aws

ユーザーガイド

AWS Secrets Manager



Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

AWS Secrets Manager: ユーザーガイド

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

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

Table of Contents

Secrets Manager とは 1
Secrets Manager の使用を開始する 1
標準への準拠 2
料金 2
Secrets Manager にアクセスする 4
Secrets Manager コンソール 4
コマンドラインツール 4
AWS SDKs 5
HTTPS クエリ API 5
Secrets Manager エンドポイント 6
ベストプラクティス 12
認証情報やその他の機密情報を に保存する AWS Secrets Manager 12
コード内の保護されていないシークレットを検索する 12
シークレットの暗号化キーを選択する 13
キャッシュを使用してシークレットを取得する 13
シークレットのローテーション 14
CLI を使用するリスクを軽減する 14
シークレットへのアクセスを制限する 14
BlockPublicPolicy 条件 14
ポリシーの IP アドレス条件に注意する 15
VPC エンドポイント条件でリクエストを制限する 16
シークレットをレプリケートする 16
シークレットを監視する 16
プライベートネットワークでインフラストラクチャを実行する 17
機密性の高いワークロードの API キー 18
に API キーを保存する AWS Secrets Manager 18
暗号化設定 19
API キーの自動ローテーション 20
ローテーション関数のライフサイクル 21
ローテーションスケジュール設定 22
シークレットを効率的に取得する 22
機密性の高いワークロードのセキュリティ強化 23
よくある質問 24
ローテーション中の重複期間を処理するにはどうすればよいですか? 24

プロバイダーがプログラムによるキー作成をサポートしていない場合はどうなりますか? 24
シークレットを取得するときに API スロットリングを回避するにはどうすればよいですか? 25
環境ごとに 1 つのシークレットを使用するか、バージョンで 1 つのシークレットを使用する必要がありますか? 25
チュートリアル 26
Amazon CodeGuru Reviewer 26
ハードコードされたシークレットの置き換え 26
ステップ 1: シークレットを作成する 27
ステップ 2: コードを更新する 29
ステップ 3: シークレットを更新する 30
次の手順 30
ハードコードされたデータベース認証情報の置き換え 31
ステップ 1: シークレットを作成する 32
ステップ 2: コードを更新する 33
ステップ 3: シークレットをローテーションする 34
次の手順 35
Salesforce クライアントシークレットを作成する 36
ステップ 1: 外部クライアントアプリを作成し、OAuth を有効にする 37
ステップ 2: マネージド外部シークレットを作成し、ローテーションを有効にする 38
ステップ 3: ローテーションされた認証情報をテストする 39
ステップ 4: リソースをクリーンアップする 40
次の手順 41
交代ユーザーローテーション 41
アクセス許可 42
前提条件 43
ステップ 1: Amazon RDS データベースユーザーを作成する 46
ステップ 2: ユーザーの認証情報用のシークレットを作成する 49
ステップ 3: ローテーションされたシークレットをテストする 50
ステップ 4: リソースをクリーンアップする 51
次の手順 51
シングルユーザーローテーション 52
アクセス許可 52
前提条件 53
ステップ 1: Amazon RDS データベースユーザーを作成する 53
ステップ 2: データベースユーザー認証情報のシークレットを作成する 54

ステップ 3: ローテーションされたパスワードをテストする 55
ステップ 4: リソースをクリーンアップする 56
次の手順 56
シークレットを作成する 57
AWS CLI 60
AWS SDK 61
シークレットとは 61
メタデータ 62
シークレットバージョン 63
シークレットの JSON 構造 64
Amazon RDS と Aurora の認証情報 65
Amazon Redshift 認証情報 67
Amazon Redshift Serverless 認証情報 68
Amazon DocumentDB 認証情報 68
Amazon Timestream for InfluxDB のシークレット構造 69
Amazon ElastiCache の認証情報 69
Active Directory 認証情報 69
シークレットの管理 72
シークレット値の更新 72
AWS CLI 73
AWS SDK 73
Secrets Manager でパスワードを生成する 74
シークレットを以前のバージョンにロールバックする 74
シークレットの暗号化キーを変更する 74
AWS CLI 76
シークレットの変更 77
AWS CLI 78
AWS SDK 78
シークレットを検索する 79
検索フィルター 79
AWS CLI 80
AWS SDK 81
シークレットの削除 81
AWS CLI 83
AWS SDK 84
シークレットを復元する 84

AWS CLI 85
AWS SDK 85
シークレットにタグ付けする 86
タグの基本を確認する 86
タグ付けを使用したコストを追跡する 87
タグの制限を理解する 87
コンソールでのシークレットのタグ付け 88
AWS CLI 89
API 90
SDK 90
マルチリージョンレプリケーション 91
AWS CLI 93
AWS SDK 93
レプリカシークレットをスタンドアロンシークレットに昇格させる 93
AWS CLI 94
AWS SDK 94
レプリケーションを防止する 95
レプリケーションをトラブルシューティングする 96
選択したリージョンに同じ名前のシークレットがある 97
KMS キーにレプリケーションを完成させるためのアクセス許可がない 97
KMS キーが無効になっているか、見つかりません 97
レプリケーションを行うリージョンが有効化されていない 98
シークレットの取得 99
Java 100
クライアント側のキャッシュを使用した Java 100
シークレット内の認証情報を使用した JDBC 接続 107
Java AWS SDK 116
Python 118
クライアント側のキャッシュを使用した Python 119
Python AWS SDK 125
シークレット値をバッチ取得する 127
.NET 128
クライアント側のキャッシュを使用した .NET 129
SDK for .NET 135
Go 138
クライアント側のキャッシュを使用した Go 139

Go AWS SDK ... 143
Rust ... 144
クライアント側のキャッシュを使用した Rust ... 145
Rust ... 147
Amazon EKS ... 148
サービスアカウントの IAM ロール (IRSA) を使用した ASCP ... 148
Pod Identity を使用した ASCP ... 148
適切なアプローチの選択 ... 149
Amazon EKS 用の ASCP をインストールする ... 149
Amazon EKS 用に ASCP を Pod Identity と統合する ... 154
Amazon EKS 用に ASCP を IRSA と統合する ... 158
ASCP の例 ... 161
AWS Lambda ... 169
Lambda でシークレットを取得する ... 169
Parameter Store の統合 ... 170
AWS ワークロード認証情報プロバイダー ... 170
AWS ワークロード認証情報プロバイダーの仕組み ... 170
AWS ワークロード認証情報プロバイダーのキャッシュについて ... 171
AWS ワークロード認証情報プロバイダーをダウンロードする ... 172
ソースから AWS ワークロード認証情報プロバイダーを構築する ... 173
AWS ワークロード認証情報プロバイダーをインストールする ... 176
AWS ワークロード認証情報プロバイダーを使用してシークレットを取得する ... 182
`refreshNow` パラメータについて ... 185
ロール連鎖を使用してアカウント間でシークレットを取得する ... 187
起動時のシークレットのプリフェッチ ... 189
設定オプション ... 191
オプション機能 ... 194
ログ記録 ... 194
セキュリティに関する考慮事項 ... 195
AI コーディングエージェント ... 195
仕組み ... 196
前提条件 ... 196
プラグインをインストールする ... 197
`{{resolve:...}}` 構文 ... 197
`asm-exec` を使用してシークレットでコマンドを実行する ... 198
クロスリージョンシークレット ... 199

セキュリティに関する考慮事項 199
フックがシークレットへの直接アクセスをブロックする方法 199
トラブルシューティング 200
C++ 200
JavaScript 201
Kotlin 203
PHP 203
Ruby 205
AWS CLI 205
を使用してバッチ内のシークレットのグループを取得する AWS CLI 206
AWS コンソール 207
AWS Batch 207
CloudFormation 207
GitHub ジョブ 209
前提条件 209
使用方法 210
環境変数の命名 211
例 212
GitLab 214
考慮事項 215
前提条件 215
GitLab AWS Secrets Manager との統合 217
トラブルシューティング 218
AWS IoT Greengrass 219
パラメータストア 219
シークレットのローテーション 220
マネージドローテーション 220
マネージド外部シークレットのローテーション 222
コンソールでローテーションを設定する 222
CLI を使用してローテーションを設定する 223
Lambda 関数によるローテーション 224
データベースシークレットの自動ローテーション (コンソール) 226
非データベースシークレットの自動ローテーション (コンソール) 230
自動ローテーション (AWS CLI) 235
Lambda 関数のローテーション戦略 238
Lambda ローテーション関数 241

ローテーション関数のテンプレート 244
ローテーションへのアクセス許可 252
AWS Lambda ローテーション関数のネットワークアクセス 257
ローテーションのトラブルシューティング 258
ローテーションスケジュール 277
ローテーションウィンドウ 277
rate 式 278
cron 式 278
すぐにシークレットをローテーションする 283
AWS CLI 283
ローテーションされていないシークレットを検索する 284
自動ローテーションをキャンセルする 284
他のサービスによって管理されるシークレット 286
シークレットを使用するサービス 287
App Runner 289
AWS App2Container 289
AWS AppConfig 289
Amazon AppFlow 290
AWS AppSync 290
Amazon Athena 290
Amazon Aurora 290
AWS CodeBuild 291
Amazon Data Firehose 291
AWS DataSync 291
Amazon DataZone 292
Direct Connect 292
AWS Directory Service 292
Amazon DocumentDB 293
AWS Elastic Beanstalk 293
Amazon Elastic Container Registry 293
Amazon Elastic Container Service 294
Amazon ElastiCache 294
AWS Elemental Live 295
AWS Elemental MediaConnect 295
AWS Elemental MediaConvert 295
AWS Elemental MediaLive 295

AWS Elemental MediaPackage 296
AWS Elemental MediaTailor 296
Amazon EMR 296
Amazon EventBridge 297
Amazon FSx 297
AWS Glue DataBrew 297
AWS Glue Studio 298
AWS IoT SiteWise 298
Amazon Kendra 298
Amazon Kinesis Video Streams 298
AWS Launch Wizard 299
Amazon Lookout for Metrics 299
Amazon Managed Grafana 299
AWS Managed Services 299
Amazon Managed Streaming for Apache Kafka 300
Amazon Managed Workflows for Apache Airflow 300
AWS Marketplace 300
AWS Migration Hub 300
AWS 並列コンピューティングサービス 301
AWS ParallelCluster 301
Amazon Q 302
Amazon OpenSearch Ingestion 302
AWS OpsWorks for Chef Automate 302
Amazon Quick 302
Amazon RDS 303
Amazon Redshift 303
Amazon Redshift Query Editor V2 304
Amazon SageMaker AI 304
AWS SCT 305
Amazon Timestream for InfluxDB 305
AWS Toolkit for JetBrains 305
AWS Transfer Family 306
AWS Wickr 306
サードパーティーアプリケーションによって管理されるシークレット 307
主な特徴 307
統合パートナー 308

ビッグ ID 更新トークン 310
Cisco セキュリティプラットフォーム API キー 310
Confluent Cloud API キー 312
Databricks 個人用アクセストークン 314
Databricks サービスプリンシパル OAuth クライアントシークレット 317
Datadog 管理キー 319
Datadog API キー 321
Datadog アプリケーションキー 322
GitLab アクセストークン 324
Jenkins API トークン 326
MongoDB Atlas データベースユーザー 328
MongoDB Atlas サービスアカウントの認証情報 330
Netskope API トークン 332
Paddle API キー 334
Salesforce クライアントシークレット 335
Snowflake キーペア 337
Snowflake プログラムによるアクセストークン 339
SonarQube トークン 341
Twilio API キー 343
Twilio 認証トークン 346
Twilio メイン API キー 347
セキュリティおよびアクセス許可 349
モニタリングとトラブルシューティング 351
既存のシークレットの移行 351
制約事項と考慮事項 352
CloudFormation 353
シークレットを作成する 354
JSON 354
YAML 355
自動ローテーション付きの Amazon RDS 認証情報を使ってシークレットを作成する 355
Amazon Redshift 認証情報を使用してシークレットを作成する 355
Amazon DocumentDB 認証情報を使用してシークレットを作成する 355
JSON 356
YAML 360
Secrets Manager が を使用する方法 CloudFormation 363
AWS CDK 364

シークレットイベント通知 365
EventBridge が Secrets Manager イベントをルーティングする方法 365
Secrets Manager の イベント 365
イベントの構造 366
イベントパターンの作成 366
シークレットラベルの更新イベント 368
シークレットを監視する 371
で口グ記録する AWS CloudTrail 371
AWS CLI 372
CloudTrail エントリ 372
CloudWatch を使用して監視する 378
CloudWatch アラーム 379
EventBridge で Secrets Manager のイベントをマッチさせる 379
指定したシークレットに対するすべての変更にマッチさせる 380
シークレット値がローテーションされたらイベントをマッチさせる 381
削除予定のシークレットの監視 382
ステップ 1: CloudTrail ログファイルの CloudWatch ログへの配信を設定します 382
ステップ 2: CloudWatch アラームを作成する 383
ステップ 3: CloudWatch アラームをテストする 384
シークレットのコンプライアンスを監視する 385
Secrets Manager のコストを監視する 386
GuardDuty で脅威を検出する 386
コンプライアンス検証 387
コンプライアンス標準 387
セキュリティ 390
を使用して AWS Secrets Manager シークレット AWS CLI を保存するリスクを軽減する 391
認証とアクセスコントロール 393
アクセス許可に関するリファレンス 394
Secrets Manager 管理者のアクセス許可 394
シークレットへのアクセス許可 394
Lambda ローテーション関数のアクセス許可 394
暗号化キーのアクセス許可 394
レプリケーション権限 395
アイデンティティベースのポリシー 395
リソースベースのポリシー 402
タグを使用したシークレットへのアクセス制御 410

AWS 管理ポリシー 411
シークレットへのアクセス許可を持つユーザーを特定する 418
クロスアカウントアクセス 419
オンプレミスアクセス 423
Secrets Manager でのデータ保護 423
保管中の暗号化 424
転送中の暗号化 424
ネットワーク間トラフィックのプライバシー 425
暗号化キーの管理 425
シークレット暗号化と復号 425
AWS KMS キーの選択 426
暗号化されるもの 427
プロセスの暗号化と復号 427
KMS キーのアクセス許可 428
Secrets Manager による KMS キーの使用方法 429
AWS マネージドキー (aws/secretsmanager) のキーポリシー 431
Secrets Manager の暗号化コンテキスト 433
Secrets Manager の とのやり取りをモニタリングする AWS KMS 435
インフラストラクチャセキュリティ 439
VPC エンドポイント (AWS PrivateLink) 440
エンドポイントポリシーを作成する 441
共有サブネット 442
IPv4 および IPv6 アクセス 442
IPv6 とは 442
デュアルスタックポリシーの使用 443
ポリシーに IPv6 を追加する 443
クライアントが IPv6 をサポートしていることを検証する 445
耐障害性 446
ポスト量子 TLS 447
トラブルシューティング 449
「アクセスが拒否されました」のメッセージ 449
一時的なセキュリティ認証情報に「アクセスが拒否されました」と表示される 450
変更がすぐに表示されない。 450
「ピアによる接続のリセット」エラー 451
シークレットの作成時に「非対称 KMS キーを持つデータキーを生成できません」 452

AWS CLI または AWS SDK オペレーションが部分的な ARN からシークレットを見つけられない 452
このシークレットは AWS サービスによって管理されるため、そのサービスを使用して更新する必要があります。 453
Transform: AWS::SecretsManager-2024-09-16 の使用時に Python モジュールのインポートが失敗する 453
マネージド外部シークレットがローテーションしていない 453
クォータ 455
Secrets Manager のクォータ 455
アプリケーションへの再試行を追加する 459
ドキュメント履歴 461
以前の更新 462
..... cdlxiii

とは AWS Secrets Manager

AWS Secrets Manager は、データベース認証情報、アプリケーション認証情報、OAuth トークン、API キー、およびその他のシークレットをライフサイクル全体で管理、取得、ローテーションするのに役立ちます。多くの AWS サービスは、Secrets Manager でシークレットを保存して使用します。

Secrets Manager は、アプリケーションのソースコードにハードコーディングされた認証情報が不要になるため、セキュリティ態勢を改善するのに役立ちます。認証情報を Secrets Manager に保存することで、アプリケーションまたはそのコンポーネントを調べることができるすべてのユーザーによる侵害の可能性を回避できます。ハードコーディングされた認証情報を Secrets Manager サービスへのランタイム呼び出しに置き換えることができるため、必要に応じて認証情報を動的に取得できます。

Secrets Manager で、シークレットの自動ローテーションスケジュールを設定することができます。これにより、長期のシークレットを短期のシークレットに置き換えることが可能となり、侵害されるリスクを大幅に減少させるのに役立ちます。認証情報はアプリケーションに保存されなくなったため、認証情報を変更しても、アプリケーションを更新したり、アプリケーションクライアントに変更を反映させたりする必要はなくなりました。

組織内に存在する可能性のある他のタイプのシークレット:

- AWS 認証情報 – をお勧めしますAWS Identity and Access Management。
- 暗号化キー — AWS Key Management Service を推奨
- SSH キー – Amazon EC2 Instance Connect を推奨
- プライベートキーと証明書 – AWS Certificate Manager を推奨

Secrets Manager の使用を開始する

Secrets Manager を初めて使用する場合は、以下のチュートリアルのいずれかから始めてください。

- the section called “ハードコードされたシークレットの置き換え ”
- the section called “ハードコードされたデータベース認証情報の置き換え ”
- the section called “交代ユーザーローテーション”
- the section called “シングルユーザーローテーション”

シークレットを使って実行できるその他のタスク:

- シークレットの管理
- シークレットへのアクセスを制御する
- シークレットの取得
- シークレットのローテーション
- シークレットイベント通知
- シークレットを監視する
- シークレットのコンプライアンスを監視する
- でシークレットを作成する AWS CloudFormation

標準への準拠

AWS Secrets Manager は、複数の標準について監査済みであり、コンプライアンス認定を取得する必要がある場合はソリューションの一部になることができます。詳細については、「コンプライアンス検証」を参照してください。

料金

Secrets Manager では、使用した分のみ料金が発生し、最低料金や設定料金はありません。削除対象としてマークされたシークレットに対しては料金は発生しません。現在の価格の詳細なリストについては、「AWS Secrets Manager 料金表」を参照してください。コストを監視するには、「the section called “Secrets Manager のコストを監視する”」を参照してください。

Secrets Manager が作成する を使用して AWS マネージドキー `aws/secretsmanager`、シークレットを無料で暗号化できます。シークレットを暗号化するために独自の KMS キーを作成すると、 は現在の AWS KMS レートで AWS 課金します。詳細については、「AWS Key Management Service 料金」を参照してください。

自動ローテーション (マネージドローテーションを除く) を有効にすると、Secrets Manager は AWS Lambda 関数を使用してシークレットをローテーションし、ローテーション関数に対して現在の Lambda レートで課金されます。詳細については、「AWS Lambda 料金」を参照してください。

アカウント AWS CloudTrail で を有効にすると、Secrets Manager が送信する API コールのログを取得できます。Secrets Manager は、すべてのイベントを管理イベントとしてログに記録します。は、すべての管理イベントの最初のコピーを無料で AWS CloudTrail 保存します。ただし、通知を有

効にすると、Amazon S3 のログストレージと Amazon SNS の料金が発生する場合があります。また、追加の証跡を設定している場合、管理イベントの追加コピーについては、料金が発生する可能性があります。詳細については、AWS CloudTrail 料金表を参照してください。

Secrets Manager のコスト配分タグを使用して、特定のシークレットまたはプロジェクトに関連する経費を追跡および分類できます。詳細については、このガイドthe section called “ シークレットにタグ付けする”の「」および AWS Billing 「 ユーザーガイド」のAWS 「コスト配分タグの使用」を参照してください。

アクセス AWS Secrets Manager

Secrets Manager は、次の方法で利用できます。

- Secrets Manager コンソール
- コマンドラインツール
- AWS SDKs
- HTTPS クエリ API
- AWS Secrets Manager エンドポイント

Secrets Manager コンソール

ブラウザベースの Secrets Manager コンソールを使用してシークレットを管理し、コンソールではシークレットに関連するほぼすべてのタスクを実行できます。

コマンドラインツール

AWS コマンドラインツールを使用すると、システムコマンドラインでコマンドを発行して、Secrets Manager やその他の AWS タスクを実行できます。これは、コンソールを使用するよりも高速でより便利になります。コマンドラインツールは、 AWS タスクを実行するスクリプトを構築する場合に便利です。

コマンドシェルにコマンドを入力すると、コマンド履歴がアクセスされたり、ユーティリティからコマンドパラメータにアクセスされたりするリスクがあります。「the section called “を使用して AWS Secrets Manager シークレット AWS CLI を保存するリスクを軽減する”」を参照してください。

コマンドラインツールは、 AWS リージョン内のサービスのデフォルトエンドポイントを自動的に使用します。API リクエストに別のエンドポイントを指定することができます。「the section called “Secrets Manager エンドポイント”」を参照してください。

AWS には、次の 2 セットのコマンドラインツールが用意されています。

- AWS Command Line Interface (AWS CLI)
- AWS Tools for Windows PowerShell

AWS SDKs

AWS SDKs は、さまざまなプログラミング言語とプラットフォーム用のライブラリとサンプルコードで構成されています。SDK には、暗号署名によるリクエスト、エラーの管理、リクエストの自動再試行などのタスクが含まれます。いずれかの SDK をダウンロードしてインストールするには、「Amazon Web サービスのツール」を参照してください。

AWS SDKs、 AWS リージョンのサービスのデフォルトエンドポイントを自動的に使用します。API リクエストに別のエンドポイントを指定することができます。「the section called “Secrets Manager エンドポイント”」を参照してください。

SDK ドキュメントについては、以下を参照してください。

- C++
- Go
- Java
- JavaScript
- Kotlin
- .NET
- PHP
- Python (Boto3)
- Ruby
- Rust
- SAP ABAP
- Swift

HTTPS クエリ API

HTTPS クエリ API を使用すると、Secrets Manager と にプログラムでアクセスできます AWS。HTTPS クエリ API を使用すると、サービスに HTTPS リクエストを直接発行できます。

Secrets Manager HTTPS クエリ API を直接呼び出すこともできますが、いずれかの SDK を使用することをお勧めします。SDK は、本来は手動で行う必要がある多くの便利なタスクを実行します。例えば、SDK は自動的にリクエストに署名し、該当言語で構文的に適切な構造になるようレスポンスを変換します。

Secrets Manager に対し HTTPS 呼び出しを行うには、??? に接続します。

AWS Secrets Manager エンドポイント

プログラムにより Secrets Manager に接続するには、エンドポイント (サービスのエントリポイントの URL) を使用します。Secrets Manager エンドポイントは、IPv4 と IPv6 の両方をサポートするデュアルスタックのエンドポイントです。

Secrets Manager では、連邦情報処理規格 (FIPS、Federal Information Processing Standard) 140-2 をサポートするエンドポイントが、一部のリージョンで利用可能です。

Secrets Manager は TLS 1.2 および 1.3 をサポートしています。Secrets Manager は、中国リージョンを除くすべてのリージョンで PQTLS をサポートしています。

> **Note**
>
> Python AWS SDK と は IPv6 と IPv4 を順番に呼び AWS CLI 出そうとするため、IPv6 を有効にしていない場合、呼び出しがタイムアウトして IPv4 で再試行されるまでに時間がかかることがあります。この問題に対処するには、IPv6 を完全に無効にするか、IPv6 に移行します。

Secrets Manager のサービスエンドポイントは次のとおりです。この命名は、一般的なデュアルスタックの命名規則とは異なることに注意してください。Secrets Manager でデュアルスタックアドレス指定を使用する方法については、「IPv4 および IPv6 アクセス」を参照してください。

リージョン名	リージョン	エンドポイント	プロトコル	
米国東部 (オハイオ)	us-east-2	secretsmanager.us-east-2.amazonaws.com	HTTPS	
		secretsmanager-fips.us-east-2.amazonaws.com	HTTPS	
米国東部 (バージニア北部)	us-east-1	secretsmanager.us-east-1.amazonaws.com	HTTPS	
		secretsmanager-fips.us-east-1.amazonaws.com	HTTPS	

リージョン名	リージョン	エンドポイント	プロトコル	
米国西部 (北カリフォルニア)	us-west-1	secretsmanager.us-west-1.amazonaws.com secretsmanager-fips.us-west-1.amazonaws.com	HTTPS HTTPS	
米国西部 (オレゴン)	us-west-2	secretsmanager.us-west-2.amazonaws.com secretsmanager-fips.us-west-2.amazonaws.com	HTTPS HTTPS	
アフリカ (ケープタウン)	af-south-1	secretsmanager.af-south-1.amazonaws.com	HTTPS	
アジアパシフィック (香港)	ap-east-1	secretsmanager.ap-east-1.amazonaws.com	HTTPS	
アジアパシフィック (ハイデラバード)	ap-south-2	secretsmanager.ap-south-2.amazonaws.com	HTTPS	
アジアパシフィック (ジャカルタ)	ap-southeast-3	secretsmanager.ap-southeast-3.amazonaws.com	HTTPS	
アジアパシフィック (マレーシア)	ap-southeast-5	secretsmanager.ap-southeast-5.amazonaws.com	HTTPS	

リージョン名	リージョン	エンドポイント	プロトコル	
アジアパシフィック (メルボルン)	ap-southeast-4	secretsmanager.ap-southeast-4.amazonaws.com	HTTPS	
アジアパシフィック (ムンバイ)	ap-south-1	secretsmanager.ap-south-1.amazonaws.com	HTTPS	
アジアパシフィック (ニュージーランド)	ap-southeast-6	secretsmanager.ap-southeast-6.amazonaws.com	HTTPS	
アジアパシフィック (大阪)	ap-northeast-3	secretsmanager.ap-northeast-3.amazonaws.com	HTTPS	
アジアパシフィック (ソウル)	ap-northeast-2	secretsmanager.ap-northeast-2.amazonaws.com	HTTPS	
アジアパシフィック (シンガポール)	ap-southeast-1	secretsmanager.ap-southeast-1.amazonaws.com	HTTPS	
アジアパシフィック (シドニー)	ap-southeast-2	secretsmanager.ap-southeast-2.amazonaws.com	HTTPS	

リージョン名	リージョン	エンドポイント	プロトコル	
アジアパシフィック (台北)	ap-east-2	secretsmanager.ap-east-2.amazonaws.com	HTTPS	
アジアパシフィック (タイ)	ap-southeast-7	secretsmanager.ap-southeast-7.amazonaws.com	HTTPS	
アジアパシフィック (東京)	ap-northeast-1	secretsmanager.ap-northeast-1.amazonaws.com	HTTPS	
カナダ (中部)	ca-central-1	secretsmanager.ca-central-1.amazonaws.com secretsmanager-fips.ca-central-1.amazonaws.com	HTTPS HTTPS	
カナダ西部 (カルガリー)	ca-west-1	secretsmanager.ca-west-1.amazonaws.com secretsmanager-fips.ca-west-1.amazonaws.com	HTTPS HTTPS	
欧州 (フランクフルト)	eu-central-1	secretsmanager.eu-central-1.amazonaws.com	HTTPS	
欧州 (アイルランド)	eu-west-1	secretsmanager.eu-west-1.amazonaws.com	HTTPS	
欧州 (ロンドン)	eu-west-2	secretsmanager.eu-west-2.amazonaws.com	HTTPS	
ヨーロッパ (ミラノ)	eu-south-1	secretsmanager.eu-south-1.amazonaws.com	HTTPS	

リージョン名	リージョン	エンドポイント	プロトコル	
欧州 (パリ)	eu-west-3	secretsmanager.eu-west-3.amazonaws.com	HTTPS	
欧州 (スペイン)	eu-south-2	secretsmanager.eu-south-2.amazonaws.com	HTTPS	
欧州 (ストックホルム)	eu-north-1	secretsmanager.eu-north-1.amazonaws.com	HTTPS	
欧州 (チューリッヒ)	eu-central-2	secretsmanager.eu-central-2.amazonaws.com	HTTPS	
イスラエル (テルアビブ)	il-central-1	secretsmanager.il-central-1.amazonaws.com	HTTPS	
メキシコ (中部)	mx-central-1	secretsmanager.mx-central-1.amazonaws.com	HTTPS	
中東 (バーレーン)	me-south-1	secretsmanager.me-south-1.amazonaws.com	HTTPS	
中東 (アラブ首長国連邦)	me-central-1	secretsmanager.me-central-1.amazonaws.com	HTTPS	
南米 (サンパウロ)	sa-east-1	secretsmanager.sa-east-1.amazonaws.com	HTTPS	

リージョン名	リージョン	エンドポイント	プロトコル	
AWS GovCloud (米国東部)	us-gov-east-1	secretsmanager.us-gov-east-1.amazonaws.com secretsmanager-fips.us-gov-east-1.amazonaws.com	HTTPS HTTPS	
AWS GovCloud (米国西部)	us-gov-west-1	secretsmanager.us-gov-west-1.amazonaws.com secretsmanager-fips.us-gov-west-1.amazonaws.com	HTTPS HTTPS	

AWS Secrets Manager ベストプラクティス

Secrets Manager には、独自のセキュリティポリシーを開発および実装する際に考慮が必要ないくつかのセキュリティ機能が用意されています。以下のベストプラクティスは一般的なガイドラインであり、完全なセキュリティソリューションを提供するものではありません。これらのベストプラクティスはお客様の環境に必ずしも適切または十分でない可能性があるため、処方箋ではなく、あくまで有用な検討事項とお考えください。

シークレットを保存および管理するには、次のベストプラクティスを検討してください。

- 認証情報やその他の機密情報を に保存する AWS Secrets Manager
- コード内の保護されていないシークレットを検索する
- シークレットの暗号化キーを選択する
- キャッシュを使用してシークレットを取得する
- シークレットのローテーション
- CLI を使用するリスクを軽減する
- シークレットへのアクセスを制限する
- シークレットをレプリケートする
- シークレットを監視する
- プライベートネットワークでインフラストラクチャを実行する

認証情報やその他の機密情報を に保存する AWS Secrets Manager

Secrets Manager は、セキュリティ体制とコンプライアンスを改善し、機密情報への不正アクセスのリスクを軽減するのに役立ちます。Secrets Manager は、 AWS Key Management Service () で所有および保存している暗号化キーを使用して、保管中のシークレットを暗号化しますAWS KMS。シークレットを取得すると、Secrets Manager はシークレットを復号化し、TLS 経由でローカル環境にセキュアに送信します。詳細については、「シークレットを作成する」を参照してください。

コード内の保護されていないシークレットを検索する

CodeGuru Reviewer は Secrets Manager と統合され、シークレットディテクターを使用して、コード内の保護されていないシークレットを探します。シークレットディテクターは、ハードコードされたパスワード、データベース接続文字列、ユーザー名などを検索します。詳細については、「the section called “Amazon CodeGuru Reviewer”」を参照してください。

Amazon Q は、コードベースをスキャンしてセキュリティの脆弱性やコード品質の問題を確認し、開発サイクル全体でアプリケーションの体制を改善できます。詳細については、「Amazon Q Developer ユーザーガイド」の「Amazon Q でコードをスキャン」を参照してください。

シークレットの暗号化キーを選択する

ほとんどの場合、 `aws/secretsmanager` AWS マネージドキーを使用してシークレットを暗号化することをお勧めします。この使用には費用は発生しません。

別のアカウントからシークレットにアクセスしたり、暗号化キーにキーポリシーを適用したりするには、カスタマーマネージドキーを使用してシークレットを暗号化します。

- キーポリシーで、値 `secretsmanager.<region>.amazonaws.com` を `kms:ViaService` 条件キーに割り当てます。これにより、キーの使用が Secrets Manager からのリクエストのみに制限されます。
- キーの使用をさらに制限して、正しいコンテキストを持つ Secrets Manager からのリクエストのみを使用するには、以下を作成して KMS キーを使用する条件として Secrets Manager 暗号化コンテキストのキーまたは値を使用します。
  - IAM ポリシーまたはキーポリシーの文字列条件演算子
  - 許可における権限の制約

詳細については、「the section called “シークレット暗号化と復号”」を参照してください。

キャッシュを使用してシークレットを取得する

シークレットを最も効率的に使用するには、サポートされている次のいずれかの Secrets Manager キャッシュコンポーネントを使用してシークレットをキャッシュし、必要な場合にのみ更新することをお勧めします。

- クライアント側のキャッシュを使用した Java
- クライアント側のキャッシュを使用した Python
- クライアント側のキャッシュを使用した .NET
- クライアント側のキャッシュを使用した Go
- クライアント側のキャッシュを使用した Rust
- AWS パラメータとシークレット Lambda 拡張機能
- the section called “Amazon EKS”

- を使用してthe section called “AWS ワークロード認証情報プロバイダー”、Amazon Elastic Container Service AWS Lambda、Amazon Elastic Kubernetes Service、Amazon Elastic Compute Cloud などの環境間で Secrets Manager からのシークレットの消費を標準化します。

シークレットのローテーション

シークレットを長期間変更しないと、シークレットが侵害される可能性が高くなります。Secrets Manager を使用すると、4 時間ごとに自動ローテーションを設定することができます。Secrets Manager には、シングルユーザー と 交代ユーザー の 2 つのローテーション戦略が用意されています。詳細については、「シークレットのローテーション」を参照してください。

CLI を使用するリスクを軽減する

を使用して AWS オペレーション AWS CLI を呼び出す場合は、これらのコマンドをコマンドシェルに入力します。ほとんどのコマンドシェルには、ログ記録や最後に入力したコマンドを表示する機能など、シークレットを危険にさらす可能性のある機能があります。 AWS CLI を使用して機密情報を入力する前に、必ず the section called “を使用して AWS Secrets Manager シークレット AWS CLI を保存するリスクを軽減する” を確認してください。

シークレットへのアクセスを制限する

シークレットへのアクセスを制御する IAM ポリシーステートメントでは、最小特権アクセスの原則を使用します。IAM ロールとポリシー、リソースポリシー、属性ベースのアクセス制御 (ABAC) を使用できます。詳細については、「the section called “認証とアクセスコントロール”」を参照してください。

トピック

- シークレットへの広範なアクセスをブロックする
- ポリシーの IP アドレス条件に注意する
- VPC エンドポイント条件でリクエストを制限する

シークレットへの広範なアクセスをブロックする

アクション `PutResourcePolicy` を許可するアイデンティティポリシーで
は、`BlockPublicPolicy: true` を使用することをお勧めします。この条件は、ポリシーが広範

なアクセスを許可していない場合、ユーザーがリソースポリシーのみをシークレットにアタッチできることを意味します。

Secrets Manager は、Zelkova の自動推論を使用して、広範なアクセスのためのリソースポリシーを分析します。Zelkova の詳細については、 AWS セキュリティブログの「 が自動推論 AWS を使用して大規模なセキュリティを実現する方法」を参照してください。

次の例は、BlockPublicPolicy の使用方法を示しています。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": "secretsmanager:PutResourcePolicy",
        "Resource": "arn:aws:secretsmanager:us-
east-1:123456789012:secret:secretName-AbCdEf",
    "Condition": {
            "Bool": {
                "secretsmanager:BlockPublicPolicy": "true"
            }
        }
    }
}
```

ポリシーの IP アドレス条件に注意する

Secrets Manager へのアクセスを許可または拒否するポリシーステートメントで、IP アドレス条件の演算子または aws:SourceIp 条件キーを指定するときは、注意が必要です。たとえば、企業ネットワーク IP アドレス範囲からのリクエストに AWS アクションを制限するポリシーをシークレットにアタッチすると、企業ネットワークからリクエストを呼び出す IAM ユーザーとしてのリクエストは期待どおりに機能します。ただし、Lambda 関数でローテーションを有効にする場合など、他のサービスがユーザーに代わってシークレットにアクセスできるようにすると、その関数は内部アドレス空間から Secrets Manager AWSオペレーションを呼び出します。IP アドレスのフィルターがあるポリシーによって影響されたリクエストは失敗します。

また、リクエストが Amazon VPC エンドポイントから送信されている場合、aws:sourceIP 条件キーは効果が低下します。特定の VPC エンドポイントに対するリクエストを制限するには、the section called “VPC エンドポイント条件でリクエストを制限する” を使用します。

VPC エンドポイント条件でリクエストを制限する

特定の VPC または VPC エンドポイントからのリクエストに対するアクセスを許可または拒否するには、aws:SourceVpc を使用して、指定された VPC からのリクエストに対するアクセスを制限するか、aws:SourceVpce を使用して、指定された VPC エンドポイントからのリクエストに対するアクセスを制限します。「the section called “例: アクセス許可と VPC”」を参照してください。

- aws:SourceVpc は、指定した VPC からのリクエストにアクセスを制限します。
- aws:SourceVpce は、指定した VPC エンドポイントからのリクエストにアクセスを制限します。

これらの条件キーをシークレットポリシーステートメントで使用し、Secrets Manager シークレットへのアクセスを許可または拒否すると、ユーザーに代わってシークレットへのアクセスに Secrets Manager を使用しているサービスへのアクセスを、意図せずに拒否してしまうことがあります。一部の AWS サービスのみが VPC 内のエンドポイントで実行できます。シークレットのリクエストを VPC または VPC エンドポイントに制限すると、サービスに設定されていないサービスからの Secrets Manager への呼び出しは失敗します。

「the section called “VPC エンドポイント (AWS PrivateLink)”」を参照してください。

シークレットをレプリケートする

Secrets Manager は、回復力またはディザスタリカバリ要件を満たすために、シークレットを複数の AWS リージョンに自動的にレプリケートできます。詳細については、「マルチリージョンレプリケーション」を参照してください。

シークレットを監視する

Secrets Manager では、 AWS ログ記録、モニタリング、通知サービスとの統合を通じて、シークレットを監査およびモニタリングできます。詳細については、以下を参照してください。

- the section called “で口グ記録する AWS CloudTrail ”
- the section called “CloudWatch を使用して監視する”

- the section called “シークレットのコンプライアンスを監視する”
- the section called “Secrets Manager のコストを監視する”
- the section called “GuardDuty で脅威を検出する”

プライベートネットワークでインフラストラクチャを実行する

パブリックインターネットからアクセスできないプライベートネットワーク上で、できるだけ多くのインフラストラクチャを実行することをお勧めします。VPC と Secrets Manager とのプライベート接続は、インターフェイス VPC エンドポイントを作成すると、確立できます。詳細については、「the section called “VPC エンドポイント (AWS PrivateLink)”」を参照してください。

セキュリティ重視のワークロードの API キーを管理する

AWS Secrets Manager は、データベース認証情報、アプリケーション認証情報、OAuth トークン、API キー、およびその他のシークレットをライフサイクル全体で管理、取得、ローテーションするのに役立ちます。このページでは、セキュリティに敏感なワークロードで API キーとサードパーティーの認証情報を管理するための規範的なガイダンスを提供します。自動ローテーションをカスタマーマネージド AWS KMS キーおよび最小特権 IAM ポリシーと組み合わせます。また、侵害された認証情報の露出ウィンドウとブラスト半径を最小限に抑えるように VPC エンドポイントアクセスを設定します。

に API キーを保存する AWS Secrets Manager

このセクションでは、最適なローテーションと取得のために API キーシークレットを構築する方法について説明します。各 API キーは、構造化された JSON 値を持つ個別のシークレットとして保存します。この構造では、アプリケーションは個々のフィールドを取得し、Secrets Manager は正しい値をローテーション関数に渡すことができます。

次の例は、サードパーティー API キーシークレットの一般的な JSON 構造を示しています。実際のフィールドはプロバイダーの要件によって異なります。保存する必要がある特定の認証情報とメタデータについては、プロバイダーのドキュメントを参照してください。

```
{
  "apiKey": "your-api-key-value",
  "apiKeyId": "key-identifier",
  "endpoint": "https://api.example.com/v1",
  "provider": "example-service"
}
```

API キーのシークレット構造の例

フィールド	値の例	目的
`apiKey`	`sk_live_abc123...`	アプリケーションがサードパーティー API による認証に使用する認証情報値。
`apiKeyId`	`key_001`	プロバイダー側のキーの識別子。ローテーション中に新しいキーを作成し、古いキーを削除するために使用します。

フィールド	値の例	目的
`endpoint`	`https://api.example.com/v1`	API エンドポイント URL。キーを使用して を保存し、取得によってアプリケーションが接続するために必要なものがすべて返されるようにします。
`provider`	`stripe`	プロバイダー名。複数のプロバイダーを処理する関数ロジックのタグ付け、フィルタリング、ローテーションに役立ちます。

IAM ポリシー条件と組織のフィルタリングをサポートするために、各シークレットにメタデータをタグ付けします。たとえば、 AWS CLI または コンソールを使用して、シークレットに所有チーム、環境、コンプライアンススコープをタグ付けできます。

```
aws secretsmanager tag-resource \
    --secret-id prod/payments/stripe-api-key \
    --tags Key=Team,Value=payments Key=Environment,Value=production \
           Key=Provider,Value=stripe Key=Compliance,Value=pci-dss
```

シークレットの作成の詳細については、「」を参照してくださいシークレットを作成する。

暗号化設定

Secrets Manager は、 AWS KMS キーを使用して保管中のすべてのシークレット値を暗号化します。セキュリティに敏感なワークロードの場合は、コンプライアンスとアクセスコントロールの要件に基づいて暗号化キーを選択します。

API キーシークレットの KMS キーオプション

キーのタイプ	どのようなときに使うか	セキュリティに関する考慮事項
AWS マネージドキー (`aws/secretsmanager`)	ほとんどのワークロードのデフォルト。追加コストやキー管理のオーバーヘッドはありません。	キーポリシーは Secrets Manager オペレーションのみに制限されており、変更することはできません。クロスアカウントアクセスには使用できません。

キーのタイプ	どのようなときに使うか	セキュリティに関する考慮事項
カスタマーマネージドキー	コンプライアンス要件 (PCI DSS、HIPAA、SOC 2、またはその他の適用可能な標準など) 。クロスアカウントシークレット共有。主要な使用状況監査要件。	キーポリシーはユーザーが制御します。復号できるプリンシパルを制限できます。シークレットとは別に、キーの削除を無効化またはスケジュールできます。を通じて独立した監査証跡を提供します。

セキュリティに敏感なワークロードの場合は、次のキーポリシー条件でカスタマーマネージドキーを使用します。

- `kms:ViaService` – Secrets Manager () から送信されるリクエストにキーの使用を制限しますsecretsmanager.<region>.amazonaws.com。
- `kms:EncryptionContext:SecretARN` – Secrets Manager の暗号化コンテキストを一致させることで、復号を特定のシークレット ARNs に制限します。
- コンプライアンス境界ごとにキーを分離する – 異なるコンプライアンススコープ (PCI と非 PCI の比較など) のシークレットに異なる AWS KMS キーを使用します。

暗号化と復号プロセスの詳細については、「」を参照してくださいthe section called “シークレット暗号化と復号”。

API キーの自動ローテーション

自動ローテーションにより、侵害された認証情報の公開期間が短縮されます。Secrets Manager は、スケジュールに従って Lambda 関数を呼び出します。関数は、プロバイダーに新しい API キーを作成し、シークレット値を更新して、古いキーを削除します。

Secrets Manager は、マネージド外部シークレットを通じて一部のサードパーティープロバイダーにマネージドローテーション機能を提供します。この機能とサポートされているプロバイダーのリストの詳細については、「」を参照してくださいthe section called “統合パートナー”。マネージドローテーションがサポートされていないプロバイダーの場合、プロバイダーの API を呼び出してキーを作成および削除するカスタム Lambda ローテーション関数を実装します。

ローテーション関数のライフサイクル

ローテーション Lambda 関数は 4 つのステップを実装します。Secrets Manager は、ステップごとに 関数を 1 回呼び出し、Stepパラメータを渡します。ステップが失敗すると、Secrets Manager は自動的にローテーション全体を再試行します。

API キーのローテーションステップ

Step	API キーのアクション	障害処理
createSecret	プロバイダー API を呼び出して新しいキーを作成します。AWSPENDINGステージングラベルを使用して、新しいキー値を Secrets Manager に保存します。	キーの作成が失敗した場合、ローテーションは次のステップに移行しません。既存のキーは としてアクティブのままですAWSCURRENT 。
setSecret	プロバイダーで作成された API キーの場合、このステップは通常 no-op です。このステップは、ランダムキーが Secrets Manager で生成され、一般的な API キーフローではないプロバイダーで設定する必要がある場合に使用されます。	このステップが失敗した場合、ローテーションは に進みませんtestSecret 。
testSecret	Secrets Manager からAWSPENDING 値を取得し、プロバイダーにテスト API コールを実行して、新しいキーが機能することを確認します。	テストが失敗した場合は、プロバイダーで保留中のキーを削除し、例外を発生させます。
finishSecret	新しいキーAWSCURRENT に移動します。古いキーは に移動しますAWSPREVIOUS 。必要に応じて、プロバイダーで古いキーを削除します。	ラベルの更新が失敗した場合、ローテーションは完了しません。新しいキーは存在しますが、まだというラベルが付けられていませんAWSCURRENT 。

完全なローテーション関数テンプレートと実装ガイダンスについては、「」を参照してくださいthe section called “Lambda ローテーション関数”。

ローテーションスケジュール設定

コンプライアンス要件と内部セキュリティポリシーに基づいてローテーション間隔を設定します。ワークロードに適用されるコンプライアンス標準を参照して、適切なローテーション頻度を決定します。

ローテーションウィンドウを使用して、ローテーションが発生するタイミングを制御します。これにより、ピークトラフィックまたはメンテナンスウィンドウ中にローテーションが実行されなくなります。

