統合環境、本番稼働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の関連付けを作成し、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、MyS3ConfigurationBackUpExtension 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 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。

カスタム AWS AppConfig 拡張機能を呼び出すアクションを実行する

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環として、 AWS AppConfig はPRE_CREATE_HOSTED_CONFIGURATION_VERSIONアクションポイントに対戦します。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。

AWS AppConfig 拡張機能と Atlassian Jira の統合

AWS AppConfig は Atlassian Jira と統合されています。統合により AWS AppConfig 、 AWS アカウント 指定した の で機能フラグを変更するたびに、Atlassian コンソールで問題を作成および更新できます 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、「の Atlassian Jira 拡張機能の使用 AWS AppConfig」を参照してください。

AWS AppConfig コードサンプル

このセクションには、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が含まれています。テスト環境でアクションを実行するには、Java 、Python 、および JavaScript SDKs でこれらのサンプルを使用することをお勧めします。このセクションには、終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が含まれています。

トピック

ホストされた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た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 APIs。

- 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た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 APIs。

- 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た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 APIs。

- 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た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 Simplified retrieval methods
        */

        // 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
# Simplified retrieval methods
#
```

```
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
// Simplified retrieval methods

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た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 }
```

GetLatestConfig API アクションを使用したフリーフォーム設定プロファイルの読み取り

次の各サンプルには、コードによって実行された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 APIs。

- GetLatestConfiguration
- StartConfigurationSession

Java

```
public void retrieveConfigFromApi() {
        /*
        The example below uses two AppConfigData APIs: StartConfigurationSession and
 GetLatestConfiguration.
        For more information on these APIs, see AWS AppConfig Data         */
        AppConfigDataClient appConfigData = AppConfigDataClient.create();

        /*
        Start a new configuration session using the StartConfigurationSession API.
 This operation does not return configuration data.
        Rather, it returns an initial configuration token that should be passed to
 GetLatestConfiguration.
        IMPORTANT: This operation should only be performed once (per configuration),
 prior to the first GetLatestConfiguration
```

```
        call you preform. Each GetLatestConfiguration will return a new
configuration token that you should then use in the
        next GetLatestConfiguration call.
        */
        StartConfigurationSessionResponse session =
            appConfigData.startConfigurationSession(req -> req
                .applicationIdentifier("MyDemoApp")
                .configurationProfileIdentifier("MyConfigProfile")
                .environmentIdentifier("Beta"));

        /*
        Retrieve configuration data using the GetLatestConfiguration API. The first
time you call this API your configuration
        data will be returned. You should cache that data (and the configuration
token) and update that cache asynchronously
        by regularly polling the GetLatestConfiguration API in a background thread.
If you already have the latest configuration
        data, subsequent GetLatestConfiguration calls will return an empty response.
If you then deploy updated configuration
        data the next time you call GetLatestConfiguration it will return that
updated data.

        You can also avoid all the complexity around writing this code yourself by
leveraging our agent instead.
        For more information about the agent, see Simplified retrieval methods
        */

        // The first getLatestConfiguration call uses the token from
StartConfigurationSession
        String configurationToken = session.initialConfigurationToken();
        GetLatestConfigurationResponse configuration =

appConfigData.getLatestConfiguration(GetLatestConfigurationRequest.builder().configurationT

        System.out.println("Configuration retrieved via API: " +
configuration.configuration().asUtf8String());

        // You'll want to hold on to the token in the getLatestConfiguration
response because you'll need to use it
        // the next time you call
        configurationToken = configuration.nextPollConfigurationToken();
        configuration =

appConfigData.getLatestConfiguration(GetLatestConfigurationRequest.builder().configurationT
```

```
        // Try creating a new deployment at this point to see how the output below
 changes.
        if (configuration.configuration().asByteArray().length != 0) {
            System.out.println("Configuration contents have changed
 since the last GetLatestConfiguration call, new contents = " +
 configuration.configuration().asUtf8String());
        } else {
            System.out.println("GetLatestConfiguration returned an empty response
 because we already have the latest configuration");
        }
    }
```

Python

```
# the example below uses two AppConfigData APIs: StartConfigurationSession and
 GetLatestConfiguration.
#
# for more information on these APIs, see
# AWS AppConfig Data
#

import boto3

application_name = 'MyDemoApp'
environment_name = 'MyEnvironment'
config_profile_name = 'MyConfigProfile'

appconfigdata = boto3.client('appconfigdata')