```
aws secretsmanager rotate-secret \
    --secret-id prod/payments/stripe-api-key \
    --rotation-rules '{
        "ScheduleExpression": "cron(0 4 ? * SUN *)",
        "Duration": "2h"
    }'
```

スケジュール式の構文については、「」を参照してくださいthe section called “ローテーションスケジュール”。

シークレットを効率的に取得する

Secrets Manager は、GetSecretValue呼び出しで 1 秒あたり 10,000 件のトランザクションをサポートします。ほとんどのアプリケーションではスロットリングは発生しません。呼び出し量やレイテンシーの影響を受けやすいパスが非常に多いアプリケーションでは、キャッシュソリューションを使用して API 呼び出しを減らし、応答時間を短縮します。

Secrets Manager は、複数の言語のキャッシュクライアントと、実行環境内でシークレットをローカルにキャッシュする Lambda 拡張機能を提供します。キャッシュオプションの詳細については、「」、the section called “クライアント側のキャッシュを使用した Python”「」、およびthe section called “クライアント側のキャッシュを使用した Java”「」を参照してくださいthe section called “クライアント側のキャッシュを使用した Go”。

すべてのアクセスパターンsecretsmanager:GetSecretValueについて、各アプリケーションが必要とする特定のシークレットに制限する IAM ポリシーを設定します。

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

```
  "Statement": [{
    "Effect": "Allow",
    "Action": "secretsmanager:GetSecretValue",
    "Resource": "arn:aws:secretsmanager:us-east-1:123456789012:secret:prod/payments/*",
    "Condition": {
      "StringEquals": {
        "aws:ResourceTag/Environment": "production"
      }
    }
  }]
}
```

機密性の高いワークロードのセキュリティ強化

以下のプラクティスは、厳格なセキュリティ要件 (PCI DSS、SOC 2、HIPAA、またはその他の該当するコンプライアンスフレームワークなど) を持つ環境の API キーを詳細に保護します。

VPC エンドポイントによるネットワークアクセスの制限

シークレットの取得がパブリックインターネットを経由しないように、Secrets Manager のインターフェイス VPC エンドポイントを作成します。エンドポイントを介してアクセスできるシークレットを制限するエンドポイントポリシーを適用します。詳細については、「the section called “VPC エンドポイント (AWS PrivateLink)”」を参照してください。

リソースポリシーをシークレットに適用する

アカウント外または特定の VPC エンドポイント外のプリンシパルからのアクセスを明示的に拒否するリソースポリシーを各シークレットにアタッチします。これにより、IAM ID ポリシーとは無関係に 2 番目の認可境界が提供されます。

でシークレットアクセスをモニタリングする

は、、`GetSecretValue`、 `PutSecretValue`を含むすべての Secrets Manager API コールを自動的にログに記録します`RotateSecret`。セキュリティに敏感なシークレットの場合は、認識されないソース IP アドレスや IAM プリンシパルからの`GetSecretValue`呼び出しなど、予期しない呼び出しでトリガーする Amazon CloudWatch アラームを作成します。

正常なローテーション**`AWSPREVIOUS`**に使用する

ローテーション中、Secrets Manager は`AWSPREVIOUS`ステージングラベルを使用して以前のキー値を維持します。新しいキーの作成時にプロバイダーが以前のキーを無効にする場合は、 が認証

エラーAWSCURRENTを返AWSPREVIOUSす場合に にフォールバックするようにアプリケーションを設定します。これにより、キーの作成からラベルの更新までの短い時間帯のダウンタイムを回避できます。

アタッチする前にリソースポリシーを検証する

リソースポリシーのないシークレットは、パブリックアクセスを既にブロックしています。シークレットにリソースポリシーをアタッチするときは、 ValidateResourcePolicy API を使用して、ポリシーが広範なパブリックアクセスを許可しないようにします。で BlockPublicPolicyパラメータを使用してPutResourcePolicy、パブリックアクセスを許可するポリシーがアタッチされないようにすることもできます。リソースポリシーで aws:PrincipalOrgID条件キーを使用して、組織外のプリンシパルからのアクセスを防止します。

よくある質問

このセクションでは、 での API キーのローテーションとシークレット管理に関する一般的な質問に回答します AWS Secrets Manager。

ローテーション中の重複期間を処理するにはどうすればよいですか?

これは、新しいキーの作成時にプロバイダーが既存のキーを無効にする場合にのみ必要です。プロバイダーが複数のアクティブなキーを同時にサポートしている場合、両方のキーはローテーション期間中にアプリケーション側のフォールバックロジックなしで機能します。古いキーを無効にするプロバイダーの場合、現在のキーが 401 または 403 エラーを返AWSPREVIOUSした場合に で再試行するようにアプリケーションを設定します。新しいキーが機能していることを確認した後にのみ、finishSecretステップのプロバイダーで古いキーを削除します。

プロバイダーがプログラムによるキー作成をサポートしていない場合はどうなりますか?

プロバイダーが手動でキーを作成する必要がある場合 (ウェブコンソール経由など) 、ローテーションを完全に自動化することはできません。代わりに、ローテーションが予定されているときに (Amazon Simple Notification Service を介して) 通知を送信するローテーション関数を使用し、オペレーターにキーを手動で作成してシークレット値を更新するように促します。コンプライアンスローテーション要件に合わせてローテーションスケジュールを設定し、 で Amazon CloudWatch アラームを使用してローテーションdays_since_last_rotationの欠落を検出します。

シークレットを取得するときに API スロットリングを回避するにはどうすればよいですか?

Secrets Manager は、 で 1 秒あたり 10,000 件のトランザクションをサポートしていますGetSecretValue。ほとんどのアプリケーションではスロットリングは発生しません。アプリケーションで非常に大量の呼び出しを行う場合は、キャッシュクライアントまたは Lambda Parameters and Secrets 拡張機能を使用します。これらはシークレット値をメモリにキャッシュし、定期的に更新して、API コールの数を減らします。キャッシュ TTL をローテーション間隔より短い値に設定すると、アプリケーションはローテーション後に新しいキーを取得します。

環境ごとに 1 つのシークレットを使用するか、バージョンで 1 つのシークレットを使用する必要がありますか?

環境ごとに個別のシークレットを使用します (例: prod/payments/stripeおよび dev/payments/stripe)。これにより、環境ごとに異なる IAM ポリシー、ローテーションスケジュール、暗号化キーが許可されます。シークレットバージョン (ステージングラベル) はローテーション状態管理用であり、環境分離用ではありません。

AWS Secrets Manager チュートリアル

トピック

- Amazon CodeGuru Reviewer を使って、コードの中の保護されていないシークレットを見つける
- ハードコードされたシークレットを に移動する AWS Secrets Manager
- ハードコードされたデータベース認証情報を に移動する AWS Secrets Manager
- Salesforce クライアントシークレットを作成する
- の交代ユーザーローテーションを設定する AWS Secrets Manager
- のシングルユーザーローテーションを設定する AWS Secrets Manager

Amazon CodeGuru Reviewer を使って、コードの中の保護されていないシークレットを見つける

Amazon CodeGuru Reviewer は、プログラム解析と機械学習により、開発者が見つけにくい潜在的な不具合を検出し、Java や Python のコードの改善案を提示するサービスです。CodeGuru Reviewer は Secrets Manager と統合し、コードにある保護されていないシークレットを見つけます。検出できるシークレットの種類については、「Amazon CodeGuru Reviewer User Guide」(Amazon CodeGuru Reviewer　ユーザーガイド) の「Types of secrets detected by CodeGuru Reviewer」(CodeGuru Reviewer で検出されるシークレットの種類) を参照してください。

ハードコードされたシークレットを見つけたら、それを置き換えるためにアクションを起こします。

- the section called “ハードコードされたデータベース認証情報の置き換え ”
- the section called “ハードコードされたシークレットの置き換え ”

ハードコードされたシークレットを に移動する AWS Secrets Manager

コード内に平文のシークレットがある場合は、ローテーションしてシークレットマネージャーに保存することをお勧めします。Secrets Managerへの移行により、コードがSecrets Managerから直接シークレットを取得するため、コードを見た人が誰でもシークレットを見ることができるという問題は解決されます。シークレットをローテーションすると、現在ハードコードされているシークレットが無効になります。

データベース認証のシークレットについては、ハードコードされたデータベース認証情報を に移動する AWS Secrets Manager を参照してください。

開始する前に、シークレットへのアクセスが必要なユーザーを決める必要があります。シークレットへのアクセス権限を管理するために、2 つの IAM ロールを使用することをお勧めします。

- 組織のシークレットを管理するロール。詳細については、「the section called “Secrets Manager 管理者のアクセス許可”」を参照してください。このロールを使用してシークレットを作成し、ローテーションします。
- 実行時に資格情報を使用できるロール、このチュートリアルでは *RoleToRetrieveSecretAtRuntime* です。コードは、このロールを担ってシークレットを取得します。このチュートリアルでは、ロールに 1 つのシークレットの値を取得する権限のみを与え、シークレットの値のリソース・ポリシーを使用して権限を付与しています。代替手段については、the section called “次の手順” を参照してください。

手順:

- ステップ 1: シークレットを作成する
- ステップ 2: コードを更新する
- ステップ 3: シークレットを更新する
- 次の手順

ステップ 1: シークレットを作成する

最初のステップは、既存のハードコードされたシークレットを、Secrets Manager にコピーすることです。シークレットが AWS リソースに関連している場合は、リソースと同じリージョンに保存します。それ以外の場合は、ユースケースに応じて最もレイテンシーの低いリージョンに保存します。

シークレットを作成するには (コンソール)

1. Secrets Manager のコンソール (https://console.aws.amazon.com/secretsmanager/) を開きます。
2. [Store a new secret] (新しいシークレットを保存する) を選択します。
3. [Choose secret type] (シークレットタイプを選択する) ページで、次の操作を行います。
    a. [Secret type] (シークレットタイプ) で、[Other type of secret] (他の種類のシークレット) を選択します。

b. シークレットのキーとバリューのペア、または平文で入力してください。例:

API key

キーと値のペアとして次を入力します。

ClientID: *client_id*

ClientSecret : *wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY*

OAuth token

プレーンテキストとして入力します。

AKIAI44QH8DHBEXAMPLE

Digital certificate

プレーンテキストとして入力します。

```
-----BEGIN CERTIFICATE-----
EXAMPLE
-----END CERTIFICATE-----
```

Private key

プレーンテキストとして入力します。

```
---- BEGIN PRIVATE KEY ----
EXAMPLE
---- END PRIVATE KEY ----
```

c. [Secrets Manager] (暗号化キー) で aws/secretsmanager を選択すると、Secrets Manager に AWS マネージドキー マネージドキーを使用します。このキーを使用してもコストは発生しません。独自のカスタマーマネージドキーを使用することもできます (例えば、別の AWS アカウントアカウントからシークレットにアクセスする場合など)。カスタマーマネージドキーの使用料金の詳細については、「料金」を参照してください。

d. [次へ] を選択します。

4. [Choose secret type] (シークレットタイプを選択する) ページで、次の操作を行います。

a. わかりやすいシークレット名と説明を入力します。

b. [Resource permissions] (リソースのアクセス許可) で、[Edit permissions] (アクセス許可の編集) を選択します。*RoleToRetrieveSecretAtRuntime* がシークレットを取得できるように以下のポリシーを貼り付け、[保存]を選択します。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS":
 "arn:aws:iam::111122223333:role/RoleToRetrieveSecretAtRuntime"
            },
            "Action": "secretsmanager:GetSecretValue",
            "Resource": "*"
        }
    ]
}
```

c. ページの最下部にある [Next] (次へ) を選択します。

5. [Configure rotation] (ローテーションを設定する) ページで、回転をオフにしておきます。[次へ] を選択します。
6. [Review] (レビュー) ページで、シークレットの詳細を確認し、[Store] (保存) を選択します。

ステップ 2: コードを更新する

コードは、シークレットを取得できるように、IAMロール *RoleToRetrieveSecretAtRuntime* を想定している必要があります。詳細については、「IAM ロールへの切り替え (AWS API)」を参照してください。

次に、Secrets Manager が提供するサンプルコードを使用して、Secrets Manager からシークレットを取得するようにコードを更新します。

サンプルコードを見るには

1. Secrets Manager コンソール (https://console.aws.amazon.com/secretsmanager/) を開きます。
2. [Secrets] (シークレット) ページで、自分のシークレットを選択します。

3. [サンプルコード] を下にスクロールします。プログラミング言語を選択し、コードスニペットをコピーします。

アプリケーションで、ハードコードされたシークレットを削除し、コードスニペットを貼り付けます。コード言語によっては、スニペットの中に関数やメソッドの呼び出しを追加する必要があります。

ハードコードされたシークレットを使用して、アプリケーションが期待通りに動作することをテストしてください。

ステップ 3: シークレットを更新する

最後のステップは、ハードコードされたシークレットを失効させ、更新することです。シークレットの発行元を参照し、シークレットの取り消しや更新の手順を確認します。例えば、現在のシークレットを無効にして、新しいシークレットを生成する必要がある場合があります。

シークレットを新しい値で更新するには

1. Secrets Manager のコンソール (https://console.aws.amazon.com/secretsmanager/) を開きます。
2. [Secrets] (シークレット)で、シークレットを選択します。
3. [Secret details] (シークレットの詳細) ページでスクロールダウンし、[Rotation configuration] (ローテーション設定) セクションの [Edit rotation] (ローテーションの編集) を選択します。
4. シークレットを更新し、[Save] (保存) を選択します。

次に、新しいシークレットを使用して、アプリケーションが期待どおりに動作するかどうかテストします。

次の手順

コードからハードコードされたシークレットを削除した後、次に検討すべきいくつかの項目を挙げます。

- Java や Python のアプリケーションでハードコードされたシークレットを見つけるには、Amazon CodeGuru Reviewer をお勧めします。
- シークレットをキャッシュすることで、パフォーマンスを向上させ、コストを削減することができます。詳細については、「シークレットの取得」を参照してください。

- 複数のリージョンからアクセスするシークレットについては、レイテンシーを改善するためにシークレットをレプリケーションすることを検討してください。詳細については、「マルチリージョンレプリケーション」を参照してください。
- このチュートリアルでは、*RoleToRetrieveSecretAtRuntime* にシークレット値を取得する権限のみを付与しました。シークレットに関するメタデータの取得やシークレットの一覧の表示など、より多くの権限をロールに付与するには、the section called “リソースベースのポリシー” を参照してください。
- このチュートリアルでは、次のアクセス許可を付与しました。*RoleToRetrieveSecretAtRuntime* *##########*シークレットのリソースポリシーを使用します。権限を付与するその他の方法については、「the section called “アイデンティティベースのポリシー”」を参照してください。

ハードコードされたデータベース認証情報を に移動する AWS Secrets Manager

コード内に平文のデータベース認証情報がある場合は、認証情報を Secrets Manager に移動して、すぐにローテートすることをお勧めします。認証情報を Secrets Manager に移動することで、コードを見た人に認証情報が見えるという問題が解決されます。なぜなら、今後、あなたのコードは Secrets Manager から直接認証情報を取得するからです。シークレットを回転させると、パスワードが更新され、現在のハードコードされたパスワードが無効となるように取り消されます。

Amazon RDS、Amazon Redshift、および Amazon DocumentDB データベースの場合、このページの手順を使用してハードコードされた認証情報を Secrets Manager に移動させることができます。他のタイプの認証情報や他のシークレットについては、「the section called “ハードコードされたシークレットの置き換え ”」を参照してください。

開始する前に、シークレットへのアクセスが必要なユーザーを決める必要があります。シークレットへのアクセス権限を管理するために、2 つの IAM ロールを使用することをお勧めします。

- 組織のシークレットを管理するロール。詳細については、「the section called “Secrets Manager 管理者のアクセス許可”」を参照してください。このロールを使用してシークレットを作成し、ローテーションします。
- 実行時に認証情報を使用できるロール、このチュートリアルでは *RoleToRetrieveSecretAtRuntime* です。コードは、このロールを担ってシークレットを取得します。

手順:

- ステップ 1: シークレットを作成する
- ステップ 2: コードを更新する
- ステップ 3: シークレットをローテーションする
- 次の手順

ステップ 1: シークレットを作成する

最初のステップは、既存のハードコードされた認証情報を、Secrets Manager でシークレットにコピーすることです。レイテンシーを最低限に抑えるために、シークレットをデータベースと同じリージョンに保存します。

シークレットを作成する

1. Secrets Manager のコンソール (https://console.aws.amazon.com/secretsmanager/) を開きます。
2. [Store a new secret] (新しいシークレットを保存する) を選択します。
3. [Choose secret type] (シークレットタイプを選択する) ページで、次の操作を行います。

    a. [Secret type] (シークレットタイプ) で、保存するデータベース認証情報のタイプを選択します。

        - Amazon RDS データベース
        - Amazon DocumentDB データベース
        - Amazon Redshift データウェアハウス。
        - その他のタイプのシークレットについては、「ハードコードされたシークレットを置き換える」を参照してください。

    b. 認証情報を使用する場合、データベースの既存のハードコードされた認証情報を入力します。

    c. [Secrets Manager] (暗号化キー) で aws/secretsmanager を選択すると、Secrets Manager に AWS マネージドキー マネージドキーを使用します。このキーを使用してもコストは発生しません。独自のカスタマーマネージドキーを使用することもできます (例えば、別の AWS アカウントアカウントからシークレットにアクセスする場合など)。カスタマーマネージドキーの使用料金の詳細については、「料金」を参照してください。

    d. [Database] (データベース) で、データベースを選択します。

e. [次へ] を選択します。

4. [Configure secret] (シークレットを設定する) ページで、次の操作を行います。

   a. わかりやすいシークレット名と説明を入力します。

   b. [Resource permissions] (リソースのアクセス許可) で、[Edit permissions] (アクセス許可の編集) を選択します。*RoleToRetrieveSecretAtRuntime* がシークレットを取得できるように以下のポリシーを貼り付け、[保存]を選択します。

      JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS":
 "arn:aws:iam::111122223333:role/RoleToRetrieveSecretAtRuntime"
            },
            "Action": "secretsmanager:GetSecretValue",
            "Resource": "*"
        }
    ]
}
```

   c. ページの最下部にある [Next] (次へ) を選択します。

5. リポジトリの [Configure rotation] (ローテーションを設定する) ページでは、今のところローテーションはオフにしておきます。後でオンにします。[Next] (次へ) を選択します。

6. [Review] (レビュー) ページで、シークレットの詳細を確認し、[Store] (保存) を選択します。

ステップ 2: コードを更新する

コードは、シークレットを取得できるように、IAMロール *RoleToRetrieveSecretAtRuntime* を想定している必要があります。詳細については、「IAM ロールへの切り替え (AWS API)」を参照してください。

次に、Secrets Manager が提供するサンプルコードを使用して、Secrets Manager からシークレットを取得するようにコードを更新します。

サンプルコードを見るには

1. Secrets Manager コンソール (https://console.aws.amazon.com/secretsmanager/) を開きます。
2. [Secrets] (シークレット) ページで、自分のシークレットを選択します。
3. [サンプルコード] を下にスクロールします。言語を選択し、コードスニペットをコピーします。

アプリケーションで、ハードコードされた認証情報を削除し、コードスニペットを貼り付けます。コード言語によっては、スニペットの中に関数やメソッドの呼び出しを追加する必要があります。

ハードコードされた認証情報の代わりにシークレットを使用して、アプリケーションが期待通りに動作することをテストしてください。

ステップ 3: シークレットをローテーションする

最後のステップは、シークレットのローテーションによってハードコードされた認証情報を失効させることです。ローテーションとは、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、シークレットとデータベースの両方で認証情報が更新されます。Secrets Manager は、設定したスケジュールで自動的にシークレットをローテーションすることができます。

ローテーションの設定の一部として、Lambda ローテーション関数が Secrets Manager とデータベースの両方にアクセスできることを確認する必要があります。自動ローテーションをオンにすると、Secrets Manager は Lambda ローテーション機能をデータベースと同じ VPC に作成し、データベースへのネットワークアクセスができるようにします。Lambda ローテーション関数は、Secrets Manager を呼び出してシークレットを更新することも可能でなければなりません。Lambda から Secrets Manager への呼び出しが AWS インフラストラクチャを離れないように、VPC に Secrets Manager エンドポイントを作成することをお勧めします。手順については、「the section called “VPC エンドポイント (AWS PrivateLink)”」を参照してください。

ローテーションを有効にするには

1. Secrets Manager コンソール (https://console.aws.amazon.com/secretsmanager/) を開きます。
2. [Secrets] (シークレット) ページで、自分のシークレットを選択します。
3. シークレットの詳細ページで、[Rotation configuration] (ローテーション設定) セクションの [Edit rotation] (ローテーションの編集) を選択します。
4. [Edit rotation configuration] (ローテーション設定の編集) ダイアログボックスで、次の操作を行います。

a. [Automatic rotation] (自動ローテーション) を有効化します。
b. [ローテーションのスケジュール]で、UTC タイムゾーンでスケジュールを入力します。
c. シークレットを保存したときにシークレットをローテーションするには、[Rotate immediately when the secret is stored] (シークレットが保存されたときにすぐにローテーションする) を選択します。
d. ローテーション関数で、[Create a new Lambda function] (新しい Lambda 関数の作成) を選択し、新しい関数の名前を入力します。Secrets Manager は、関数名の先頭に「SecretsManager」を追加します。
e. [ローテーション戦略] で、[単一ユーザー] を選択します。
f. [保存] を選択します。

シークレットがローテーションしたことを確認するには

1. Secrets Manager のコンソール (https://console.aws.amazon.com/secretsmanager/) を開きます。
2. [Secrets] (シークレット) で、シークレットを選択します。
3. [Secret details] (シークレットの詳細) ページで、下へスクロールし、[Retrieve secret value] (シークレットの値を取得する) を選択します。

   シークレットの値が変更された場合、ローテーションは成功したことになります。シークレットの値が変更されなかった場合、ローテーション関数の CloudWatch Logs を見て ローテーションのトラブルシューティング する必要があります。

ローテーションされたシークレットを使用して、アプリケーションが期待どおりに動作するかどうかテストします。

次の手順

コードからハードコードされたシークレットを削除した後、次に検討すべきいくつかの項目を挙げます。

- シークレットをキャッシュすることで、パフォーマンスを向上させ、コストを削減することができます。詳細については、「シークレットの取得」を参照してください。
- 別のローテーションスケジュールを選択できます。詳細については、「the section called “ローテーションスケジュール”」を参照してください。

- Java や Python のアプリケーションでハードコードされたシークレットを見つけるには、Amazon CodeGuru Reviewer をお勧めします。

Salesforce クライアントシークレットを作成する

このチュートリアルでは、Salesforce クライアントシークレットを作成して にマネージド外部シークレット AWS Secrets Manager として保存し、マネージドローテーションを有効にしてから、ローテーションされた認証情報をアクセストークンと交換してテストします。このチュートリアルの所要時間は約 20 ~ 30 分です。

このチュートリアルでは AWS アカウント、Secrets Manager に保存するシークレットの料金や、使用を選択した場合はカスタマーマネージドキーなど、 に料金が発生する可能性のある AWS サービスを使用します。詳細については、「料金」を参照してください。

> **Note**
>
> このチュートリアルでは、例として SalesforceClientSecret シークレットタイプを使用します。他のマネージド外部シークレットタイプも同様のフローに従います。シークレットは、統合パートナーが定義する形式を使用して作成し、マネージドローテーションを有効にしてから、ローテーションされた認証情報を使用します。オンボーディングされたパートナーとその必要なシークレット形式のリストについては、「」を参照してください統合パートナー。

このチュートリアルを開始する前に、以下の要件を満たしていることを確認してください。

- 外部クライアントアプリケーション (ECAs) を作成できる Salesforce アカウント (開発者またはその他) 。
- への管理アクセス許可 AWS アカウント。本番稼働用設定では、タスクごとに異なるロールを使用するのがベストプラクティスです。たとえば、組織内のシークレットを管理するロールはシークレットを作成し、別のロールは実行時にシークレットを使用します。チュートリアルの各ステップでは同じアイデンティティを使用することが推奨されます。本番稼働環境で許可を設定する方法については、「the section called “認証とアクセスコントロール”」を参照してください。

このチュートリアルには、以下のステップが含まれています。

- ステップ 1: 外部クライアントアプリを作成し、OAuth を有効にする

- ステップ 2: マネージド外部シークレットを作成し、ローテーションを有効にする
- ステップ 3: ローテーションされた認証情報をテストする
- ステップ 4: リソースをクリーンアップする
- 次の手順

ステップ 1: 外部クライアントアプリを作成し、OAuth を有効にする

最初のステップは、Salesforce で外部クライアントアプリ (ECA) を作成し、Secrets Manager がクライアントシークレットをローテーションできるように OAuth 用に設定することです。Salesforce の手順の詳細については、Salesforce ドキュメントの「外部クライアントアプリで認可を設定する」を参照してください。

ECA を設定するときは、必ず以下を設定してください。

- 組織レベルで、REST API を介して外部クライアントアプリのコンシューマーシークレットへのアクセスを許可するを有効にします。この設定により、Secrets Manager は Salesforce REST API を介してコンシューマーシークレットを取得してローテーションできます。
- ECA で、OAuth スコープを有効にします APIs (api) を介してユーザーデータを管理します。
- ECA で、クライアント認証情報フローの有効化を有効にします。
- ECA で、トークン交換フローを有効にし、トークン交換フローのシークレットを要求します。

ECA を作成したら、次のステップでシークレットに保存する必要がある値を収集します。

- ECA Manager OAuth 設定のコンシューマーキー (クライアント ID) とコンシューマーシークレット (クライアントシークレット) 。
- Salesforce My Domain のベース URL。例: `https://MyDomainName.my.salesforce.com`。
- ECA のアプリ ID とコンシューマー ID。これらは、Salesforce ドキュメントの `external_client_app_identifier`および `consumer_id`フィールドに対応しています。これらの値を取得する方法については、Salesforce ドキュメントの「ステージ OAuth 認証情報」を参照してください。

シークレットで必要な各フィールドの説明については、「」を参照してくださいSalesforce クライアントシークレット。

ステップ 2: マネージド外部シークレットを作成し、ローテーションを有効にする

次に、Salesforce クライアントシークレットを Secrets Manager にマネージド外部シークレットとして保存し、マネージドローテーションを有効にします。マネージドローテーションでは、Lambda ローテーション関数を作成または維持しません。Secrets Manager はシークレットをローテーションします。

シークレットを作成してローテーションを有効にするには (コンソール)

1. https://console.aws.amazon.com/secretsmanager/ から Secrets Manager コンソールを開きます。
2. [Store a new secret] (新しいシークレットを保存する) を選択します。
3. [Choose secret type] (シークレットタイプを選択する) ページで、次の操作を行います。

   a. シークレットタイプで、Salesforce クライアントシークレットマネージド外部シークレットタイプ (SalesforceClientSecret) を選択します。

   b. Salesforce クライアントシークレットのシークレット値フィールドとして、コンシューマーキー、コンシューマーシークレット、基本 URI、アプリケーション ID、コンシューマー ID を入力します。各フィールドの説明については、「」を参照してくださいSalesforce クライアントシークレット。

   c. [Secrets Manager] (暗号化キー) で aws/secretsmanager を選択すると、Secrets Manager に AWS マネージドキー マネージドキーを使用します。このキーを使用してもコストは発生しません。独自のカスタマーマネージドキーを使用することもできます。カスタマーマネージドキーの使用料金の詳細については、「料金」を参照してください。

   d. [次へ] を選択します。
4. シークレットの設定ページで、わかりやすいシークレット名と説明を入力し、次へを選択します。
5. ローテーションの設定ページで、次の操作を行います。マネージド外部シークレットの場合、シークレットの作成中にローテーションがデフォルトで有効になります。

   a. 自動ローテーションがオンになっていることを確認します。

   b. ローテーションメタデータの API バージョン (`apiVersion`) に、Salesforce 組織の API バージョンを入力します。このフィールドは必須です。これは の形式である必要があります。v*XX.XX* は数字で、少なくとも である必要がありますv65.0。Secrets Manager で別の一連の管理認証情報を使用してシークレットを更新する場合は、管理者シークレット

ARN も指定します。各メタデータフィールドの説明については、「」を参照してくださいSalesforce クライアントシークレット。

c. シークレットローテーションのサービスアクセス許可で、新しいロールの作成を選択して、Secrets Manager がシークレットローテーションに必要なアクセス許可を持つ IAM ロールを自動的に作成します。デフォルトでは、アクセス許可はシークレットが作成された AWS リージョンの統合パートナーに限定されます。詳細については、「セキュリティおよびアクセス許可」を参照してください。

d. ローテーションスケジュールに、UTC タイムゾーンでスケジュールを入力します。たとえば、30 日ごとに自動的にローテーションします。

e. [次へ] を選択します。

6. [Review] (レビュー) ページで、シークレットの詳細を確認し、[Store] (保存) を選択します。

など、マネージド外部シークレットのローテーションを設定するその他の方法については、 AWS CLI AWS 「」を参照してくださいSecrets Manager が管理する外部シークレットのローテーション。

ステップ 3: ローテーションされた認証情報をテストする

最後のステップは、シークレット値を取得し、Salesforce アクセストークンの認証情報を交換することで、ローテーションされた認証情報が機能することを確認することです。

シークレット値を取得するには (コンソール)

1. Secrets Manager コンソール (https://console.aws.amazon.com/secretsmanager/) を開きます。
2. [Secrets] (シークレット) ページで、自分のシークレットを選択します。
3. [Secret details] (シークレットの詳細) ページで、下へスクロールし、[Retrieve secret value] (シークレットの値を取得する) を選択します。現在の `consumerKey`と `consumerSecret`の値を書き留めます。

> **Note**
>
> `consumerKey` と の`consumerSecret`値が、最初に保存した値から実際に変更されたことを確認します。変更されていない場合は、数分待ってからもう一度確認してください。まだ変更されていない場合は、ローテーションが失敗した理由の詳細を含むシークレット

のRotationFailedイベント AWS CloudTrail を確認します。詳細については、「マネージド外部シークレットがローテーションしていない」を参照してください。

次に、取得した認証情報を使用して Salesforce からアクセストークンをリクエストします。*MyDomainName* を Salesforce My ドメイン名に置き換え、*consumerKey* と *consumerSecret* を取得した値に置き換えます。

```
curl -X POST "https://MyDomainName.my.salesforce.com/services/oauth2/token" \
  -H "Content-Type: application/x-www-form-urlencoded" \
  -d "grant_type=client_credentials" \
  -d "client_id=consumerKey" \
  -d "client_secret=consumerSecret"
```

認証情報が有効な場合、Salesforce は を含む JSON レスポンスを返しますaccess_token。アクセストークンを受け取ると、ローテーションされた認証情報が機能することを確認します。

コマンドシェルにコマンドを入力すると、コマンド履歴がアクセスされたり、ユーティリティからコマンドパラメータにアクセスされたりするリスクがあります。「the section called “を使用して AWS Secrets Manager シークレット AWS CLI を保存するリスクを軽減する”」を参照してください。

ステップ 4: リソースをクリーンアップする

継続的な料金が発生しないようにするには、不要になったときにこのチュートリアルで作成したリソース、つまり Secrets Manager が作成したシークレットと IAM ローテーションロールを削除します。シークレットを削除すると、自動ローテーションも停止します。

シークレットを削除するには (コンソール)

1. Secrets Manager コンソール (https://console.aws.amazon.com/secretsmanager/) を開きます。
2. [Secrets] (シークレット) ページで、自分のシークレットを選択します。
3. [Secrets details] (シークレットの詳細) セクションで、[Actions] (アクション) を選択し、[Delete secret] (シークレットの削除) を選択します。
4. シークレットの無効化と削除のスケジュールダイアログボックスで、待機期間を入力し、削除のスケジュールを選択します。Secrets Manager はシークレットにすぐにアクセスできなくなり、復旧ウィンドウ後に削除するようにスケジュールします。詳細については、「AWS Secrets Manager シークレットを削除する」を参照してください。

ローテーションロールを削除するには (コンソール)

1. https://console.aws.amazon.com/iam/ で IAM コンソールを開き、ロールを選択します。
2. Secrets Manager がローテーション用に作成したロールを見つけ、それを選択してから、削除を選択します。詳細については、「IAM ユーザーガイド」の「ロールまたはインスタンスプロファイルを削除する」を参照してください。

次の手順

- マネージド外部シークレットと Secrets Manager が と統合するパートナーの詳細については、「」を参照してくださいAWS Secrets Manager マネージド外部シークレットを使用したサードパーティーシークレットの管理。
- Salesforce に必要なシークレット値とローテーションメタデータフィールドを確認するには、「」を参照してくださいSalesforce クライアントシークレット。
- マネージド外部シークレットのローテーションを設定するその他の方法については、「」を参照してくださいマネージド外部シークレットのローテーション。
- シークレットをキャッシュすることで、パフォーマンスを向上させ、コストを削減することができます。詳細については、「シークレットの取得」を参照してください。
- Secrets Manager の使用コストについては、「」を参照してください料金。

の交代ユーザーローテーションを設定する AWS Secrets Manager

このチュートリアルでは、データベース認証情報を含むシークレットに対して、交代ユーザーローテーションを設定する方法について学びます。交代ユーザーローテーションとは、Secrets Manager がユーザーのクローンを作成し、ユーザーの認証情報を交代で更新するローテーション戦略です。この戦略は、シークレットの高可用性が必要な場合に適しています。これは、代替ユーザーの 1 人がデータベースへの現在の認証情報を保持し、もう 1 人が更新されているためです。詳細については、「the section called “交代ユーザー”」を参照してください。

交代ユーザーローテーションを設定するには、次の 2 つのシークレットが必要です。

- ローテーションさせたい認証情報を持つ 1 つのシークレット。
- 管理者認証情報を持つ 2 番目のシークレット。

  このユーザーには、最初のユーザーのクローンを作成し、最初のユーザーのパスワードを変更する権限があります。このチュートリアルでは、Amazon RDS で管理者ユーザー用にこのシークレッ

トを作成します。Amazon RDS は、管理者パスワードのローテーションも管理します。詳細については、「the section called “マネージドローテーション”」を参照してください。

このチュートリアルの最初の部分では、現実的な環境をセットアップします。ローテーションの仕組みをご覧いただくため、このチュートリアルでは Amazon RDS MySQL データベースの例を使用します。セキュリティのため、データベースはインバウンドのインターネットアクセスを制限する VPC 内にあります。お使いのローカルコンピュータからインターネット経由でこのデータベースに接続するときは、このデータベースに接続できるが、インターネット経由での SSH 接続も可能な VPC 内のサーバー、踏み台ホストを使用します。このチュートリアルで使用する踏み台ホストは Amazon EC2 インスタンスであり、このインスタンスのセキュリティグループでは、他のタイプの接続は禁止されています。

チュートリアルを終了したら、チュートリアルからリソースをクリーンアップすることをお勧めします。本番環境では使用しないでください。

Secrets Manager のローテーションでは、 AWS Lambda 関数を使用してシークレットとデータベースを更新します。Lambda 関数を使用する場合のコストについては、「料金」を参照してください。

チュートリアル:

- アクセス許可
- 前提条件
- ステップ 1: Amazon RDS データベースユーザーを作成する
- ステップ 2: ユーザーの認証情報用のシークレットを作成する
- ステップ 3: ローテーションされたシークレットをテストする
- ステップ 4: リソースをクリーンアップする
- 次の手順

アクセス許可

このチュートリアルの前提条件として、 AWS アカウントの管理者権限が必要となります。本番稼働環境では、各ステップで異なるロールを使用するのがベストプラクティスとなります。例えば、Amazon RDS データベースの作成にはデータベース管理者権限を持つロールを使用し、VPC とセキュリティグループの設定にはネットワーク管理者権限を持つロールを使用します。チュートリアルの各ステップでは同じアイデンティティを使用することが推奨されます。

本番稼働環境で許可を設定する方法については、「the section called “認証とアクセスコントロール”」を参照してください。

前提条件

このチュートリアルでは、以下が必要になります。

- 前提条件 A: Amazon VPC
- 前提条件 B: Amazon EC2 インスタンス
- 前提条件 C: Amazon RDS データベースと、管理者認証情報の Secrets Manager シークレット
- 前提条件 D: ローカル コンピューターが EC2 インスタンスに接続できるようにする

前提条件 A: Amazon VPC

このステップでは、Amazon RDS データベースと Amazon EC2 インスタンスを起動できる VPC を作成します。後のステップで、コンピューターを使用してインターネット経由で踏み台に接続し、データベースに接続するため、VPC からのトラフィックを許可する必要があります。これを行うために、Amazon VPC はインターネットゲートウェイを VPC にアタッチし、ルートテーブルにルートを追加して、VPC の外部に向かうトラフィックがインターネットゲートウェイに送信されるようにします。

VPC 内で、Secrets Manager エンドポイントと Amazon RDS エンドポイントを作成します。後のステップで自動ローテーションを設定すると、Secrets Manager は VPC 内に Lambda ローテーション関数を作成して、データベースにアクセスできるようにします。また、Lambda ローテーション関数は Secrets Manager を呼び出してシークレットを更新し、Amazon RDS を呼び出してデータベース接続情報を取得します。VPC 内にエンドポイントを作成することで、Lambda 関数から Secrets Manager と Amazon RDS への呼び出しが AWS インフラストラクチャを離れないようにします。代わりに、VPC 内のエンドポイントにルーティングされます。

VPC を作成するには

1. Amazon VPC コンソールの https://console.aws.amazon.com/vpc/ を開いてください。
2. [Create VPC (VPC の作成)] を選択します。
3. [VPC の作成] ページで、[VPC など] を選択します。
4. [名前タグの自動生成] で、[自動生成] に **`SecretsManagerTutorial`** と入力します。
5. DNS オプションでは、**`Enable DNS hostnames`** と **`Enable DNS resolution`** の両方を選択します。

6. [Create VPC (VPC の作成)] を選択します。

VPC 内に Secrets Manager エンドポイントを作成するには

1. Amazon VPC コンソールの [エンドポイント] 下で、[エンドポイントの作成] を選択します。
2. [Endpoint Settings] (エンドポイントの設定) の [Name] (名前) に **SecretsManagerTutorialEndpoint** と入力します。
3. [Services] (サービス) で **secretsmanager** と入力して、リストをフィルタリングし、 AWS リージョンで Secrets Manager エンドポイントを選択します。米国東部 (バージニア北部) であれば、`com.amazonaws.us-east-1.secretsmanager` を選択します。
4. [VPC] で、**vpc**** **(SecretsManagerTutorial)** を選択します。
5. [Subnets] (サブネット) で、すべての [Availability Zones] (アベイラビリティーゾーン) を選択し、各アベイラビリティーゾーンに [Subnet ID] (サブネット ID) を挿入します。
6. [IP address type] (IP アドレスタイプ) で、**IPv4** を選択します。
7. [Security groups] (セキュリティグループ) で、デフォルトのセキュリティグループを選択します。
8. [Policy type] (ポリシーの種類) で、**Full access** を選択します。
9. エンドポイントの作成 を選択します。

VPC 内に Amazon RDS エンドポイントを作成するには

1. Amazon VPC コンソールの [エンドポイント] 下で、[エンドポイントの作成] を選択します。
2. [Endpoint Settings] (エンドポイントの設定) の [Name] (名前) に **RDSTutorialEndpoint** と入力します。
3. [Services] (サービス) で、**rds** と入力してリストをフィルタリングし、 AWS リージョンで Amazon RDS エンドポイントを選択します。米国東部 (バージニア北部) であれば、`com.amazonaws.us-east-1.rds` を選択します。
4. [VPC] で、**vpc**** **(SecretsManagerTutorial)** を選択します。
5. [Subnets] (サブネット) で、すべての [Availability Zones] (アベイラビリティーゾーン) を選択し、各アベイラビリティーゾーンに [Subnet ID] (サブネット ID) を挿入します。
6. [IP address type] (IP アドレスタイプ) で、**IPv4** を選択します。
7. [Security groups] (セキュリティグループ) で、デフォルトのセキュリティグループを選択します。

8. [Policy type] (ポリシーの種類) で、**Full access** を選択します。
9. エンドポイントの作成 を選択します。

前提条件 B: Amazon EC2 インスタンス

後のステップで作成する Amazon RDS データベースは VPC にあるため、それにアクセスするには踏み台ホストが必要です。踏み台ホストも VPC 内にありますが、後のステップで、セキュリティグループを構成して、ローカルコンピューターが SSH で踏み台ホストに接続できるようにします。

踏み台ホストの EC2 インスタンスを作成するには

1. Amazon EC2 コンソールの https://console.aws.amazon.com/ec2/ を開いてください。
2. [Instances] (インスタンス) を選択し、[Launch instance] (インスタンスの作成) を選択します。
3. [Name and tags] (名前とタグ) の、[Name] (名前) で、**SecretsManagerTutorialInstance** と入力します。
4. [アプリケーションと OS イメージ] で、デフォルトの **Amazon Linux 2 AMI (HMV) Kernel 5.10** をそのまま使用します。
5. [インスタンス タイプ] で、デフォルトの **t2.micro** をそのまま使用します。
6. [Key pair] (キーペア) で [Create key pair] (キーペアの作成) を選択します。

   [キーペア作成] ダイアログボックスの [キーペア名] に **SecretsManagerTutorialKeyPair** を入力し、[作成] を選択します。

   キーペアは自動的にダウンロードされます。
7. [Network settings] (ネットワーク設定) で、[Edit] (編集) を選択し、次の操作を実行します。
   a. VPCで、**vpc-**** SecretsManagerTutorial** を選択します。
   b. [Auto-assign Public IP] (自動割り当てパブリック IP) で、**Enable** を選択します。
   c. [Firewall] (ファイアウォール) の 既存のセキュリティグループ を選択します。
   d. [Common security groups] (共通のセキュリティグループ) で、**default** を選択します。
8. [インスタンスを起動] を選択します。

前提条件 C: Amazon RDS データベースと、管理者認証情報の Secrets Manager シークレット

このステップでは、Amazon RDS MySQL データベースを作成し、Amazon RDS が管理者認証情報を含むシークレットを作成するように設定します。次に、Amazon RDS が管理者シークレットのローテーションを自動的に管理します。詳細については、「マネージドローテーション」を参照してください。

データベース作成の一環として、前のステップで作成した踏み台ホストを指定します。次に、Amazon RDS は、データベースとインスタンスが相互にアクセスできるようにセキュリティグループを設定します。インスタンスにアタッチされたセキュリティグループにルールを追加して、ローカルコンピューターもインスタンスに接続できるようにします。

管理者認証情報を含む Secrets Manager シークレットを使用して Amazon RDS データベースを作成するには

1. Amazon RDS コンソールで、[Databases] (データベース) を選択します。
2. [Engine options] (エンジンオプション) セクションで、エンジンタイプとして **MySQL** を選択します。
3. [Templates] (テンプレート) セクションで、**Free tier** を選択します。
4. [Settings] (設定) セクションで、以下の手順を実行します:
   a. [DB instance identifier] (DB インスタンス識別子) に**SecretsManagerTutorial** と入力します。
   b. 認証情報設定で、 でマスター認証情報を管理する AWS Secrets Managerを選択します。
5. [Connect] (接続) セクションの [Computer resource] (コンピューターリソース) で、[EC2 computer resource] を選択し、[EC2 インスタンス] (EC2 コンピュータリソースに接続) で **SecretsManagerTutorialInstance** を選択します。
6. [データベースの作成] を選択します。

前提条件 D: ローカル コンピューターが EC2 インスタンスに接続できるようにする

このステップでは、前提条件 B で作成した EC2 インスタンスを構成して、ローカルコンピューターが EC2 インスタンスに接続できるようにします。これを行うには、Amazon RDS が Prereq C に追加したセキュリティ グループを編集して、コンピューターの IP アドレスが SSH に接続できるようにするルールを含めます。このルールにより、ローカルコンピューター (現在の IP アドレスで識別される) は、インターネット経由で SSH を使用して踏み台ホストに接続できます。

ローカルコンピューターが EC2 インスタンスに接続できるようにするには

1. Amazon EC2 コンソールの https://console.aws.amazon.com/ec2/ を開いてください。
2. EC2 インスタンスの [SecretsManager TutorialInstance] (セキュリティマネージャー チュートリアルインスタンス) の [Security] (セキュリティ) タブの [Security groups] (セキュリティグループ) で、**sg-*** (ec2-rds-X)** を選択します。
3. [Input rules] (インプットルール) で、[Edit inbound rules] (インバウンドルールを編集) を選択します。
4. [Add rule] (ルールを追加) を選択し、次の操作を行います。
   a. [Type] (タイプ) で、**SSH** を選択します。
   b. [Source type] (ソースタイプ) で **My IP** を選択します。

ステップ 1: Amazon RDS データベースユーザーを作成する

まず、シークレットにその認証情報を保存するユーザーが必要になります。ユーザーを作成するには、管理者の認証情報を使用して Amazon RDS データベースにログインします。簡単にするために、このチュートリアルでは、データベースへの完全なアクセス許可を持つユーザーを作成します。運用環境では、これは一般的ではないため、最小権限の原則に従うことをお勧めします。

データベースに接続するには、MySQL クライアントツールを使用します。このチュートリアルでは、GUI ベースのアプリケーション、MySQL Workbench を使用します。MySQL Workbench のインストール方法については、「Download MySQL Workbench」を参照してください。

データベースに接続するには、MySQL Workbench で接続構成を作成します。設定には、Amazon EC2 と Amazon RDS の両方からの情報が必要です。

MySQL Workbench でデータベースの接続を設定するには

1. MySQL Workbench で、[MySQL Connections] (MySQL 接続) の横にある (+) ボタンをクリックします。
2. [Setup New Connection] (新しい接続の設定) ダイアログボックスで、次を実行します。
   a. [Connection Name] (接続名) に、**SecretsManagerTutorial** と入力します。
   b. [Connection method] (接続方法) で、**Standard TCP/IP over SSH** を選択します。
   c. [Parameters] (パラメータ) タブで、次を実行します。

i. [SSH Hostname] (SSH ホスト名) に、Amazon EC2 インスタンスのパブリック IP アドレスを入力します。

この IP アドレスは、インスタンス [SecretsManagerTutorialInstance] を選択すると、Amazon EC2 コンソールに表示されます。[Public IPv4 DNS] の下に表示された IP アドレスをコピーします。

ii. [SSH Username] (SSH ユーザーネーム) に、**ec2-user** と入力します。

iii. [SSH Keyfile] (SSH キーファイル) で、前回の前提条件でダウンロードしたキーペアファイル [SecretsManagerTutorialKeyPair.pem] を選択します。

iv. [MySQL Hostname] (MySQL ホスト名) に、Amazon RDS エンドポイントアドレスを入力します。

このエンドポイントアドレスは、データベースインスタンス [secretsmanagertutorialdb] を選択すると、Amazon RDS コンソールに表示されます。[Endpoint] (エンドポイント) の下に表示されたアドレスをコピーします。

v. [Username] (ユーザーネーム) に、**admin** と入力します。

d. [OK] を選択してください。

管理者パスワードを取得するには

1. Amazon RDS コンソールで、データベースに移動します。
2. [Configuration] (設定) タブの [Master Credentials ARN] (マスター認証情報 ARN)で、[Manage in Secrets Manager] (Secrets Managerの管理) を選択します。

   Secrets Manager コンソールが開きます。
3. シークレットの詳細ページで、[Retrieve secret value] (シークレット値の取得) を選択します。
4. パスワードは [Secret value] (シークレット値) セクションに表示されます。

データベースユーザーを作成するには

1. MySQL Workbench で、接続 [SecretsManagerTutorial] を選択します。
2. シークレットから取得した管理者パスワードを入力します。
3. MySQL Workbenchの[Query] (クエリ) ウィンドウで、次のコマンド (強力なパスワードを含む) を入力し、[Execute] (実行) を選択します。ローテーション関数は SELECT を使用して更新されたシークレットをテストするため、**appuser** には少なくともその権限が必要です。

```
CREATE DATABASE myDB;
CREATE USER 'appuser'@'%' IDENTIFIED BY 'EXAMPLE-PASSWORD';
GRANT SELECT ON myDB . * TO 'appuser'@'%';
```

[Output] (出力) ウィンドウに、コマンドが実行されたことが表示されます。

ステップ 2: ユーザーの認証情報用のシークレットを作成する

次に、いま作成したユーザーの認証情報を保存するための、シークレットを作成します。これが、後ほどローテーションするシークレットです。自動ローテーションをオンにし、交代ユーザー戦略を指定するために、1 人目のユーザーのパスワードを変更する権限を持った、別のスーパーユーザーのシークレットを選択します。

1. Secrets Manager のコンソール (https://console.aws.amazon.com/secretsmanager/) を開きます。
2. [Store a new secret] (新しいシークレットを保存する) を選択します。
3. [Choose secret type] (シークレットタイプを選択する) ページで、次の操作を行います。
   a. [Secret type] (シークレットタイプ) で、[Credentials for Amazon RDS database] (Amazon RDS データベースの認証情報) を選択します。
   b. [Credentials] (認証情報) で、ユーザーネーム **appuser** と、MySQL Workbench を使って作成したデータベースユーザー向けに入力したパスワードを入力します。
   c. [Database] (データベース) で、secretsmanagertutorialdb を選択します。
   d. [次へ] を選択します。
4. [Configure secret] (シークレットを設定する) ページで、[Secret name] (シークレット名) に **SecretsManagerTutorialAppuser** と入力し、[Next] (次) を選択します。
5. [Configure Routing] (ローテーションを設定する) ページで、以下を実行します。
   a. [Automatic rotation] (自動ローテーション) を有効化します。
   b. [Rotation schedule] (ローテーションスケジュール) で、[Days] (日数) のスケジュールを設定します: [Duration] (期間) の **2** 日間: **2h**。[Rotate immediately] (すぐにローテーションする) は選択したままにしておきます。
   c. [Rotation function] (ローテーション関数) で [Create a rotation function] (ローテーション関数を作成) を選択し、[Function Name] (関数名) に **tutorial-alternating-users-rotation** と入力します。

d. [ローテーション戦略] で [代替ユーザー] を選択し、[管理者認証情報シークレット] で rds!cluster... という名前のシークレットを選択します。これには、このチュートリアル **secretsmanagertutorial** で作成したデータベースの名前を含む [説明] が含まれています (例: `Secret associated with primary RDS DB instance: arn:aws:rds:Region:AccountId:db:secretsmanagertutorial`)。

e. [次へ] を選択します。

6. [Review] (確認) ページで [Store] (保存) を選択します。

   Secrets Manager はシークレットの詳細ページに戻ります。ローテーション設定のステータスは、ページの上部で確認できます。Secrets Manager では、Lambda ローテーション関数や Lambda 関数を実行する実行ロールなどのリソースを作成するときは CloudFormation を使用します。CloudFormation が終了すると、バナーがローテーションが予定されているシークレットに切り替わります。これで、最初のローテーションは完了です。

ステップ 3: ローテーションされたシークレットをテストする

シークレットがローテーションされたので、シークレットに有効な新しい認証情報が含まれていることを確認できます。シークレット内のパスワードが、元の認証情報から変更されました。

シークレットから新しいパスワードを取得するには

1. Secrets Manager のコンソール (https://console.aws.amazon.com/secretsmanager/) を開きます。
2. [Secrets] (シークレット) を選択し、シークレット **SecretsManagerTutorialAppuser** を選択します。
3. [Secret details] (シークレットの詳細) ページで、下へスクロールし、[Retrieve secret value] (シークレットの値を取得する) を選択します。
4. [Key/value] (キー/バリュー) テーブルで、**password** のシークレットの値をコピーします。

認証情報をテストするには

1. MySQL ワークベンチで、接続 [SecretsManagerTutorial] を右クリックし、[Edit Connection] (接続を編集) を選択します。
2. [Manage Server Connections] (サーバー接続を管理) ダイアログボックスで、[Username] (ユーザーネーム) に **appuser** と入力し、[Close] (閉じる) を選択します。
3. MySQL Workbench に戻り、接続 [SecretsManagerTutorial] を選択します。

4. [Open SSH Connection] (SSH 接続を開く) ダイアログボックスで、シークレットから取得したパスワードを [Password] (パスワード) に貼り付けて、[OK] をクリックします。

   認証情報が有効であれば、MySQL Workbench でデータベースの設計ページが開きます。

こちらは、シークレットのローテーションが完了したことを示しています。シークレット内の認証情報が更新されます。これが、データベースに接続するための有効なパスワードとなります。

ステップ 4: リソースをクリーンアップする

別のローテーション戦略であるシングルユーザーローテーションを試したい場合は、リソースのクリーンアップをスキップして the section called “シングルユーザーローテーション” に進みます。

それ以外の場合は、潜在的な課金を回避し、インターネットにアクセスできる EC2 インスタンスを削除するために、このチュートリアルで作成した次のリソースとその前提条件を削除します。

- Amazon RDS データベースインスタンス。手順については、「Amazon RDS ユーザーガイド」の「DB インスタンスの削除」を参照してください。
- Amazon EC2 インスタンス。手順については、「Amazon EC2 ユーザーガイド」の「インスタンスの終了」を参照してください。
- Secrets Manager シークレット `SecretsManagerTutorialAppuser`。手順については、「the section called “シークレットの削除”」を参照してください。
- Secrets Manager エンドポイント。手順については、AWS PrivateLink ガイドの「VPC エンドポイントを削除する」を参照してください。
- VPC エンドポイント 手順については、AWS PrivateLink ガイドの「VPC の削除」を参照してください。

次の手順

- アプリケーションでシークレットを取得する方法について説明します。
- その他のローテーションスケジュールについて説明します。

のシングルユーザーローテーションを設定する AWS Secrets Manager

このチュートリアルでは、データベース認証情報を含むシークレットの単一ユーザーローテーションを設定する方法を学習します。シングルユーザーローテーションは、Secrets Manager がシークレットとデータベースの両方でユーザーの認証情報を更新するローテーション戦略です。詳細については、「the section called “シングルユーザー”」を参照してください。

チュートリアルを終了したら、チュートリアルからリソースをクリーンアップすることをお勧めします。本番環境では使用しないでください。

Secrets Manager のローテーションでは、 AWS Lambda 関数を使用してシークレットとデータベースを更新します。Lambda 関数を使用する場合のコストについては、「料金」を参照してください。

目次

- アクセス許可
- 前提条件
- ステップ 1: Amazon RDS データベースユーザーを作成する
- ステップ 2: データベースユーザー認証情報のシークレットを作成する
- ステップ 3: ローテーションされたパスワードをテストする
- ステップ 4: リソースをクリーンアップする
- 次の手順

アクセス許可

このチュートリアルの前提条件として、 AWS アカウントの管理者権限が必要となります。本番稼働環境では、各ステップで異なるロールを使用するのがベストプラクティスとなります。例えば、Amazon RDS データベースの作成にはデータベース管理者権限を持つロールを使用し、VPC とセキュリティグループの設定にはネットワーク管理者権限を持つロールを使用します。チュートリアルの各ステップでは同じアイデンティティを使用することが推奨されます。

本番稼働環境で許可を設定する方法については、「the section called “認証とアクセスコントロール”」を参照してください。

前提条件

このチュートリアルの前提条件は the section called “交代ユーザーローテーション” です。最初のチュートリアルが完了した時点で、リソースのクリーンアップを実行しないでください。このチュートリアルの後、Amazon RDS データベースと、データベースの管理者認証情報を含む Secrets Manager シークレットを備えた現実的な環境ができました。データベースユーザーの認証情報を含む 2 番目のシークレットもありますが、このチュートリアルではそのシークレットを使用しません。

また、管理者認証情報を使用してデータベースに接続するための接続が、MySQL Workbench に設定されます。

ステップ 1: Amazon RDS データベースユーザーを作成する

まず、シークレットにその認証情報を保存するユーザーが必要になります。ユーザーを作成するには、シークレットに保存されている管理者認証情報を使用して Amazon RDS データベースにログインします。簡単にするために、このチュートリアルでは、データベースへの完全なアクセス許可を持つユーザーを作成します。運用環境では、これは一般的ではないため、最小権限の原則に従うことをお勧めします。

管理者パスワードを取得するには

1. Amazon RDS コンソールで、データベースに移動します。
2. [Configuration] (設定) タブの [Master Credentials ARN] (マスター認証情報 ARN)で、[Manage in Secrets Manager] (Secrets Managerの管理) を選択します。

   Secrets Manager コンソールが開きます。
3. シークレットの詳細ページで、[Retrieve secret value] (シークレット値の取得) を選択します。
4. パスワードは [Secret value] (シークレット値) セクションに表示されます。

データベースユーザーを作成するには

1. MySQL ワークベンチで、接続 [SecretsManagerTutorial] を右クリックし、[Edit Connection] (接続を編集) を選択します。
2. [Manage Server Connections] (サーバー接続を管理) ダイアログボックスで、[Username] (ユーザーネーム) に **`admin`** と入力し、[Close] (閉じる) を選択します。
3. MySQL Workbench に戻り、接続 [SecretsManagerTutorial] を選択します。

4. シークレットから取得した管理者パスワードを入力します。
5. MySQL Workbenchの[Query] (クエリ) ウィンドウで、次のコマンド (強力なパスワードを含む) を入力し、[Execute] (実行) を選択します。ローテーション関数は SELECT を使用して更新されたシークレットをテストするため、**dbuser** には少なくともその権限が必要です。

```
CREATE USER 'dbuser'@'%' IDENTIFIED BY 'EXAMPLE-PASSWORD';
GRANT SELECT ON myDB . * TO 'dbuser'@'%';
```

[Output] (出力) ウィンドウに、コマンドが実行されたことが表示されます。

ステップ 2: データベースユーザー認証情報のシークレットを作成する

次に、作成したばかりのユーザーの認証情報を格納するためのシークレットを作成し、即時ローテーションを含む自動ローテーションをオンにします。Secrets Manager はシークレットをローテーションします。これは、パスワードがプログラムによって生成されることを意味します - 誰もこの新しいパスワードを見たことはありません。ローテーションをすぐに開始すると、ローテーションが正しく設定されているかどうかを判断するのにも役立ちます。

1. https://console.aws.amazon.com/secretsmanager/ から Secrets Manager コンソールを開きます。
2. [Store a new secret] (新しいシークレットを保存する) を選択します。
3. [Choose secret type] (シークレットタイプを選択する) ページで、次の操作を行います。
   a. [Secret type] (シークレットタイプ) で、[Credentials for Amazon RDS database] (Amazon RDS データベースの認証情報) を選択します。
   b. [Credentials] (認証情報) で、ユーザーネーム **dbuser** と、MySQL Workbench を使って作成したデータベースユーザー向けに入力したパスワードを入力します。
   c. [Database] (データベース) で、secretsmanagertutorialdb を選択します。
   d. [次へ] を選択します。
4. [Configure secret] (シークレットを設定する) ページで、[Secret name] (シークレット名) に **SecretsManagerTutorialDbuser** と入力し、[Next] (次) を選択します。
5. [Configure Routing] (ローテーションを設定する) ページで、以下を実行します。
   a. [Automatic rotation] (自動ローテーション) を有効化します。

b. [Rotation schedule] (ローテーションスケジュール) で、[Days] (日数) のスケジュールを設定します: [Duration] (期間) の **2** 日間: **2h**。[Rotate immediately] (すぐにローテーションする) は選択したままにしておきます。

c. [Rotation function] (ローテーション関数) で [Create a rotation function] (ローテーション関数を作成) を選択し、[Function Name] (関数名) に **`tutorial-single-user-rotation`** と入力します。

d. [ローテーション戦略] で、[単一ユーザー] を選択します。

e. [次へ] を選択します。

6. [Review] (確認) ページで [Store] (保存) を選択します。

   Secrets Manager はシークレットの詳細ページに戻ります。ローテーション設定のステータスは、ページの上部で確認できます。Secrets Manager では、Lambda ローテーション関数や Lambda 関数を実行する実行ロールなどのリソースを作成するときは CloudFormation を使用します。CloudFormation が終了すると、バナーがローテーションが予定されているシークレットに切り替わります。これで、最初のローテーションは完了です。

ステップ 3: ローテーションされたパスワードをテストする

最初のシークレットローテーションが完了すると (所要時間は数秒)、シークレットに有効な認証情報が残っているかどうかを確認できます。シークレット内のパスワードが、元の認証情報から変更されました。

シークレットから新しいパスワードを取得するには

1. Secrets Manager のコンソール (https://console.aws.amazon.com/secretsmanager/) を開きます。
2. [Secrets] (シークレット) を選択し、シークレット **`SecretsManagerTutorialDbuser`** を選択します。
3. [Secret details] (シークレットの詳細) ページで、下へスクロールし、[Retrieve secret value] (シークレットの値を取得する) を選択します。
4. [Key/value] (キー/バリュー) テーブルで、**`password`** のシークレットの値をコピーします。

認証情報をテストするには

1. MySQL ワークベンチで、接続 [SecretsManagerTutorial] を右クリックし、[Edit Connection] (接続を編集) を選択します。

2. [Manage Server Connections] (サーバー接続を管理) ダイアログボックスで、[Username] (ユーザーネーム) に **`dbuser`** と入力し、[Close] (閉じる) を選択します。
3. MySQL Workbench に戻り、接続 [SecretsManagerTutorial] を選択します。
4. [Open SSH Connection] (SSH 接続を開く) ダイアログボックスで、シークレットから取得したパスワードを [Password] (パスワード) に貼り付けて、[OK] をクリックします。

   認証情報が有効であれば、MySQL Workbench でデータベースの設計ページが開きます。

ステップ 4: リソースをクリーンアップする

料金を発生させないために、このチュートリアルで作成したシークレットは削除します。手順については、「the section called “シークレットの削除”」を参照してください。

前のチュートリアルで作成したリソースをクリーンアップするには、the section called “ステップ 4: リソースをクリーンアップする” を参照してください。

次の手順

- アプリケーションでシークレットを取得する方法について説明します。「シークレットの取得」を参照してください。
- その他のローテーションスケジュールについて説明します。「the section called “ローテーションスケジュール”」を参照してください。

AWS Secrets Manager シークレットを作成する

シークレットは、パスワード、ユーザーネームやパスワードなどの一連の認証情報、OAuth トークン、または、暗号化された形式で Secrets Manager に保存されるその他のシークレット情報にすることができます。

> **Tip**
>
> Amazon RDS および Amazon Redshift 管理ユーザーの認証情報には、マネージドシークレットを使用することをお勧めします。管理サービスを通じてマネージドシークレットを作成すると、マネージドローテーションを使用できます。

コンソールを使用して、他のリージョンにレプリケートされているソースデータベースのデータベース認証情報を保存すると、シークレットにはソースデータベースの接続情報が含まれます。その後、シークレットをレプリケートすると、レプリカはソースシークレットのコピーとなり、同じ接続情報が含まれます。リージョン接続情報のシークレットにキー/値ペアを追加できます。

シークレットを作成するには、SecretsManagerReadWrite マネージドポリシーで付与されるアクセス許可が必要です。

Secrets Manager では、シークレットを作成すると CloudTrail ログエントリを生成します。詳細については、「the section called “でログ記録する AWS CloudTrail ”」を参照してください。

シークレットを作成するには (コンソール)

1. Secrets Manager のコンソール (https://console.aws.amazon.com/secretsmanager/) を開きます。
2. [Store a new secret] (新しいシークレットを保存する) を選択します。
3. [Choose secret type] (シークレットタイプを選択する) ページで、次の操作を行います。

    a. [Secret type] (シークレットの種類) で、次のいずれかを実行します。

    - データベース認証情報を保存するには、保存するデータベース認証情報のタイプを選択します。次に、[データベース] を選択し、[認証情報] を入力します。
    - API キー、アクセストークン、データベース用ではない認証情報を保存するには、[その他のタイプのシークレット] を選択します。

[キーと値のペア] に、JSON の キーと値 ペアでシークレットを入力するか、[プレーンテキスト] タブを開き、任意の形式でシークレットを入力します。シークレットには最大 65536 バイトまで保存できます。例:

API key

キーと値のペアとして次を入力します。

ClientID: *client_id*

ClientSecret : *wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY*

OAuth token

プレーンテキストとして入力します。

AKIAI44QH8DHBEXAMPLE

Digital certificate

プレーンテキストとして入力します。

```
-----BEGIN CERTIFICATE-----
EXAMPLE
-----END CERTIFICATE-----
```

Private key

プレーンテキストとして入力します。

```
---- BEGIN PRIVATE KEY ----
EXAMPLE
---- END PRIVATE KEY ----
```

- Secrets Manager パートナーからマネージド外部シークレットを保存するには、パートナーシークレットを選択します。次に、パートナーを選択し、パートナーのシークレットを識別する詳細を指定します。詳細については、「AWS Secrets Manager マネージド外部シークレットを使用したサードパーティーシークレットの管理」を参照してください。

b. 暗号化キーでは、Secrets Manager AWS KMS key がシークレット値を暗号化するために使用する を選択します。詳細については、「シークレット暗号化と復号」を参照してください。

- ほとんどの場合、aws/secretsmanager を選択して、Secrets Manager AWS マネージドキー の を使用します。このキーを使用してもコストは発生しません。
- 別の からシークレットにアクセスする必要がある場合 AWS アカウント、または独自の KMS キーを使用してキーポリシーを更新または適用する場合は、リストからカスタマーマネージドキーを選択するか、カスタマーマネージドキーのキー ARN または エイリアス ARN を入力するか、新しいキーを追加を選択して作成します。カスタマーマネージドキーの使用料金の詳細については、「料金」を参照してください。

  必要なもの: the section called “KMS キーのアクセス許可” クロスアカウントアクセスの詳細については、「the section called “クロスアカウントアクセス”」を参照してください。

c. [次へ] を選択します。

4. [Configure secret] (シークレットを設定する) ページで、次の操作を行います。

a. わかりやすいシークレット名と説明を入力します。シークレット名には、1～512 文字の英数字と /_+=.@- の文字を含めることができます。

b. (オプション) 外部シークレットを作成した場合は、シークレットを保持する Secrets Manager パートナーに必要なメタデータを入力します。

c. (オプション) [Tags] (タグ) セクションで、タグをシークレットに追加します。タグ付け戦略については、「the section called “ シークレットにタグ付けする”」を参照してください。機密情報は暗号化されていないため、タグに保存しないでください。

d. (オプション) [Resource permissions] (リソースに対するアクセス許可) でリソースポリシーをシークレットに追加するには、[Edit permissions] (アクセス許可の編集) をクリックします。詳細については、「the section called “リソースベースのポリシー”」を参照してください。

e. (オプション) シークレットのレプリケートで、シークレットを別のシークレットにレプリケートするには AWS リージョン、シークレットのレプリケートを選択します。シークレットのレプリケーションは、この段階で実行することも、後に戻ってきて実行することもできます。詳細については、「マルチリージョンレプリケーション」を参照してください。

f. [Next] (次へ) を選択します。

5. (オプション) [Configure rotation] (ローテーションを設定する) ページで、シークレットの自動ローテーションを有効にできます。ローテーションをオフにしておいて、後でオンにすることも

できます。詳しくは、「シークレットのローテーション」 を参照してください。[Next] (次へ) を選択します。

6. [Review] (レビュー) ページで、シークレットの詳細を確認し、[Store] (保存) を選択します。

   Secrets Manager はシークレットのリストに戻ります。新しいシークレットが表示されない場合は、更新ボタンを選択します。

AWS CLI

コマンドシェルにコマンドを入力すると、コマンド履歴がアクセスされたり、ユーティリティからコマンドパラメータにアクセスされたりするリスクがあります。「the section called “を使用して AWS Secrets Manager シークレット AWS CLI を保存するリスクを軽減する”」を参照してください。

Example JSON ファイルのデータベース認証情報からシークレットを作成する

次の `create-secret` の例は、ファイル内の認証情報からシークレットを作成します。詳細については、「 AWS CLI ユーザーガイド」の「ファイルからのパラメータのロード AWS CLI」を参照してください。

Secrets Manager がシークレットをローテーションできるようにするには、必ず JSON が シークレットの JSON 構造 にマッチしている必要があります。

```
aws secretsmanager create-secret \
    --name MyTestSecret \
    --secret-string file://mycreds.json
```

mycreds.json の内容:

```
{
    "engine": "mysql",
    "username": "saanvis",
    "password": "EXAMPLE-PASSWORD",
    "host": "my-database-endpoint.us-west-2.rds.amazonaws.com",
    "dbname": "myDatabase",
    "port": "3306"
}
```

Exampleシークレットを作成する

次に、2 つのキーと値のペアを持つシークレットを作成する、`create-secret` の例をします。

```
aws secretsmanager create-secret \
    --name MyTestSecret \
    --description "My test secret created with the CLI." \
    --secret-string '{"user":"diegor","password":"EXAMPLE-PASSWORD"}'
```

Exampleシークレットを作成する

次に、2 つのタグを持つシークレットを作成する、create-secret の例を示します。

```
aws secretsmanager create-secret \
    --name MyTestSecret \
    --description "My test secret created with the CLI." \
    --secret-string '{"user":"diegor","password":"EXAMPLE-PASSWORD"}'  \
    --tags '[{"Key": "FirstTag", "Value": "FirstValue"}, {"Key": "SecondTag", "Value":
 "SecondValue"}]'
```

AWS SDK

AWS SDKs のいずれかを使用してシークレットを作成するには、CreateSecretアクションを使用します。詳細については、「the section called “AWS SDKs”」を参照してください。

Secrets Manager シークレットの概要

Secrets Manager では、シークレットは、シークレット情報、シークレット値、およびシークレットに関するメタデータで構成されます。シークレット値には、文字列またはバイナリを使用できます。

複数の文字列値をシークレットに保存するには、キーと値のペアを使用した JSON テキスト文字列を使用することをお勧めします。次に例を示します。

```
{
  "host"       : "ProdServer-01.databases.example.com",
  "port"       : "8888",
  "username"   : "administrator",
  "password"   : "EXAMPLE-PASSWORD",
  "dbname"     : "MyDatabase",
  "engine"     : "mysql"
}
```

データベースシークレットで自動ローテーションを有効にするには、シークレットに正しい JSON 構造でデータベースの接続情報が含まれている必要があります。詳細については、「the section called “シークレットの JSON 構造”」を参照してください。

メタデータ

シークレットのメタデータには以下が含まれます。

- 次の形式の Amazon リソースネーム (ARN)

  ```
  arn:aws:secretsmanager:<Region>:<AccountId>:secret:SecretName-6RandomCharacters
  ```

  Secrets Manager は、シークレット ARN が確実に一意であるようにするのに役立つよう、シークレット名の末尾に 6 つのランダムな文字を含めます。元のシークレットが削除され、同じ名前で新しいシークレットが作成された場合、これらの文字により 2 つのシークレット ARN は異なったものとなります。ARN が異なるため、古いシークレットにアクセスできるユーザーであっても、新しいシークレットへのアクセスを自動的に取得するわけではありません。
- シークレットの名前、説明、リソースポリシー、タグ
- 暗号化キーの ARN。Secrets Manager AWS KMS key がシークレット値を暗号化および復号するために使用する です。Secrets Manager はシークレットテキストを常に暗号化された形式で保存し、転送中のシークレットを暗号化します。the section called “シークレット暗号化と復号” を参照してください。
- シークレットをローテーションする方法に関する情報 (ローテーションを設定した場合)「シークレットのローテーション」を参照してください。

Secrets Manager は、IAM アクセス許可ポリシーを使用して、認証されたユーザーのみがシークレットにアクセスまたは変更できるようにします。「の認証とアクセスコントロール AWS Secrets Manager」を参照してください。

シークレットには、暗号化されたシークレット値のコピーを保持している複数のバージョンがあります。シークレットの値を変更するか、シークレットをローテーションすると、Secrets Manager は新しいバージョンを作成します。「the section called “シークレットバージョン”」を参照してください。

シークレットをレプリケート AWS リージョン することで、複数の でシークレットを使用できます。シークレットをレプリケートする場合、元のシークレットをコピーするか、レプリカシークレットと呼ばれるプライマリシークレットを作成します。レプリカシークレットは、プライマリシーク

レットにリンクされたままになっています。「マルチリージョンレプリケーション」を参照してください

「シークレットの管理」を参照してください

シークレットバージョン

シークレットには、暗号化されたシークレット値のコピーを保持している複数のバージョンがあります。シークレットの値を変更するか、シークレットをローテーションすると、Secrets Manager は新しいバージョンを作成します。

Secrets Manager は、シークレットの履歴をバージョン順には保存しません。代わりに、以下の 3 つの特定のバージョンにラベルを付け、追跡します。

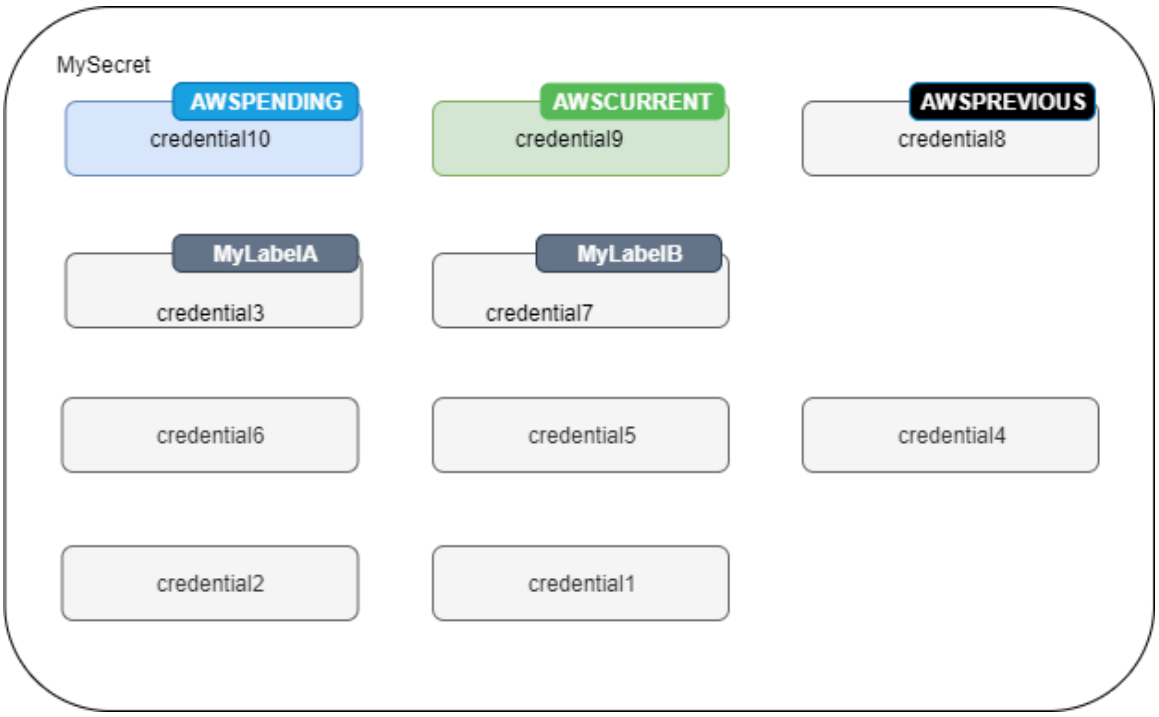
- 現在のバージョン – AWSCURRENT
- 以前のバージョン – AWSPREVIOUS
- 保留中のバージョン (ローテーション中) – AWSPENDING

シークレットには常に、AWSCURRENT というラベルが付いたバージョンがあり、シークレット値を取得する際は、そのバージョンがデフォルトで Secrets Manager から返されます。

update-secret-version-stage で を呼び出すことで、独自のラベルでバージョンにラベルを付けることもできます AWS CLI。シークレットには、最大 20 個のラベルをアタッチできます。シークレットの 2 つのバージョンで同じステージングラベルを持つことはできません。各バージョンには、複数のラベル付けることが可能です。

ラベル付きのバージョンであれば、Secrets Manager により削除されることはありませんが、ラベルのないバージョンは非推奨と見なされます。非推奨のバージョンの数が 100 を超えた場合、Secrets Manager はそれらを削除します。ただし、Secrets Manager は 24 時間以内に作成されたバージョンは削除しません。

次の図は、 AWS ラベル付きバージョンとカスタマーラベル付きバージョンを持つシークレットを示しています。ラベルのないバージョンは非推奨と見なされ、将来のある時点で Secrets Manager によって削除されます。



AWS Secrets Manager シークレットの JSON 構造

Secrets Manager シークレットには、最大サイズ 65,536 バイトまでのテキストまたはバイナリを保存できます。

the section called “Lambda 関数によるローテーション” を使用する場合、シークレットには、ローテーション関数が想定する特定の JSON フィールドが含まれている必要があります。例えば、データベース認証情報を含むシークレットの場合、ローテーション関数はデータベースに接続して認証情報を更新するため、シークレットにデータベース接続情報が含まれている必要があります。

コンソールを使用して、データベースシークレットのローテーションを編集する場合、シークレットにデータベースを識別する特定の JSON キーと値のペアが含まれている必要があります。Secrets Manager はこれらのフィールドを使用して、データベースをクエリし、ローテーション関数を保存する正しい VPC を検索します。

JSON キー名では大文字と小文字が区別されます。

トピック

- Amazon RDS と Aurora の認証情報
- Amazon Redshift 認証情報
- Amazon Redshift Serverless 認証情報

- Amazon DocumentDB 認証情報
- Amazon Timestream for InfluxDB のシークレット構造
- Amazon ElastiCache の認証情報
- Active Directory 認証情報

Amazon RDS と Aurora の認証情報

Secrets Manager が提供するローテーション関数テンプレートを使用するには、次の JSON 構造を使用します。キーと値のペアを追加して、例えば、他のリージョンのレプリカデータベースの接続情報を含めることができます。

DB2

Amazon RDS Db2 インスタンスの場合、ユーザーは自分のパスワードを変更できないため、管理者の認証情報を別のシークレットで提供する必要があります。

```
{
  "engine": "db2",
  "host": "<instance host name/resolvable DNS name>",
  "username": "<username>",
  "password": "<password>",
  "dbname": "<database name. If not specified, defaults to None>",
  "port": <TCP port number. If not specified, defaults to 3306>,
  "masterarn": "<ARN of the elevated secret>",
  "dbInstanceIdentifier": <optional: ID of the instance. Alternately, use
 dbClusterIdentifier. Required for configuring rotation in the console.>",
  "dbClusterIdentifier": <optional: ID of the cluster. Alternately, use
 dbInstanceIdentifier. Required for configuring rotation in the console.>"
}
```

MariaDB

```
{
  "engine": "mariadb",
  "host": "<instance host name/resolvable DNS name>",
  "username": "<username>",
  "password": "<password>",
  "dbname": "<database name. If not specified, defaults to None>",
  "port": <TCP port number. If not specified, defaults to 3306>,
```

```
  "masterarn": "<optional: ARN of the elevated secret. Required for the the section
 called “######”.>",
  "dbInstanceIdentifier": <optional: ID of the instance. Alternately, use
 dbClusterIdentifier. Required for configuring rotation in the console.>",
  "dbClusterIdentifier": <optional: ID of the cluster. Alternately, use
 dbInstanceIdentifier. Required for configuring rotation in the console.>"
}
```

MySQL

```
{
  "engine": "mysql",
  "host": "<instance host name/resolvable DNS name>",
  "username": "<username>",
  "password": "<password>",
  "dbname": "<database name. If not specified, defaults to None>",
  "port": <TCP port number. If not specified, defaults to 3306>,
  "masterarn": "<optional: ARN of the elevated secret. Required for the the section
 called “######”.>",
  "dbInstanceIdentifier": <optional: ID of the instance. Alternately, use
 dbClusterIdentifier. Required for configuring rotation in the console.>",
  "dbClusterIdentifier": <optional: ID of the cluster. Alternately, use
 dbInstanceIdentifier. Required for configuring rotation in the console.>"
}
```

Oracle

```
{
  "engine": "oracle",
  "host": "<instance host name/resolvable DNS name>",
  "username": "<username>",
  "password": "<password>",
  "dbname": "<database name>",
  "port": <TCP port number. If not specified, defaults to 1521>,
  "masterarn": "<optional: ARN of the elevated secret. Required for the the section
 called “######”.>",
  "dbInstanceIdentifier": <optional: ID of the instance. Alternately, use
 dbClusterIdentifier. Required for configuring rotation in the console.>",
  "dbClusterIdentifier": <optional: ID of the cluster. Alternately, use
 dbInstanceIdentifier. Required for configuring rotation in the console.>"
}
```

Postgres

```
{
  "engine": "postgres",
  "host": "<instance host name/resolvable DNS name>",
  "username": "<username>",
  "password": "<password>",
  "dbname": "<database name. If not specified, defaults to 'postgres'>",
  "port": <TCP port number. If not specified, defaults to 5432>,
  "masterarn": "<optional: ARN of the elevated secret. Required for the the section called “######”.>",
  "dbInstanceIdentifier": <optional: ID of the instance. Alternately, use dbClusterIdentifier. Required for configuring rotation in the console.>",
  "dbClusterIdentifier": <optional: ID of the cluster. Alternately, use dbInstanceIdentifier. Required for configuring rotation in the console.>"
}
```

SQLServer

```
{
  "engine": "sqlserver",
  "host": "<instance host name/resolvable DNS name>",
  "username": "<username>",
  "password": "<password>",
  "dbname": "<database name. If not specified, defaults to 'master'>",
  "port": <TCP port number. If not specified, defaults to 1433>,
  "masterarn": "<optional: ARN of the elevated secret. Required for the the section called “######”.>",
  "dbInstanceIdentifier": <optional: ID of the instance. Alternately, use dbClusterIdentifier. Required for configuring rotation in the console.>",
  "dbClusterIdentifier": <optional: ID of the cluster.Alternately, use dbInstanceIdentifier.  Required for configuring rotation in the console.>"
}
```

Amazon Redshift 認証情報

Secrets Manager が提供するローテーション関数テンプレートを使用するには、次の JSON 構造を使用します。キーと値のペアを追加して、例えば、他のリージョンのレプリカデータベースの接続情報を含めることができます。

```
{
```

```
  "engine": "redshift",
  "host": "<instance host name/resolvable DNS name>",
  "username": "<username>",
  "password": "<password>",
  "dbname": "<database name. If not specified, defaults to None>",
  "dbClusterIdentifier": "<optional: database ID. Required for configuring rotation in
 the console.>"
  "port": <optional: TCP port number. If not specified, defaults to 5439>
  "masterarn": "<optional: ARN of the elevated secret. Required for the the section
 called “######”.>"
}
```

Amazon Redshift Serverless 認証情報

Secrets Manager が提供するローテーション関数テンプレートを使用するには、次の JSON 構造を使用します。キーと値のペアを追加して、例えば、他のリージョンのレプリカデータベースの接続情報を含めることができます。

```
{
  "engine": "redshift",
  "host": "<instance host name/resolvable DNS name>",
  "username": "<username>",
  "password": "<password>",
  "dbname": "<database name. If not specified, defaults to None>",
  "namespaceName": "<optional: namespace name, Required for configuring rotation in the
 console.> "
  "port": <optional: TCP port number. If not specified, defaults to 5439>
  "masterarn": "<optional: ARN of the elevated secret. Required for the the section
 called “######”.>"
}
```

Amazon DocumentDB 認証情報

Secrets Manager が提供するローテーション関数テンプレートを使用するには、次の JSON 構造を使用します。キーと値のペアを追加して、例えば、他のリージョンのレプリカデータベースの接続情報を含めることができます。

```
{
  "engine": "mongo",
  "host": "<instance host name/resolvable DNS name>",
  "username": "<username>",
```

```
  "password": "<password>",
  "dbname": "<database name. If not specified, defaults to None>",
  "port": <TCP port number. If not specified, defaults to 27017>,
  "ssl": <true|false. If not specified, defaults to false>,
  "masterarn": "<optional: ARN of the elevated secret. Required for the the section
 called “######”.>",
  "dbClusterIdentifier": "<optional: database cluster ID. Alternately, use
 dbInstanceIdentifier. Required for configuring rotation in the console.>"
  "dbInstanceIdentifier": "<optional: database instance ID. Alternately, use
 dbClusterIdentifier. Required for configuring rotation in the console.>"
}
```

Amazon Timestream for InfluxDB のシークレット構造

Timestream シークレットをローテーションするには、the section called “Amazon Timestream for InfluxDB” ローテーションテンプレートを使用できます。

詳細については、「Amazon Timestream Developer Guide」の「How Amazon Timestream for InfluxDB uses secrets」を参照してください。

ローテーションテンプレートを使用するには、Timestream シークレットが正しい JSON 構造になっている必要があります。詳細については、「Amazon Timestream Developer Guide」の「What's in the secret」を参照してください。

Amazon ElastiCache の認証情報

次の例は、ElastiCache の認証情報を保存するシークレットの JSON 構造を示しています。