# start a new configuration session.
# this operation does not return configuration data.
# rather, it returns an initial configuration token that should be passed to
 GetLatestConfiguration.
#
# note: this operation should only be performed once (per configuration).
#   all subsequent calls to AppConfigData should be via GetLatestConfiguration.
scs = appconfigdata.start_configuration_session(
    ApplicationIdentifier=application_name,
    EnvironmentIdentifier=environment_name,
    ConfigurationProfileIdentifier=config_profile_name)
initial_token = scs['InitialConfigurationToken']
```

```
# retrieve configuration data from the session.
# this operation returns your configuration data.
# each invocation of this operation returns a unique token that should be passed to
 the subsequent invocation.
#
# note: this operation does not always return configuration data after the first
 invocation.
#   data is only returned if the configuration has changed within AWS AppConfig
 (i.e. a deployment occurred).
#   therefore, you should cache the data returned by this call so that you can use
 it later.
glc = appconfigdata.get_latest_configuration(ConfigurationToken=initial_token)
config = glc['Configuration'].read()
```

JavaScript

```
// the example below uses two AppConfigData APIs: StartConfigurationSession and
 GetLatestConfiguration.

// for more information on these APIs, see
// AWS AppConfig Data

import {
  AppConfigDataClient,
  GetLatestConfigurationCommand,
  StartConfigurationSessionCommand,
} from "@aws-sdk/client-appconfigdata";

const appconfigdata = new AppConfigDataClient();

const application_name = "MyDemoApp";
const environment_name = "MyEnvironment";
const config_profile_name = "MyConfigProfile";

// start a new configuration session.
// this operation does not return configuration data.
// rather, it returns an initial configuration token that should be passed to
 GetLatestConfiguration.
//
// note: this operation should only be performed once (per configuration).
//   all subsequent calls to AppConfigData should be via GetLatestConfiguration.
const scs = await appconfigdata.send(
  new StartConfigurationSessionCommand({
```

```
    ApplicationIdentifier: application_name,
    EnvironmentIdentifier: environment_name,
    ConfigurationProfileIdentifier: config_profile_name,
  })
);
const { InitialConfigurationToken } = scs;