```
{
  "password": "<password>",
  "username": "<username>"
  "user_arn": "ARN of the Amazon EC2 user"
}
```

詳細については、「Amazon ElastiCache ユーザーガイド」の「Automatically rotating passwords for users」(ユーザーのパスワードの自動ローテーション) を参照してください。

Active Directory 認証情報

AWS Directory Service はシークレットを使用して Active Directory 認証情報を保存します。詳細については、「AWS Directory Service Administration Guide」の「Seamlessly join an Amazon EC2 Linux

instance to your Managed AD Active Directory」を参照してください。シームレスなドメイン参加には、次の例にあるキー名が必要です。シームレスなドメイン参加を使用しない場合は、ローテーション関数テンプレートコードで説明されているように、環境変数を使用してシークレット内のキーの名前を変更できます。

Active Directory シークレットをローテーションするには、Active Directory ローテーションテンプレートを使用できます。

Active Directory credential

```
{
  "awsSeamlessDomainUsername": "<username>",
  "awsSeamlessDomainPassword": "<password>"
}
```

シークレットをローテーションするには、ドメインディレクトリ ID を含めます。

```
{
  "awsSeamlessDomainDirectoryId": "d-12345abc6e",
  "awsSeamlessDomainUsername": "<username>",
  "awsSeamlessDomainPassword": "<password>"
}
```

シークレットをキータブを含むシークレットと組み合わせて使用する場合は、キータブシークレット ARN を含めます。

```
{
  "awsSeamlessDomainDirectoryId": "d-12345abc6e",
  "awsSeamlessDomainUsername": "<username>",
  "awsSeamlessDomainPassword": "<password>",
  "directoryServiceSecretVersion": 1,
  "schemaVersion": "1.0",
  "keytabArns": [
    "<ARN of child keytab secret 1>,
    "<ARN of child keytab secret 2>,
    "<ARN of child keytab secret 3>,
  ],
  "lastModifiedDateTime": "2021-07-19 17:06:58"
}
```

Active Directory keytab

キータブファイルを使用して Amazon EC2 の Active Directory アカウントを認証する方法については、「Deploying and configuring Active Directory authentication with SQL Server 2017 on Amazon Linux 2」を参照してください。

```
{
  "awsSeamlessDomainDirectoryId": "d-12345abc6e",
  "schemaVersion": "1.0",
  "name": "< name>",
  "principals": [
    "aduser@MY.EXAMPLE.COM",
    "MSSQLSvc/test:1433@MY.EXAMPLE.COM"
  ],
  "keytabContents": "<keytab>",
  "parentSecretArn": "<ARN of parent secret>",
  "lastModifiedDateTime": "2021-07-19 17:06:58"
  "version": 1
}
```

でシークレットを管理する AWS Secrets Manager

トピック

- AWS Secrets Manager シークレットの値を更新する
- Secrets Manager でパスワードを生成する
- シークレットを以前のバージョンにロールバックする
- AWS Secrets Manager シークレットの暗号化キーを変更する
- AWS Secrets Manager シークレットを変更する
- でシークレットを検索する AWS Secrets Manager
- AWS Secrets Manager シークレットを削除する
- AWS Secrets Manager シークレットを復元する
- でのシークレットのタグ付け AWS Secrets Manager

AWS Secrets Manager シークレットの値を更新する

シークレットの値を更新するには、コンソール、CLI、または SDK を使用できます。シークレット値を更新すると、Secrets Manager は、ステージングラベル `AWSCURRENT` 付きのシークレットの新しいバージョンを作成します。ラベル `AWSPREVIOUS` の付いた古いバージョンには引き続きアクセスできます。独自のラベルを追加することもできます。詳細については、「Secrets Manager のバージョニング」を参照してください。

シークレット値を更新するには (コンソール)

1. Secrets Manager のコンソール (https://console.aws.amazon.com/secretsmanager/) を開きます。
2. シークレットのリストから、自分のシークレットを選択します。
3. シークレットの詳細ページの [概要] タブの [シークレットの値] セクションで、[シークレットの値を取得する] を選択し、[編集] を選択します。

AWS CLI

シークレット値を更新するには (AWS CLI)

- コマンドシェルにコマンドを入力すると、コマンド履歴がアクセスされたり、ユーティリティからコマンドパラメータにアクセスされたりするリスクがあります。「the section called “を使用して AWS Secrets Manager シークレット AWS CLI を保存するリスクを軽減する”」を参照してください。

  次の `put-secret-value` は、キーと値のペア 2 つを含むシークレットの新しいバージョンを作成します。

  ```
  aws secretsmanager put-secret-value \
        --secret-id MyTestSecret \
        --secret-string "{\"user\":\"diegor\",\"password\":\"EXAMPLE-PASSWORD\"}"
  ```

  次の `put-secret-value` は、カスタムステージングラベル付きの新しいバージョンを作成します。新しいバージョンには、`MyLabel` と `AWSCURRENT` のラベルが付けられます。

  ```
  aws secretsmanager put-secret-value \
        --secret-id MyTestSecret \
        --secret-string "{\"user\":\"diegor\",\"password\":\"EXAMPLE-PASSWORD\"}"
        --version-stages "MyLabel"
  ```

AWS SDK

10 分に 1 回以上の持続頻度で `PutSecretValue` または `UpdateSecret` を呼び出すことは避けることが推奨されます。`PutSecretValue` または `UpdateSecret` を呼び出してシークレット値を更新すると、Secrets Manager はシークレットの新しいバージョンを作成します。Secrets Manager は、ラベルのないバージョンが 100 を超えると削除しますが、24 時間以内に作成されたバージョンは削除しません。10 分に 1 回以上の頻度でシークレット値を更新すると、Secrets Manager が削除した数よりも多くバージョンが作成され、シークレットバージョンのクォータに達します。

シークレット値を更新するには、`UpdateSecret` アクションまたは `PutSecretValue` アクションを使用します。詳細については、「the section called “AWS SDKs”」を参照してください。

Secrets Manager でパスワードを生成する

Secrets Manager を使用する一般的なパターンでは、Secrets Manager でパスワードを生成し、そのパスワードをデータベースまたはサービスで使用します。これを行うには、次の方法を使用します。

- CloudFormation – 「」を参照してくださいCloudFormation。
- AWS CLI – 「」を参照してくださいget-random-password。
- AWS SDKs「」を参照してくださいGetRandomPassword。

シークレットを以前のバージョンにロールバックする

AWS CLIを使用してシークレットバージョンにアタッチされたラベルを移動することで、シークレットを以前のバージョンに戻すことができます。Secrets Manager がシークレットのバージョンを保存する方法については、「the section called “シークレットバージョン”」を参照してください。

次の update-secret-version-stage の例では、AWSCURRENT ステージングラベルを以前のバージョンのシークレットに移動し、シークレットを以前のバージョンに戻します。以前のバージョンの ID を検索するには、list-secret-version-ids を使用するか、Secrets Manager コンソールでバージョンを表示します。

この例では、AWSCURRENT ラベルのバージョンは a1b2c3d4-5678-90ab-cdef-EXAMPLE11111 で、AWSPREVIOUS ラベルのバージョンは a1b2c3d4-5678-90ab-cdef-EXAMPLE22222 です。この例では、AWSCURRENT ラベルをバージョン 11111 から 22222 に移動します。AWSCURRENT ラベルはバージョンから削除されるため、update-secret-version-stage は AWSPREVIOUS ラベルをそのバージョン (11111) に自動的に移動します。その結果、AWSCURRENT と AWSPREVIOUS のバージョンが交換されます。

```
aws secretsmanager update-secret-version-stage \
  --secret-id MyTestSecret \
  --version-stage AWSCURRENT \
  --move-to-version-id a1b2c3d4-5678-90ab-cdef-EXAMPLE22222 \
  --remove-from-version-id a1b2c3d4-5678-90ab-cdef-EXAMPLE11111
```

AWS Secrets Manager シークレットの暗号化キーを変更する

Secrets Manager は AWS KMS 、キーとデータキーによるエンベロープ暗号化を使用して各シークレット値を保護します。シークレットごとに使用する KMS キーを選択できます。を使用するか AWS マネージドキー aws/secretsmanager、カスタマーマネージドキーを使用できます。ほとんど

のケースでは、aws/secretsmanager の使用をお勧めします。利用料金は発生しません。別の からシークレットにアクセスする必要がある場合 AWS アカウント、または独自の KMS キーを使用してローテーションしたり、キーポリシーを適用したりする場合は、 を使用します カスタマー管理キー。必要なもの: the section called “KMS キーのアクセス許可” カスタマーマネージドキーの使用料金の詳細については、「料金」を参照してください。

シークレットの暗号化キーを変更できます。たとえば、別のアカウントからシークレットにアクセスし、そのシークレットが現在 AWS マネージドキー を使用して暗号化されている場合はaws/secretsmanager、 に切り替えることができます カスタマー管理キー。

> **Tip**
>
> をローテーションする場合は カスタマー管理キー、 AWS KMS 自動キーローテーションを使用することをお勧めします。詳細については、「キーのローテーション」を参照してください AWS KMS。

暗号化キーを変更すると、Secrets Manager は新しいキーを使用して、AWSCURRENT、AWSPENDING、AWSPREVIOUS バージョンを再暗号化します。シークレットからロックアウトされないように、Secrets Manager は既存のすべてのバージョンを以前のキーで暗号化したままの状態にします。つまり、AWSCURRENT、AWSPENDING、AWSPREVIOUS のバージョンを以前のキーまたは新しいキーで復号化できます。以前のキーに対する kms:Decrypt アクセス許可がない場合、暗号化キーを変更すると、Secrets Manager はシークレットバージョンを復号して再暗号化することはできません。この場合、既存のバージョンは再暗号化されません。

AWSCURRENT を新しい暗号化キーでのみ復号できるようにするには、新しいキーを使用してシークレットの新しいバージョンを作成します。次に、AWSCURRENT シークレットのバージョンを復号するには、新しいキーに対するアクセス許可が必要です。

以前の暗号化キーを非アクティブ化すると、AWSCURRENT、AWSPENDING、AWSPREVIOUS 以外のシークレットバージョンを復号できなくなります。アクセスを保持する必要のある他のラベル付きシークレットバージョンがある場合、the section called “AWS CLI” を使用して新しい暗号化キーでそのバージョンを再作成する必要があります。

シークレットの暗号化キーを変更するには (コンソール)

1. Secrets Manager のコンソール (https://console.aws.amazon.com/secretsmanager/) を開きます。
2. シークレットのリストから、自分のシークレットを選択します。

3. シークレットの詳細ページの [シークレットの詳細] セクションで、[アクション] を選択して [暗号化キーの編集] を選択します。

AWS CLI

シークレットの暗号化キーを変更した後で以前の暗号化キーを非アクティブ化すると、AWSCURRENT、AWSPENDING、AWSPREVIOUS 以外のシークレットバージョンを復号できなくなります。アクセスを保持する必要のある他のラベル付きシークレットバージョンがある場合、the section called “AWS CLI” を使用して新しい暗号化キーでそのバージョンを再作成する必要があります。

シークレットの暗号化キーを変更するには (AWS CLI)

1. 次の `update-secret` の例は、シークレット値の暗号化に使用される KMS キーを更新します。KMS キーは、シークレットと同じリージョンに存在する必要があります。

```
aws secretsmanager update-secret \
      --secret-id MyTestSecret \
      --kms-key-id arn:aws:kms:us-west-2:123456789012:key/EXAMPLE1-90ab-cdef-fedc-
ba987EXAMPLE
```

2. (オプション) カスタムラベルの付いたシークレットバージョンがある場合、新しいキーを使用してそれらを再暗号化するには、そのバージョンを再作成する必要があります。

   コマンドシェルにコマンドを入力すると、コマンド履歴がアクセスされたり、ユーティリティからコマンドパラメータにアクセスされたりするリスクがあります。「the section called “を使用して AWS Secrets Manager シークレット AWS CLI を保存するリスクを軽減する”」を参照してください。

   a. シークレットバージョンの値を取得します。

```
aws secretsmanager get-secret-value \
      --secret-id MyTestSecret \
      --version-stage MyCustomLabel
```

      シークレット値を書き留めておきます。

   b. その値で新しいバージョンを作成します。

```
aws secretsmanager put-secret-value \
```

```
    --secret-id testDescriptionUpdate \
    --secret-string "SecretValue" \
    --version-stages "MyCustomLabel"
```

AWS Secrets Manager シークレットを変更する

シークレットの作成者に応じて、そのシークレットの作成後にそのメタデータを変更できます。他のサービスによって作成されたシークレットについては、他のサービスを使用して更新またはローテーションする必要がある場合があります。

シークレットを管理しているユーザーを特定するには、シークレット名を確認します。他のサービスによって管理されるシークレットには、そのサービスの ID がプレフィックスとして付けられます。または、 で describe-secret を AWS CLI呼び出し、フィールド を確認しますOwningService。詳細については、「他のサービスによって管理されるシークレット」を参照してください。

管理するシークレットについては、説明、リソースベースのポリシー、暗号化キー、およびタグを変更できます。また、暗号化されたシークレット値を変更することもできますが、ローテーションを使用して認証情報を含むシークレット値を更新することが推奨されます。ローテーションによって、Secrets Manager のシークレットと、データベースまたはサービスの認証情報の両方が更新されます。これによりシークレットが自動的に同期されるため、クライアントがシークレット値をリクエストしたとき、常に有効な認証情報を取得できます。詳細については、「 シークレットのローテーション」を参照してください。

Secrets Manager では、シークレットを編集すると CloudTrail ログエントリが生成されます。詳細については、「the section called “でログ記録する AWS CloudTrail ”」を参照してください。

管理するシークレットを更新するには (コンソール)

1. Secrets Manager のコンソール (https://console.aws.amazon.com/secretsmanager/) を開きます。
2. シークレットのリストから、自分のシークレットを選択します。
3. シークレットの詳細ページで、次のいずれかの操作を実行します。

   シークレットの名前や ARN は変更できないことに注意してください。

   - 説明を更新するには、[Secrets details] (シークレットの詳細) セクションで [Actions] (アクション) を選択し、[Edit description] (説明の編集) を選択します。
   - 暗号化キーを更新するには、「the section called “シークレットの暗号化キーを変更する”」を参照してください。

- タグを更新するには、[タグ] タブで [タグを編集] を選択します。「the section called “ シークレットにタグ付けする”」を参照してください。
- シークレット値を更新するには、「the section called “シークレット値の更新”」を参照してください。
- シークレットの許可を更新するには、[概要] タブで、[許可を編集] を選択します。「the section called “リソースベースのポリシー”」を参照してください。

- シークレットのローテーションを更新するには、[ローテーション] タブで [ローテーションを編集] を選択します。「 シークレットのローテーション」を参照してください。
- シークレットを他のリージョンにレプリケーションするには、「マルチリージョンレプリケーション」を参照してください。
- シークレットにレプリカがある場合は、レプリカの暗号化キーを変更できます。[レプリケーション] タブで、レプリカのラジオボタンを選択し、[アクション] メニューで、[暗号化キーを編集] を選択します。「the section called “シークレット暗号化と復号”」を参照してください。
- シークレットを別のサービスによって管理されるように変更するには、そのサービスでシークレットを再作成する必要があります。「他のサービスによって管理されるシークレット」を参照してください。

AWS CLI

Exampleシークレットの説明を更新する

次の `update-secret` の例では、シークレットの説明が更新されます。

```
aws secretsmanager update-secret \
    --secret-id MyTestSecret \
    --description "This is a new description for the secret."
```

AWS SDK

10 分に 1 回以上の持続頻度で PutSecretValue または UpdateSecret を呼び出すことは避けることが推奨されます。PutSecretValue または UpdateSecret を呼び出してシークレット値を更新すると、Secrets Manager はシークレットの新しいバージョンを作成します。Secrets Manager は、ラベルのないバージョンが 100 を超えると削除しますが、24 時間以内に作成されたバージョンは削除しません。10 分に 1 回以上の頻度でシークレット値を更新すると、Secrets Manager が削除した数よりも多くバージョンが作成され、シークレットバージョンのクォータに達します。

シークレットを更新するには、UpdateSecret アクションまたは ReplicateSecretToRegions アクションを使用します。詳細については、「the section called “AWS SDKs”」を参照してください。

でシークレットを検索する AWS Secrets Manager

フィルタなしでシークレットを検索すると、Secrets Manager はシークレット名、説明、タグキー、およびタグ値のキーワードを照合します。フィルタなしで検索すると、大文字と小文字は区別されず、スペース、/、_、=、# などの特殊文字は無視され、数字と文字のみが使用されます。フィルタなしで検索すると、Secrets Manager は検索文字列を分析して個別の単語に変換します。単語は、大文字から小文字、文字から数字、または数字/文字から句読点への変更によって区切られます。たとえば、名前、説明、およびタグのキーと値で、creds、Database、および 892 を検索する検索語 credsDatabase#892 を入力するとします。

Secrets Manager では、シークレットをリストアップする際に CloudTrail のログエントリーが生成されます。詳細については、「the section called “で口グ記録する AWS CloudTrail ”」を参照してください。

Secrets Manager はリージョンのサービスであり、検索は、選択されたリージョンのシークレットのみを返します。

検索フィルター

フィルターを使用しない場合、Secrets Manager は検索文字列を単語に分割し、すべての属性で一致するものを検索します。この検索では、大文字と小文字は区別されません。例えば、**My_Secret** を検索すると、名前、説明、またはタグの my または secret という単語が含まれるシークレットが一致します。

検索に次のフィルタを適用できます。

名前

シークレット名の先頭に一致します。大文字と小文字は区別されます。たとえば、名前: **Data** は、databaseSecret でも MyData でもない DatabaseSecret という名前のシークレットを返します。

説明

シークレットの説明内の単語に一致します。大文字と小文字は区別されません。たとえば、説明: **My Description** は、シークレットを次の説明で照合します。

- `My Description`
- `my description`
- `My basic description`
- `Description of my secret`

管理者

外部のサービスによって管理されるシークレットを検索します AWS。次に例を示します。

- 1Password
- Akeyless
- CyberArk
- HashiCorp

所有しているサービス

管理サービスの ID プレフィックスの先頭に一致します。大文字と小文字は区別されません。例えば、**my-ser** は、プレフィックス `my-serv` および `my-service` を持つサービスが管理するシークレットと一致します。詳細については、「他のサービスによって管理されるシークレット」を参照してください。

レプリケート:

プライマリシークレット、レプリカシークレット、またはレプリケートされないシークレットをフィルタリングできます。

タグキー

タグキーの先頭に一致します。大文字と小文字は区別されます。たとえば、タグキー: **Prod** は、タグ `Production` そして `Prod1` 付きのシークレットを返しますが、タグ `prod` または `1 Prod` 付きのシークレットは返しません。

タグ 値

タグ値の先頭に一致します。大文字と小文字は区別されます。たとえば、タグ値: **Prod** はタグ `Production` そして `Prod1` 付きのシークレットを返しますが、タグ値 `prod` または `1 Prod` を持つシークレットは返しません。

AWS CLI

Exampleアカウントにあるシークレットを一覧表示する

以下の `list-secrets` 例は、 アカウント内にあるシークレットの一覧を取得します。

```
aws secretsmanager list-secrets
```

Exampleアカウントにあるシークレットの一覧をフィルタリングする

次の `list-secrets` の例は、アカウント内にあり、名前に Test が含まれているシークレットの一覧を取得します。名前によるフィルタリングでは、大文字と小文字が区別されます。

```
aws secretsmanager list-secrets \
    --filters Key="name",Values="Test"
```

Example他の AWS サービスによって管理されているシークレットを検索する

次の `list-secrets` の例では、サービスによって管理されているシークレットの一覧を取得します。サービスは ID で指定します。詳細については、「他のサービスによって管理されるシークレット」を参照してください。

```
aws secretsmanager list-secrets \
    --filters Key="owning-service",Values="<service ID prefix>"
```

AWS SDK

AWS SDKs のいずれかを使用してシークレットを検索するには、 を使用します`ListSecrets`。詳細については、「the section called “AWS SDKs”」を参照してください。

AWS Secrets Manager シークレットを削除する

シークレットは重要な性質があるため、 AWS Secrets Manager は意図的にシークレットの削除を困難にします。Secrets Manager は、シークレットをすぐには削除しません。Secrets Manager は、シークレットをすぐにアクセス不能にし、最短で 7 日間の復旧期間が経過した後に削除されるようスケジュールを設定します。ウインドウの復旧期間が終了するまで、以前に削除したシークレットを復旧することができます。削除対象としてマークしたシークレットに対しては料金は発生しません。

プライマリシークレットが他のリージョンにレプリケートされている場合、プライマリシークレットを削除することはできません。最初にレプリカを削除してから、プライマリシークレットを削除します。レプリカを削除すると、すぐに削除されます。

シークレットのバージョンを直接削除することはできません。代わりに、 AWS CLI または AWS SDK を使用して、 バージョンからすべてのステージングラベルを削除します。これにより、その

バージョンは非推奨とマークされ、Secrets Manager はバックグラウンドでバージョンを自動的に削除できるようになります。

アプリケーションでシークレットがまだ使用されているかどうかがわからない場合は、復旧期間中にシークレットにアクセスしようとしたときに警告する Amazon CloudWatch アラームを作成します。詳しくは、「削除が予定されている AWS Secrets Manager シークレットがいつアクセスされるかをモニタリングする」を参照してください。

シークレットを削除するには、`secretsmanager:ListSecrets` と `secretsmanager:DeleteSecret` のアクセス許可が必要です。

Secrets Manager では、シークレットを削除すると CloudTrail ログエントリが生成されます。詳細については、「the section called “でログ記録する AWS CloudTrail ”」を参照してください。

シークレットを削除するには (コンソール)

1. Secrets Manager のコンソール (https://console.aws.amazon.com/secretsmanager/) を開きます。
2. シークレットのリストで、削除するシークレットを選択します。
3. [Secrets details] (シークレットの詳細) セクションで、[Actions] (アクション) を選択し、[Delete secret] (シークレットの削除) を選択します。
4. [Disable secret and schedule deletion] (シークレットの無効化と削除のスケジュール) ダイアログボックスの、[Waiting period] (待機期間) に、永続的に削除するまでの待機日数を入力します。Secrets Manager は `DeletionDate` というフィールドをアタッチし、現在の日付と時刻に、復旧期間として指定した日数を加えたものを設定します。
5. [Schedule deletion] (削除をスケジュールする) を選択します。

削除済みのシークレットを表示するには

1. https://console.aws.amazon.com/secretsmanager/ から Secrets Manager コンソールを開きます。
2. [Secrets] (シークレット) ページで、[Preferences] (設定) () を選択します。
3. [設定] ダイアログボックスで、[削除予定のシークレットを表示] を選択し、[保存] を選択します。

レプリカシークレットを削除する

1. Secrets Manager コンソール (https://console.aws.amazon.com/secretsmanager/を開く。
2. プライマリシークレットを選択します。
3. [Replicate Secret] (シークレットのレプリケーション) セクションで、レプリカのシークレットを選択します。
4. [Actions] (アクション) メニューから [Delete Replica] (レプリカの削除) を選択します。

AWS CLI

Exampleシークレットの削除

次の `delete-secret` の例では、シークレットの削除を行います。DeletionDate レスポンスフィールドで示される日時までは、`restore-secret` により、このシークレットを回復できます。他のリージョンにレプリカが作成されているシークレットを削除する場合は、まずそのレプリカを `remove-regions-from-replication` で削除してから、`delete-secret` を呼び出します。

```
aws secretsmanager delete-secret \
    --secret-id MyTestSecret \
    --recovery-window-in-days 7
```

Exampleシークレットを直ちに削除する

次の `delete-secret` の例では、復旧期間なしでシークレットを直ちに削除します。この場合のシークレットは復元できません。

```
aws secretsmanager delete-secret \
    --secret-id MyTestSecret \
    --force-delete-without-recovery
```

Exampleレプリカシークレットを削除する

次の `remove-regions-from-replication` の例では、eu-west-3 にあるレプリカシークレットを削除しています。他のリージョンにレプリカが作成されているプライマリシークレットを削除するには、まずそのレプリカを削除してから `delete-secret` を呼び出します。

```
aws secretsmanager remove-regions-from-replication \
    --secret-id MyTestSecret \
```

```
    --remove-replica-regions eu-west-3
```

AWS SDK

シークレットを削除するには、DeleteSecret コマンドを使用します。シークレットのバージョンを削除するには、UpdateSecretVersionStage コマンドを使用します。レプリカを削除するには、StopReplicationToReplica コマンドを使用します。詳しくは、「the section called “AWS SDKs”」を参照してください。

AWS Secrets Manager シークレットを復元する

Secrets Manager は、削除が予定されているシークレットを非推奨とみなし、ユーザーはこれに直接アクセスすることはできなくなります。復旧期間が過ぎると、Secrets Manager はシークレットを完全に削除します。Secrets Manager がシークレットを削除すると、復元することはできません。復旧期間が終了する前に、シークレットを復元して再度アクセス可能にすることができます。これにより、スケジュールされた完全削除をキャンセルする DeletionDate フィールドが削除されます。

コンソールでシークレットとそのメタデータを復元するには、secretsmanager:ListSecrets と secretsmanager:RestoreSecret のアクセス許可が必要です。

Secrets Manager では、シークレットを復元すると CloudTrail ログエントリを生成します。詳細については、「the section called “で口グ記録する AWS CloudTrail ”」を参照してください。

シークレットを復元するには (コンソール)

1. Secrets Manager のコンソール (https://console.aws.amazon.com/secretsmanager/) を開きます。
2. シークレットのリストで、復元するシークレットを選択します。

   削除したシークレットがシークレットのリストに表示されないときは、[Preferences] (設定) (⚙) を選択します。[設定] ダイアログボックスで、[削除予定のシークレットを表示] を選択し、[保存] を選択します。
3. [Secret details] (シークレットの詳細) ページで、[Cancel deletion] (削除のキャンセル) を選択します。
4. [Cancel secret deletion] (シークレットの削除のキャンセル) ダイアログボックスで、[Cancel deletion] (削除のキャンセル) を選択します。

AWS CLI

Example以前に削除したシークレットを復元する

次の `restore-secret` の例では、スケジュールにより以前に削除されたシークレットを復元します。

```
aws secretsmanager restore-secret \
    --secret-id MyTestSecret
```

AWS SDK

削除対象としてマークされたシークレットを復元するには、`RestoreSecret` コマンドを実行します。詳細については、「the section called “AWS SDKs”」を参照してください。

でのシークレットのタグ付け AWS Secrets Manager

では AWS Secrets Manager、タグを使用してシークレットにメタデータを割り当てることができます。タグは、シークレットに対して定義するキーと値のペアです。タグは、 AWS リソースを管理し、請求情報を含むデータを整理するのに役立ちます。

タグを使用すると、次を行えます。

- 自分の AWS アカウントでシークレットとその他のリソースを管理、検索、フィルタリングする
- アタッチされたタグに基づいてシークレットへのアクセスを制御する
- 特定のシークレットまたはプロジェクトに関連する経費を追跡および分類する

タグを使用してアクセスをコントロールする方法については、「the section called “タグを使用したシークレットへのアクセス制御”」を参照してください。

コスト配分タグの詳細については、「 AWS Billing ユーザーガイド」のAWS 「コスト配分タグの使用」を参照してください。

タグクォータと名前付けの制限については、「AWS の全般的なリファレンスガイド」の「タグ付けのサービスクォータ」を参照してください。タグは、大文字と小文字が区別します。

Secrets Manager では、シークレットのタグ付けやタグ解除を行うと、CloudTrail のログエントリーが生成されます。詳細については、「the section called “で口グ記録する AWS CloudTrail ”」を参照してください。

> **Tip**
>
> すべての AWS リソースで一貫したタグ付けスキームを使用します。ベストプラクティスについては、「タグ付けのベストプラクティス」ホワイトペーパーを参照してください。

タグの基本を確認する

コンソール、、および SDKs でシークレットを見つけることができます。 には AWS CLI、タグに基づいてリソースを統合して整理するカスタムコンソールを作成するための Resource Groups ツール AWS も用意されています。特定のタグを持つシークレットを検索するには、「the section called “シークレットを検索する”」を参照してください。

Secrets Manager コンソール AWS CLI、または Secrets Manager API を使用して、次のことができます。

- タグ付きのシークレットを作成する
- シークレットにタグを追加する
- シークレットのタグを一覧表示する
- シークレットからタグを削除する

タグを使用すると、シークレットを分類できます。たとえば、目的、所有者、環境などに基づいてシークレットを分類できます。タグごとにキーと値を定義するため、特定のニーズに合わせてカテゴリのカスタムセットを作成できます。次にいくつかのタグの例を示します。

- `Project: Project name`
- `Owner: Name`
- `Purpose: Load testing`
- `Application: Application name`
- `Environment: Production`

タグ付けを使用したコストを追跡する

タグを使用して、 AWS コストを分類および追跡できます。シークレットを含む AWS リソースにタグを適用すると、 AWS コスト配分レポートにはタグ別に集計された使用量とコストが含まれます。自社のカテゴリ、たとえばコストセンター、アプリケーション名、所有者を表すタグを適用すると、複数のサービスにわたってコストを分類することができます。詳細については、AWS Billing ユーザーガイドのコスト配分タグを使用したカスタム請求レポートを参照してください。

タグの制限を理解する

タグには次の制限があります。

基本制限

- リソース (シークレット) あたりのタグの最大数は 50 です。
- タグキーと値は大文字と小文字が区別されます。
- 削除されたシークレットのタグを変更または編集することはできません。

タグキーの制限

- 各タグキーは一意である必要があります。既に使用されているキーを持つタグを追加すると、新しいタグによって既存のキーと値のペアが上書きされます。
- このプレフィックスはユーザーに代わってこのプレフィックスで始まるタグ AWS AWS を作成するaws:ため、 でタグキーを開始することはできませんが、編集または削除することはできません。
- タグキーの長さは 1 ~ 128 文字 (Unicode) にする必要があります。
- タグキーは、次の文字で構成する必要があります。Unicode 文字、数字、空白、特殊文字 (_ . / = + - @)。

タグ値の制限

- タグ値は 0 ~ 256 Unicode 文字の長さである必要があります。
- タグ値は空白にすることができます。空白にしない場合は、次の文字で構成する必要があります。Unicode 文字、数字、空白、特殊文字 (_ . / = + - @)。

Secrets Manager コンソールを使用してシークレットにタグ付けする

Secrets Manager コンソールを使用して、シークレットのタグを管理できます。

タグ付け機能にアクセスするには、以下を実行します。

1. Secrets Manager コンソールを開きます。
2. ナビゲーションバーで、任意のリージョンを選択します。
3. [シークレット] ページで、シークレットを選択します。

シークレットのタグを表示するには

- [シークレットの詳細] ページで、[タグ] タブを選択します。

タグを使用してシークレットを作成するには

- 「シークレットを作成する」のステップを実行してください。

シークレットのタグを追加または編集するには

1. [シークレットの詳細] ページで、[タグ] タブ、[タグの編集] の順に選択します。
2. [キー] フィールドにタグキーを入力します。[値] フィールドに、必要に応じてタグ値を入力します。
3. [保存] を選択します。新規または更新されたタグがタグのリストに表示されます。

> **Note**
>
> [保存] ボタンが有効でない場合は、タグキーまたはタグ値のいずれかがタグの制限を満たしていない可能性があります。詳細については、「タグの制限を理解する」を参照してください。

シークレットからタグを削除するには

1. [シークレットの詳細] ページで、[タグ] タブを選択し、削除するタグの横にある [削除] アイコンを選択します。
2. [保存] を選択して削除を確定するか、[元に戻す] を選択してキャンセルします。

を使用してシークレットにタグを付ける AWS CLI

AWS CLI 例

Exampleシークレットにタグを追加する

次の `tag-resource` の例は、短縮構文を使用してタグをアタッチする方法を示しています。

```
aws secretsmanager tag-resource \
            --secret-id MyTestSecret \
            --tags Key=FirstTag,Value=FirstValue
```

Exampleシークレットに複数のタグを追加する

次の `tag-resource` の例では、キーと値のタグ 2 個がシークレットにアタッチされます。

```
aws secretsmanager tag-resource \
            --secret-id MyTestSecret \
```

```
            --tags '[{"Key": "FirstTag", "Value": "FirstValue"}, {"Key": "SecondTag",
 "Value": "SecondValue"}]'
```

Exampleシークレットからタグを削除する

次の `untag-resource` の例では、シークレットから 2 個のタグが削除されます。タグごとに、キーと値の両方が削除されます。

```
aws secretsmanager untag-resource \
            --secret-id MyTestSecret \
            --tag-keys '[ "FirstTag", "SecondTag"]'
```

Secrets Manager API を使用してシークレットにタグ付けする

Secrets Manager API を使用してタグの追加、一覧表示、および削除を行うことができます。例については、次のドキュメントを参照してください。

- ListSecrets: `ListSecrets` を使用して、シークレットに適用されたタグを表示します
- TagResource: シークレットにタグを追加します
- Untag: シークレットからタグを削除します

Secrets Manager AWS SDK を使用してシークレットにタグを付ける

シークレットのタグを変更するには、次の API オペレーションを使用します:

- ListSecrets: `ListSecrets` を使用して、シークレットに適用されたタグを表示します
- `TagResource`: シークレットにタグを追加します
- `UntagResource`: シークレットからタグを削除します

SDK の使用の詳細については、「the section called “AWS SDKs”」を参照してください。

リージョン間でシー AWS Secrets Manager クレットをレプリケートする

シークレットを複数の にレプリケート AWS リージョン して、それらのリージョンにまたがるアプリケーションをサポートし、リージョンアクセスと低レイテンシーの要件を満たすことができます。後で必要な場合は、レプリカシークレットをスタンドアロンに昇格させ、レプリケーション用に個別に設定できます。Secrets Manager は、指定したリージョン全体で、タグ、リソースポリシー、シークレットの更新など、暗号化されたシークレットデータおよびメタデータをレプリケートします。

複製されたシークレットの ARN は、リージョンを除いてプライマリシークレットと同じです。以下にその例を示します。

- プライマリシークレット:
  `arn:aws:secretsmanager:Region1:123456789012:secret:MySecret-a1b2c3`
- レプリカシークレット:
  `arn:aws:secretsmanager:Region2:123456789012:secret:MySecret-a1b2c3`

レプリカシークレットの料金情報については、AWS Secrets Manager の料金を参照してください。

他のリージョンにレプリケートされているソースデータベースのデータベース認証情報を保存すると、シークレットにはソースデータベースの接続情報が含まれます。その後、シークレットをレプリケートすると、レプリカはソースシークレットのコピーとなり、同じ接続情報が含まれます。リージョン接続情報のシークレットにキー/値ペアを追加できます。

プライマリシークレットのローテーションを設定すると、Secrets Manager はプライマリリージョンでシークレットのローテーションを実行し、新しいシークレット値が関連するすべてのレプリカシークレットに反映されます。すべてのレプリカシークレットのローテーションを個別に管理する必要はありません。

シークレットは、有効なすべての AWS リージョンにレプリケートできます。ただし、 AWS GovCloud (US) や中国リージョンなどの特殊な AWS リージョンで Secrets Manager を使用する場合は、これらの特殊な AWS リージョン内でのみシークレットとレプリカを設定できます。有効な AWS リージョンのシークレットを特殊なリージョンにレプリケートしたり、特殊なリージョンから商用リージョンにシークレットをレプリケートしたりすることはできません。

シークレットを別のリージョンにレプリケートするには、そのリージョンを有効にする必要があります。詳細については、「AWS リージョンの管理」を参照してください。

シークレットが保存されているリージョンで Secrets Manager エンドポイントを呼び出すことで、レプリケートせずに複数のリージョンでシークレットを使用できます。; エンドポイントのリストについては、「the section called “Secrets Manager エンドポイント”」を参照してください。レプリケーションを使用してワークロードの耐障害性を向上させるには、「 でのディザスタリカバリ (DR) アーキテクチャ AWS、パート I: クラウドでの復旧戦略」を参照してください。

Secrets Manager は、シークレットをレプリケートすると CloudTrail ログエントリを生成します。詳細については、「the section called “で口グ記録する AWS CloudTrail ”」を参照してください。

シークレットを他のリージョンにレプリケートするには (コンソール)

1. Secrets Manager コンソール (https://console.aws.amazon.com/secretsmanager/) を開きます。
2. シークレットのリストから、自分のシークレットを選択します。
3. シークレットの詳細ページの [レプリケーション] タブで、次のいずれかを実行します。
   - シークレットがレプリケートされない場合は、[Replicate secret] (シークレットをレプリケート) をクリックします。
   - シークレットがレプリケートされている場合は、[Replicate secret] (シークレットをレプリケート) セクションで、[Add Regions] (リージョンを追加) をクリックします。
4. [Add replica regions] (レプリカリージョンの追加) ダイアログボックスで、次の操作を行います。

   a. [AWS Region] (AWS リージョン) で、シークレットのレプリケート先となるリージョン選択します。

   b. (オプション) [Encryption key] (暗号化キー) で、シークレットの暗号化に使用する KMS キーを選択します。このキーは、レプリカのリージョンに存在する必要があります。

   c. (オプション) 別のリージョンを追加するには、[Add more regions] (リージョンを追加) をクリックします。

   d. [Replicate] (レプリケート) を選択します。

   シークレットの詳細ページに戻ります。[Replicate Secret] (シークレットをレプリケート) セクションで、[Replication Status] (レプリケーションステータス) がそれぞれのリージョンを表示します。

AWS CLI

Exampleシークレットを異なるリージョンにレプリケートする

次に、シークレットをeu-west-3 にレプリケートする、`replicate-secret-to-regions` の例を示します。レプリカは AWS マネージドキー で暗号化されますaws/secretsmanager。

```
aws secretsmanager replicate-secret-to-regions \
        --secret-id MyTestSecret \
        --add-replica-regions Region=eu-west-3
```

Exampleシークレットを作成し、レプリケートする

次の例ではシークレットを作成し、eu-west-3 にレプリケートします。レプリカは で暗号化されます AWS マネージドキー aws/secretsmanager。

```
aws secretsmanager create-secret \
    --name MyTestSecret \
    --description "My test secret created with the CLI." \
    --secret-string "{\"user\":\"diegor\",\"password\":\"EXAMPLE-PASSWORD\"}"
    --add-replica-regions Region=eu-west-3
```

AWS SDK

シークレットをレプリケートするには、`ReplicateSecretToRegions` コマンドを使用してください。詳細については、「the section called “AWS SDKs”」を参照してください。

レプリカシークレットを のスタンドアロンシークレットに昇格させる AWS Secrets Manager

レプリカシークレットは、別の のプライマリからレプリケートされるシークレットです AWS リージョン。プライマリと同じシークレット値とメタデータを持ちますが、別の KMS キーで暗号化できます。レプリカシークレットは、暗号化キーを除き、プライマリシークレットとは別に更新することはできません。レプリカシークレットを昇格すると、プライマリシークレットからレプリカシークレットが切断され、レプリカシークレットがスタンドアロンのシークレットになります。プライマリシークレットに加えた変更は、スタンドアロンのシークレットにレプリケーションされません。

プライマリシークレットが使用できなくなった場合、災害対策ソリューションとして、レプリカシークレットをスタンドアロンのインスタンスに昇格させることができます。または、レプリカの口ー

テーションを有効にする場合は、レプリカをスタンドアロンのシークレットに昇格させることができます。

レプリカを昇格する場合は、スタンドアロンシークレットを使用するために、必ず対応するアプリケーションを更新します。

Secrets Manager では、シークレットをプロモートすると CloudTrail ログエントリを生成します。詳細については、「the section called “で口グ記録する AWS CloudTrail ”」を参照してください。

レプリカシークレットを昇格するには (コンソール)

1. 次の場所から Secrets Manager にログインします (https://console.aws.amazon.com/secretsmanager/)。
2. レプリカのリージョンに移動します。
3. [Secrets] (シークレット) ページで、レプリカシークレットを選択します。
4. レプリカシークレットの詳細ページで、[Promote to standalone secret] (スタンドアロンシークレットに昇格させる) を選択します。
5. [Promote replica to standalone secret] (レプリカのスタンドアロンシークレットへの昇格) ダイアログボックスで、リージョンを入力してから、[Promote replica] (レプリカを昇格させる) を選択します。

AWS CLI

Exampleレプリカシークレットをプライマリに昇格させる

次の `stop-replication-to-replica` の例は、レプリカシークレットからプライマリへのリンクを削除します。このレプリカシークレットは、レプリカのリージョンでプライマリシークレットに昇格されます。`stop-replication-to-replica` は、レプリカリージョン内から呼び出す必要があります。

```
aws secretsmanager stop-replication-to-replica \
    --secret-id MyTestSecret
```

AWS SDK

レプリカをスタンドアロンシークレットに昇格させるには、`StopReplicationToReplica` コマンドを使用します。このコマンドはレプリカシークレットリージョンから呼び出す必要があります。詳細については、「the section called “AWS SDKs”」を参照してください。

AWS Secrets Manager レプリケーションの防止

シークレットは、ReplicateSecretToRegions を使用した場合、または CreateSecret を使用して作成されたときにレプリケートできるため、ユーザーがシークレットをレプリケートできないようにする場合は、AddReplicaRegions パラメータを含むアクションを禁止することを推奨します。アクセス許可ポリシーで Condition ステートメントを使用して、レプリカリージョンを追加しないアクションのみを許可できます。使用できる条件ステートメントについては、次のポリシー例を参照してください。

Exampleレプリケーション権限の防止する

次のポリシー例は、レプリカリージョンを追加しないすべてのアクションを許可する方法を示しています。これにより、ユーザーは ReplicateSecretToRegions と CreateSecret の両方を経由して、シークレットをレプリケートできなくなります。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "secretsmanager:*",
      "Resource": "*",
      "Condition": {
        "Null": {
          "secretsmanager:AddReplicaRegions": "true"
        }
      }
    }
  ]
}
```

Exampleレプリケーション権限を特定のリージョンにのみ許可する

次のポリシーは、以下のすべてを許可する方法を示しています。

- シークレットをレプリケーションなしで作成する
- 米国およびカナダリージョンのみにレプリケーションするシークレットを作成する

- シークレットを米国およびカナダのリージョンにのみレプリケートする

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "secretsmanager:CreateSecret",
        "secretsmanager:ReplicateSecretToRegions"
      ],
      "Resource": "*",
      "Condition": {
        "ForAllValues:StringLike": {
          "secretsmanager:AddReplicaRegions": [
            "us-*",
            "ca-*"
          ]
        }
      }
    }
  ]
}
```

AWS Secrets Manager レプリケーションのトラブルシューティング

AWS Secrets Manager レプリケーションは、さまざまな理由で失敗することがあります。シークレットのレプリケートに失敗した理由を確認するには、次のいずれかを実行します。

- `DescribeSecret` API オペレーションを呼び出す
- AWS CloudTrail イベントの確認

レプリケーションが失敗した場合:

- 使用可能なシークレットバージョンがない場合、Secrets Manager はレプリカリージョンからシークレットを削除します。
- 正常にレプリケートされたシークレットバージョンがある場合は、RemoveRegionsFromReplication API オペレーションを使用して明示的に削除するまで、レプリカリージョンに残ります。

以下のセクションでは、レプリケーションの失敗の一般的な理由について説明します。

選択したリージョンに同じ名前のシークレットがある

この問題を解決するには、レプリカリージョンにある重複した名前のシークレットを上書きします。レプリケーションを再試行し、[レプリケーションを再試行] ダイアログボックスで [上書き]をクリックします。

KMS キーにレプリケーションを完成させるためのアクセス許可がない

Secrets Manager は、レプリカリージョンにある新しい KMS キーを使用して再暗号化する前に、まずシークレットを復号します。プライマリリージョンの暗号化キーに対する kms:Decrypt アクセス許可がない場合、このエラーが発生します。aws/secretsmanager 以外の KMS キーでレプリケートされたシークレットを暗号化するには、キーに kms:GenerateDataKey と kms:Encrypt が必要です。「the section called “KMS キーのアクセス許可”」を参照してください。

KMS キーが無効になっているか、見つかりません

プライマリリージョンの暗号化キーが無効化または削除されている場合、Secrets Manager はシークレットをレプリケートできません。このエラーは、暗号化キーを変更した場合でも、無効化または削除された暗号化キーで暗号化されたカスタムラベルが付いたバージョンがシークレットにある場合に発生することがあります。Secrets Manager が暗号化を行う方法については、「the section called “シークレット暗号化と復号”」を参照してください。この問題を回避するには、Secrets Manager が現在の暗号化キーで暗号化するようにシークレットバージョンを再作成できます。詳細については、「シークレットの暗号化キーを変更する」を参照してください。その後でレプリケーションを再試行します。

```
aws secretsmanager put-secret-value \
  --secret-id testDescriptionUpdate \
  --secret-string "SecretValue" \
  --version-stages "MyCustomLabel"
```

レプリケーションを行うリージョンが有効化されていない

リージョンを有効化する方法については、「AWS Account Management リファレンスガイド」の「Managing AWS Regions」を参照してください。

からシークレットを取得する AWS Secrets Manager

Secrets Manager では、シークレットを取得すると CloudTrail ログエントリを生成します。詳細については、「the section called “でログ記録する AWS CloudTrail ”」を参照してください。

シークレット値を取得するには、以下のコードを使用します。

- Java を使用して Secrets Manager シークレット値を取得する
- Python を使用して Secrets Manager のシークレット値を取得する
- .NET を使用して Secrets Manager シークレット値を取得する
- Go を使用して Secrets Manager シークレット値を取得する
- Rust を使用して Secrets Manager シークレット値を取得する
- Amazon Elastic Kubernetes Service で AWS Secrets Manager シークレットを使用する
- AWS Lambda 関数で AWS Secrets Manager シークレットを使用する
- AWS ワークロード認証情報プロバイダーの使用
- AI コーディングエージェントでシークレットを安全に使用する
- C++ AWS SDK を使用して Secrets Manager シークレット値を取得する
- JavaScript AWS SDK を使用して Secrets Manager シークレット値を取得する
- Kotlin AWS SDK を使用して Secrets Manager のシークレット値を取得する
- PHP AWS SDK を使用して Secrets Manager シークレット値を取得する
- Ruby AWS SDK を使用して Secrets Manager のシークレット値を取得する
- を使用してシークレット値を取得する AWS CLI
- AWS コンソールを使用してシークレット値を取得する
- で AWS Secrets Manager シークレットを使用する AWS Batch
- CloudFormation リソースで AWS Secrets Manager シークレットを取得する
- GitHub ジョブで AWS Secrets Manager シークレットを使用する
- GitLab AWS Secrets Manager で を使用する
- で AWS Secrets Manager シークレットを使用する AWS IoT Greengrass
- パラメータストアで AWS Secrets Manager シークレットを使用する

Java を使用して Secrets Manager シークレット値を取得する

アプリケーションでは、任意の AWS SDKs `BatchGetSecretValue`で `GetSecretValue`または を呼び出してシークレットを取得できます。ただし、クライアント側のキャッシュを使用してシークレット値をキャッシュすることをお勧めします。シークレットをキャッシュすると、速度が向上し、コストが削減されます。

シークレットの認証情報を使用してデータベースに接続するには、基本 JDBC ドライバーをラップする Secrets Manager SQL Connection ドライバーを使用します。これはまた、クライアント側のキャッシュを使用するため、Secrets Manager API を呼び出すコストを削減できます。

トピック

- Java とクライアント側のキャッシュを使用して、Secrets Manager のシークレット値を取得する
- AWS Secrets Manager シークレットの認証情報を使用して JDBC を使用して SQL データベースに接続する
- Java AWS SDK を使用して Secrets Manager シークレット値を取得する

Java とクライアント側のキャッシュを使用して、Secrets Manager のシークレット値を取得する

シークレットを取得するときに、Secrets Manager の Java ベースのキャッシュコンポーネントを使用して、将来使用するためにキャッシュすることができます。キャッシュされたシークレットの取得は、Secrets Manager からの取得よりも高速です。Secrets Manager API を呼び出すにはコストがかかるため、キャッシュを使用するとコストを削減できます。シークレットを取得するすべての方法については、「シークレットの取得」を参照してください。

キャッシュポリシーは LRU (最近最も使われていない) であるため、キャッシュでシークレットを破棄する必要が生じた場合は、最も最近使われていないシークレットが破棄されます。デフォルトでは、1 時間ごとにキャッシュでシークレットが更新されます。キャッシュでシークレットが更新される頻度を設定できるだけでなく、シークレットの取得にフックさせて機能を追加することもできます。

キャッシュ参照が解放されると、キャッシュはガベージコレクションを強制しません。キャッシュの実装には、キャッシュの無効化は含まれていません。キャッシュを実装するのはキャッシュを使用するためであり、セキュリティを強化するためでもセキュリティに焦点を当てるためでもありません。キャッシュ内のアイテムを暗号化するなど、セキュリティを強化する必要がある場合は、所定のインターフェイスと抽象メソッドを使用してください。

このコンポーネントを使用するには、以下が必要です。

- Java 8 以上の開発環境。Oracle のウェブサイトの「Java SE Downloads」(Java SEのダウンロード) を参照してください。

ソースコードをダウンロードするには、GitHub の「Secrets Manager Java-based caching client component」(Secrets Manager の Java ベースのキャッシュクライアントコンポーネント) を参照してください。

コンポーネントをプロジェクトに追加するには、Maven pom.xml ファイルに、次の依存関係を含めます。Maven の詳細については、Apache Maven プロジェクトのウェブサイトの「Getting Started Guide」(入門ガイド) を参照してください。

```
<dependency>
  <groupId>com.amazonaws.secretsmanager</groupId>
  <artifactId>aws-secretsmanager-caching-java</artifactId>
  <version>1.0.2</version>
</dependency>
```

必要な許可:

- `secretsmanager:DescribeSecret`
- `secretsmanager:GetSecretValue`

詳細については、「アクセス許可に関するリファレンス」を参照してください。

リファレンス

- SecretCache
- SecretCacheConfiguration
- SecretCacheHook

Exampleシークレットを取得する

次のコード例は、シークレット文字列を取得する Lambda 関数を示しています。これは関数ハンドラーの外部でのキャッシュのインスタンス化のベストプラクティスに従うため、Lambda 関数を再度呼び出しても、API は継続して呼び出されません。

```
package com.amazonaws.secretsmanager.caching.examples;
```

```
    import com.amazonaws.services.lambda.runtime.Context;
    import com.amazonaws.services.lambda.runtime.RequestHandler;
    import com.amazonaws.services.lambda.runtime.LambdaLogger;

    import com.amazonaws.secretsmanager.caching.SecretCache;

    public class SampleClass implements RequestHandler<String, String> {

         private final SecretCache cache  = new SecretCache();

         @Override public String handleRequest(String secretId,  Context context) {
             final String secret  = cache.getSecretString(secretId);

            // Use the secret, return success;

        }
    }
```

SecretCache

Secrets Manager からリクエストされたシークレットのインメモリキャッシュ。the section called “getSecretString” または the section called “getSecretBinary” を使用して、キャッシュからシークレットを取得します。キャッシュの設定は、コンストラクタで the section called “SecretCacheConfiguration” オブジェクトを渡すことで設定できます。

詳細と例については、「the section called “クライアント側のキャッシュを使用した Java”」を参照してください。

コンストラクタ

`public SecretCache()`

SecretCache オブジェクトのデフォルトコンストラクタ。

`public SecretCache(AWSSecretsManagerClientBuilder builder)`

提供された `AWSSecretsManagerClientBuilder` を用いて作成された Secrets Manager クライアントを使用して、新しいキャッシュを構築します。このコンストラクタを使用して、Secrets Manager クライアントをカスタマイズします (特定のリージョンまたはエンドポイントを使用するなど)。

`public SecretCache(AWSSecretsManager client)`

提供された AWSSecretsManagerClient を使用して、新しいシークレットキャッシュを構築します。このコンストラクタを使用して、Secrets Manager クライアントをカスタマイズします (特定のリージョンまたはエンドポイントを使用するなど)。

`public SecretCache(SecretCacheConfiguration config)`

提供された the section called “SecretCacheConfiguration” を使用して、新しいシークレットキャッシュを構築します。

方法

getSecretString

`public String getSecretString(final String secretId)`

Secrets Manager から文字列シークレットを取得します。戻り値は String。

getSecretBinary

`public ByteBuffer getSecretBinary(final String secretId)`

Secrets Manager からバイナリシークレットを取得します。戻り値は ByteBuffer。

refreshNow

`public boolean refreshNow(final String secretId) throws InterruptedException`

キャッシュを強制的に更新します。エラーが発生せずに更新が完了した場合は `true` を返し、そうでない場合は `false` を返します。

close

`public void close()`

キャッシュを終了します。

SecretCacheConfiguration

キャッシュされるシークレットの最大キャッシュサイズや有効期限 (TTL) などの、the section called “SecretCache” のキャッシュ設定オプション。

コンストラクタ

`public SecretCacheConfiguration`

`SecretCacheConfiguration` オブジェクトのデフォルトコンストラクタ。

方法

getClient

`public AWSSecretsManager getClient()`

キャッシュがシークレットを取得する `AWSSecretsManagerClient` を返します。

setClient

`public void setClient(AWSSecretsManager client)`

キャッシュがシークレットを取得する `AWSSecretsManagerClient` クライアントを設定します。

getCacheHook

`public SecretCacheHook getCacheHook()`

キャッシュ更新に接続するために使用される the section called “SecretCacheHook” インターフェイスを返します。

setCacheHook

`public void setCacheHook(SecretCacheHook cacheHook)`

キャッシュ更新に接続するために使用される the section called “SecretCacheHook” インターフェイスを設定します。

getMaxCacheSize

`public int getMaxCacheSize()`

最大キャッシュサイズを返します。デフォルトは 1,024 個のシークレットです。

setMaxCacheSize

`public void setMaxCacheSize(int maxCacheSize)`

最大キャッシュサイズを設定します。デフォルトは 1,024 個のシークレットです。

getCacheItemTTL

```
public long getCacheItemTTL()
```

キャッシュされた項目の TTL をミリ秒単位で返します。キャッシュされたシークレットがこの TTL を超えると、キャッシュは AWSSecretsManagerClient から新しいシークレットのコピーを取得します。デフォルトは 1 時間 (ミリ秒単位) です。

TTL の後にシークレットがリクエストされると、キャッシュはシークレットを同期的に更新します。同期更新が失敗した場合、キャッシュは古いシークレットを返します。

setCacheItemTTL

```
public void setCacheItemTTL(long cacheItemTTL)
```

キャッシュされた項目の TTL をミリ秒単位で設定します。キャッシュされたシークレットがこの TTL を超えると、キャッシュは AWSSecretsManagerClient から新しいシークレットのコピーを取得します。デフォルトは 1 時間 (ミリ秒単位) です。

getVersionStage

```
public String getVersionStage()
```

キャッシュするシークレットのバージョンを返します。詳細については、「Secret versions」(シークレットバージョン) を参照してください。デフォルトは "AWSCURRENT" です。

setVersionStage

```
public void setVersionStage(String versionStage)
```

キャッシュするシークレットのバージョンを設定します。詳細については、「Secret versions」(シークレットバージョン) を参照してください。デフォルトは "AWSCURRENT" です。

SecretCacheConfiguration withClient

```
public SecretCacheConfiguration withClient(AWSSecretsManager client)
```

シークレットを取得する AWSSecretsManagerClient を設定します。新しい設定を持つ更新された SecretCacheConfiguration オブジェクトを返します。

SecretCacheConfiguration withCacheHook

```
public SecretCacheConfiguration withCacheHook(SecretCacheHook cacheHook)
```

インメモリキャッシュに接続するために使用されるインターフェイスを設定します。新しい設定を持つ更新された SecretCacheConfiguration オブジェクトを返します。

SecretCacheConfiguration withMaxCacheSize

```
public SecretCacheConfiguration withMaxCacheSize(int maxCacheSize)
```

最大キャッシュサイズを設定します。新しい設定を持つ更新された SecretCacheConfiguration オブジェクトを返します。

SecretCacheConfiguration withCacheItemTTL

```
public SecretCacheConfiguration withCacheItemTTL(long cacheItemTTL)
```

キャッシュされた項目の TTL をミリ秒単位で設定します。キャッシュされたシークレットがこの TTL を超えると、キャッシュは AWSSecretsManagerClient から新しいシークレットのコピーを取得します。デフォルトは 1 時間 (ミリ秒単位) です。新しい設定を持つ更新された SecretCacheConfiguration オブジェクトを返します。

SecretCacheConfiguration withVersionStage

```
public SecretCacheConfiguration withVersionStage(String versionStage)
```

キャッシュするシークレットのバージョンを設定します。詳細については、「Secret versions」(シークレットバージョン) を参照してください。新しい設定を持つ更新された SecretCacheConfiguration オブジェクトを返します。

SecretCacheHook

the section called “SecretCache” に接続して、キャッシュに保存されているシークレットに対してアクションを実行するインターフェイス。

put

```
Object put(final Object o)
```

キャッシュに保存するオブジェクトを準備します。

キャッシュに保存するオブジェクトを返します。

get

```
Object get(final Object cachedObject)
```

キャッシュされたオブジェクトからオブジェクトを派生させます。

キャッシュから返すオブジェクトを返します

AWS Secrets Manager シークレットの認証情報を使用して JDBC を使用して SQL データベースに接続する

Java アプリケーションでは、Secrets Manager SQL 接続ドライバーを使用して、Secrets Manager に保存された認証情報を用いて MySQL、PostgreSQL、Oracle、MSSQLServer、Db2、Redshift のデータベースに接続できます。各ドライバーはベース JDBC ドライバーをラップしているため、JDBC 呼び出しを使用してデータベースにアクセスすることができます。ただし、接続用のユーザーネームとパスワードを渡す代わりに、シークレットの ID を指定します。ドライバーは、Secrets Manager を呼び出してシークレット値を取得してから、シークレット内の認証情報と接続情報を使用してデータベースに接続します。また、ドライバーは Java のクライアント側のキャッシュライブラリを使用して認証情報をキャッシュするため、その後の接続では Secrets Manager を呼び出す必要はありません。デフォルトでは、1 時間ごと、およびシークレットがローテーションされたときに、キャッシュが更新されます。キャッシュを設定するには、the section called “SecretCacheConfiguration” を参照してください。

GitHub からソースコードをダウンロードすることができます。

Secrets Manager SQL 接続ドライバーを使用するには、以下が必要です。

- アプリケーションが Java 8 以降である必要があります。
- シークレットが次のいずれかである必要があります。
  - 期待される JSON 構造のデータベースシークレット。シークレットの形式を確認するには、Secrets Manager コンソールで、シークレットを表示して [Retrieve secret value] を選択します。または、 で AWS CLI getget-secret-value。
  - Amazon RDS マネージドシークレット。このタイプのシークレットでは、接続を確立するときにエンドポイントとポートを指定する必要があります。
  - Amazon Redshift マネージドシークレット。このタイプのシークレットでは、接続を確立するときにエンドポイントとポートを指定する必要があります。

データベースが他のリージョンにレプリケートされている場合、別のリージョンのレプリカデータベースに接続するには、接続の作成時にリージョンのエンドポイントとポートを指定します。リージョン接続情報は、追加のキー/値のペア、SSM パラメータストアパラメータ、またはコード構成でシークレットに格納できます。

ドライバーをプロジェクトに追加するには、Maven ビルドファイル pom.xml で、次のドライバーの依存関係を追加します。詳細については、Maven Central Repository の web サイトの「Secrets Manager SQL Connection Library」(Secrets Manager SQL 接続ライブラリ) を参照してください。

```
<dependency>
    <groupId>com.amazonaws.secretsmanager</groupId>
    <artifactId>aws-secretsmanager-jdbc</artifactId>
    <version>1.0.12</version>
</dependency>
```

このドライバーではデフォルトの認証情報プロバイダーチェーンを使用します。Amazon EKS でドライバーを実行すると、サービスアカウントロールの代わりに、実行中のノードの認証情報が取得される可能性があります。これに対処するには、com.amazonaws:aws-java-sdk-sts のバージョン 1 を Gradle または Maven プロジェクトファイルに依存関係として追加します。

ファイルで DNS AWS PrivateLink エンドポイント URL とリージョンを設定するにはsecretsmanager.properties:

```
drivers.vpcEndpointUrl = endpoint URL
drivers.vpcEndpointRegion = endpoint region
```

プライマリリージョンをオーバーライドするには、AWS_SECRET_JDBC_REGION 環境変数を設定するか、secretsmanager.properties ファイルに次の変更を加えます。

```
drivers.region = region
```

必要な許可:

- secretsmanager:DescribeSecret
- secretsmanager:GetSecretValue

詳細については、「アクセス許可に関するリファレンス」を参照してください。

例:

- データベースへの接続を確立する
- エンドポイントとポートを指定して接続を確立する
- c3p0 接続プールを使用して接続を確立する
- c3p0 接続プールを使用して、エンドポイントとポートを指定して接続を確立する

データベースへの接続を確立する

次の例では、シークレット内の認証情報と接続情報を使用してデータベースへの接続を確立する方法を示しています。接続が確立すると、JDBC 呼び出しを使用してデータベースにアクセスすることができます。詳細については、Java ドキュメントのウェブサイトの「JDBC Basics」(JDBC の基本)を参照してください。

MySQL

```
// Load the JDBC driver
Class.forName( "com.amazonaws.secretsmanager.sql.AWSSecretsManagerMySQLDriver" ).newInstance

// Retrieve the connection info from the secret using the secret ARN
String URL = "secretId";

// Populate the user property with the secret ARN to retrieve user and password from
 the secret
Properties info = new Properties( );
info.put( "user", "secretId" );

// Establish the connection
conn = DriverManager.getConnection(URL, info);
```

PostgreSQL

```
// Load the JDBC driver
Class.forName( "com.amazonaws.secretsmanager.sql.AWSSecretsManagerPostgreSQLDriver" ).newIns

// Retrieve the connection info from the secret using the secret ARN
String URL = "secretId";

// Populate the user property with the secret ARN to retrieve user and password from
 the secret
Properties info = new Properties( );
info.put( "user", "secretId" );

// Establish the connection
conn = DriverManager.getConnection(URL, info);
```

Oracle

```
// Load the JDBC driver
```

```
Class.forName( "com.amazonaws.secretsmanager.sql.AWSSecretsManagerOracleDriver" ).newInstanc

// Retrieve the connection info from the secret using the secret ARN
String URL = "secretId";

// Populate the user property with the secret ARN to retrieve user and password from
 the secret
Properties info = new Properties( );
info.put( "user", "secretId" );

// Establish the connection
conn = DriverManager.getConnection(URL, info);
```

MSSQLServer

```
// Load the JDBC driver
Class.forName( "com.amazonaws.secretsmanager.sql.AWSSecretsManagerMSSQLServerDriver" ).newIn

// Retrieve the connection info from the secret using the secret ARN
String URL = "secretId";

// Populate the user property with the secret ARN to retrieve user and password from
 the secret
Properties info = new Properties( );
info.put( "user", "secretId" );

// Establish the connection
conn = DriverManager.getConnection(URL, info);
```

Db2

```
// Load the JDBC driver
Class.forName( "com.amazonaws.secretsmanager.sql.AWSSecretsManagerDb2Driver" ).newInstance()

// Retrieve the connection info from the secret using the secret ARN
String URL = "secretId";

// Populate the user property with the secret ARN to retrieve user and password from
 the secret
Properties info = new Properties( );
info.put( "user", "secretId" );

// Establish the connection
```

```
conn = DriverManager.getConnection(URL, info);
```

Redshift

```
// Load the JDBC driver
Class.forName( "com.amazonaws.secretsmanager.sql.AWSSecretsManagerRedshiftDriver" ).newInsta

// Retrieve the connection info from the secret using the secret ARN
String URL = "secretId";

// Populate the user property with the secret ARN to retrieve user and password from
 the secret
Properties info = new Properties( );
info.put( "user", "secretId" );

// Establish the connection
conn = DriverManager.getConnection(URL, info);
```

エンドポイントとポートを指定して接続を確立する

次の例は、シークレット内の認証情報を使用して、指定したエンドポイントとポートでデータベースへの接続を確立する方法を示しています。

Amazon RDS マネージドシークレットには、データベースのエンドポイントおよびポートは含まれていません。Amazon RDS が管理するシークレットのマスター認証情報を使用してデータベースに接続するには、コードでマスター認証情報を指定します。

他のリージョンにレプリケートされるシークレットは、リージョンデータベースへの接続のレイテンシーを改善できますが、ソースシークレット以外の接続情報を保持しません。各レプリカは、ソースシークレットのコピーです。リージョン接続情報をシークレットに保存するには、リージョンのエンドポイントとポート情報のキー/値のペアを追加します。

接続が確立すると、JDBC 呼び出しを使用してデータベースにアクセスすることができます。詳細については、Java ドキュメントのウェブサイトの「JDBC Basics」(JDBC の基本) を参照してください。

MySQL

```
// Load the JDBC driver
Class.forName( "com.amazonaws.secretsmanager.sql.AWSSecretsManagerMySQLDriver" ).newInstance
```

```
// Set the endpoint and port. You can also retrieve it from a key/value pair in the
 secret.
String URL = "jdbc-secretsmanager:mysql://example.com:3306";

// Populate the user property with the secret ARN to retrieve user and password from
 the secret
Properties info = new Properties( );
info.put( "user", "secretId" );

// Establish the connection
conn = DriverManager.getConnection(URL, info);
```

PostgreSQL

```
// Load the JDBC driver
Class.forName( "com.amazonaws.secretsmanager.sql.AWSSecretsManagerPostgreSQLDriver" ).newIns

// Set the endpoint and port. You can also retrieve it from a key/value pair in the
 secret.
String URL = "jdbc-secretsmanager:postgresql://example.com:5432/database";

// Populate the user property with the secret ARN to retrieve user and password from
 the secret
Properties info = new Properties( );
info.put( "user", "secretId" );

// Establish the connection
conn = DriverManager.getConnection(URL, info);
```

Oracle

```
// Load the JDBC driver
Class.forName( "com.amazonaws.secretsmanager.sql.AWSSecretsManagerOracleDriver" ).newInstanc

// Set the endpoint and port. You can also retrieve it from a key/value pair in the
 secret.
String URL = "jdbc-secretsmanager:oracle:thin:@example.com:1521/ORCL";

// Populate the user property with the secret ARN to retrieve user and password from
 the secret
Properties info = new Properties( );
info.put( "user", "secretId" );
```

```
// Establish the connection
conn = DriverManager.getConnection(URL, info);
```

MSSQLServer

```
// Load the JDBC driver
Class.forName( "com.amazonaws.secretsmanager.sql.AWSSecretsManagerMSSQLServerDriver" ).newIn

// Set the endpoint and port. You can also retrieve it from a key/value pair in the
 secret.
String URL = "jdbc-secretsmanager:sqlserver://example.com:1433";

// Populate the user property with the secret ARN to retrieve user and password from
 the secret
Properties info = new Properties( );
info.put( "user", "secretId" );

// Establish the connection
conn = DriverManager.getConnection(URL, info);
```

Db2

```
// Load the JDBC driver
Class.forName( "com.amazonaws.com.amazonaws.secretsmanager.sql.AWSSecretsManagerDb2Driver" )

// Set the endpoint and port. You can also retrieve it from a key/value pair in the
 secret.
String URL = "jdbc-secretsmanager:db2://example.com:50000";

// Populate the user property with the secret ARN to retrieve user and password from
 the secret
Properties info = new Properties( );
info.put( "user", "secretId" );

// Establish the connection
conn = DriverManager.getConnection(URL, info);
```

Redshift

```
// Load the JDBC driver
Class.forName( "com.amazonaws.com.amazonaws.secretsmanager.sql.AWSSecretsManagerRedshiftDriv
```

```
// Set the endpoint and port. You can also retrieve it from a key/value pair in the
 secret.
String URL = "jdbc-secretsmanager:redshift://example.com:5439";

// Populate the user property with the secret ARN to retrieve user and password from
 the secret
Properties info = new Properties( );
info.put( "user", "secretId" );

// Establish the connection
conn = DriverManager.getConnection(URL, info);
```

c3p0 接続プールを使用して接続を確立する

次の例は、ドライバーを使用してシークレットから認証情報および接続情報を取得する c3p0.properties ファイルで接続プールを確立する方法を示しています。user と jdbcUrl には、シークレット ID を入力して接続プールを設定します。その後、プールから接続を取得し、他のデータベース接続として使用することができます。詳細については、Java ドキュメントのウェブサイトの「JDBC Basics」(JDBC の基本) を参照してください。

c3p0 の詳細については、Machinery For Change のウェブサイトの「c3p0」を参照してください。

MySQL

```
c3p0.user=secretId
c3p0.driverClass=com.amazonaws.secretsmanager.sql.AWSSecretsManagerMySQLDriver
c3p0.jdbcUrl=secretId
```

PostgreSQL

```
c3p0.user=secretId
c3p0.driverClass=com.amazonaws.secretsmanager.sql.AWSSecretsManagerPostgreSQLDriver
c3p0.jdbcUrl=secretId
```

Oracle

```
c3p0.user=secretId
c3p0.driverClass=com.amazonaws.secretsmanager.sql.AWSSecretsManagerOracleDriver
c3p0.jdbcUrl=secretId
```

MSSQLServer

```
c3p0.user=secretId
c3p0.driverClass=com.amazonaws.secretsmanager.sql.AWSSecretsManagerMSSQLServerDriver
c3p0.jdbcUrl=secretId
```

Db2

```
c3p0.user=secretId
c3p0.driverClass=com.amazonaws.secretsmanager.sql.AWSSecretsManagerDb2Driver
c3p0.jdbcUrl=secretId
```

Redshift

```
c3p0.user=secretId
c3p0.driverClass=com.amazonaws.secretsmanager.sql.AWSSecretsManagerRedshiftDriver
c3p0.jdbcUrl=secretId
```

c3p0 接続プールを使用して、エンドポイントとポートを指定して接続を確立する

次の例は、ドライバーを使用して指定されたエンドポイントとポートでシークレットの認証情報を取得する `c3p0.properties` ファイルで接続プールを確立する方法を示しています。その後、プールから接続を取得し、他のデータベース接続として使用することができます。詳細については、Java ドキュメントのウェブサイトの「JDBC Basics」(JDBC の基本) を参照してください。

Amazon RDS マネージドシークレットには、データベースのエンドポイントおよびポートは含まれていません。Amazon RDS が管理するシークレットのマスター認証情報を使用してデータベースに接続するには、コードでマスター認証情報を指定します。

他のリージョンにレプリケートされるシークレットは、リージョンデータベースへの接続のレイテンシーを改善できますが、ソースシークレット以外の接続情報を保持しません。各レプリカは、ソースシークレットのコピーです。リージョン接続情報をシークレットに保存するには、リージョンのエンドポイントとポート情報のキー/値のペアを追加します。

MySQL

```
c3p0.user=secretId
c3p0.driverClass=com.amazonaws.secretsmanager.sql.AWSSecretsManagerMySQLDriver
c3p0.jdbcUrl=jdbc-secretsmanager:mysql://example.com:3306
```

PostgreSQL

```
c3p0.user=secretId
c3p0.driverClass=com.amazonaws.secretsmanager.sql.AWSSecretsManagerPostgreSQLDriver
c3p0.jdbcUrl=jdbc-secretsmanager:postgresql://example.com:5432/database
```

Oracle

```
c3p0.user=secretId
c3p0.driverClass=com.amazonaws.secretsmanager.sql.AWSSecretsManagerOracleDriver
c3p0.jdbcUrl=jdbc-secretsmanager:oracle:thin:@example.com:1521/ORCL
```

MSSQLServer

```
c3p0.user=secretId
c3p0.driverClass=com.amazonaws.secretsmanager.sql.AWSSecretsManagerMSSQLServerDriver
c3p0.jdbcUrl=jdbc-secretsmanager:sqlserver://example.com:1433
```

Db2

```
c3p0.user=secretId
c3p0.driverClass=com.amazonaws.secretsmanager.sql.AWSSecretsManagerDb2Driver
c3p0.jdbcUrl=jdbc-secretsmanager:db2://example.com:50000
```

Redshift

```
c3p0.user=secretId
c3p0.driverClass=com.amazonaws.secretsmanager.sql.AWSSecretsManagerRedshiftDriver
c3p0.jdbcUrl=jdbc-secretsmanager:redshift://example.com:5439
```

Java AWS SDK を使用して Secrets Manager シークレット値を取得する

アプリケーションでは、任意の AWS SDKs `BatchGetSecretValue`で `GetSecretValue`または を呼び出してシークレットを取得できます。ただし、クライアント側のキャッシュを使用してシークレット値をキャッシュすることをお勧めします。シークレットをキャッシュすると、速度が向上し、コストが削減されます。

- データベースの認証情報をシークレットに保存する場合は、Secrets Manager SQL 接続ドライバーを使用し、シークレット内の認証情報を用いてデータベースに接続します。

- 他のタイプのシークレットの場合は、Secrets Manager の Java ベースのキャッシュコンポーネントを使用するか、GetSecretValue または BatchGetSecretValue を使用して SDK を直接呼び出します。

次のサンプルコードは、GetSecretValue を使用する方法を説明しています。

必要な許可:secretsmanager:GetSecretValue

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.secretsmanager.SecretsManagerClient;
import software.amazon.awssdk.services.secretsmanager.model.GetSecretValueRequest;
import software.amazon.awssdk.services.secretsmanager.model.GetSecretValueResponse;
import software.amazon.awssdk.services.secretsmanager.model.SecretsManagerException;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 *
 * We recommend that you cache your secret values by using client-side caching.
 *
 * Caching secrets improves speed and reduces your costs. For more information,
 * see the following documentation topic:
 *
 * https://docs.aws.amazon.com/secretsmanager/latest/userguide/retrieving-secrets.html
 */
public class GetSecretValue {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <secretName>\s

                Where:
                    secretName - The name of the secret (for example, tutorials/
MyFirstSecret).\s
                """;

        if (args.length != 1) {
            System.out.println(usage);
```

```
            System.exit(1);
        }

        String secretName = args[0];
        Region region = Region.US_EAST_1;
        SecretsManagerClient secretsClient = SecretsManagerClient.builder()
                .region(region)
                .build();

        getValue(secretsClient, secretName);
        secretsClient.close();
    }

    public static void getValue(SecretsManagerClient secretsClient, String secretName)
 {
        try {
            GetSecretValueRequest valueRequest = GetSecretValueRequest.builder()
                    .secretId(secretName)
                    .build();

            GetSecretValueResponse valueResponse =
 secretsClient.getSecretValue(valueRequest);
            String secret = valueResponse.secretString();
            System.out.println(secret);

        } catch (SecretsManagerException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

Python を使用して Secrets Manager のシークレット値を取得する

アプリケーションでは、任意の AWS SDKs `BatchGetSecretValue`で `GetSecretValue`または を呼び出してシークレットを取得できます。ただし、クライアント側のキャッシュを使用してシークレット値をキャッシュすることをお勧めします。シークレットをキャッシュすると、速度が向上し、コストが削減されます。

トピック

- Python とクライアント側のキャッシュを使用して、Secrets Manager のシークレット値を取得する
- Python AWS SDK を使用して Secrets Manager シークレット値を取得する
- Python AWS SDK を使用して Secrets Manager シークレット値のバッチを取得する

Python とクライアント側のキャッシュを使用して、Secrets Manager のシークレット値を取得する

シークレットを取得するときに、Secrets Manager の Python ベースのキャッシュコンポーネントを使用して、将来使用するためにキャッシュすることができます。キャッシュされたシークレットの取得は、Secrets Manager からの取得よりも高速です。Secrets Manager API を呼び出すにはコストがかかるため、キャッシュを使用するとコストを削減できます。シークレットを取得するすべての方法については、「シークレットの取得」を参照してください。

キャッシュポリシーは LRU (最近最も使われていない) であるため、キャッシュでシークレットを破棄する必要が生じた場合は、最も最近使われていないシークレットが破棄されます。デフォルトでは、1 時間ごとにキャッシュでシークレットが更新されます。キャッシュでシークレットが更新される頻度を設定できるだけでなく、シークレットの取得にフックさせて機能を追加することもできます。

キャッシュ参照が解放されると、キャッシュはガベージコレクションを強制しません。キャッシュの実装には、キャッシュの無効化は含まれていません。キャッシュを実装するのはキャッシュを使用するためであり、セキュリティを強化するためでもセキュリティに焦点を当てるためでもありません。キャッシュ内のアイテムを暗号化するなど、セキュリティを強化する必要がある場合は、所定のインターフェイスと抽象メソッドを使用してください。

このコンポーネントを使用するには、以下が必要です。

- Python 3.6 以降。
- botocore 1.12 以降。「AWS SDK for Python」および「Botocore」を参照してください。
- setuptools_scm 3.2 以降。https://pypi.org/project/setuptools-scm/ を参照してください。

ソースコードをダウンロードするには、GitHub の「Secrets Manager Python-based caching client component」(Secrets Manager の Python ベースのキャッシュクライアントコンポーネント) を参照してください。

コンポーネントをインストールするには、次のコマンドを使用します。

```
$ pip install aws-secretsmanager-caching
```

必要な許可:

- `secretsmanager:DescribeSecret`
- `secretsmanager:GetSecretValue`

詳細については、「アクセス許可に関するリファレンス」を参照してください。

リファレンス

- SecretCache
- SecretCacheConfig
- SecretCacheHook
- @InjectSecretString
- @InjectKeywordedSecretString

Exampleシークレットを取得する

次の例は、*mysecret* という名前のシークレットのシークレット値を取得する方法を示しています。

```
import botocore
import botocore.session
from aws_secretsmanager_caching import SecretCache, SecretCacheConfig

client = botocore.session.get_session().create_client('secretsmanager')
cache_config = SecretCacheConfig()
cache = SecretCache( config = cache_config, client = client)

secret = cache.get_secret_string('mysecret')
```

SecretCache

Secrets Manager から取得されたシークレットのインメモリキャッシュ。the section called “get_secret_string” または the section called “get_secret_binary” を使用して、キャッシュからシークレットを取得します。キャッシュの設定は、コンストラクタで the section called “SecretCacheConfig” オブジェクトを渡すことで設定できます。

詳細と例については、「the section called “クライアント側のキャッシュを使用した Python”」を参照してください。

```
cache = SecretCache(
    config = the section called “SecretCacheConfig”,
    client = client
)
```

使用できるメソッドは次のとおりです。

- get_secret_string
- get_secret_binary

get_secret_string

シークレット文字列値を取得します。

リクエストの構文

```
response = cache.get_secret_string(
    secret_id='string',
    version_stage='string' )
```

パラメータ

- `secret_id` (string): [必須] シークレットの名前または ARN。
- `version_stage` (string): 取得するシークレットのバージョン。詳細については、「secret versions」を参照してください。デフォルトは「AWSCURRENT」です。

戻り型

string

get_secret_binary

シークレットバイナリ値を取得します。

リクエストの構文

```
response = cache.get_secret_binary(
    secret_id='string',
```

```
    version_stage='string'
)
```

パラメータ

- secret_id (string): [必須] シークレットの名前または ARN。
- version_stage (string): 取得するシークレットのバージョン。詳細については、「secret versions」を参照してください。デフォルトは「AWSCURRENT」です。

戻り型

base64 でエンコードされた文字列

SecretCacheConfig

キャッシュされるシークレットの最大キャッシュサイズや有効期限 (TTL) などの、the section called "SecretCache" のキャッシュ設定オプション。

パラメータ

max_cache_size (int)

最大キャッシュサイズ。デフォルトは 1024 個のシークレットです。

exception_retry_delay_base (int)

例外が発生してから、リクエストを再試行するまで待機する秒数。デフォルトは 1 です。

exception_retry_growth_factor (int)pur

失敗したリクエストの再試行間の待機時間を計算するために使用する増加係数。デフォルトは 2 です。

exception_retry_delay_max (int)

失敗したリクエスト間の最大待機時間 (秒)。デフォルトは 3600 です。

default_version_stage (str)

キャッシュするシークレットのバージョン。詳細については、「Secret versions」(シークレットバージョン) を参照してください。デフォルトは 'AWSCURRENT' です。

secret_refresh_interval (int)

キャッシュされたシークレット情報の更新間の待機秒数。デフォルトは 3600 です。

`secret_cache_hook` (SecretCacheHook)

`SecretCacheHook` 抽象クラスの実装。デフォルト値は `None` です。

SecretCacheHook

the section called “SecretCache” に接続して、キャッシュに保存されているシークレットに対してアクションを実行するインターフェイス。

使用できるメソッドは次のとおりです。

- put
- get

put

キャッシュに保存するオブジェクトを準備します。

リクエストの構文

```
response = hook.put(
    obj='secret_object'
)
```

パラメータ

- `obj` (object) -- [必須] シークレットまたはシークレットを含むオブジェクト。

戻り型

オブジェクト

get

キャッシュされたオブジェクトからオブジェクトを派生させます。

リクエストの構文

```
response = hook.get(
    obj='secret_object'
)
```

パラメータ

- obj (object): [必須] シークレットまたはシークレットを含むオブジェクト。

戻り型

オブジェクト

@InjectSecretString

このデコレータは、1 番目と 2 番目の引数として、シークレット ID 文字列と the section called "SecretCache" を必要とします。このデコレータはシークレット文字列値を返します。シークレットに文字列が含まれている必要があります。

```
from aws_secretsmanager_caching import SecretCache
from aws_secretsmanager_caching import InjectKeywordedSecretString,
 InjectSecretString

cache = SecretCache()

@InjectSecretString ( 'mysecret' ,  cache )
def function_to_be_decorated( arg1,  arg2,  arg3):
```

@InjectKeywordedSecretString

このデコレータは、1 番目と 2 番目の引数として、シークレット ID 文字列と the section called "SecretCache" を必要とします。残りの引数は、ラップされた関数のパラメータをシークレット内の JSON キーにマッピングします。シークレットに JSON 構造の文字列が含まれている必要があります。

この JSON を含むシークレットの場合は、次のようになります。

```
{
  "username": "saanvi",
  "password": "EXAMPLE-PASSWORD"
}
```

次の例では、シークレットから username および password の JSON 値を抽出する方法を示しています。

```
from aws_secretsmanager_caching import SecretCache
```

```
 from aws_secretsmanager_caching import InjectKeywordedSecretString,
InjectSecretString

 cache = SecretCache()

 @InjectKeywordedSecretString ( secret_id = 'mysecret' ,  cache = cache ,
func_username = 'username' ,  func_password = 'password' )
 def function_to_be_decorated( func_username,  func_password):
      print( 'Do something with the func_username and func_password parameters')
```

Python AWS SDK を使用して Secrets Manager シークレット値を取得する

アプリケーションでは、任意の AWS SDKs BatchGetSecretValueで GetSecretValueまたは を呼び出してシークレットを取得できます。ただし、クライアント側のキャッシュを使用してシークレット値をキャッシュすることをお勧めします。シークレットをキャッシュすると、速度が向上し、コストが削減されます。

Python アプリケーションの場合は、Secrets Manager の Python ベースのキャッシュコンポーネントを使用するか、get_secret_value または batch_get_secret_value を使用して SDK を直接呼び出します。

次のサンプルコードは、GetSecretValue を使用する方法を説明しています。

必要な許可:secretsmanager:GetSecretValue

```
"""
Purpose

Shows how to use the AWS SDK for Python (Boto3) with AWS
Secrets Manager to get a specific of secrets that match a
specified name
"""
import boto3
import logging

from get_secret_value import GetSecretWrapper

# Configure logging
logging.basicConfig(level=logging.INFO)


def run_scenario(secret_name):
    """
```

```
    Retrieve a secret from AWS Secrets Manager.

    :param secret_name: Name of the secret to retrieve.
    :type secret_name: str
    """
    try:
        # Validate secret_name
        if not secret_name:
            raise ValueError("Secret name must be provided.")
        # Retrieve the secret by name
        client = boto3.client("secretsmanager")
        wrapper = GetSecretWrapper(client)
        secret = wrapper.get_secret(secret_name)
        # Note: Secrets should not be logged.
        return secret
    except Exception as e:
        logging.error(f"Error retrieving secret: {e}")
        raise

class GetSecretWrapper:
    def __init__(self, secretsmanager_client):
        self.client = secretsmanager_client


    def get_secret(self, secret_name):
        """
        Retrieve individual secrets from AWS Secrets Manager using the get_secret_value
 API.
        This function assumes the stack mentioned in the source code README has been
 successfully deployed.
        This stack includes 7 secrets, all of which have names beginning with
 "mySecret".

        :param secret_name: The name of the secret fetched.
        :type secret_name: str
        """
        try:
            get_secret_value_response = self.client.get_secret_value(
                SecretId=secret_name
            )
            logging.info("Secret retrieved successfully.")
            return get_secret_value_response["SecretString"]
        except self.client.exceptions.ResourceNotFoundException:
            msg = f"The requested secret {secret_name} was not found."
```

```
            logger.info(msg)
            return msg
        except Exception as e:
            logger.error(f"An unknown error occurred: {str(e)}.")
            raise
```

Python AWS SDK を使用して Secrets Manager シークレット値のバッチを取得する

次のコード例は、Secrets Manager のシークレットの値をバッチ取得する方法を示しています。

必要な許可:

- `secretsmanager:BatchGetSecretValue`
- 取得するシークレットごとに `secretsmanager:GetSecretValue` のアクセス許可が必要です。
- フィルターを使用する場合は、`secretsmanager:ListSecrets` も必要です。

アクセス許可ポリシーの例については、「the section called “例: バッチでシークレット値のグループを取得するためのアクセス許可”」を参照してください。

> **Important**
>
> 取得しようとしているグループ内の個別シークレットを取得するためのアクセス許可を拒否する VPCE ポリシーを設定している場合、`BatchGetSecretValue` はシークレット値を返さず、エラーが返されます。

```
class BatchGetSecretsWrapper:
    def __init__(self, secretsmanager_client):
        self.client = secretsmanager_client


    def batch_get_secrets(self, filter_name):
        """
        Retrieve multiple secrets from AWS Secrets Manager using the
 batch_get_secret_value API.
```

```
        This function assumes the stack mentioned in the source code README has been
successfully deployed.
        This stack includes 7 secrets, all of which have names beginning with
"mySecret".