// retrieve configuration data from the session.
// this operation returns your configuration data.
// each invocation of this operation returns a unique token that should be passed to
 the subsequent invocation.
//
// note: this operation does not always return configuration data after the first
 invocation.
//   data is only returned if the configuration has changed within AWS AppConfig
 (i.e. a deployment occurred).
//   therefore, you should cache the data returned by this call so that you can use
 it later.
const glc = await appconfigdata.send(
  new GetLatestConfigurationCommand({
    ConfigurationToken: InitialConfigurationToken,
  })
);
const config = glc.Configuration.transformToString();
```

環境のクリーンアップ

このセクションのコードサンプルを 1 つ以上実行した場合は、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リソースを検索し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 APIs。

- 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 のセキュリティ

AWS では、クラウドセキュリティが最優先事項です。AWS のお客様は、セキュリティを最も重視する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利点を得られます。

セキュリティは、AWS とお客様の間の共有責任です。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して説明しています。

- クラウドのセキュリティ - AWS は、AWS クラウド で AWS の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を担います。また、AWS は、ユーザーが安全に使用できるサービスも提供します。AWSコンプライアンスプログラムg11AWSコンプライアンスプログラム/g11g10AWSコンプライアンスプログラム/g10の一環として、サードパーティーの監査が定期的にセキュリティの有効性をテストおよび検証しています。AWS Systems Manager に適用するコンプライアンスプログラムの詳細については、「コンプライアンスプログラムによる対象範囲内の AWS のサービス」「」を参照してください。
- クラウド内のセキュリティ - ユーザーの責任は、使用する AWS のサービスに応じて異なります。またお客様は、データの機密性、企業要件、適用法令と規制などのその他の要因に対しても責任を担います。

AWS AppConfig は AWS Systems Manager の一機能です。AWS AppConfig を使用する際に責任共有モデルを適用する方法を理解するには、「AWS Systems Manager でのセキュリティ」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システム· マネージャ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バックアップとディスクへの書き込みの 2 つの機能があります。これらの機能を有効にする場合は、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が AWS AppConfig エージェントのみにあ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ために必要なプロセスだけが、その機能を持っていることを確認します。最小限の特権アクセスの実装は、セキュリティリスクはもちろん、エラーや悪意ある行動によってもたらされる可能性のある影響を減らす上での基本となります。

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 最小特権アクセスの付与」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その他の取り出し機能。

AWS AppConfig での静止時のデータの暗号化

AWS AppConfig を使用して保存中の顧客データを保護するために、デフォルトで AWS 所有のキー暗号化機能を備えています。

デフォルトでは、AWS 所有のキー — AWS AppConfig はこれらのキーを使用して、サービスによってデプロイされ、AWS AppConfig データストアでホストされているデータを自動的に暗号化します。AWS 所有のキーは表示、管理、使用することはできず、その使用を監査することもできません。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、別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 データストアにホストされている設定データを保存するときや設定データを展開するときに使用するカスタマー管理キーを指定できます。

顧客管理キー — AWS AppConfig は、お客様が作成、所有、管理する対称型顧客管理キーの使用をサポートするようになりました。これにより、AWS 所有のキー の既存の暗号化に加えて第二レイヤーが追加されます。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
- 暗号化素材のローテーション
- 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顧客管理キーへのサポート

AWS AppConfig 設定データ用の顧客管理型キー暗号化をサポートします。AWS AppConfig ホストされたデータストアに保存された設定バージョンについては、お客様は対応設定プロファイルに KmsKeyIdentifier 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を作成するたびに、AWS AppConfig は AWS KMS から KmsKeyIdentifier データキーを生成してデータを暗号化してから保存します。後で、GetHostedConfigurationVersion または StartDeployment API 操作中にデータにアクセスすると、AWS AppConfig 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復号化します。

AWS AppConfig はまた、設定データのデプロイ用の顧客管理型キー暗号化をサポートします。設定データを暗号化するために、お客様は KmsKeyIdentifier をデプロイ提供できます。AWS AppConfig AWS KMS このキーを使用して、StartDeployment API 操作でデータを暗号化するための KmsKeyIdentifier データキーを生成します。

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、認証された暗号化をサポートする追加の認証データとして暗号化コンテキストを使用します。データの暗号化リクエストに暗号化コンテキストを組み込むと、AWS KMS は暗号化コンテキストを暗号化後のデータにバインドします。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:AWS AppConfig は、ホスト設定データの暗号化とデプロイメントのすべての AWS KMS 暗号化操作に暗号化コンテキストを使用します。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。

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、 が AWS AppConfigに送信するリクエストを追跡できますAWS KMS。

次の例は、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Decryptするためにによって呼び出されるAWS KMSオペレーションをモニタリングAWS AppConfigする 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を使用して AWS AppConfig にアクセス (AWS PrivateLink)

AWS PrivateLink を使用して、VPC と AWS AppConfig API の間にプライベート接続を作成できます。インターネットゲートウェイ、NAT デバイス、VPN 接続、AWS AppConfig 接続のいずれかを

使用せずに、VPC 内にあるかのように Amazon EKS にアクセスできます。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。

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 ガイド)』の「AWS のサービス でアクセスする)」を参照してください。

AWS AppConfig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 PrivateLink ガイド」の「考慮事項」を確認してください。

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

AWS AppConfig のインターフェイスエンドポイントは、Amazon VPC コンソールまたは AWS Command Line Interface (AWS CLI) を使用して作成できます。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の作成を参照してください。

以下のサービス名を使用して、AWS AppConfig API 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
```

インターフェイスエンドポイントのプライベート DNS を有効にすると、リージョンのデフォルト DNS 名を使用して、AWS AppConfig への API リクエストを実行できます。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経由での AWS AppConfig API へのフルアクセスが許可されています。VPC から AWS AppConfig API への許可されたアクセスをコントロールするには、カスタムエンドポイントポリシーをインターフェイスエンドポイントにアタッチします。

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 アクション用の 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 AWS AppConfig と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。

AWS AppConfig 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

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。Secrets Manager では、シークレットとデータベースの更新に AWS Lambda 関数が使用され、シークレットとデータベースの更新にも使用されます。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。

AWS AppConfig 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、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後に、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))
```

AWS AppConfig のモニタリング

モニタリングは、AWS AppConfig とその他 AWS ソリューションの信頼性、可用性、およびパフォーマンスの維持における重要な要素です。AWS は、AWS AppConfig をモニタリングし、問題が発生した場合には報告を行い、必要に応じて自動アクションを実行するために以下のモニタリングツールを提供しています。

- AWS CloudTrail は、AWSアカウントにより、またはそのアカウントに代わって行われた API コールや関連イベントを取得し、指定した Amazon S3 バケットにログファイルを配信します。AWS を呼び出したユーザーとアカウント、呼び出し元の IP アドレス、および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』を参照してください。
- Amazon CloudWatch Logs を使用すると、Amazon EC2 インスタンスや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 CloudTrail、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、特定のしきい値に達したときに通知できます。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できます。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。

トピック

- AWS AppConfigを使用したAWS CloudTrailAPI コールのログ記録
- AWS AppConfig データプレーン呼び出しのログ記録メトリクス

AWS AppConfigを使用したAWS CloudTrailAPI コールのログ記録

AWS AppConfig は、 のユーザーAWS CloudTrail、ロール、または AWSのサービス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 AppConfig。 は、 のすべての API コールをイベントAWS AppConfigとして CloudTrail キャプチャします。キャプチャされたコールには、AWS AppConfig コンソールのコールと、AWS AppConfig API オペレーションへのコードのコールが含まれます。証跡を作成する場合は、 の CloudTrailイベントなど、Amazon S3 バケットへのイベントの継続的な配信を有効にすることができますAWS AppConfig。証跡を設定しない場合でも、イベント履歴 で CloudTrail コンソールで最新のイベントを表示できます。 で収集された情報を使用して CloudTrail、 に対するリクエストAWS AppConfig、リクエスト元の IP アドレス、リクエスト者、リクエスト日時などの詳細を確認できます。

の詳細については CloudTrail、「 AWS CloudTrailユーザーガイド」を参照してください。

AWS AppConfig 内の情報 CloudTrail

CloudTrail アカウントを作成するAWS アカウントと、 は で有効になります。でアクティビティが発生するとAWS AppConfig、そのアクティビティは CloudTrail イベント履歴 の他のAWSサービスイベントとともに イベントに記録されます。 最近のイベントは、AWS アカウント で表示、検索、ダウンロードできます。詳細については、「イベント履歴 での CloudTrail イベントの表示」を参照してください。

AWS AppConfig のイベントなど、AWS アカウント のイベントの継続的な記録に対して、追跡を作成します。証跡により、 はログファイル CloudTrail を Amazon S3 バケットに配信できます。デフォルトでは、コンソールで証跡を作成するときに、証跡がすべての AWS リージョン に適用されます。証跡は、AWS パーティションのすべてのリージョンからのイベントをログに記録し、指定した Amazon S3 バケットにログファイルを配信します。さらに、 CloudTrail ログで収集されたデータをより詳細に分析し、それに基づく対応を行うように他の AWSサービスを設定できます。詳細については、次を参照してください:

- 「証跡作成の概要」
- CloudTrail でサポートされているサービスと統合
- の Amazon SNS 通知の設定 CloudTrail
- 複数のリージョンからの CloudTrail ログファイルの受信と複数のアカウントからの CloudTrail ログファイルの受信

すべての AWS AppConfig アクションは によってログに記録 CloudTrail され、 AWS AppConfig API リファレンス に記載されています。例えば、 および `ListApplications`アクションを呼び出す`GetApplication`と`CreateApplication`、 CloudTrail ログファイルにエントリが生成されます。

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、以下を判別するために役立ちます。

- リクエストが、ルート認証情報と AWS Identity and Access Management (IAM) ユーザー認証情報のどちらを使用して送信されたか。
- 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 サービスによって送信されたかどうか。

詳細については、「CloudTrail userIdentity 要素」を参照してください。

AWS AppConfig での データイベント CloudTrail

データイベントは、リソースで、またはリソース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、 を呼び出してデプロイされた最新の設定を取得するなど GetLatestConfiguration)。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 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、AWS CLI、または CloudTrail API オペレーションを使用して、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、 で使用できる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、証跡またはイベントデータストアを作成してデータイベントを記録するか、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口グに記録します。

  1. データイベントを選択して、データイベントを口グに記録します。
  2. データイベントタイプのリストから、 を選択しますAWS AppConfig。
  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口グに記録する、すべての`readOnly`イベントを口グに記録する、すべての`writeOnly`イベントを口グに記録する`eventName`、または `readOnly`、、および `resources.ARN`フィールドでフィルタリングするカスタム口グセレクタテンプレートを作成できます。
  4. セレクタ名 に、 と入力しますAppConfigDataEvents。データイベント証跡の Amazon CloudWatch Logs を有効にする方法については、「」を参照してくださいAWS AppConfig データプレーン呼び出しの口グ記録メトリクス。

- を使用してデータイベントを口グに記録するにはAWS CLI、 `eventCategory`フィールドを `Data`に、 `resources.type`フィールドをリソースタイプ値に等しく設定するように `--advanced-event-selectors`パラメータを設定します (表 を参照)。、`readOnly`、`eventName`および `resources.ARN`フィールドの値でフィルタリングする条件を追加できます。
  - データイベントを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、「 を使用した証跡のデータイベントの口グ記録AWS CLI」を参照してください。

- データイベントをログに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に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、「 を使用したイベントデータストアのデータイベントのログ記録 AWS CLI」を参照してください。

以下の表に示しているのは、AWS AppConfig リソースタイプです。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。列にログ記録された Data APIs CloudTrail には、リソースタイプについて に CloudTrail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	ログに記録されたデータ APIs CloudTrail*
AWS AppConfig	`AWS::AppConfig::Configuration`	• GetLatestConfiguration • StartConfigurationSession

* 、`eventName`、および `resources.ARN`フィールドでフィルタリングして`readOnly`、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 での 管理イベント CloudTrail

管理イベントでは、AWS アカウントのリソースで実行される管理オペレーションについての情報が得られます。これらのイベントは、コントロールプレーンオペレーションとも呼ばれます。デフォルトでは、 は管理イベント CloudTrail を記録します。

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記録します。が に記録する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 CloudTrail、AWS AppConfigAWS AppConfig「 API リファレンス」を参照してください。

AWS AppConfig ログファイルエントリについて

証跡は、指定した Amazon S3 バケットにイベントをログファイルとして配信できるようにする設定です。 CloudTrail ログファイルには、1 つ以上のログエントリが含まれます。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、リクエストされたアクション、アクションの日時、リクエストパラメータなどに関する情報が含まれます。 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、特定の順序では表示されません。

次の例は、 StartConfigurationSessionアクションを示す CloudTrail ログエントリを示しています。