        :param filter_name: The full or partial name of secrets to be fetched.
        :type filter_name: str
        """
        try:
            secrets = []
            response = self.client.batch_get_secret_value(
                Filters=[{"Key": "name", "Values": [f"{filter_name}"]}]
            )
            for secret in response["SecretValues"]:
                secrets.append(json.loads(secret["SecretString"]))
            if secrets:
                logger.info("Secrets retrieved successfully.")
            else:
                logger.info("Zero secrets returned without error.")
            return secrets
        except self.client.exceptions.ResourceNotFoundException:
            msg = f"One or more requested secrets were not found with filter:
{filter_name}"
            logger.info(msg)
            return msg
        except Exception as e:
            logger.error(f"An unknown error occurred:\n{str(e)}.")
            raise
```

.NET を使用して Secrets Manager シークレット値を取得する

アプリケーションでは、任意の AWS SDKs `BatchGetSecretValue`で `GetSecretValue`または を呼び出してシークレットを取得できます。ただし、クライアント側のキャッシュを使用してシークレット値をキャッシュすることをお勧めします。シークレットをキャッシュすると、速度が向上し、コストが削減されます。

トピック

- .NET とクライアント側のキャッシュを使用して、Secrets Manager のシークレット値を取得する
- を使用して Secrets Manager シークレット値を取得する SDK for .NET

.NET とクライアント側のキャッシュを使用して、Secrets Manager のシークレット値を取得する

シークレットを取得するときに、Secrets Manager の .NET ベースのキャッシュコンポーネントを使用して、将来使用するためにキャッシュすることができます。キャッシュされたシークレットの取得は、Secrets Manager からの取得よりも高速です。Secrets Manager API を呼び出すにはコストがかかるため、キャッシュを使用するとコストを削減できます。シークレットを取得するすべての方法については、「シークレットの取得」を参照してください。

キャッシュポリシーは LRU (最近最も使われていない) であるため、キャッシュでシークレットを破棄する必要が生じた場合は、最も最近使われていないシークレットが破棄されます。デフォルトでは、1 時間ごとにキャッシュでシークレットが更新されます。キャッシュでシークレットが更新される頻度を設定できるだけでなく、シークレットの取得にフックさせて機能を追加することもできます。

キャッシュ参照が解放されると、キャッシュはガベージコレクションを強制しません。キャッシュの実装には、キャッシュの無効化は含まれていません。キャッシュを実装するのはキャッシュを使用するためであり、セキュリティを強化するためでもセキュリティに焦点を当てるためでもありません。キャッシュ内のアイテムを暗号化するなど、セキュリティを強化する必要がある場合は、所定のインターフェイスと抽象メソッドを使用してください。

このコンポーネントを使用するには、以下が必要です。

- .NET Framework 4.6.2 以上、または .NET Standard 2.0 以上。Microsoft .NET のウェブサイトの「.NET のダウンロード」を参照してください。
- AWS SDK for .NET。「the section called “AWS SDKs”」を参照してください。

ソースコードをダウンロードするには、GitHub の「.NET キャッシュクライアント」を参照してください。

キャッシュを使用するには、まずキャッシュをインスタンス化してから、`GetSecretString` または `GetSecretBinary` を使用して自分のシークレットを取得する必要があります。連続して取得すると、キャッシュはシークレットのキャッシュされたコピーを返します。

キャッシュパッケージを入手するには

- 次のいずれかを行います。
  - プロジェクトディレクトリで次の .NET CLI コマンドを実行します。

```
dotnet add package AWSSDK.SecretsManager.Caching --version 1.0.6
```

- .csproj ファイルに次のパッケージリファレンスを追加します。

```
<ItemGroup>
    <PackageReference Include="AWSSDK.SecretsManager.Caching" Version="1.0.6" /
>
</ItemGroup>
```

必要な許可:

- secretsmanager:DescribeSecret
- secretsmanager:GetSecretValue

詳細については、「アクセス許可に関するリファレンス」を参照してください。

リファレンス

- SecretsManagerCache
- SecretCacheConfiguration
- ISecretCacheHook

Exampleシークレットを取得する

次のコード例では、*MySecret* という名前のシークレットを取得するメソッドを示しています。

```
using Amazon.SecretsManager.Extensions.Caching;

namespace LambdaExample
{
    public class CachingExample
    {
        private const string MySecretName ="MySecret";

        private SecretsManagerCache cache = new SecretsManagerCache();

        public async Task<Response>  FunctionHandlerAsync(string input, ILambdaContext
 context)
        {
```

```
            string MySecret = await cache.GetSecretString(MySecretName);

            // Use the secret, return success

        }
    }
}
```

Example有効期限 (TTL) キャッシュ更新期間を設定する

次のコード例は、*MySecret* という名前のシークレットを取得し、TTL キャッシュ更新期間を 24 時間に設定する方法を示しています。

```
using Amazon.SecretsManager.Extensions.Caching;

namespace LambdaExample
{
    public class CachingExample
    {
        private const string MySecretName = "MySecret";

        private static SecretCacheConfiguration cacheConfiguration = new
 SecretCacheConfiguration
        {
            CacheItemTTL = 86400000
        };
        private SecretsManagerCache cache = new
 SecretsManagerCache(cacheConfiguration);
        public async Task<Response> FunctionHandlerAsync(string input, ILambdaContext
 context)
        {
            string mySecret = await cache.GetSecretString(MySecretName);

            // Use the secret, return success
        }
    }
}
```

SecretsManagerCache

Secrets Manager からリクエストされたシークレットのインメモリキャッシュ。the section called “GetSecretString” または the section called “GetSecretBinary” を使用して、キャッシュ

からシークレットを取得します。キャッシュの設定は、コンストラクタで the section called “SecretCacheConfiguration” オブジェクトを渡すことで設定できます。

詳細と例については、「the section called “クライアント側のキャッシュを使用した .NET”」を参照してください。

コンストラクタ

```
public SecretsManagerCache()
```

SecretsManagerCache オブジェクトのデフォルトコンストラクタ。

```
public SecretsManagerCache(IAmazonSecretsManager secretsManager)
```

提供された AmazonSecretsManagerClient を用いて作成された Secrets Manager クライアントを使用して、新しいキャッシュを構築します。このコンストラクタを使用して、Secrets Manager クライアントをカスタマイズします (特定のリージョンまたはエンドポイントを使用するなど)。

パラメータ

secretsManager

シークレットを取得する AmazonSecretsManagerClient。

```
public SecretsManagerCache(SecretCacheConfiguration config)
```

提供された the section called “SecretCacheConfiguration” を使用して、新しいシークレットキャッシュを構築します。このコンストラクタを使用してキャッシュを設定します (キャッシュするシークレットの数や更新頻度など)。

パラメータ

config

キャッシュの設定情報が含まれている the section called “SecretCacheConfiguration”。

```
public SecretsManagerCache(IAmazonSecretsManager secretsManager,
SecretCacheConfiguration config)
```

提供された AmazonSecretsManagerClient および the section called “SecretCacheConfiguration” を使用して作成された Secrets Manager クライアントを使用して、新しいキャッシュを構築します。このコンストラクタを使用して Secrets Manager クライアントをカスタマイズし (特定のリージョンまたはエンドポイントを使用するなど)、キャッシュを構成します (キャッシュするシークレットの数や更新頻度など)。

パラメータ

secretsManager

シークレットを取得する AmazonSecretsManagerClient。

config

キャッシュの設定情報が含まれている the section called “SecretCacheConfiguration”。

方法

GetSecretString

```
public async Task<String> GetSecretString(String secretId)
```

Secrets Manager から文字列シークレットを取得します。

パラメータ

secretId

取得するシークレットの ARN または名前。

GetSecretBinary

```
public async Task<byte[]> GetSecretBinary(String secretId)
```

Secrets Manager からバイナリシークレットを取得します。

パラメータ

secretId

取得するシークレットの ARN または名前。

RefreshNowAsync

```
public async Task<bool> RefreshNowAsync(String secretId)
```

Secrets Manager からのシークレット値をリクエストし、変更があればキャッシュを更新します。既存のキャッシュエントリがない場合は、新しいキャッシュエントリを作成します。更新に成功した場合は、`true` を返します。

パラメータ

secretId

取得するシークレットの ARN または名前。

GetCachedSecret

```
public SecretCacheItem GetCachedSecret(string secretId)
```

指定されたシークレットのキャッシュエントリがキャッシュに存在する場合、そのキャッシュエントリを返します。それ以外の場合は、Secrets Manager からシークレットを取得し、新しいキャッシュエントリを作成します。

パラメータ

secretId

取得するシークレットの ARN または名前。

SecretCacheConfiguration

キャッシュされるシークレットの最大キャッシュサイズや有効期限 (TTL) などの、the section called “SecretsManagerCache” のキャッシュ設定オプション。

プロパティ

CacheItemTTL

```
public uint CacheItemTTL { get; set; }
```

キャッシュ項目の TTL (ミリ秒単位)。デフォルトは 3600000 ミリ秒 (1 時間) です。最大値は 4294967295 ms で、約 49.7 日です。

MaxCacheSize

```
public ushort MaxCacheSize { get; set; }
```

最大キャッシュサイズ。デフォルトは 1,024 個のシークレットです。最大値は 65,535 です。

VersionStage

```
public string VersionStage { get; set; }
```

キャッシュするシークレットのバージョン。詳細については、「Secret versions」(シークレットバージョン) を参照してください。デフォルトは "AWSCURRENT" です。

クライアント

```
public IAmazonSecretsManager Client { get; set; }
```

シークレットを取得する AmazonSecretsManagerClient。null の場合、キャッシュは新しいクライアントをインスタンス化します。デフォルトは null です。

CacheHook

```
public ISecretCacheHook CacheHook { get; set; }
```

the section called “ISecretCacheHook”。

ISecretCacheHook

the section called “SecretsManagerCache” に接続して、キャッシュに保存されているシークレットに対してアクションを実行するインターフェイス。

方法

プット

```
object Put(object o);
```

キャッシュに保存するオブジェクトを準備します。

キャッシュに保存するオブジェクトを返します。

Get

```
object Get(object cachedObject);
```

キャッシュされたオブジェクトからオブジェクトを派生させます。

キャッシュから返すオブジェクトを返します

を使用して Secrets Manager シークレット値を取得する SDK for .NET

アプリケーションでは、任意の AWS SDKs BatchGetSecretValueで GetSecretValueまたはを呼び出してシークレットを取得できます。ただし、クライアント側のキャッシュを使用してシーク

レット値をキャッシュすることをお勧めします。シークレットをキャッシュすると、速度が向上し、コストが削減されます。

.NET アプリケーションの場合は、Secrets Manager の .NET ベースのキャッシュコンポーネントを使用するか、GetSecretValue または BatchGetSecretValue を使用して SDK を直接呼び出します。

次のサンプルコードは、GetSecretValue を使用する方法を説明しています。

必要な許可:secretsmanager:GetSecretValue

```
using System;
using System.IO;
using System.Threading.Tasks;
using Amazon.SecretsManager;
using Amazon.SecretsManager.Model;

/// <summary>
/// This example uses the Amazon Web Service Secrets Manager to retrieve
/// the secret value for the provided secret name.
/// </summary>
public class GetSecretValue
{
    /// <summary>
    /// The main method initializes the necessary values and then calls
    /// the GetSecretAsync and DecodeString methods to get the decoded
    /// secret value for the secret named in secretName.
    /// </summary>
    public static async Task Main()
    {
        string secretName = "<<{{MySecretName}}>>";
        string secret;

        IAmazonSecretsManager client = new AmazonSecretsManagerClient();

        var response = await GetSecretAsync(client, secretName);

        if (response is not null)
        {
            secret = DecodeString(response);

            if (!string.IsNullOrEmpty(secret))
            {
```

```
                Console.WriteLine($"The decoded secret value is: {secret}.");
            }
            else
            {
                Console.WriteLine("No secret value was returned.");
            }
        }
    }

    /// <summary>
    /// Retrieves the secret value given the name of the secret to
    /// retrieve.
    /// </summary>
    /// <param name="client">The client object used to retrieve the secret
    /// value for the given secret name.</param>
    /// <param name="secretName">The name of the secret value to retrieve.</param>
    /// <returns>The GetSecretValueReponse object returned by
    /// GetSecretValueAsync.</returns>
    public static async Task<GetSecretValueResponse> GetSecretAsync(
        IAmazonSecretsManager client,
        string secretName)
    {
        GetSecretValueRequest request = new GetSecretValueRequest()
        {
            SecretId = secretName,
            VersionStage = "AWSCURRENT", // VersionStage defaults to AWSCURRENT if
unspecified.
        };

        GetSecretValueResponse response = null;

        // For the sake of simplicity, this example handles only the most
        // general SecretsManager exception.
        try
        {
            response = await client.GetSecretValueAsync(request);
        }
        catch (AmazonSecretsManagerException e)
        {
            Console.WriteLine($"Error: {e.Message}");
        }

        return response;
    }
```

```
        /// <summary>
        /// Decodes the secret returned by the call to GetSecretValueAsync and
        /// returns it to the calling program.
        /// </summary>
        /// <param name="response">A GetSecretValueResponse object containing
        /// the requested secret value returned by GetSecretValueAsync.</param>
        /// <returns>A string representing the decoded secret value.</returns>
        public static string DecodeString(GetSecretValueResponse response)
        {
            // Decrypts secret using the associated AWS Key Management Service
            // Customer Master Key (CMK.) Depending on whether the secret is a
            // string or binary value, one of these fields will be populated.
            if (response.SecretString is not null)
            {
                var secret = response.SecretString;
                return secret;
            }
            else if (response.SecretBinary is not null)
            {
                var memoryStream = response.SecretBinary;
                StreamReader reader = new StreamReader(memoryStream);
                string decodedBinarySecret =
System.Text.Encoding.UTF8.GetString(Convert.FromBase64String(reader.ReadToEnd()));
                return decodedBinarySecret;
            }
            else
            {
                return string.Empty;
            }
        }
    }
```

Go を使用して Secrets Manager シークレット値を取得する

アプリケーションでは、任意の AWS SDKs BatchGetSecretValueで GetSecretValueまたは を呼び出してシークレットを取得できます。ただし、クライアント側のキャッシュを使用してシークレット値をキャッシュすることをお勧めします。シークレットをキャッシュすると、速度が向上し、コストが削減されます。

トピック

- Go とクライアント側のキャッシュを使用して、Secrets Manager のシークレット値を取得する
- Go AWS SDK を使用して Secrets Manager のシークレット値を取得する

Go とクライアント側のキャッシュを使用して、Secrets Manager のシークレット値を取得する

シークレットを取得するときに、Secrets Manager の GO ベースのキャッシュコンポーネントを使用して、将来の使用のためにキャッシュすることができます。キャッシュされたシークレットの取得は、Secrets Manager からの取得よりも高速です。Secrets Manager API を呼び出すにはコストがかかるため、キャッシュを使用するとコストを削減できます。シークレットを取得するすべての方法については、「シークレットの取得」を参照してください。

キャッシュポリシーは LRU (最近最も使われていない) であるため、キャッシュでシークレットを破棄する必要が生じた場合は、最も最近使われていないシークレットが破棄されます。デフォルトでは、1 時間ごとにキャッシュでシークレットが更新されます。キャッシュでシークレットが更新される頻度を設定できるだけでなく、シークレットの取得にフックさせて機能を追加することもできます。

キャッシュ参照が解放されると、キャッシュはガベージコレクションを強制しません。キャッシュの実装には、キャッシュの無効化は含まれていません。キャッシュを実装するのはキャッシュを使用するためであり、セキュリティを強化するためでもセキュリティに焦点を当てるためでもありません。キャッシュ内のアイテムを暗号化するなど、セキュリティを強化する必要がある場合は、所定のインターフェイスと抽象メソッドを使用してください。

このコンポーネントを使用するには、以下が必要です。

- AWS SDK for Go。「the section called “AWS SDKs”」を参照してください。

ソースコードをダウンロードするには、GitHub の「Secrets Manager Go caching client」(Secrets Manager の Go キャッシュクライアント) を参照してください。

Go の開発環境を設定するには、Go プログラミング言語のウェブサイトの「Golang Getting Started」(Go 言語の使用開始) を参照してください。

必要な許可:

- `secretsmanager:DescribeSecret`
- `secretsmanager:GetSecretValue`

詳細については、「アクセス許可に関するリファレンス」を参照してください。

リファレンス

- type Cache
- type CacheConfig
- type CacheHook

Exampleシークレットを取得する

次のコード例は、シークレットを取得する Lambda 関数を示しています。

```
package main

import (
  "github.com/aws/aws-lambda-go/lambda"
  "github.com/aws/aws-secretsmanager-caching-go/v2/secretcache"
)

var (
  secretCache, _ = secretcache.New()
)

func HandleRequest(secretId string) string {
  result, _ := secretCache.GetSecretString(secretId)

  // Use the secret, return success
}

 func main() {
  lambda. Start( HandleRequest)
}
```

type Cache

Secrets Manager からリクエストされたシークレットのインメモリキャッシュ。the section called “GetSecretString” または the section called “GetSecretBinary” を使用して、キャッシュからシークレットを取得します。

次に、キャッシュの設定方法の例を示します。

```
// Create a custom secretsmanager client
```

```
client := getCustomClient()

// Create a custom CacheConfig struct
config := secretcache. CacheConfig{
    MaxCacheSize:  secretcache.DefaultMaxCacheSize + 10,
    VersionStage:  secretcache.DefaultVersionStage,
    CacheItemTTL:  secretcache.DefaultCacheItemTTL,
}

// Instantiate the cache
cache, _ := secretcache.New(
    func( c *secretcache.Cache) {  c. CacheConfig = config },
    func( c *secretcache.Cache) {  c. Client = client },
)
```

詳細と例については、「the section called “クライアント側のキャッシュを使用した Go”」を参照してください。

方法

新

```
func New(optFns ...func(*Cache)) (*Cache, error)
```

New は機能オプションを使用してシークレットキャッシュを構築します。それ以外の場合はデフォルトが使用されます。新しいセッションから Secrets Manager クライアントを初期化します。CacheConfig をデフォルト値に初期化します。LRU キャッシュをデフォルトの最大サイズで初期化します。

GetSecretString

```
func (c *Cache) GetSecretString(secretId string) (string, error)
```

GetSecretString は、指定されたシークレット ID のキャッシュからシークレット文字列値を取得します。シークレット文字列を返し、オペレーションが失敗した場合はエラーを返します。

GetSecretStringWithStage

```
func (c *Cache) GetSecretStringWithStage(secretId string, versionStage
string) (string, error)
```

GetSecretStringWithStage は、指定されたシークレット ID とバージョンステージのキャッシュからシークレット文字列値を取得します。シークレット文字列を返し、オペレーションが失敗した場合はエラーを返します。

GetSecretBinary

```
func (c *Cache) GetSecretBinary(secretId string) ([]byte, error) {
```

GetSecretBinary は、指定されたシークレット ID のキャッシュからシークレットバイナリ値を取得します。シークレットバイナリを返し、オペレーションが失敗した場合はエラーを返します。

GetSecretBinaryWithStage

```
func (c *Cache) GetSecretBinaryWithStage(secretId string, versionStage
string) ([]byte, error)
```

GetSecretBinaryWithStage は、指定されたシークレット ID とバージョンステージのキャッシュからシークレットバイナリ値を取得します。シークレットバイナリを返し、オペレーションが失敗した場合はエラーを返します。

type CacheConfig

キャッシュされるシークレットの最大キャッシュサイズ、デフォルトバージョンステージ、および有効期限 (TTL) などの、Cache のキャッシュ設定オプション。

```
type CacheConfig struct {

    // The maximum cache size. The default is 1024 secrets.
    MaxCacheSize int

    // The TTL of a cache item in nanoseconds. The default is
    // 3.6e10^12 ns or 1 hour.
    CacheItemTTL int64

    // The version of secrets that you want to cache. The default
    // is "AWSCURRENT".
    VersionStage string

    // Used to hook in-memory cache updates.
    Hook CacheHook
}
```

type CacheHook

Cache に接続して、キャッシュに保存されているシークレットに対してアクションを実行するインターフェイス。

方法

プット

`Put(data interface{}) interface{}`

キャッシュに保存するオブジェクトを準備します。

Get

`Get(data interface{}) interface{}`

キャッシュされたオブジェクトからオブジェクトを派生させます。

Go AWS SDK を使用して Secrets Manager のシークレット値を取得する

アプリケーションでは、任意の AWS SDKs `BatchGetSecretValue`で `GetSecretValue`または を呼び出してシークレットを取得できます。ただし、クライアント側のキャッシュを使用してシークレット値をキャッシュすることをお勧めします。シークレットをキャッシュすると、速度が向上し、コストが削減されます。

Go アプリケーションの場合は、Secrets Manager の Go ベースのキャッシュコンポーネントを使用するか、`GetSecretValue` または `BatchGetSecretValue` を使用して SDK を直接呼び出します。

次の例で、Secrets Manager シークレットの値を取得する方法を示します。

必要な許可:`secretsmanager:GetSecretValue`

```
 // Use this code snippet in your app.
 // If you need more information about configurations or implementing the sample code,
visit the AWS docs:
 // https://aws.github.io/aws-sdk-go-v2/docs/getting-started/

 import (
   "context"
   "log"

   "github.com/aws/aws-sdk-go-v2/aws"
```

```
    "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/service/secretsmanager"
  )

  func main() {
    secretName := "<<{{MySecretName}}>>"
    region := "<<{{MyRegionName}}>>"

    config, err := config.LoadDefaultConfig(context.TODO(), config.WithRegion(region))
    if err != nil {
      log.Fatal(err)
    }

    // Create Secrets Manager client
    svc := secretsmanager.NewFromConfig(config)

    input := &secretsmanager.GetSecretValueInput{
      SecretId:     aws.String(secretName),
      VersionStage: aws.String("AWSCURRENT"), // VersionStage defaults to AWSCURRENT if
 unspecified
    }

    result, err := svc.GetSecretValue(context.TODO(), input)
    if err != nil {
      // For a list of exceptions thrown, see
      // https://<<{{DocsDomain}}>>/secretsmanager/latest/apireference/
API_GetSecretValue.html
      log.Fatal(err.Error())
    }

    // Decrypts secret using the associated KMS key.
    var secretString string = *result.SecretString

    // Your code goes here.
  }
```

Rust を使用して Secrets Manager シークレット値を取得する

アプリケーションでは、任意の AWS SDKs `BatchGetSecretValue`で `GetSecretValue`または を呼び出してシークレットを取得できます。ただし、クライアント側のキャッシュを使用してシークレット値をキャッシュすることをお勧めします。シークレットをキャッシュすると、速度が向上し、コストが削減されます。

トピック

- Rust とクライアント側のキャッシュを使用して、Secrets Manager のシークレット値を取得する
- Rust AWS SDK を使用して Secrets Manager のシークレット値を取得する

Rust とクライアント側のキャッシュを使用して、Secrets Manager のシークレット値を取得する

シークレットを取得するときに、Secrets Manager の Rust ベースのキャッシュコンポーネントを使用して、将来使用するためにキャッシュすることができます。キャッシュされたシークレットの取得は、Secrets Manager からの取得よりも高速です。Secrets Manager API を呼び出すにはコストがかかるため、キャッシュを使用するとコストを削減できます。シークレットを取得するすべての方法については、「シークレットの取得」を参照してください。

キャッシュポリシーは先入れ先出し (FIFO) であるため、キャッシュがシークレットを破棄する必要がある場合、最も古いシークレットは破棄されます。デフォルトでは、1 時間ごとにキャッシュでシークレットが更新されます。次のオプションを設定できます。

- `max_size` – 最近アクセスされていないシークレットを削除する前に保持するキャッシュされたシークレットの最大数。
- `ttl` – シークレット状態の更新を要求する前にキャッシュされた項目が有効と見なされる期間。

キャッシュの実装には、キャッシュの無効化は含まれていません。キャッシュを実装するのはキャッシュを使用するためであり、セキュリティを強化するためでもセキュリティに焦点を当てるためでもありません。キャッシュ内のアイテムを暗号化するなど、セキュリティを強化する必要がある場合は、提供されている特性を使用してキャッシュを変更します。

コンポーネントを使用するには、`tokio` を備えた Rust 2021 開発環境が必要です。詳細については、Rust プログラミング言語ウェブサイトで「はじめに」を参照してください。

ソースコードをダウンロードするには、GitHub の「Secrets Manager Rust-based caching client component」を参照してください。

キャッシュコンポーネントをインストールするには、次のコマンドを使用します。

```
cargo add aws_secretsmanager_caching
```

必要な許可:

- secretsmanager:DescribeSecret
- secretsmanager:GetSecretValue

詳細については、「アクセス許可に関するリファレンス」を参照してください。

Exampleシークレットを取得する

次の例は、*MyTest* という名前のシークレットのシークレット値を取得する方法を示しています。

```
use aws_secretsmanager_caching::SecretsManagerCachingClient;
use std::num::NonZeroUsize;
use std::time::Duration;

let client = match SecretsManagerCachingClient::default(
    NonZeroUsize::new(10).unwrap(),
    Duration::from_secs(60),
)
.await
{
    Ok(c) => c,
    Err(_) => panic!("Handle this error"),
};

let secret_string = match client.get_secret_value("MyTest", None, None).await {
    Ok(s) => s.secret_string.unwrap(),
    Err(_) => panic!("Handle this error"),
};

// Your code here
```

Exampleカスタム設定とカスタムクライアントによるキャッシュのインスタンス化

次の例は、キャッシュを構成し、*MyTest* という名前のシークレットのシークレット値を取得する方法を示しています。

```
let config = aws_config::load_defaults(BehaviorVersion::latest())
    .await
    .into_builder()
    .region(Region::from_static("us-west-2"))
    .build();
```

```
let asm_builder = aws_sdk_secretsmanager::config::Builder::from(&config);

let client = match SecretsManagerCachingClient::from_builder(
    asm_builder,
    NonZeroUsize::new(10).unwrap(),
    Duration::from_secs(60),
)
.await
{
    Ok(c) => c,
    Err(_) => panic!("Handle this error"),
};

let secret_string = client
    .get_secret_value("MyTest", None, None)
    .await
    {
        Ok(c) => c.secret_string.unwrap(),
        Err(_) => panic!("Handle this error"),
    };

// Your code here
```

Rust AWS SDK を使用して Secrets Manager のシークレット値を取得する

アプリケーションでは、任意の AWS SDKs `BatchGetSecretValue`で `GetSecretValue`または を呼び出してシークレットを取得できます。ただし、クライアント側のキャッシュを使用してシークレット値をキャッシュすることをお勧めします。シークレットをキャッシュすると、速度が向上し、コストが削減されます。

Rust アプリケーションの場合は、Secrets Manager の Rust ベースのキャッシュコンポーネントを使用するか、GetSecretValue または BatchGetSecretValue を使用して SDK を直接呼び出します。

次の例で、Secrets Manager シークレットの値を取得する方法を示します。

必要な許可:`secretsmanager:GetSecretValue`

```
async fn show_secret(client: &Client, name: &str) -> Result<(), Error> {
    let resp = client.get_secret_value().secret_id(name).send().await?;

    println!("Value: {}", resp.secret_string().unwrap_or("No value!"));
```

```
    Ok(())
}
```

Amazon Elastic Kubernetes Service で AWS Secrets Manager シークレットを使用する

AWS Secrets Manager (ASCP) からのシークレットを Amazon EKS Pod にマウントされたファイルとして表示するには、Kubernetes AWS Secrets Store CSI ドライバーのシークレットと設定プロバイダーを使用できます。ASCP は、Amazon EC2 ノードグループを実行する Amazon Elastic Kubernetes Service 1.17+ で動作します。 AWS Fargate ノードグループはサポートされていません。ASCP を使用すると、秘密マネジャー でシークレットを保存および管理し、Amazon EKS で実行されているワークロードからシークレットを取得できます。シークレットに JSON 形式の複数のキー/値ペアが含まれている場合は、Amazon EKS にマウントするキー/値ペアを選択できます。ASCP は JMESPath 構文を使用して、シークレット内のキーと値のペアをクエリします。ASCP はパラメータストアパラメータを使用しても動作します。ASCP には、Amazon EKS による 2 つの認証方法が用意されています。最初のアプローチでは、サービスアカウントの IAM ロール (IRSA) を使用します。2 番目のアプローチでは、Pod Identity を使用します。各アプローチにはそれぞれの利点とユースケースがあります。

サービスアカウントの IAM ロール (IRSA) を使用した ASCP

ASCP with IAM Roles for Service Accounts (IRSA) を使用すると、Amazon EKS ポッド内のファイル AWS Secrets Manager として からシークレットをマウントできます。このアプローチは、次の場合に適しています。

- シークレットをポッドのファイルとしてマウントする必要がある。
- Amazon EC2 ノードグループで Amazon EKS バージョン 1.17 以降を使用している。
- JSON 形式のシークレットから特定のキーと値のペアを取得する。

詳細については、「the section called “Amazon EKS 用に ASCP を IRSA と統合する”」を参照してください。

Pod Identity を使用した ASCP

EKS Pod Identity を使用した ASCP

Pod Identity を使用した ASCP の方法では、セキュリティが強化され、Amazon EKS のシークレットにアクセスするための設定が簡素化されます。このアプローチは、次の場合に役立ちます。

- ポッドレベルでより詳細にアクセス許可を管理する必要がある。
- Amazon EKS バージョン 1.24 以降を使用している。
- パフォーマンスとスケーラビリティを向上させる必要がある。

詳細については、「the section called “Amazon EKS 用に ASCP を Pod Identity と統合する”」を参照してください。

適切なアプローチの選択

IRSA を使用した ASCP と Pod Identity を使用した ASCP のどちらを選択するかを決めるときは、次の要素を考慮します。

- Amazon EKSversion: Pod Identity には Amazon EKS 1.24 以降が必要ですが、CSI ドライバーは Amazon EKS 1.17 以降で動作します。
- セキュリティ要件: Pod Identity は、ポッドレベルでのより詳細な制御を提供します。
- パフォーマンス: 通常、大規模な環境では Pod Identity の方がパフォーマンスが向上します。
- 複雑さ: Pod Identity には個別のサービスアカウントが必要ないため、設定が簡素化されます。

特定の要件と Amazon EKS 環境に最適な方法を選択します。

Amazon EKS 用の ASCP をインストールする

このセクションでは、Amazon EKS の AWS シークレットと設定プロバイダーをインストールする方法について説明します。ASCP を使用すると、Secrets Manager のシークレットと のパラメータを Amazon EKS Pods のファイル AWS Systems Manager としてマウントできます。

前提条件

- Amazon EKS クラスター
  - Pod Identity 用にバージョン 1.24 以降
  - IRSA 用にバージョン 1.17 以降
- AWS CLI のインストールと設定
- kubectl が、Amazon EKS クラスター用にインストールされ、設定されている

- Helm (バージョン 3.0 以降)

ASCP をインストールして設定する

ASCP は secrets-store-csi-provider-aws リポジトリの GitHub で入手できます。リポジトリには、シークレットを作成してマウントするための YAML ファイルの例も含まれています。

インストール中に、FIPS エンドポイントを使用するように ASCP を設定できます。; エンドポイントのリストについては、「the section called “Secrets Manager エンドポイント”」を参照してください。

> **アドオンスキーマ検証スコープ**
>
> インストール中、アドオンはスキーマに対して AWS プロバイダー設定オプションのみを検証します。Secrets Store CSI ドライバーオプションは検証されません。Secrets Store CSI ドライバーの設定オプションの詳細については、GitHub ウェブサイトの Helm チャート README の設定を参照してください。

ASCP を EKS アドオンとしてインストールするには

1. のインストール `eksctl` (インストール手順)
2. 次のコマンドを実行して、デフォルト設定でアドオンをインストールします。

   ```
   eksctl create addon --cluster <your_cluster> --name aws-secrets-store-csi-driver-provider
   ```

   アドオンを設定する場合は、代わりに次のインストールコマンドを実行します。

   ```
   aws eks create-addon --cluster-name <your_cluster> --addon-name aws-secrets-store-csi-driver-provider --configuration-values 'file://path/to/config.yaml'
   ```

   設定ファイルは、YAML ファイルまたは JSON ファイルです。アドオンの設定スキーマを表示するには:

   a. 次のコマンドを実行し、最新バージョンのアドオンを書き留めます。

```
aws eks describe-addon-versions --addon-name aws-secrets-store-csi-driver-
provider
```

b. 次のコマンドを実行してアドオンの設定スキーマを確認し、 を前のステップのバージョン<version>に置き換えます。

```
aws eks describe-addon-configuration --addon-name aws-secrets-store-csi-driver-
provider --addon-version <version>
```

Helm を使用して ASCP をインストールするには

1. リポジトリが最新のチャートを指していることを確認するには、helm repo update . を使用します。
2. グラフをインストールします。次に helm install コマンドの例を示します。

```
helm install -n kube-system secrets-provider-aws aws-secrets-manager/secrets-store-
csi-driver-provider-aws
```

a. FIPS エンドポイントを使用するには、--set useFipsEndpoint=true のフラグを追加します。

b. スロットリングを設定するには、--set-json 'k8sThrottlingParams={"qps": "*number of queries per second*", "burst": "*number of queries per second*"}' のフラグを追加します。

c. Secrets Store CSI ドライバーがクラスターに既にインストールされている場合は、--set secrets-store-csi-driver.install=false フラグを追加します。これにより、依存関係としての Secrets Store CSI ドライバーのインストールはスキップされます。

リポジトリ内の YAML を使用してインストールするには

- 次のコマンドを使用します。

```
helm repo add secrets-store-csi-driver https://kubernetes-sigs.github.io/secrets-
store-csi-driver/charts
helm install -n kube-system csi-secrets-store secrets-store-csi-driver/secrets-
store-csi-driver
```

```
kubectl apply -f https://raw.githubusercontent.com/aws/secrets-store-csi-driver-
provider-aws/main/deployment/aws-provider-installer.yaml
```

インストールを検証する

EKS クラスター、Secrets Store CSI ドライバー、ASCP プラグインのインストールを検証するには、次の手順に従います。

1. EKS クラスターを検証します。

   ```
   eksctl get cluster --name clusterName
   ```

   このコマンドは、クラスターに関する情報を返します。

2. Secrets Store CSI ドライバーのインストールを検証します。

   ```
   kubectl get pods -n kube-system -l app=secrets-store-csi-driver
   ```

   `csi-secrets-store-secrets-store-csi-driver-xxx` のような名前の実行されているポッドが表示されます。

3. ASCP プラグインのインストールを検証します。

   YAML installation

   ```
   $ kubectl get pods -n kube-system -l app=csi-secrets-store-provider-aws
   ```

   出力の例:

   ```
   NAME                                     READY   STATUS    RESTARTS   AGE
   csi-secrets-store-provider-aws-12345      1/1     Running   0          2m
   ```

   Helm installation

   ```
   $  kubectl get pods -n kube-system -l app=secrets-store-csi-driver-provider-aws
   ```

   出力の例:

```
NAME                                                     READY   STATUS    RESTARTS
 AGE
secrets-provider-aws-secrets-store-csi-driver-provider-67890       1/1
 Running   0          2m
```

Running 状態のポッドが表示されます。

これらのコマンドを実行した後に、すべてが正しく設定されている場合は、すべてのコンポーネントがエラーなく実行されます。問題が発生した場合は、問題が発生している特定のポッドのログを確認してトラブルシューティングする必要がある場合があります。

トラブルシューティング

1. ASCP プロバイダーのログを確認するには、以下を実行します。

   ```
   kubectl logs -n kube-system -l app=csi-secrets-store-provider-aws
   ```

2. kube-system 名前空間内のすべてのポッドのステータスを確認します。

   ```
   kubectl -n kube-system get pods
   ```

   ```
   kubectl -n kube-system logs pod/PODID
   ```

   CSI ドライバーと ASCP に関連するすべてのポッドが「Running」状態である必要があります。

3. CSI ドライバーのバージョンを確認します。

   ```
   kubectl get csidriver secrets-store.csi.k8s.io -o yaml
   ```

   このコマンドは、インストールされている CSI ドライバーに関する情報を返します。

その他のリソース

Amazon EKS での ASCP の使用に関する詳細については、次のリソースを参照してください。

- Amazon EKS で Pod Identity を使用する
- GitHub のAWS Secrets Store CSI ドライバー

Amazon EKS の Pod Identity で AWS シークレットと設定プロバイダー CSI を使用する

AWS Secrets and Configuration Provider と Pod Identity Agent for Amazon Elastic Kubernetes Service の統合により、Amazon EKS で実行されているアプリケーションのセキュリティが強化され、設定が簡素化され、パフォーマンスが向上します。Pod Identity は、Secrets Manager からシークレットを取得するとき、またはパラメータストアからパラメータを取得するときに、Amazon EKS の IAM AWS Systems Manager 認証を簡素化します。

Amazon EKS Pod Identity によって Amazon EKS インターフェイスを介してアクセス許可を直接設定できるようになるため、Kubernetes アプリケーションの IAM アクセス許可を設定するプロセスが合理化され、ステップ数が減少し、Amazon EKS と IAM サービスを切り替える必要がなくなります。Pod Identity では、信頼ポリシーを更新することなく複数のクラスターで 1 つの IAM ロールを使用でき、より詳細なアクセスコントロールのためのロールセッションタグがサポートされます。このアプローチは、ロール間でアクセス許可ポリシーを再利用できるようにすることでポリシー管理を簡素化するだけでなく、一致するタグに基づいて AWS リソースへのアクセスを有効にすることでセキュリティを強化します。

仕組み

1. Pod Identity はポッドに IAM ロールを割り当てます。
2. ASCP はこのロールを使用して認証します AWS のサービス。
3. 承認されると、ASCP はリクエストされたシークレットを取得し、ポッドで使用できるようにします。

詳細については、「Amazon EKS ユーザーガイド」の「EKS Pod Identity の仕組みを理解する」を参照してください。

前提条件

> **Important**
>
> Pod Identity は、クラウド内の Amazon EKS でのみサポートされています。Amazon EC2 インスタンスの Amazon EKS Anywhere、Red Hat OpenShift Service on AWS、またはセルフマネージド Kubernetes クラスターではサポートされていません。

- Amazon EKS クラスター (バージョン 1.24 以降)

- を介した AWS CLI および Amazon EKS クラスターへのアクセス kubectl
- 2 へのアクセス AWS アカウント (クロスアカウントアクセス用)

Amazon EKS Pod Identity Agent をインストールする

クラスターで Pod Identity を使用するには、Amazon EKS Pod Identity Agent アドオンをインストールする必要があります。

Pod Identity Agent をインストールするには

- Pod Identity Agent アドオンをクラスターにインストールします。

```
eksctl create addon \
  --name eks-pod-identity-agent \
  --cluster clusterName \
  --region region
```

Pod Identity を使用して ASCP を設定する

1. ポッドがアクセスする必要があるシークレットに secretsmanager:GetSecretValue と secretsmanager:DescribeSecret のアクセス許可を付与するアクセス許可ポリシーを作成します。ポリシーの例については「the section called “例: 個々のシークレットを読み、記述するアクセス許可”」を参照してください。
2. Pod Identity の Amazon EKS サービスプリンシパルが引き受けることができる IAM ロールを作成します。

   JSON

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

```
            "sts:TagSession"
        ]
      }
    ]
  }
```

IAM ポリシーをロールにアタッチします。

```
aws iam attach-role-policy \
  --role-name MY_ROLE \
  --policy-arn POLICY_ARN
```

3. Pod Identity の関連付けを作成します。例については、「Amazon EKS ユーザーガイド」の「Pod Identity の関連付けを作成する」を参照してください。
4. ポッドにマウントするシークレットを指定する SecretProviderClass を作成します。

```
kubectl apply -f https://raw.githubusercontent.com/aws/secrets-store-csi-driver-
provider-aws/main/examples/ExampleSecretProviderClass-PodIdentity.yaml
```

SecretProviderClass における IRSA と Pod Identity の主な違いは、オプションのパラメータ usePodIdentity です。これは、認証アプローチを決定するオプションのフィールドです。指定しない場合、デフォルトではサービスアカウントの IAM ロール (IRSA) が使用されます。

- EKS Pod Identity を使用するには、"true", "True", "TRUE", "t", "T" のいずれかの値を使用します。
- IRSA を明示的に使用するには、"false", "False", "FALSE", "f", or "F" のいずれかの値に設定します。

5. /mnt/secrets-store の下に、シークレットをマウントするポッドをデプロイします。

```
kubectl apply -f https://raw.githubusercontent.com/aws/secrets-store-csi-driver-
provider-aws/main/examples/ExampleDeployment-PodIdentity.yaml
```

6. プライベート Amazon EKS クラスターを使用する場合は、クラスターがある VPC に AWS STS エンドポイントがあることを確認してください。エンドポイントの作成の詳細については、「AWS Identity and Access Management ユーザーガイド」の「インターフェイス VPC エンドポイント」を参照してください。

シークレットのマウントを検証する

シークレットが正しくマウントされていることを検証するには、次のコマンドを実行します。

```
kubectl exec -it $(kubectl get pods | awk '/pod-identity-deployment/{print $1}' | head
 -1) -- cat /mnt/secrets-store/MySecret
```

Amazon EKS Pod Identity に Secrets Manager のシークレットへのアクセスを許可するには

1. ポッドがアクセスする必要があるシークレットに `secretsmanager:GetSecretValue` と `secretsmanager:DescribeSecret` のアクセス許可を付与するアクセス許可ポリシーを作成します。ポリシーの例については「the section called “例: 個々のシークレットを読み、記述するアクセス許可”」を参照してください。
2. シークレットをまだ作成していない場合は、Secrets Manager でシークレットを作成します。

トラブルシューティング

ポッドのデプロイを記述すると、ほとんどのエラーを表示できます。

コンテナのエラーメッセージを表示するには

1. 次のコマンドで、ポッド名のリストを取得します。デフォルトの名前空間を使用していない場合は、`-n` *NAMESPACE* を使用してください。

   ```
   kubectl get pods
   ```

2. ポッドを記述するには、次のコマンドで、前のステップで見つけたポッドの ポッド ID を使用します (*PODID* の場合)。デフォルトの名前空間を使用していない場合は、`-n` *NAMESPACE* を使用してください。

   ```
   kubectl describe pod/PODID
   ```

ASCP のエラーを表示するには

- プロバイダーログで詳細情報を検索するには、次のコマンドの *PODID* に、csi-secrets-store-provider-aws ポッドの ID を入力します。

  ```
  kubectl -n kube-system get pods
  ```

```
kubectl -n kube-system logs pod/PODID
```

サービスアカウント (IRSA) の IAM ロールで AWS シークレットと設定プロバイダー CSI を使用する

トピック

- 前提条件
- アクセス制御を設定する
- マウントするシークレットを特定する
- トラブルシューティング

前提条件

- Amazon EKS クラスター (バージョン 1.17 以降)
- を介した AWS CLI および Amazon EKS クラスターへのアクセス `kubectl`

アクセス制御を設定する

ASCP は Amazon EKS Pod Identity を取得し、IAM ロールと交換します。その IAM ロールの IAM ポリシーでアクセス許可を設定します。ASCP が IAM ロールを引き受けると、認証されたシークレットへのアクセス許可が付与されます。他のコンテナは、IAM ロールに関連付けられていない限り、シークレットにアクセスできません。

Amazon EKS ポッドに Secrets Manager のシークレットへのアクセスを許可するには

1. ポッドがアクセスする必要があるシークレットに `secretsmanager:GetSecretValue` と `secretsmanager:DescribeSecret` のアクセス許可を付与するアクセス許可ポリシーを作成します。ポリシーの例については「the section called “例: 個々のシークレットを読み、記述するアクセス許可”」を参照してください。
2. クラスター用に IAM OpenID Connect (OIDC) プロバイダーがない場合は作成します。詳細については、「Amazon EKS ユーザーガイド」の「クラスターに IAM OIDC プロバイダーを作成する」を参照してください。
3. サービスアカウントの IAM ロールを作成して、ポリシーをアタッチします。詳細については、「Amazon EKS ユーザーガイド」の「サービスアカウントの IAM ロールを作成する」を参照してください。

4. プライベート Amazon EKS クラスターを使用する場合は、クラスターがある VPC に AWS STS エンドポイントがあることを確認してください。エンドポイントの作成の詳細については、「AWS Identity and Access Management ユーザーガイド」の「インターフェイス VPC エンドポイント」を参照してください。

マウントするシークレットを特定する

ASCP が (ファイルシステム上のファイルとして) Amazon EKS 内にマウントするシークレットを判別するには、the section called “SecretProviderClass” YAML ファイルを作成します。`SecretProviderClass` では、マウントするシークレットと、それらのマウントに使用されるファイル名がリストされます。`SecretProviderClass` は、参照する Amazon EKS ポッドと同じ名前空間にある必要があります。

シークレットをファイルとしてマウントする

次の手順は、サンプル YAML ファイル (ExampleSecretProviderClass.yaml と ExampleDeployment.yaml) を使用してシークレットをファイルとしてマウントする方法を示します。

Amazon EKS でシークレットをマウントするには

1. `SecretProviderClass` をポッドに適用します。

```
kubectl apply -f ExampleSecretProviderClass.yaml
```

2. ポッドをデプロイします。

```
kubectl apply -f ExampleDeployment.yaml
```

3. ASCP はファイルをマウントします。

トラブルシューティング

ポッドのデプロイを記述すると、ほとんどのエラーを表示できます。

コンテナのエラーメッセージを表示するには

1. 次のコマンドで、ポッド名のリストを取得します。デフォルトの名前空間を使用していない場合は、`-n` *`nameSpace`* を使用してください。

```
kubectl get pods
```

2. ポッドを記述するには、次のコマンドで、前のステップで見つけたポッドの ポッド ID を使用します (*podId* の場合)。デフォルトの名前空間を使用していない場合は、-n *nameSpace* を使用してください。

```
kubectl describe pod/podId
```

ASCP のエラーを表示するには

- プロバイダーログで詳細情報を検索するには、次のコマンドの *podId* に、csi-secrets-store-provider-aws ポッドの ID を入力します。

```
kubectl -n kube-system get pods
kubectl -n kube-system logs Pod/podId
```

- **SecretProviderClass** CRD がインストールされていることを確認します。

```
kubectl get crd secretproviderclasses.secrets-store.csi.x-k8s.io
```

このコマンドは、SecretProviderClass カスタムリソース定義に関する情報を返します。

- SecretProviderClass オブジェクトが作成されたことを確認します。

```
kubectl get secretproviderclass SecretProviderClassName -o yaml
```

AWS シークレットと設定プロバイダーのコード例

ASCP の認証とアクセスコントロールの例

例: Amazon EKS Pod Identity サービス (pods.eks.amazonaws.com) がロールを引き受け、セッションにタグを付けることを許可する IAM ポリシー:

JSON

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

SecretProviderClass

ASCP を使用して Amazon EKS にマウントするシークレットを YAML で記述します。例については、the section called “SecretProviderClass の使用” を参照してください。

SecretProviderClass YAML 構造

```
apiVersion: secrets-store.csi.x-k8s.io/v1
kind: SecretProviderClass
metadata:
   name: name
spec:
  provider: aws
  parameters:
    region:
```

```
    failoverRegion:
    pathTranslation:
    usePodIdentity:
    preferredAddressType:
    objects:
```

パラメータフィールドには、以下のマウントリクエストの詳細が含まれます。

リージョン

(オプション) シークレット AWS リージョン の 。このフィールドを使用しない場合、ASCP はノード上のアノテーションからリージョンを検索します。この検索では、マウントリクエストにオーバーヘッドが追加されるため、大量のポッドを使用するクラスターでリージョンを指定することをお勧めします。

同時に `failoverRegion` も指定すると、ASCP は両方のリージョンからシークレットを試行します。どちらかのリージョンが認証の問題などで 4xx エラーを返した場合、ASCP はいずれのシークレットもマウントしません。シークレットが `region` から正常に取得されると、ASCP はそのシークレット値をマウントします。`region` からはシークレットが正常に取得されないものの、`failoverRegion` からは正常に取得された場合、ASCP は取得された方のシークレット値をマウントします。

failoverRegion

(オプション) このフィールドを含めると、ASCP は `region` で定義されているリージョンと、このフィールドで定義されているリージョンからのシークレット取得を試行します。どちらかのリージョンが認証の問題などで 4xx エラーを返した場合、ASCP はいずれのシークレットもマウントしません。シークレットが `region` から正常に取得されると、ASCP はそのシークレット値をマウントします。`region` からはシークレットが正常に取得されないものの、`failoverRegion` からは正常に取得された場合、ASCP は取得された方のシークレット値をマウントします。このフィールドの使用例については、「マルチリージョンシークレットのフェイルオーバー」を参照してください。

pathTranslation

(オプション) Amazon EKS のファイル名に、Linux のスラッシュ (/) のようなパス区切り文字が含まれている場合に使用する単一の置換文字。ASCP では、パス区切り文字が含まれているファイルを作成し、マウントすることはできません。代わりに、ASCP はパス区切り文字を別の文字に置き換えます。このフィールドを使用しない場合、置換文字にはアンダースコア (_) が使用されます。例えば、`My/Path/Secret` は `My_Path_Secret` としてマウントされます。

文字の置換を防ぐには、文字列 `False` を入力してください。

usePodIdentity

(オプション) 認証アプローチを決定します。指定しない場合、デフォルトではサービスアカウントの IAM ロール (IRSA) になります。

- EKS Pod Identity を使用するには、`"true"`、`"True"`、`"TRUE"`、`"t"`、または `"T"` のいずれかの値を使用します。
- IRSA を明示的に使用するには、`"false"`、`"False"`、`"FALSE"`、`"f"`、または `"F""= のいずれかの値に設定します。

preferredAddressType

(オプション) Pod Identity Agent のエンドポイント通信の優先 IP アドレスタイプを指定します。このフィールドは EKS Pod Identity 機能を使用する場合にのみ適用され、サービスアカウントの IAM ロールを使用する場合は無視されます。値では大文字と小文字は区別されません。次の値を指定できます。

- `"ipv4"`、`"IPv4"`、または `"IPV4"` – Pod Identity Agent IPv4 エンドポイントの使用を強制します
- `"ipv6"`、`"IPv6"`、または `"IPV6"` – Pod Identity Agent IPv6 エンドポイントの使用を強制します
- 指定なし – 自動エンドポイント選択を使用し、最初に IPv4 エンドポイントを試して、IPv4 が失敗した場合は IPv6 エンドポイントにフォールバックします

objects

マウントするシークレットの YAML 宣言を含む文字列 YAML 複数行の文字列またはパイプ (|) 文字を使用することをお勧めします。

objectName

必須。取得するシークレットまたはパラメータの名前を指定します。Secrets Manager の場合、これは `SecretId` パラメータであり、シークレットのわかりやすい名前または完全な ARN のいずれかを使用できます。SSM Parameter Store の場合、これはパラメータの `Name` であり、パラメータの名前または完全な ARN のいずれかを使用できます。

objectType

`objectName` で Secrets Manager ARN を使用しない場合は必須。`secretsmanager` または `ssmparameter` のいずれかになります。

objectAlias

(オプション) Amazon EKS ポッド内のシークレットのファイル名。このフィールドを指定しなかった場合、objectName がファイル名として表示されます。

filePermission

(オプション) シークレットをマウントするファイルアクセス許可を指定する 4 桁の 8 進文字列。このフィールドを指定しない場合は、デフォルトで "0644" になります。

objectVersion

(オプション) シークレットのバージョン ID バージョン ID は、シークレットを変更するたびに更新の必要が生じるため、これは推奨されません。デフォルトでは、最新バージョンが使用されます。failoverRegion を含めると、このフィールドはプライマリの objectVersion になります。

objectVersionLabel

(オプション) バージョンのエイリアス デフォルトでは、最新バージョン AWSCURRENT になります。詳細については、「the section called “シークレットバージョン”」を参照してください。failoverRegion を含めると、このフィールドはプライマリの objectVersionLabel になります。

jmesPath

(オプション) シークレット内のキーから Amazon EKS にマウントされるファイルへのマッピング このフィールドを使用するには、シークレット値を JSON 形式にする必要があります。このフィールドを使用する場合は、サブフィールド path および objectAlias を含める必要があります。

パス

シークレット値の JSON 内のキー/値のペアからのキー フィールドにハイフンが含まれている場合は、一重引用符でエスケープします。例: path: '"hyphenated-path"'

objectAlias

Amazon EKS ポッドにマウントされるファイル名。フィールドにハイフンが含まれている場合は、一重引用符でエスケープします。例: objectAlias: '"hyphenated-alias"'

filePermission

(オプション) シークレットをマウントするファイルアクセス許可を指定する 4 桁の 8 進文字列。このフィールドを指定しない場合、デフォルトで親オブジェクトのファイルアクセス許可になります。

failoverObject

(オプション) このフィールドを指定した場合、ASCP はプライマリ `objectName` で指定されたシークレットと、`failoverObject`、`objectName` サブフィールドで指定されたシークレットの両方の取得を試行します。どちらかが認証の問題などで 4xx エラーを返した場合、ASCP はいずれのシークレットもマウントしません。シークレットがプライマリ `objectName` から正常に取得されると、ASCP はそのシークレット値のマウントを行います。シークレットが、プライマリ `objectName` からは正常に取得されなかったものの、フェイルオーバー `objectName` からは正常に取得された場合、ASCP はその (取得された) シークレット値をマウントします。このフィールドを含める場合は、フィールド `objectAlias` も含める必要があります。このフィールドの使用例については、「別のシークレットにフェイルオーバーする」を参照してください。

通常、このフィールドはフェイルオーバーシークレットがレプリカではない場合に使用します。レプリカを指定する際の例については、「マルチリージョンシークレットのフェイルオーバー」を参照してください。

objectName

フェイルオーバーシークレットの名前または完全な ARN ARN を使用する場合、ARN 内のリージョンはフィールド `failoverRegion` と一致する必要があります。

objectVersion

(オプション) シークレットのバージョン ID プライマリ `objectVersion` と一致している必要があります。バージョン ID は、シークレットを変更するたびに更新の必要が生じるため、これは推奨されません。デフォルトでは、最新バージョンが使用されます。

objectVersionLabel

(オプション) バージョンのエイリアス デフォルトでは、最新バージョン AWSCURRENT になります。詳細については、「the section called “シークレットバージョン”」を参照してください。

Amazon EKS ポッドにシークレットをマウントするための基本的な SecretProviderClass 設定を作成します。

Pod Identity

同じ Amazon EKS クラスターのシークレットを使用する SecretProviderClass:

```
apiVersion: secrets-store.csi.x-k8s.io/v1
```

```
kind: SecretProviderClass
metadata:
  name: aws-secrets-manager
spec:
  provider: aws
  parameters:
    objects: |
      - objectName: "mySecret"
        objectType: "secretsmanager"
    usePodIdentity: "true"
```

IRSA

```
apiVersion: secrets-store.csi.x-k8s.io/v1
kind: SecretProviderClass
metadata:
  name: deployment-aws-secrets
spec:
  provider: aws
  parameters:
    objects: |
        - objectName: "MySecret"
          objectType: "secretsmanager"
```

SecretProviderClass の使用

これらの例を使用して、さまざまなシナリオの SecretProviderClass 設定を作成します。

例: 名前または ARN でシークレットをマウントする

この例では、3 つの異なるタイプのシークレットをマウントする方法を示します。

- 完全な ARN によって指定されたシークレット
- 名前で指定されたシークレット
- シークレットの特定のバージョン

```
apiVersion: secrets-store.csi.x-k8s.io/v1
kind: SecretProviderClass
metadata:
  name: aws-secrets
spec:
```

```
  provider: aws
  parameters:
    objects: |
      - objectName: "arn:aws:secretsmanager:us-east-2:777788889999:secret:MySecret2-
d4e5f6"
      - objectName: "MySecret3"
        objectType: "secretsmanager"
      - objectName: "MySecret4"
        objectType: "secretsmanager"
        objectVersionLabel: "AWSCURRENT"
```

例: シークレットからキーと値のペアをマウントする

この例では、JSON 形式のシークレットから特定のキーと値のペアをマウントする方法を示します。

```
apiVersion: secrets-store.csi.x-k8s.io/v1
kind: SecretProviderClass
metadata:
  name: aws-secrets
spec:
  provider: aws
  parameters:
    objects: |
      - objectName: "arn:aws:secretsmanager:us-east-2:777788889999:secret:MySecret-
a1b2c3"
        jmesPath:
            - path: username
              objectAlias: dbusername
            - path: password
              objectAlias: dbpassword
```

例: ファイルアクセス許可でシークレットをマウントする

この例では、特定のファイルアクセス許可を使用してシークレットをマウントする方法を示します。

```
apiVersion: secrets-store.csi.x-k8s.io/v1
kind: SecretProviderClass
metadata:
  name: aws-secrets
spec:
  provider: aws
  parameters:
    objects: |
```

```
      - objectName: "mySecret"
        objectType: "secretsmanager"
        filePermission: "0600"
        jmesPath:
            - path: username
              objectAlias: dbusername
              filePermission: "0400"
```

例: フェイルオーバー設定の例

これらの例では、シークレットのフェイルオーバーを設定する方法を示します。

マルチリージョンシークレットのフェイルオーバー

この例では、複数のリージョンにレプリケートされたシークレットの自動フェイルオーバーを設定する方法を示します。

```
apiVersion: secrets-store.csi.x-k8s.io/v1
kind: SecretProviderClass
metadata:
  name: aws-secrets
spec:
  provider: aws
  parameters:
    region: us-east-1
    failoverRegion: us-east-2
    objects: |
      - objectName: "MySecret"
```

別のシークレットにフェイルオーバーする

この例では、(レプリカではない) 別のシークレットへのフェイルオーバーを設定する方法を示します。

```
apiVersion: secrets-store.csi.x-k8s.io/v1
kind: SecretProviderClass
metadata:
  name: aws-secrets
spec:
  provider: aws
  parameters:
    region: us-east-1
    failoverRegion: us-east-2
```

```
    objects: |
      - objectName: "arn:aws:secretsmanager:us-east-1:777788889999:secret:MySecret-
a1b2c3"
        objectAlias: "MyMountedSecret"
        failoverObject:
          - objectName: "arn:aws:secretsmanager:us-
east-2:777788889999:secret:MyFailoverSecret-d4e5f6"
```

その他のリソース

Amazon EKS での ASCP の使用に関する詳細については、次のリソースを参照してください。

- Amazon EKS で Pod Identity を使用する
- AWS シークレットと設定プロバイダーの使用
- GitHub のAWS Secrets Store CSI ドライバー

AWS Lambda 関数で AWS Secrets Manager シークレットを使用する

AWS Lambda は、サーバーのプロビジョニングや管理を行わずにコードを実行できるようにするサーバーレスコンピューティングサービスです。の一機能である Parameter Store AWS Systems Managerは、設定データ管理とシークレット管理のための安全な階層型ストレージを提供します。AWS Parameters and Secrets Lambda Extension を使用して、 SDK を使用せずに Lambda 関数でシーク AWS Secrets Manager レットと Parameter Store パラメータを取得してキャッシュできます。この拡張機能の使用の詳細については、「Lambda デベロッパーガイド」の「Lambda 関数で Secrets Manager シークレットを使用する」を参照してください。

Lambda で Secrets Manager シークレットを使用する

Lambda デベロッパーガイドには、Lambda 関数で Secrets Manager シークレットを使用するための包括的な手順が記載されています。開始するには、以下の手順を実行します。

1. 「Lambda 関数で Secrets Manager シークレットを使用する」の段階的チュートリアルに従います。これには、以下が含まれます。
   - 任意のランタイム (Python、Node.js、Java) で Lambda 関数を作成する
   - AWS パラメータとシークレットの Lambda 拡張機能をレイヤーとして追加する
   - 必要なアクセス許可を設定する

- 拡張機能からシークレットを取得するためのコードを記述する
- 関数をテストする

2. キャッシュ設定やタイムアウトなど、拡張機能の動作を設定するための環境変数について説明します
3. シークレットローテーションを使用する際のベストプラクティスを理解する

VPC で Secrets Manager と Lambda を使用する

Lambda 関数が VPC で実行されている場合は、拡張機能が Secrets Manager を呼び出すことができるように VPC エンドポイントを作成する必要があります。詳細については、「the section called "VPC エンドポイント (AWS PrivateLink)"」を参照してください。

AWS パラメータとシークレットの Lambda 拡張機能の使用

この拡張機能では、Secrets Manager のシークレットと Parameter Store のパラメータの両方を取得できます。拡張機能で Parameter Store のパラメータを使用する方法の詳細については、「AWS Systems Manager ユーザーガイド」の「Lambda 関数での Parameter Store のパラメータの使用」を参照してください。

Systems Manager のドキュメントには以下が含まれます。

- 拡張機能が Parameter Store と連携する方法の詳細な説明
- Lambda 関数に拡張機能を追加する方法の説明
- 拡張機能を設定するための環境変数
- パラメータを取得するためのサンプルコマンド
- サポートされているすべてのアーキテクチャとリージョンの拡張機能 ARN の完全なリスト

AWS ワークロード認証情報プロバイダーの使用

AWS ワークロード認証情報プロバイダーの仕組み

AWS ワークロード認証情報プロバイダー (以前の AWS Secrets Manager エージェント) は、コンピューティング環境全体で Secrets Manager からシークレットを使用する方法を標準化するのに役立つクライアント側の HTTP サービスを提供します。これは次のサービスと併用できます。

- AWS Lambda

- Amazon Elastic Container Service
- アマゾン エラスティックKubernetesサービス
- Amazon Elastic Compute Cloud

AWS ワークロード認証情報プロバイダーはシークレットを取得してメモリにキャッシュするため、アプリケーションは Secrets Manager を直接呼び出す代わりに localhost からシークレットを取得できます。 AWS ワークロード認証情報プロバイダーはシークレットのみを読み取ることができ、変更することはできません。

AWS ワークロード認証情報プロバイダーはオープンソースです。ソースコード、インストール手順、最新のリリース情報は、GitHub で入手できます。

> **Important**
>
> AWS ワークロード認証情報プロバイダーは、環境の AWS 認証情報を使用して Secrets Manager を呼び出します。これには、シークレットセキュリティの向上に役立つ Server Side Request Forgery (SSRF) に対する保護が含まれています。 AWS ワークロード認証情報プロバイダーは、デフォルトで最高優先度のキー交換としてポスト量子 ML-KEM キー交換を使用します。

AWS ワークロード認証情報プロバイダーのキャッシュについて

AWS ワークロード認証情報プロバイダーは、 AWS ワークロード認証情報プロバイダーの再起動時にリセットされるインメモリキャッシュを使用します。以下に基づいて、キャッシュされたシークレット値を定期的に更新します。

- デフォルトの更新頻度 (TTL) は 300 秒です
- 設定ファイルを使用して TTL を変更できます。
- 更新は、TTL の有効期限が切れた後にシークレットをリクエストしたときに発生します。

> **Note**
>
> AWS ワークロード認証情報プロバイダーにはキャッシュの無効化は含まれません。キャッシュエントリの有効期限が切れる前にシークレットがローテーションした場合、 AWS ワークロード認証情報プロバイダーは古いシークレット値を返す可能性があります。

AWS ワークロード認証情報プロバイダーは、 のレスポンスと同じ形式でシークレット値を返しますGetSecretValue。シークレット値はキャッシュ内で暗号化されません。

トピック

- AWS ワークロード認証情報プロバイダーをダウンロードする
- ソースから AWS ワークロード認証情報プロバイダーを構築する
- AWS ワークロード認証情報プロバイダーをインストールする
- AWS ワークロード認証情報プロバイダーを使用してシークレットを取得する
- refreshNow パラメータについて
- ロール連鎖を使用してアカウント間でシークレットを取得する
- 起動時のシークレットのプリフェッチ
- AWS ワークロード認証情報プロバイダーを設定する
- オプション機能
- ログ記録
- セキュリティに関する考慮事項

AWS ワークロード認証情報プロバイダーをダウンロードする

AWS ワークロード認証情報プロバイダーのビルド済みバイナリをダウンロードするには、次のリンクを使用します。Windows のリリースは署名されています。

プラットフォーム	アーキテクチャ	URL のダウンロード	チェックサム (SHA-256)
Linux	x86-64	Linux x86-64 用 (3.1.1) のダウンロード	471c1978f8bf63a0aefb425e974ac3c816c7e04feb302236512eea375160de4
Linux	AArch64	for Linux AArch64 (3.1.1) のダウンロード	8b09dc4e58b84379f580a9ae179940bcb3de0338421f638a43376bf73380ffb6

プラットフォーム	アーキテクチャ	URL のダウンロード	チェックサム (SHA-256)
Windows	x86-64	Windows x86-64 用のダウンロード (3.1.1)	5fbdac4017e630556b94b90e5ed9ed11be69ac7a47e67655e20181ce990dbe12

ソースから AWS ワークロード認証情報プロバイダーを構築する

または、ソースから AWS ワークロード認証情報プロバイダーを構築することもできます。開始する前に、プラットフォーム用の標準開発ツールとRustツールがインストールされていることを確認してください。

> **Note**
>
> macOS でfipsこの機能を有効にしてプロバイダーを構築するには、現在次の回避策が必要です。
>
> - `xcrun --show-sdk-path` の実行の結果に設定された SDKROOT という環境変数を作成する

RPM-based systems

RPM ベースのシステム上に構築するには

1. リポジトリで提供されている `install` スクリプトを使用します。

   スクリプトは、起動時にランダムな SSRF トークンを生成し、ファイル `/var/run/awssmatoken` に保存します。トークンは、インストールスクリプトが作成する `aws-wcp-token` グループによって読み取り可能です。
2. アプリケーションがトークンファイルを読み取れるようにするには、アプリケーションが実行するユーザーアカウントを `aws-wcp-token` グループに追加する必要があります。例えば、次の usermod コマンドを使用してトークンファイルを読み取るアクセス許可をアプリケーションに付与できます。ここで *<APP_USER>* は、アプリケーションを実行するユーザー ID です。

```
sudo usermod -aG aws-wcp-token <APP_USER>
```

開発ツールのインストール

AL2023 などの RPM ベースのシステムでは、開発ツールグループをインストールします。

```
sudo yum -y groupinstall "Development Tools"
```

3. Rust のインストール

「Rust ドキュメント」の「Rust のインストール」の指示に従います。

```
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh # Follow the on-
screen instructions
. "$HOME/.cargo/env"
```

4. プロバイダーを構築する

貨物ビルドコマンドを使用して AWS ワークロード認証情報プロバイダーを構築します。

```
cargo build --release
```

実行ファイルは `target/release/aws-workload-credentials-provider` にあります。

Debian-based systems

Debian ベースのシステム上に構築するには

1. 開発ツールのインストール

Ubuntu などの Debian ベースのシステムでは、build-essential パッケージをインストールします。

```
sudo apt install build-essential
```

2. Rust のインストール

「Rust ドキュメント」の「Rust のインストール」の指示に従います。

```
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh # Follow the on-
screen instructions
. "$HOME/.cargo/env"
```

3. プロバイダーを構築する

   貨物ビルドコマンドを使用して AWS ワークロード認証情報プロバイダーを構築します。

   ```
   cargo build --release
   ```

   実行ファイルは `target/release/aws-workload-credentials-provider` にあります。

Windows

Windows でビルドするには

1. 開発環境のセットアップ

   「Microsoft Windows ドキュメント」の「Windows for Rust で開発環境をセットアップする」の手順に従います。

2. プロバイダーを構築する

   貨物ビルドコマンドを使用して AWS ワークロード認証情報プロバイダーを構築します。

   ```
   cargo build --release
   ```

   実行ファイルは `target/release/aws-workload-credentials-provider.exe` にあります。

Cross-compile natively

ネイティブにクロスコンパイルするには

1. クロスコンパイルツールをインストールする

   `cargo-xwin` をインストールします。

```
cargo install cargo-xwin
```

2. Rust ビルドターゲットを追加する

   Windows MSVC ビルドターゲットをインストールします。

   ```
   rustup target add x86_64-pc-windows-msvc
   ```

3. Windows 用にビルドする

   Windows のプロバイダーをクロスコンパイルします。

   ```
   cargo xwin build --release --target x86_64-pc-windows-msvc
   ```

   実行ファイルは `target/x86_64-pc-windows-msvc/release/aws-workload-credentials-provider.exe` にあります。

AWS ワークロード認証情報プロバイダーをインストールする

次のインストールオプションからコンピューティング環境を選択します。

Amazon EC2

Amazon EC2 Linuxインスタンスでは、RPM パッケージ (AL2023 で利用可能) または手動インストールスクリプトを使用して AWS 、ワークロード認証情報プロバイダーをインストールできます。

オプション A: RPM パッケージをインストールする (AL2023)

1. パッケージをインストールする

   RPM パッケージをインストールします。パッケージマネージャーは最新バージョンを見つけます。

   ```
   sudo dnf install aws-workload-credentials-provider
   ```

   RPM パッケージは、以下を自動的にセットアップします。

   - `aws-wcp` サービスユーザー、`awscreds`および `aws-wcp-token`グループ。

- のバイナリスクリプトとヘルパースクリプト/opt/aws/workload-credentials-provider/。
- AWS Secrets Manager および トークンsystemdサービス。自動的に開始されます。
- のランダム SSRF トークン/var/run/awssmatoken。

Secrets Manager 機能は、インストール後すぐに使用できます。RPM パッケージを使用して をインストールすると、 AWS ワークロード認証情報プロバイダーはデフォルトパスから設定オプションを読み取ります/etc/aws-workload-credentials-provider/config.toml。設定をカスタマイズするには、そのファイルを作成または編集します。

2. (オプション) 証明書管理機能を有効にする

   から証明書を取得および更新するには AWS Certificate Manager、AWS Certificate Manager「 ユーザーガイド」の「証明書の自動化」を参照してください。

3. アプリケーションのアクセス許可を設定する

   アプリケーションが SSRF トークンファイルを読み取れるようにするには、アプリケーションのユーザーアカウントを aws-wcp-tokenグループに追加します。

   ```
   sudo usermod -aG aws-wcp-token APP_USER
   ```

   APP_USER を、アプリケーションが実行されるユーザー ID に置き換えます。

4. (オプション) プロバイダーログへのアクセスを許可する

   プロバイダーログを読み取るには、ユーザーをawscredsグループに追加し、変更を有効にするためにログアウトして再度ログイン (または を実行newgrp awscreds) します。

   ```
   sudo usermod -aG awscreds APP_USER
   ```

> **AWS ワークロード認証情報プロバイダーのアンインストール**
>
> AWS ワークロード認証情報プロバイダーをアンインストールするには、 を実行しますsudo dnf remove aws-workload-credentials-provider。アンインストールすると、すべてのサービスが停止し、バイナリとサービスユニットが削除されます。アンインストールプロセスでは、aws-wcpユーザー、グループ、ログディレクトリが保持されます。

オプション B: インストールスクリプトを実行する (任意の Linux)

1. 設定ディレクトリに移動する

   設定ディレクトリに変更します:

   ```
   cd aws_workload_credentials_provider_common/configuration
   ```

2. インストールスクリプトを実行する

   リポジトリで提供されている `install` スクリプトを実行します。

   スクリプトは、起動時にランダムな SSRF トークンを生成し、ファイル `/var/run/awssmatoken` に保存します。トークンは、インストールスクリプトが作成する `aws-wcp-token` グループによって読み取り可能です。

3. アプリケーションのアクセス許可を設定する

   アプリケーションが実行されるユーザーアカウントを `aws-wcp-token` グループに追加します:

   ```
   sudo usermod -aG aws-wcp-token APP_USER
   ```

   APP_USER を、アプリケーションが実行されるユーザー ID に置き換えます。

Container Sidecar

Docker を使用して AWS 、ワークロード認証情報プロバイダーをサイドカーコンテナとしてアプリケーションと一緒に実行できます。その後、アプリケーションは AWS ワークロード認証情報プロバイダーが提供するローカル HTTP サーバーからシークレットを取得できます。Docker の詳細については、「Docker ドキュメント」を参照してください。

AWS ワークロード認証情報プロバイダーのサイドカーコンテナを作成するには

1. プロバイダー Dockerfile を作成する

   AWS ワークロード認証情報プロバイダーサイドカーコンテナの Dockerfile を作成します。

   ```
   # Use the latest Debian image as the base
   FROM debian:latest
   ```

```
# Set the working directory inside the container
WORKDIR /app

# Copy the Workload Credentials Provider binary to the container
COPY aws-workload-credentials-provider .

# Install any necessary dependencies
RUN apt-get update && apt-get install -y ca-certificates

# Set the entry point to run the provider
ENTRYPOINT ["./aws-workload-credentials-provider", "sm", "start"]
```

2. アプリケーション Dockerfile を作成する

   クライアントアプリケーション用の Dockerfile を作成します。

3. Docker Compose ファイルを作成する

   Docker Compose ファイルを作成して、共有ネットワークインターフェイスで両方のコンテナを実行します:

> **Important**
>
> AWS ワークロード AWS 認証情報プロバイダーを使用するには、アプリケーションの認証情報と SSRF トークンをロードする必要があります。Amazon EKS と Amazon ECS については、以下を参照してください。
>
> - 「Amazon EKS ユーザーガイド」の「アクセスの管理」
> - 「Amazon ECS デベロッパーガイド」の「Amazon ECS タスク IAM ロール」

```
version: '3'
services:
    client-application:
    container_name: client-application
    build:
        context: .
        dockerfile: Dockerfile.client
    command: tail -f /dev/null  # Keep the container running


    workload-credentials-provider:
```

```
    container_name: workload-credentials-provider
    build:
        context: .
        dockerfile: Dockerfile.provider
    network_mode: "container:client-application"  # Attach to the client-
application container's network
    depends_on:
        - client-application
```

4. プロバイダーバイナリをコピーする

   `aws-workload-credentials-provider` バイナリを Dockerfiles および Docker Compose ファイルと同じディレクトリにコピーします。

5. コンテナの構築と実行

   Docker Compose を使用し、コンテナを構築して実行します:

   ```
   docker-compose up --build
   ```

6. 次の手順

   AWS ワークロード認証情報プロバイダーを使用して、クライアントコンテナからシークレットを取得できるようになりました。詳細については、「the section called “AWS ワークロード認証情報プロバイダーを使用してシークレットを取得する”」を参照してください。

Lambda

AWS ワークロード認証情報プロバイダーを Lambda 拡張機能としてパッケージ化できます。次に、Lambda 関数にレイヤーとして追加し、Lambda 関数から AWS ワークロード認証情報プロバイダーを呼び出してシークレットを取得できます。

次の手順は、aws-workload-credentials-provider GitHub リポジトリ`secrets-manager-provider-extension.sh`のサンプルスクリプトを使用して AWS 、ワークロード認証情報プロバイダーを Lambda 拡張機能としてインストールすることで、MyTest という名前のシークレットを取得する方法を示しています。

AWS ワークロード認証情報プロバイダーの Lambda 拡張機能を作成するには

1. プロバイダーレイヤーをパッケージ化する

AWS ワークロード認証情報プロバイダーコードパッケージのルートから、次のコマンドを実行します。

```
AWS_ACCOUNT_ID=AWS_ACCOUNT_ID
LAMBDA_ARN=LAMBDA_ARN

# Build the release binary
cargo build --release --target=x86_64-unknown-linux-gnu

# Copy the release binary into the `bin` folder
mkdir -p ./bin
cp ./target/x86_64-unknown-linux-gnu/release/aws-workload-credentials-
provider ./bin/aws-workload-credentials-provider

# Copy the `secrets-manager-provider-extension.sh` example script into the
 `extensions` folder.
mkdir -p ./extensions
cp aws_secretsmanager_provider/examples/example-lambda-extension/secrets-
manager-provider-extension.sh ./extensions

# Zip the extension shell script and the binary
zip secrets-manager-provider-extension.zip bin/* extensions/*

# Publish the layer version
LAYER_VERSION_ARN=$(aws lambda publish-layer-version \
    --layer-name secrets-manager-provider-extension \
    --zip-file "fileb://secrets-manager-provider-extension.zip" | jq -r
 '.LayerVersionArn')
```

2. SSRF トークンを設定する

   プロバイダーのデフォルト設定では、SSRF トークンを事前設定変数AWS_SESSION_TOKENまたはAWS_CONTAINER_AUTHORIZATION_TOKEN環境変数 (SnapStart が有効になっている Lambda 関数の後者変数) で設定された値に自動的に設定します。または、Lambda 関数の任意の値を使用して AWS_TOKEN 環境変数を定義することもできます。これは、この変数が他の 2 つよりも優先されるためです。AWS_TOKEN 環境変数を使用する場合は、その環境変数を lambda:UpdateFunctionConfiguration 呼び出しで設定する必要があります。

3. レイヤーを関数にアタッチする

   レイヤーバージョンを Lambda 関数にアタッチします:

```
# Attach the layer version to the Lambda function
aws lambda update-function-configuration \
    --function-name $LAMBDA_ARN \
    --layers "$LAYER_VERSION_ARN"
```

4. 関数コードの更新

   X-Aws-codes-Secrets-Token ヘッダー値を上記の環境変数から取得した SSRF トークンの値に設定して http://localhost:2773/secretsmanager/get?secretId=MyTest のクエリを実行するように Lambda 関数を更新し、シークレットを取得します。Lambda 拡張機能の初期化と登録の遅延に対応するため、アプリケーションコードに再試行ロジックを実装してください。

5. 関数をテストする

   Lambda 関数を呼び出して、シークレットが正しく取得されていることを確認します。

AWS ワークロード認証情報プロバイダーを使用してシークレットを取得する

シークレットを取得するには、シークレット名または ARN をクエリパラメータとしてローカル AWS ワークロード認証情報プロバイダーエンドポイントを呼び出します。デフォルトでは、 AWS ワークロード認証情報プロバイダーはシークレットAWSCURRENTのバージョンを取得します。別のバージョンを取得するには、versionStage または versionId パラメータを使用します。

> **Important**
>
> AWS ワークロード認証情報プロバイダーを保護するには、各リクエストの一部として SSRF トークンヘッダーを含める必要があります: X-Aws-Parameters-Secrets-Token。 AWS ワークロード認証情報プロバイダーは、このヘッダーを持たないリクエスト、または無効な SSRF トークンを持つリクエストを拒否します。the section called “設定オプション” で SSRF ヘッダー名をカスタマイズできます。

必要なアクセス許可

AWS ワークロード認証情報プロバイダーは、AWS 認証情報プロバイダーチェーンを使用する AWS SDK for Rust を使用します。これらの IAM 認証情報の ID によって、 AWS ワークロード認証情報プロバイダーがシークレットを取得するためのアクセス許可が決まります。

- `secretsmanager:DescribeSecret`
- `secretsmanager:GetSecretValue`

権限の詳細については、「the section called “アクセス許可に関するリファレンス”」を参照してください。

> **Important**
>
> シークレット値が AWS ワークロード認証情報プロバイダーにプルされると、コンピューティング環境と SSRF トークンにアクセスできるすべてのユーザーは、 AWS ワークロード認証情報プロバイダーキャッシュからシークレットにアクセスできます。詳細については、「the section called “セキュリティに関する考慮事項”」を参照してください。

リクエストの例

curl

Example例 – curl を使用してシークレットを取得する

次の curl の例は、 AWS ワークロード認証情報プロバイダーからシークレットを取得する方法を示しています。この例では、SSRF がインストールスクリプトによって保存されるファイルに存在するかどうかに依存します。

```
curl -v -H \
    "X-Aws-Parameters-Secrets-Token: $(</var/run/awssmatoken)" \
    'http://localhost:2773/secretsmanager/get?secretId=YOUR_SECRET_ID'
```

Python

Example例 – Python を使用してシークレットを取得する

次の Python の例は、 AWS ワークロード認証情報プロバイダーからシークレットを取得する方法を示しています。この例では、SSRF がインストールスクリプトによって保存されるファイルに存在するかどうかに依存します。

```
import requests
import json

# Function that fetches the secret from AWS Workload Credentials Provider for the
 provided secret id.
def get_secret():
    # Construct the URL for the GET request
    url = f"http://localhost:2773/secretsmanager/get?secretId=YOUR_SECRET_ID"

    # Get the SSRF token from the token file
    with open('/var/run/awssmatoken') as fp:
        token = fp.read()

    headers = {
        "X-Aws-Parameters-Secrets-Token": token.strip()
    }

    try:
        # Send the GET request with headers
        response = requests.get(url, headers=headers)

        # Check if the request was successful
        if response.status_code == 200:
            # Return the secret value
            return response.text
        else:
            # Handle error cases
            raise Exception(f"Status code {response.status_code} - {response.text}")

    except Exception as e:
        # Handle network errors
        raise Exception(f"Error: {e}")
```

refreshNow パラメータについて

AWS ワークロード認証情報プロバイダーは、インメモリキャッシュを使用してシークレット値を保存し、定期的に更新します。デフォルトでは、この更新は、有効期限 (TTL) が切れた後、通常 300 秒ごとにシークレットをリクエストしたときに発生します。ただし、このアプローチでは、特にキャッシュエントリの有効期限が切れる前にシークレットがローテーションした場合、シークレット値が古くなることがあります。

この制限に対応するため、 AWS ワークロード認証情報プロバイダーは URL refreshNowで というパラメータをサポートしています。このパラメータを使用して、シークレットの値の即時更新を強制し、キャッシュをバイパスして最新の情報を取得できます。

デフォルトの動作 (**refreshNow** なし)

- TTL の有効期限が切れるまでキャッシュされた値を使用します
- TTL の後にのみシークレットを更新します (デフォルトは 300 秒)
- キャッシュの有効期限が切れる前にシークレットがローテーションすると、古い値を返す可能性があります

refreshNow=true での動作

- キャッシュを完全にバイパスします
- Secrets Manager から直接、最新のシークレット値を取得します
- キャッシュを新しい値で更新し、TTL をリセットします
- 常に最新のシークレット値を取得するようにします

シークレット値を強制更新する

> **Important**
>
> refreshNow の初期値は false です。に設定するとtrue、 AWS ワークロード認証情報プロバイダー設定ファイルで指定された TTL が上書きされ、Secrets Manager に API コールが実行されます。

curl

Example例 – curl を使用してシークレットを強制更新する

次の curl の例は、 AWS ワークロード認証情報プロバイダーにシークレットの更新を強制する方法を示しています。この例では、SSRF がインストールスクリプトによって保存されるファイルに存在するかどうかに依存します。

```
curl -v -H \
"X-Aws-Parameters-Secrets-Token: $(</var/run/awssmatoken)" \
'http://localhost:2773/secretsmanager/get?secretId=YOUR_SECRET_ID&refreshNow=true'
```

Python

Example例 – Python を使用してシークレットを強制更新する

次の Python の例は、 AWS ワークロード認証情報プロバイダーからシークレットを取得する方法を示しています。この例では、SSRF がインストールスクリプトによって保存されるファイルに存在するかどうかに依存します。

```
import requests
import json

# Function that fetches the secret from AWS Workload Credentials Provider for the
 provided secret id.
def get_secret():
    # Construct the URL for the GET request
    url = f"http://localhost:2773/secretsmanager/get?
secretId=YOUR_SECRET_ID&refreshNow=true"

    # Get the SSRF token from the token file
    with open('/var/run/awssmatoken') as fp:
        token = fp.read()

    headers = {
        "X-Aws-Parameters-Secrets-Token": token.strip()
    }

    try:
        # Send the GET request with headers
        response = requests.get(url, headers=headers)
```

```
        # Check if the request was successful
        if response.status_code == 200:
            # Return the secret value
            return response.text
        else:
            # Handle error cases
            raise Exception(f"Status code {response.status_code} - {response.text}")

    except Exception as e:
        # Handle network errors
        raise Exception(f"Error: {e}")
```

ロール連鎖を使用してアカウント間でシークレットを取得する

ロールの連鎖により、 AWS ワークロード認証情報プロバイダーは を使用して IAM ロールを引き受けることで、他の AWS アカウントからシークレットを取得できます AWS STS `AssumeRole`。AWS ワークロード認証情報プロバイダーは、一意のロール ARN ごとに個別のキャッシュクライアントを作成してキャッシュします。各ロールクライアントは独自の独立したキャッシュを保持するため、異なるロールでフェッチされた同じシークレットには個別のキャッシュエントリがあります。

必要なアクセス許可

ロールチェーンを使用するには、以下が必要です。

- AWS ワークロード認証情報プロバイダーの環境認証情報には、ターゲットロール ARN に対するアクセス`sts:AssumeRole`許可が必要です。
- ターゲットロールには、アクセスするシークレットの `secretsmanager:GetSecretValue`および アクセス`secretsmanager:DescribeSecret`許可が必要です。
- ターゲットロールの信頼ポリシーは、 AWS ワークロード認証情報プロバイダーの ID がそれを引き受けることを許可する必要があります。

クロスアカウントシークレットを取得する

AWS ワークロード認証情報プロバイダーへのリクエストに `roleArn`クエリパラメータを含めて、シークレットの取得のために引き受けるロールを指定します。

curl

Example例 – curl を使用したクロスアカウントシークレット

```
curl -v -H \
    "X-Aws-Parameters-Secrets-Token: $(</var/run/awssmatoken)" \
    'http://localhost:2773/secretsmanager/get?
secretId=YOUR_SECRET_ID&roleArn=arn:aws:iam::ACCOUNT_ID:role/ROLE_NAME'
```

Python

Example例 – Python を使用したクロスアカウントシークレット