```
{
      "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"
      }
    }
```

AWS AppConfig データプレーン呼び出しのログ記録メトリクス

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ログに記録できます。その後、1 つ以上のメトリクスフィルターを作成して、 CloudWatch ログ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ーは、Logs に送信されるログデータで検索する用語とパターンを定義します。 CloudWatch Logs CloudWatch は、メトリクスフィルターを使用してログデータを数値 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、アラームで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、 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CloudTrail。AWS AppConfig データプラン呼び出しの 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 での データイベント CloudTrail。

次の手順を使用して、 CloudWatch ログがAWS AppConfigデータプレーンへの呼び出しのメトリクスをログ記録できるようにします。

CloudWatch ログを有効にしてAWS AppConfigデータプレーンへの呼び出しのメトリクスをログに記録するには

1. https://console.aws.amazon.com/cloudtrail/ で CloudTrail コンソールを開きます。
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。ロググループは、後で CloudWatch ログコンソールで選択します。
6. [Role name] (ロール名) に名前を入力します。
7. [変更の保存] をクリックします。

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、次の手順に従います。この手順では、 による呼び出しoperationと、 および による (オプションで) operation 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Amazon Resource Name (ARN)。

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. で CloudWatch コンソールを開きますhttps://console.aws.amazon.com/cloudwatch/。
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 に名前を入力します。
6. フィルターパターン には、次のように入力します。

```
{ $.eventSource = "appconfig.amazonaws.com" }
```

7. (オプション) テストパターンセクションで、テストするログデータの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. [次へ] をクリックします。
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、 と入力します**operation**。
14. ディメンション値 には、 と入力します**$.eventName**。

(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 と入力しますresource。ディメンション値 には、 と入力します$.resources[0].ARN。

[次へ] をクリックします。

15. フィルターの詳細を確認し、メトリクスフィルター を作成します。

(オプション) この手順を繰り返して、 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AccessDenied。その場合は、以下の詳細を入力します。

1. フィルター名 に名前を入力します。
2. フィルターパターン には、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. ユニット 、ディメンション 、アラーム をスキップします。

が API コールを CloudTrail ログに記録すると、 でメトリクスを表示できます CloudWatch。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とログの表示」を参照してください。作成したメトリクスを見つける方法については、「利用可能なメトリクスの検索」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すると、ディメンションなしでメトリクスページでそれらのメトリクスを表示できます。

メトリクスの CloudWatchアラームの作成

メトリクスを作成したら、 でメトリクスアラームを作成できます CloudWatch。例えば、前の手順で作成したAWS AppConfigコール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 AWS AppConfig `StartConfigurationSession` API アクションへの呼び出しに対して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。

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
AWS AppConfig カスタム拡張のサンプル	「チュートリアル: カスタム AWS AppConfig 拡張機能の作成」トピックに、 の次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 GitHub。 • Systems Manager Change Calendar `blocked` day を使用してモレーター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 28 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、 のユーザー AWS CloudTrail、ロール、または AWS のサービス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 AppConfig。 は、 のすべての	2024 年 1 月 18 日

	API コールをイベント AWS AppConfig として CloudTrail 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固有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。	
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 AppConfig を使用したアクセス (AWS PrivateLink) 」を参照してください。	2023 年 12 月 6 日

AWS AppConfig エージェント取得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の取得に役立つ以下の追加機能があります。 その他の取り出し機能 • マルチアカウント取得： 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。 • ディスクへの設定のコピーの書き込み: 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。この機能により、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するお客様は、 と統合できます AWS AppConfig。 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れません。設	2023 年 12 月 1 日

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」または「Amazon ECS BACKUP_DIRECTORY と Amazon EKS でエージェントを使用する」の環境変数と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。指定し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。次のユースケースには、ローカル開発モードをお勧めします。

- を使用してデプロイする前に、さまざまな設定バー

	ジョンをテストします AWS AppConfig。 • 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します。 • 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します。	
新しいコードサンプルトピック	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。このトピックには、Java、Python、および の 6 つの一般的な AWS AppConfig アクションをプログラムで実行 JavaScript するため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 は、`AppConfig.Linear20PercentEvery6Minutes` 定義済みのデプロイ戦略を提供および推奨す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 と Amazon EC2 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 での新しい AWS AppConfig リソースと機能フラグのサポートの例	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および 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 および AWS::AppConfig::HostedConfiguration/バージョンのリソースに、ホストされた設定ストアで AWS AppConfig 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 と統合します 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「 とは 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 と Amazon ECS および Amazon EKS 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 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、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	2022 年 12 月 2 日

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されている間、エージェントは定期的に AWS AppConfig データプレーンをポーリングして設定データの更新を確認します。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。

新しい拡張機能: CloudWatch Evidently の 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指定した割合のユーザーに提供することで、新しい機能を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。拡張機能を設定して使用する方法については、「Amazon ユーザーガイド」の「Evidently で CloudWatch起動と A/B 実験を実行する」を参照してください。 CloudWatch	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との 2 つの新しい API アクションを使用しますGetLatestConfiguration。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	拡張機能を使用すると、設定を作成またはデプロイする AWS AppConfig ワークフロー中に、ロジックや動作をさまざまなポイントに注入できます。 AWSが作成し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 と Atlassian Jira の統合	Atlassian Jira と統合すると AWS AppConfig 、 AWS アカウント 指定した の で機能フラグを変更するたびに、Atlassian コンソールで問題を作成および更新できます 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、ユーザーから機能を隠しながら、新しい機能を開発して本番環境にデプロイできます。まず、 フラグを設定データ AWS AppConfig として 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 拡張機能が、新しいアジアパシフィック (大阪) リージョンで利用可能になりました。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、Lambda AWS AppConfig 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。AWS AppConfig サービスへの API コールが少なくなり、個別に Lambda 関数の処理時間が短縮されるため、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 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 PowerShell。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	の一機能 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、あらゆる規模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、検証チェックとモニタリングが組み込まれています。 は AWS AppConfig 、EC2 インスタンス、、コンテナ AWS Lambda、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使用できます。	2020 年 7 月 31 日

AWS 用語集

AWS の最新の用語については、「AWS の用語集リファレンス」の「AWS 用語集」を参照してください。

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。