```
import requests

def get_secret_cross_account():
    secret_id = "YOUR_SECRET_ID"
    role_arn = "arn:aws:iam::ACCOUNT_ID:role/ROLE_NAME"
    url = f"http://localhost:2773/secretsmanager/get?
secretId={secret_id}&roleArn={role_arn}"

    with open('/var/run/awssmatoken') as fp:
        token = fp.read()

    headers = {
        "X-Aws-Parameters-Secrets-Token": token.strip()
    }

    try:
        response = requests.get(url, headers=headers)

        if response.status_code == 200:
            return response.text
        else:
            raise Exception(f"Status code {response.status_code} - {response.text}")

    except Exception as e:
        raise Exception(f"Error: {e}")
```

ロールの連鎖設定と制限

TOML 設定ファイルの `max_roles`オプションを使用してロールの連鎖を設定します。これにより、同時引き受けロールの最大数が 1～20 の範囲で設定されます。デフォルトは 20 です。

> **Important**
>
> 引き受けたロールは AWS 、ワークロード認証情報プロバイダーのロールキャッシュから削除されません。ロールの最大数に達すると、 AWS ワークロード認証情報プロバイダーが再起動されるまで、新しいロール ARNs を持つリクエストは`400`エラーで拒否されます。

ロール連鎖のエラーレスポンス

`400`

`roleArn` 形式が無効であるか、引き受けたロールの最大数に達しています。

`403`

AWS STS `AssumeRole` の呼び出しに失敗しました。ターゲットロールの信頼ポリシーで AWS 、ワークロード認証情報プロバイダーの ID がそれを引き受けることが許可されていることを確認します。

起動時のシークレットのプリフェッチ

デフォルトでは、 AWS ワークロード認証情報プロバイダーは、アプリケーションがシークレットをリクエストしたときに、オンデマンドでシークレットを取得します。プリフェッチでは、 AWS ワークロード認証情報プロバイダーは起動時に指定されたシークレットをキャッシュにロードするため、アプリケーションは最初の API コールを待たずにすぐにアクセスできます。プリフェッチはバックグラウンドタスクとして実行されます。 AWS ワークロード認証情報プロバイダーはリクエストの受け入れをすぐに開始し、プリフェッチの完了時にブロックしません。

シークレットを指定すると、次の 2 つの方法でプリフェッチできます。

- 明示的なシークレット – 特定のシークレット IDsまたは ARNs。
- タグベースの検出 – タグキーでシークレットを検出します。 AWS ワークロード認証情報プロバイダーは、指定されたタグを持つすべてのシークレットを取得します。

必要なアクセス許可

シークレットを取得するための標準のアクセス許可に加えて、プリフェッチには以下が必要です。

- `secretsmanager:BatchGetSecretValue` – すべてのプリフェッチオペレーションに必要です。
- `secretsmanager:ListSecrets` – タグベースの検出を使用する場合にのみ必要です。

プリフェッチを設定する

TOML 設定ファイルに`[capabilities.secrets_manager.prefetch]`セクションを追加します。以下のオプションが利用できます。

`cache_buffer_ratio`

プリフェッチ中にクライアントごとに埋めるキャッシュの最大割合。範囲は 0.1 ~ 1.0 です。デフォルトは 0.8 です。バッファ制限に達すると、 AWS ワークロード認証情報プロバイダーは残りのシークレットのプリフェッチを停止します。既存のキャッシュエントリは削除されません。プリフェッチ中にロードされなかったシークレットは、引き続きオンデマンドで使用できます。

`max_jitter_seconds`

プリフェッチが開始されるまでの秒単位のランダムな遅延。範囲は 0 ~ 10 です。デフォルトは 0 です。これを使用して、複数のプロバイダーが同時に起動したときにフリート全体の API コールが同期されるのを防ぎます。

Example明示的なシークレットを使用したプリフェッチ設定

```
[capabilities.secrets_manager.prefetch]
cache_buffer_ratio = 0.6
max_jitter_seconds = 5
secrets = [
    { secret_id = "arn:aws:secretsmanager:us-west-2:123456789012:secret:MySecret-
AbCdEf" },
    { secret_id = "MyOtherSecret" },
]
```

Exampleタグベースの検出を使用したプリフェッチ設定

```
[capabilities.secrets_manager.prefetch]
```

```
cache_buffer_ratio = 0.8
filter_tags = [
    { key = "Environment" },
    { key = "Team" },
]
```

同じ設定で明示的なシークレットとタグベースの検出を組み合わせることもできます。クロスアカウントプリフェッチの場合は、 role_arnフィールドを追加します。詳細については、「the section called “ロール連鎖を使用してアカウント間でシークレットを取得する”」を参照してください。

Exampleクロスアカウントアクセスによるプリフェッチ設定

```
[capabilities.secrets_manager.prefetch]
cache_buffer_ratio = 0.6
max_jitter_seconds = 5
secrets = [
    { secret_id = "arn:aws:secretsmanager:us-west-2:123456789012:secret:MySecret-
AbCdEf" },
    { secret_id = "cross-account-secret", role_arn = "arn:aws:iam::987654321098:role/
SecretAccessRole" },
]
filter_tags = [
    { key = "Environment" },
    { key = "Team", role_arn = "arn:aws:iam::987654321098:role/SecretAccessRole" },
]
```

AWS ワークロード認証情報プロバイダーを設定する

AWS ワークロード認証情報プロバイダーの設定を変更するには、TOML 設定ファイルを作成し、 を呼び出します./aws-workload-credentials-provider sm start --config config.toml。

設定ファイルは、ネストされた形式をサポートしています。Secrets Manager オプションは の下に配置され[capabilities.secrets_manager]、キャッシュとセキュリティ設定のサブセクションがあります。ログ記録オプションは に配置されます[logging]。

Exampleネストされた設定ファイルの例

```
[logging]
log_level = "INFO"
log_to_file = true
```

```
[capabilities.secrets_manager]
enabled = true
http_port = 2773
region = "us-east-1"
path_prefix = "/v1/"
max_conn = 800
max_roles = 20

[capabilities.secrets_manager.cache]
ttl_seconds = 300
cache_size = 1000

[capabilities.secrets_manager.security]
ssrf_headers = ["X-Aws-Parameters-Secrets-Token", "X-Vault-Token"]
ssrf_env_variables = ["AWS_TOKEN", "AWS_SESSION_TOKEN",
 "AWS_CONTAINER_AUTHORIZATION_TOKEN"]
```

> **Note**
>
> ルートレベルのフラットキー (などhttp_port = 2773) は、既存の設定ファイルとの下位互換性のために引き続きサポートされています。

Secrets Manager の設定オプション

`enabled`

Secrets Manager 機能がアクティブかどうか: `true`または `false`。デフォルトは `true` です。

`http_port`

ローカル HTTP サーバーのポート範囲は 1024 ~ 65535 です。デフォルトは 2773 です。

`region`

リクエストに使用する AWS リージョン。リージョンが指定されていない場合、 AWS ワークロード認証情報プロバイダーは SDK からリージョンを決定します。詳細については、「AWS SDK for Rust Developer Guide」の「Specify your credentials and default Region」を参照してください。

`path_prefix`

リクエストがパスベースのリクエストかどうかを判断するために使用される URI プレフィックス。デフォルトは「/v1/」です。

max_conn

AWS ワークロード認証情報プロバイダーが許可する HTTP クライアントからの接続の最大数。範囲は 1～1000 です。デフォルトは 800 です。

max_roles

クロスアカウントアクセスの同時 IAM ロールの最大数。範囲は 1～20 です。デフォルトは 20 です。詳細については、「the section called “ロール連鎖を使用してアカウント間でシークレットを取得する”」を参照してください。

キャッシュオプション (**[capabilities.secrets_manager.cache]**)

ttl_seconds

キャッシュされた項目の TTL (秒単位) の範囲は 0 ～ 3600 です。デフォルトは 300 です。0 はキャッシュがないことを示します。

cache_size

キャッシュに保存できるシークレットの最大数。範囲は 1～1000 です。デフォルトは 1000 です。

セキュリティオプション (**[capabilities.secrets_manager.security]**)

ssrf_headers

AWS ワークロード認証情報プロバイダーが SSRF トークンをチェックするヘッダー名のリスト。デフォルトは「X-Aws-Parameters-Secrets-Token, X-Vault-Token」です。

ssrf_env_variables

AWS ワークロード認証情報プロバイダーが SSRF トークンの順番にチェックする環境変数名のリスト。環境変数には、`AWS_TOKEN=file:///var/run/awssmatoken` のようにトークンまたはトークンファイルへの参照を含めることができます。デフォルトは、AWS_TOKEN、AWS_SESSION_TOKEN、AWS_CONTAINER_AUTHORIZATION_TOKEN です。

ログ記録オプション (**[logging]**)

log_level

AWS ワークロード認証情報プロバイダーのログで報告される詳細レベル: DEBUG、INFO、WARN、ERROR、NONE。デフォルトは INFO です。

log_to_file

ファイルまたは stdout/stderr にログを記録するかどうか: `true` または `false`。デフォルトは `true` です。

オプション機能

AWS ワークロード認証情報プロバイダーは、 `--features`フラグを に渡すことで、オプションの機能を使用して構築できます`cargo build`。利用できる機能は次のとおりです。

構築機能

prefer-post-quantum

`X25519MLKEM768` を最も優先度の高いキー交換アルゴリズムにします。それ以外にした場合も利用可能ですが、最高の優先度にはなりません。`X25519MLKEM768` は、ハイブリッドのポスト量子セキュアキー交換アルゴリズムです。

fips

プロバイダーが使用する暗号スイートを FIPS 承認の暗号のみに制限します。

ログ記録

ローカルなログ記録

AWS ワークロード認証情報プロバイダーは、設定変数に応じて、エラーをローカルで ファイル`logs/secrets_manager_provider.log`または stdout/stderr `log_to_file` に記録します。アプリケーションが AWS ワークロード認証情報プロバイダーを呼び出してシークレットを取得すると、それらの呼び出しはローカルログに表示されます。CloudTrail ログには表示されません。

ログローテーション

AWS ワークロード認証情報プロバイダーは、ファイルが 10 MB に達すると新しいログファイルを作成し、合計で最大 5 つのログファイルを保存します。

AWS サービスログ記録

ログは Secrets Manager、CloudTrail、または CloudWatch には送信されません。 AWS ワークロード認証情報プロバイダーからシークレットを取得するリクエストは、それらのログに表示されません。 AWS ワークロード認証情報プロバイダーが Secrets Manager を呼び出してシークレットを取得すると、その呼び出しは を含むユーザーエージェント文字列を使用して CloudTrail に記録されます`aws-workload-credentials-provider`。

the section called “設定オプション” でログ記録オプションを設定できます。

セキュリティに関する考慮事項

信頼ドメイン

ローカルプロバイダーアーキテクチャの場合、信頼ドメインはプロバイダーエンドポイントと SSRF トークンにアクセスできる場所であり、通常はホスト全体です。 AWS ワークロード認証情報プロバイダーの信頼ドメインは、同じセキュリティ体制を維持するために、Secrets Manager 認証情報が利用可能なドメインと一致する必要があります。例えば、Amazon EC2 では、 AWS ワークロード認証情報プロバイダーの信頼ドメインは、Amazon EC2 のロールを使用する場合の認証情報のドメインと同じになります。

> **Important**
>
> Secrets Manager 認証情報がアプリケーションにロックされたプロバイダーベースのソリューションをまだ使用していないセキュリティ意識の高いアプリケーションでは、言語固有の AWS SDKs またはキャッシュソリューションの使用を検討する必要があります。シークレットの詳細については、「シークレットの取得」を参照してください。

AI コーディングエージェントでシークレットを安全に使用する

AI コーディングエージェントにシェルまたはAWS API アクセスがある場合、コンテキストウィンドウでプレーンテキストシークレットを呼び出し`get-secret-value`て受信できます。これにより、

シークレット値が会話履歴、ログ、またはダウンストリームツール呼び出しに漏れる可能性があるという複数のリスクが生じます。

これを防ぐには、 エージェントツールキットAWSのシークレット安全スキルを使用します。このスキルは、実行時に解決された動的参照を使用するように AI エージェントに教えるため、エージェントはプレーンテキストの値を見ずにシークレットの使用を調整します。

> **Important**
>
> これはベストエフォートの防御であり、セキュリティの境界ではありません。これにより、最も一般的な漏洩経路は防止されますが、すべての回避ベクトルを停止することはできません。を IAM 最小特権、CloudTrail モニタリング、VPC エンドポイントポリシーと組み合わせます。

仕組み

シークレット安全スキルは、次の 2 つの保護レイヤーを提供します。

1. スキルガイダンス – 実行時に参照を解決するラッパースクリプトasm-execである{{resolve:secretsmanager:...}}動的参照を使用するようにエージェントにトレーニングします。プレーンテキスト値は子プロセスにのみ存在し、エージェントのコンテキストウィンドウに入ることはありません。
2. 構造適用 (フック) – PreToolUseフックは、SDK AWS CLI、MCP ツールを呼び出しget-secret-valueたりbatch-get-secret-value、経由で呼び出したり、AWSワークロード認証情報プロバイダーデーモンに直接アクセスしたりしようとする試みを自動的にブロックします。手動設定は必要ありません。

前提条件

- Claude Code や OpenAI Codex などのプラグインをサポートする AI コーディングエージェント。
- エージェントツールキット for AWS aws-coreプラグインがインストールされました。
- 次のいずれかのシークレット解決バックエンド:
  - で実行されているAWSワークロード認証情報プロバイダーlocalhost:2773。「the section called “AWS ワークロード認証情報プロバイダー”」を参照してください。
  - AWS MCP エンドポイントへのリクエストに署名できる AWS認証情報。

- IAM アクセス許可: 解決するシークレット`secretsmanager:GetSecretValue`。

プラグインをインストールする

エージェントプラットフォーム用の `aws-core`プラグインをインストールします。シークレット安全スキルとフックは自動的にアクティブ化されます。

Claude コードの場合:

```
claude plugin add ./plugins/aws-core
```

OpenAI Codex の場合:

```
codex plugin add ./plugins/aws-core
```

サポートされている他のプラットフォームについては、「エージェントツールキット for AWS README」を参照してください。

{{resolve:...}} 構文

エージェントがシークレットをコマンドに渡す必要がある場合、 を呼び出す代わりに動的参照を使用します`get-secret-value`。

```
{{resolve:secretsmanager:<secret-id>:<field-type>:<json-key>:<version-stage>}}
```

コンポーネント	[Required] (必須)	デフォルト	[Description] (説明)
`secret-id`	はい	–	シークレット名または完全な ARN
`field-type`	いいえ	`SecretString`	`SecretString` を指定してください
`json-key`	いいえ	(完全な値)	JSON シークレット値から抽出するキー
`version-stage`	いいえ	`AWSCURRENT`	バージョンステージラベル

asm-exec を使用してシークレットでコマンドを実行する

asm-exec は、コマンド引数の{{resolve:...}}参照を解決し、ターゲットコマンドを実行するラッパースクリプトです。シークレット値は子プロセスにのみ存在します。

```
asm-exec -- <command> [arguments with {{resolve:...}} references]
```

asm-exec は、利用可能な最初のバックエンドを通じて参照を解決します。

1. でのAWSワークロード認証情報プロバイダー localhost:2773 – ローカルにキャッシュされます。
2. AWS MCP エンドポイント – 使用可能なAWS認証情報を使用した SigV4-signedリクエスト。

Example PostgreSQL データベースに接続する

```
asm-exec -- psql \
  "host=mydb.example.com \
  user={{resolve:secretsmanager:prod/db-creds:SecretString:username}} \
  password={{resolve:secretsmanager:prod/db-creds:SecretString:password}}" \
  -c "SELECT * FROM users LIMIT 10"
```

Exampleベアラートークンを使用して API コールを行う

```
asm-exec -- curl -H "Authorization: Bearer {{resolve:secretsmanager:prod/api-token}}" \
  https://api.example.com/data
```

Example複数のシークレットを使用して MySQL に接続する

```
asm-exec -- mysql \
  -h {{resolve:secretsmanager:prod/mysql:SecretString:host}} \
  -u {{resolve:secretsmanager:prod/mysql:SecretString:username}} \
  -p{{resolve:secretsmanager:prod/mysql:SecretString:password}} \
  -e "SHOW TABLES"
```

Exampleシークレットを環境変数として Docker コンテナに渡す

```
asm-exec -- docker run \
  -e "DB_PASSWORD={{resolve:secretsmanager:prod/db:SecretString:password}}" \
```

```
  myapp:latest
```

クロスリージョンシークレット

デフォルトのリージョンとは異なるリージョンに保存されているシークレットの場合は、完全な ARN (リージョンを含む) を使用するか、AWS_REGION環境変数を設定します。

```
# Using full ARN (region is extracted automatically)
asm-exec -- curl -H "X-Api-Key: {{resolve:secretsmanager:arn:aws:secretsmanager:eu-
west-1:123456789012:secret:prod/key-a1b2c3}}" \
  https://eu.api.example.com/data

# Using AWS_REGION
export AWS_REGION=eu-west-1
asm-exec -- curl -H "X-Api-Key: {{resolve:secretsmanager:prod/key}}" \
  https://eu.api.example.com/data
```

セキュリティに関する考慮事項

- サブプロセス分離 – ターゲットコマンドは を介して実行されますsubprocess.run。シークレット値は、asm-execプロセスメモリと子プロセス引数にのみ存在します。

フックがシークレットへの直接アクセスをブロックする方法

aws-core プラグインを有効にすると、PreToolUseフックは実行前に を傍受します。これは以下をブロックします。

- aws secretsmanager get-secret-value および CLI batch-get-secret-value経由
- get_secret_value スクリプトでの SDK 呼び出しbatch_get_secret_valueによる および
- AWSワークロード認証情報プロバイダーデーモンパスへの直接アクセス (localhost:2773/secretsmanager/get)
- GetSecretValue MCP ツールまたは構造化AWS API コールによる オペレーション

通話がブロックされると、エージェントは拒否メッセージを受信し、asm-exec代わりに で{{resolve:...}}参照を使用するように指示します。

トラブルシューティング

「シークレットが見つかりません」エラー

シークレットが存在し、IAM ロールに アクセスsecretsmanager:GetSecretValue許可があることを確認します。シークレット名では大文字と小文字が区別されます。

AWSワークロード認証情報プロバイダーの接続が拒否されました

AWSワークロード認証情報プロバイダーが実行されていない可能性があります。これは致命的ではありません。SigV4-signed付き MCP エンドポイントにasm-execフォールスルーされます。バックエンドが認証できるように、AWS認証情報が使用可能であることを確認します。

「失敗」エラー

両方のバックエンドに到達できませんでした。AWSワークロード認証情報プロバイダーが実行されているか、AWS認証情報が有効 (`aws sts get-caller-identity`) であること、シークレットのリージョンが正しいこと、および ID がシークレットsecretsmanager:GetSecretValueにあることを確認します。

解決により空の文字列が生成されます

JSON キーがシークレット値に存在しない可能性があります。AWSコンソールでシークレット構造を確認するか、シークレット所有者に使用可能なキーの確認を依頼します。

フックは通話をブロックしません

エージェントセッションの開始時にフックロードします。セッション中にプラグインをインストールした場合は、フックを有効にするエージェントセッションを再起動します。

C++ AWS SDK を使用して Secrets Manager シークレット値を取得する

C++ アプリケーションの場合は、GetSecretValue または BatchGetSecretValue を使用して SDK を直接呼び出します。

次の例で、Secrets Manager シークレットの値を取得する方法を示します。

必要な許可:`secretsmanager:GetSecretValue`

```
//! Retrieve an AWS Secrets Manager encrypted secret.
```

```
/*!
  \param secretID: The ID for the secret.
  \return bool: Function succeeded.
 */
bool AwsDoc::SecretsManager::getSecretValue(const Aws::String &secretID,
                                            const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::SecretsManager::SecretsManagerClient
 secretsManagerClient(clientConfiguration);

    Aws::SecretsManager::Model::GetSecretValueRequest request;
    request.SetSecretId(secretID);

    Aws::SecretsManager::Model::GetSecretValueOutcome getSecretValueOutcome =
 secretsManagerClient.GetSecretValue(
            request);
    if (getSecretValueOutcome.IsSuccess()) {
        std::cout << "Secret is: "
                  << getSecretValueOutcome.GetResult().GetSecretString() << std::endl;
    }
    else {
        std::cerr << "Failed with Error: " << getSecretValueOutcome.GetError()
                  << std::endl;
    }

    return getSecretValueOutcome.IsSuccess();
}
```

JavaScript AWS SDK を使用して Secrets Manager シークレット値を取得する

JavaScript アプリケーションの場合は、getSecretValue または batchGetSecretValue を使用して SDK を直接呼び出します。

次の例で、Secrets Manager シークレットの値を取得する方法を示します。

必要な許可:secretsmanager:GetSecretValue

```
import {
  GetSecretValueCommand,
  SecretsManagerClient,
} from "@aws-sdk/client-secrets-manager";
```

```
export const getSecretValue = async (secretName = "SECRET_NAME") => {
  const client = new SecretsManagerClient();
  const response = await client.send(
    new GetSecretValueCommand({
      SecretId: secretName,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '584eb612-f8b0-48c9-855e-6d246461b604',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   ARN: 'arn:aws:secretsmanager:us-east-1:xxxxxxxxxxxx:secret:binary-
secret-3873048-xxxxxx',
  //   CreatedDate: 2023-08-08T19:29:51.294Z,
  //   Name: 'binary-secret-3873048',
  //   SecretBinary: Uint8Array(11) [
  //      98, 105, 110, 97, 114,
  //     121,  32, 100, 97, 116,
  //      97
  //   ],
  //   VersionId: '712083f4-0d26-415e-8044-16735142cd6a',
  //   VersionStages: [ 'AWSCURRENT' ]
  // }

  if (response.SecretString) {
    return response.SecretString;
  }

  if (response.SecretBinary) {
    return response.SecretBinary;
  }
};
```

Kotlin AWS SDK を使用して Secrets Manager のシークレット値を取得する

Kotlin アプリケーションの場合は、GetSecretValue または BatchGetSecretValue を使用して SDK を直接呼び出します。

次の例で、Secrets Manager シークレットの値を取得する方法を示します。

必要な許可:`secretsmanager:GetSecretValue`

```
suspend fun getValue(secretName: String?) {
    val valueRequest =
        GetSecretValueRequest {
            secretId = secretName
        }

    SecretsManagerClient.fromEnvironment { region = "us-east-1" }.use { secretsClient -
>
        val response = secretsClient.getSecretValue(valueRequest)
        val secret = response.secretString
        println("The secret value is $secret")
    }
}
```

PHP AWS SDK を使用して Secrets Manager シークレット値を取得する

PHP アプリケーションの場合は、`GetSecretValue` または `BatchGetSecretValue` を使用して SDK を直接呼び出します。

次の例で、Secrets Manager シークレットの値を取得する方法を示します。

必要な許可:`secretsmanager:GetSecretValue`

```
<?php

  /**
    * Use this code snippet in your app.
    *
    * If you need more information about configurations or implementing the sample
 code, visit the AWS docs:
```

```
    * https://aws.amazon.com/developer/language/php/
    */

  require 'vendor/autoload.php';

  use Aws\SecretsManager\SecretsManagerClient;
  use Aws\Exception\AwsException;

  /**
    * This code expects that you have AWS credentials set up per:
    * https://<<{{DocsDomain}}>>/sdk-for-php/v3/developer-guide/guide_credentials.html
    */

  // Create a Secrets Manager Client
  $client = new SecretsManagerClient([
      'profile' => 'default',
      'version' => '2017-10-17',
      'region' => '<<{{MyRegionName}}>>',
  ]);

  $secret_name = '<<{{MySecretName}}>>';

  try {
      $result = $client->getSecretValue([
          'SecretId' => $secret_name,
      ]);
  } catch (AwsException $e) {
      // For a list of exceptions thrown, see
      // https://<<{{DocsDomain}}>>/secretsmanager/latest/apireference/
API_GetSecretValue.html
      throw $e;
  }

  // Decrypts secret using the associated KMS key.
  $secret = $result['SecretString'];

  // Your code goes here
```

Ruby AWS SDK を使用して Secrets Manager のシークレット値を取得する

Ruby アプリケーションの場合は、get_secret_value または batch_get_secret_value を使用して SDK を直接呼び出します。

次の例で、Secrets Manager シークレットの値を取得する方法を示します。

必要な許可:secretsmanager:GetSecretValue

```
  # Use this code snippet in your app.
  # If you need more information about configurations or implementing the sample code,
 visit the AWS docs:
  # https://aws.amazon.com/developer/language/ruby/

  require 'aws-sdk-secretsmanager'

  def get_secret
    client = Aws::SecretsManager::Client.new(region: '<<{{MyRegionName}}>>')

    begin
      get_secret_value_response = client.get_secret_value(secret_id:
 '<<{{MySecretName}}>>')
    rescue StandardError => e
      # For a list of exceptions thrown, see
      # https://<<{{DocsDomain}}>>/secretsmanager/latest/apireference/
API_GetSecretValue.html
      raise e
    end

    secret = get_secret_value_response.secret_string
    # Your code goes here.
  end
```

を使用してシークレット値を取得する AWS CLI

必要な許可:secretsmanager:GetSecretValue

Exampleシークレットの暗号化されたシークレット値を取得する

次の get-secret-value の例では、現在のシークレット値が取得されます。

```
aws secretsmanager get-secret-value \
    --secret-id MyTestSecret
```

Example前のシークレット値を取得する

次の get-secret-value の例では、前のシークレット値が取得されます。

```
aws secretsmanager get-secret-value \
        --secret-id MyTestSecret
        --version-stage AWSPREVIOUS
```

を使用してバッチ内のシークレットのグループを取得する AWS CLI

必要な許可:

- secretsmanager:BatchGetSecretValue
- 取得するシークレットごとに secretsmanager:GetSecretValue のアクセス許可が必要です。
- フィルターを使用する場合は、secretsmanager:ListSecrets も必要です。

アクセス許可ポリシーの例については、「the section called “例: バッチでシークレット値のグループを取得するためのアクセス許可”」を参照してください。

> **Important**
>
> 取得しようとしているグループ内の個別シークレットを取得するためのアクセス許可を拒否する VPCE ポリシーを設定している場合、BatchGetSecretValue はシークレット値を返さず、エラーが返されます。

Example名前順に表示されたシークレットグループのシークレット値を取得する

次の batch-get-secret-value の例では、3 つのシークレットのシークレット値を取得します。

```
aws secretsmanager batch-get-secret-value \
         --secret-id-list MySecret1 MySecret2 MySecret3
```

Exampleフィルターで選択したシークレットグループのシークレット値を取得する

次の batch-get-secret-value の例では、「Test」という名前のタグが付いたシークレットのシークレット値を取得します。

```
aws secretsmanager batch-get-secret-value \
          --filters Key="tag-key",Values="Test"
```

AWS コンソールを使用してシークレット値を取得する

シークレットを取得するには (コンソール)

1. Secrets Manager のコンソール (https://console.aws.amazon.com/secretsmanager/) を開きます。
2. シークレットのリストで、取得するシークレットを選択します。
3. [シークレット値] セクションで、[シークレット値の取得] を選択します。

   Secrets Manager にシークレットの現在のバージョン (AWSCURRENT) が表示されます。シークレットの他のバージョン (AWSPREVIOUS やカスタムラベル付きバージョンなど) を表示するには、the section called “AWS CLI” を使用します。

で AWS Secrets Manager シークレットを使用する AWS Batch

AWS Batch は、 でバッチコンピューティングワークロードを実行するのに役立ちます AWS クラウド。を使用すると AWS Batch、機密データを AWS Secrets Manager シークレットに保存し、ジョブ定義で参照することで、機密データをジョブに挿入できます。詳細については、「Secrets Manager を使用した機密データの指定」を参照してください。

CloudFormation リソースで AWS Secrets Manager シークレットを取得する

を使用すると CloudFormation、別の CloudFormation リソースで使用するシークレットを取得できます。一般的なシナリオでは、まず シークレットマネージャーが生成したパスワードを使ってシークレットを作成し、次に、シークレットから、新しいデータベースの認証情報として使用するユーザーネームとパスワードを取得します。でシークレットを作成する方法については CloudFormation、「」を参照してくださいCloudFormation。

CloudFormation テンプレートでシークレットを取得するには、動的リファレンスを使用します。スタックを作成すると、動的リファレンスはシークレット値を CloudFormation リソースにプルするため、シークレット情報をハードコードする必要はありません。代わりに、シークレットを名前または ARN で参照します。ダイナミックリファレンスは、任意のリソースプロパティ内のシークレットに対し使用できます。リソースメタデータ (`AWS::CloudFormation::Init` など) 内にあるシークレットに対しては、ダイナミックリファレンスを使用することはできません。これにより、シークレット値がコンソールに表示されてしまうためです。

シークレットに対するダイナミックリファレンスのパターンを、次に示します。

```
{{resolve:secretsmanager:secret-id:SecretString:json-key:version-stage:version-id}}
```

シークレットID

シークレット名またはシークレット ARN。 AWS アカウントのシークレットにアクセスするには、シークレット名を使用できます。別の AWS アカウントのシークレットにアクセスするには、シークレットの ARN を使用します。

json-key (オプション)

値を取得するペアのキー名を指定します。を指定しない場合`json-key`、 はシークレットテキスト全体 CloudFormation を取得します。このセグメントにはコロン文字 (`:`) を含めることはできません。

version-stage (オプション)

使用するクエリのバージョン シークレットマネージャーは、ステージングラベルがローテーション処理中にさまざまなバージョンを追跡するために使用されます。`version-stage` を使用する場合は、`version-id` を指定することはできません。`version-stage` または `version-id`、を指定しない場合、デフォルトでは `AWSCURRENT` というラベルの付いたバージョンが取得されます。このセグメントにはコロン文字 (`:`) を含めることはできません。

version-id (オプション)

使用したいシークレットのバージョンの固有識別子を指定します。`version-id` を指定した場合は、`version-stage` を指定しないでください。`version-stage` または `version-id` を指定しない場合、次にデフォルトでは `AWSCURRENT` というバージョンが取得されます。このセグメントにはコロン文字 (`:`) を含めることはできません。

詳細については、「シークレットマネージャーのシークレットを指定するための、ダイナミックレファレンスを使用する」を参照してください。

> **Note**
>
> 最終値(\)としてバックスラッシュを使用して動的参照を作成しないでください。CloudFormation はこれらの参照を解決できないため、リソースに障害が発生します。

GitHub ジョブで AWS Secrets Manager シークレットを使用する

GitHub ジョブでシークレットを使用するには、GitHub アクションを使用して からシークレットを取得し AWS Secrets Manager 、GitHub ワークフローでマスクされた環境変数として追加できます。GitHub アクションの詳細については、GitHub ドキュメントの「Understanding GitHub Actions」(GitHub アクションについて) を参照してください。

GitHub 環境にシークレットを追加すると、GitHub ジョブの他のすべてのステップでもそのシークレットが使用できるようになります。環境内のシークレットが悪用されるのを防ぐには、「Security hardening for GitHub Actions」(GitHub アクションのセキュリティ強化) のガイダンスに従ってください。

シークレット値の文字列全体を環境変数値として設定できます。また、文字列が JSON の場合は、JSON を解析して JSON キーと値のペアごとに環境変数を設定することもできます。シークレット値がバイナリの場合は文字列に変換されます。

シークレットから作成された環境変数を表示するには、デバッグログを有効にします。詳細については、GitHub Docs の「デバッグログの有効化」を参照してください。

シークレットから作成された環境変数を使用するには、GitHub Docs の「環境変数」を参照してください。

前提条件

このアクションを使用するには、まず AWS 認証情報を設定し、 `configure-aws-credentials` ステップを使用して GitHub AWS リージョン 環境で を設定する必要があります。GitHub OIDC プロバイダを使用して直接ロールを引き受けるには、「GitHub アクションの AWS 認証情報アクションの設定」の指示に従ってください。これにより、有効期間の短い認証情報を使用でき、追加のアクセスキーを Secrets Manager の外部に保存する必要がなくなります。

アクションが引き受ける IAM ロールには、以下のアクセス許可が必要です。

- 取得するシークレットに対する `GetSecretValue`

- すべてのシークレットに対する ListSecrets
- (オプション) シークレットが で暗号化 KMS key されている場合は、 Decryptの カスタマー管理キー。

詳細については、「the section called “認証とアクセスコントロール”」を参照してください。

使用方法

アクションを使用するには、次の構文を使用するステップをワークフローに追加します。

```
- name: Step name
  uses: aws-actions/aws-secretsmanager-get-secrets@v2
  with:
    secret-ids: |
      secretId1
      ENV_VAR_NAME, secretId2
    name-transformation: (Optional) uppercase|lowercase|none
    parse-json-secrets: (Optional) true|false
```

パラメータ

`secret-ids`

シークレット ARN、名前、名前プレフィックス。

環境変数名を設定するには、まず環境変数名を入力し、その後にシークレット ID とカンマを順に入力します。例えば、ENV_VAR_1, secretId はシークレット secretId から ENV_VAR_1 という名前の環境変数を作成します。環境変数名には、大文字の英字、数字、およびアンダースコアを使用できます。

プレフィックスを使用するには、3 文字以上を入力し、その後にアスタリスクを付けます。例えば、dev* は名前が dev で始まるすべてのシークレットに一致します。取得できる一致シークレットの最大数は 100 です。変数名を設定し、プレフィックスが複数のシークレットと一致する場合、アクションは失敗します。

`name-transformation`

デフォルトでは、このステップはシークレット名から各環境変数名を作成し、大文字の英字、数字、アンダースコアのみが含まれるように変換し、数字で始まらないようにします。名前の文字には、lowercase で小文字を使用するか、none で大文字と小文字を変更しないようにステップを設定できます。デフォルト値は uppercase です。

`parse-json-secrets`

(オプション) デフォルトでは、環境変数値がシークレット値内の JSON 文字列全体に設定されます。`parse-json-secrets` を `true` に設定すると、JSON 内のキーと値のペアごとに環境変数が作成されます。

「name」や「Name」など JSON で使用されるキーが大文字と小文字を区別する場合、アクション名が重複して競合することに注意してください。この場合は、`parse-json-secrets` を `false` に設定し、JSON シークレット値を別途解析します。

環境変数の命名

アクションによって作成された環境変数の名前は、その変数の元となるシークレットと同じになります。環境変数にはシークレットよりも厳しい命名要件があるため、そうした要件を満たすようにシークレット名が変換されます。例えば、小文字は大文字に変換されます。シークレットの JSON を解析する場合、環境変数名にはシークレット名と JSON キー名の両方が含まれます。(例えば `MYSECRET_KEYNAME`)。小文字を変換しないようにアクションを設定できます。

2 つの環境変数が同じ名前で終わると、アクションは失敗します。そのため、環境変数に使用する名前をエイリアスとして指定する必要があります。

名前が競合する可能性がある場合:

- 例えば、「MySecret」という名前のシークレットと「mysecret」という名前のシークレットはどちらも「MYSECRET」という名前の環境変数になります。
- 「Secret_keyname」という名前のシークレットと、「keyname」という名前のキーを持つ「Secret」という名前の JSON 解析されたシークレットは、どちらも「SECRET_KEYNAME」という名前の環境変数になります。

次の例に示すように、`ENV_VAR_NAME` という名前の変数を作成するエイリアスを指定することで、環境変数名を設定できます。

```
secret-ids: |
  ENV_VAR_NAME, secretId2
```

空のエイリアス

- parse-json-secrets: true を設定して、空のエイリアスを入力し、次にカンマ、シークレット ID の順に入力すると、アクションは環境変数に解析された JSON キーと同じ名前を付けます。変数名にはシークレット名は含まれません。

  シークレットに有効な JSON が含まれていない場合、アクションは 1 つの環境変数を作成し、シークレット名と同じ名前を付けます。
- parse-json-secrets: false を設定して、空のエイリアスを入力し、次にカンマとシークレット ID の順に入力すると、アクションは環境変数にエイリアスを指定しなかった場合と同じように名前を付けます。

次の例は、空のエイリアスを示しています。

```
,secret2
```

例

Example 1 名前と ARN でシークレットを取得する

次の例では、名前と ARN で識別されるシークレットの環境変数を作成しています。

```
- name: Get secrets by name and by ARN
  uses: aws-actions/aws-secretsmanager-get-secrets@v2
  with:
    secret-ids: |
      exampleSecretName
      arn:aws:secretsmanager:us-east-2:123456789012:secret:test1-a1b2c3
      0/test/secret
      /prod/example/secret
      SECRET_ALIAS_1,test/secret
      SECRET_ALIAS_2,arn:aws:secretsmanager:us-east-2:123456789012:secret:test2-a1b2c3
      ,secret2
```

作成された環境変数

```
EXAMPLESECRETNAME: secretValue1
TEST1: secretValue2
_0_TEST_SECRET: secretValue3
```

```
_PROD_EXAMPLE_SECRET: secretValue4
SECRET_ALIAS_1: secretValue5
SECRET_ALIAS_2: secretValue6
SECRET2: secretValue7
```

Example 2 プレフィックスで始まるシークレットをすべて取得する

次の例では、名前が *beta* で始まるすべてのシークレットの環境変数を作成しています。

```
- name: Get Secret Names by Prefix
  uses: 2
  with:
    secret-ids: |
      beta*    # Retrieves all secrets that start with 'beta'
```

作成された環境変数

```
BETASECRETNAME: secretValue1
BETATEST: secretValue2
BETA_NEWSECRET: secretValue3
```

Example 3 シークレット内の JSON を解析する

次の例では、シークレット内の JSON を解析して環境変数を作成しています。

```
- name: Get Secrets by Name and by ARN
  uses: aws-actions/aws-secretsmanager-get-secrets@v2
  with:
    secret-ids: |
      test/secret
      ,secret2
    parse-json-secrets: true
```

シークレット `test/secret` には、次のシークレット値があります。

```
{
  "api_user": "user",
  "api_key": "key",
  "config": {
    "active": "true"
```

```
  }
}
```

シークレット secret2 には、次のシークレット値があります。

```
{
  "myusername": "alejandro_rosalez",
  "mypassword": "EXAMPLE_PASSWORD"
}
```

作成された環境変数

```
TEST_SECRET_API_USER: "user"
TEST_SECRET_API_KEY: "key"
TEST_SECRET_CONFIG_ACTIVE: "true"
MYUSERNAME: "alejandro_rosalez"
MYPASSWORD: "EXAMPLE_PASSWORD"
```

Example 4 環境変数名に小文字を使用する

次の例では、小文字の名前で環境変数を作成します。

```
- name: Get secrets
  uses: aws-actions/aws-secretsmanager-get-secrets@v2
  with:
    secret-ids: exampleSecretName
    name-transformation: lowercase
```

作成された環境変数:

```
examplesecretname: secretValue
```

GitLab AWS Secrets Manager で を使用する

AWS Secrets Manager は GitLab と統合されています。Secrets Manager シークレットを活用して GitLab 認証情報を保護し、GitLab でハードコードされなくなるようにします。代わりに、アプリケーションが GitLab CI/CD パイプラインでジョブを実行すると、GitLab Runner は Secrets Manager からこれらのシークレットを取得します。

この統合を使用するには、IAM と IAM ロールに OpenID Connect (OIDC) ID プロバイダー AWS Identity and Access Management を作成します。これにより、GitLab Runner は Secrets Manager シークレットにアクセスできるようになります。GitLab CI/CD と OIDC の詳細については、「GitLab ドキュメント」を参照してください。

考慮事項

非公開の GitLab インスタンスを使用している場合、この Secrets Manager 統合を使用することはできません。代わりに、非パブリックインスタンスに関する GitLab のドキュメントを参照してください。

前提条件

Secrets Manager を GitLab と統合するには、次の前提条件を完了します。

1. AWS Secrets Manager シークレットを作成する

   GitLab ジョブで取得され、これらの認証情報をハードコードする必要がなくなる Secrets Manager シークレットが必要です。GitLab パイプラインを設定するときは、Secrets Manager シークレット ID が必要です。詳細については「AWS Secrets Manager シークレットを作成する」を参照してください。
2. IAM コンソールで GitLab を OIDC プロバイダーにします。

   このステップでは、IAM コンソールで GitLab を OIDC プロバイダーにします。詳細については、「OpenID Connect (OIDC) ID プロバイダーの作成」および「GitLab ドキュメント」を参照してください。

   IAM コンソールで OIDC プロバイダーを作成するときは、次の設定を使用します。

   a. `provider URL` を GitLab インスタンスに設定します。例えば、**`gitlab.example.com`** です。

   b. `audience` または `aud` を **`sts.amazonaws.com`** に設定します。
3. IAM ポリシーとロールを作成する

   IAM ロールとポリシーを作成する必要があります。このロールは AWS Security Token Service (STS) を使用する GitLab によって引き受けられます。詳細については、「カスタム信頼ポリシーを使用してロールを作成する」を参照してください。

a. IAM コンソールで、IAM ロールを作成するときに次の設定を使用します。

- `Trusted entity type` を **`Web identity`** に設定します。
- `Group` を **`your GitLab group`** に設定します。
- ステップ 2 で使用したものと同じプロバイダー URL (GitLab インスタンス) に `Identity provider` を設定します。
- ステップ 2 で使用したものと同じ対象者に `Audience` を設定します。

b. 次に示すのは、GitLab がロールを引き受けることを許可する信頼ポリシーの例です。信頼ポリシーには AWS アカウント、、GitLab URL、およびプロジェクトパスがリストされている必要があります。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Principal": {
        "Federated": "arn:aws:iam::111122223333:oidc-
provider/gitlab.example.com"
      },
      "Condition": {
        "StringEquals": {
          "gitlab.example.com:aud": [
            "sts.amazon.com"
          ]
        },
        "StringLike": {
          "gitlab.example.com:sub": [
            "project_path:mygroup/project-*:ref_type:branch-*:ref:main*"
          ]
        }
      }
    }
  ]
}
```

c. また、GitLab に AWS Secrets Managerへのアクセスを許可する IAM ポリシーを作成する必要があります。このポリシーを信頼ポリシーに追加できます。詳細については、「IAM ポリシーを作成する」を参照してください。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "secretsmanager:GetSecretValue",
      "Resource": "arn:aws:secretsmanager:us-
east-1:111122223333:secret:your-secret"
    }
  ]
}
```

GitLab AWS Secrets Manager との統合

前提条件を完了したら、Secrets Manager を使用して認証情報を保護するように GitLab を設定できます。

Secrets Manager を使用するように GitLab パイプラインを設定する

GitLab CI/CD 設定ファイルを以下の情報で更新する必要があります。

- STS に設定されたトークンの対象者。
- Secrets Manager のシークレット ID。
- GitLab パイプラインでジョブを実行するときに GitLab Runner が引き受ける IAM ロール。
- シークレットが保存され AWS リージョン ている 。

GitLab は Secrets Manager からシークレットを取得し、その値を一時ファイルに保存します。このファイルへのパスは、ファイルタイプ CI/CD 変数と同様に CI/CD 変数に保存されます。

以下は、GitLab CI/CD 設定ファイルの YAML ファイルのスニペットです。

```
variables:
  AWS_REGION: us-east-1
```

```
  AWS_ROLE_ARN: 'arn:aws:iam::111122223333:role/gitlab-role'
job:
  id_tokens:
    AWS_ID_TOKEN:
      aud: 'sts.amazonaws.com'
  secrets:
    DATABASE_PASSWORD:
      aws_secrets_manager:
        secret_id: "arn:aws:secretsmanager:us-east-1:111122223333:secret:secret-name"
```

詳細については、「GitLab Secrets Manager 統合のドキュメント」を参照してください。

必要に応じて、GitLab で OIDC 設定をテストできます。詳細については、「OIDC 設定のテストに関する GitLab のドキュメント」を参照してください。

トラブルシューティング

以下は、Secrets Manager を GitLab と統合するときに発生する可能性がある一般的な問題のトラブルシューティングに役立ちます。

GitLab Pipeline の問題

GitLab パイプラインの問題が発生した場合は、以下を確認してください。

- YAML ファイルが適切にフォーマットされている。詳細については、「GitLab のドキュメント」を参照してください。
- GitLab パイプラインは正しいロールを引き受けており、適切なアクセス許可と正しい AWS Secrets Manager シークレットへのアクセスを持っています。

その他のリソース

以下のリソースは、GitLab と AWS Secrets Managerに関する問題のトラブルシューティングに役立ちます。

- GitLab OIDC のトラブルシューティング
- GitLab CI/CD パイプラインのデバッグ
- トラブルシューティング

で AWS Secrets Manager シークレットを使用する AWS IoT Greengrass

AWS IoT Greengrass は、クラウド機能をローカルデバイスに拡張するソフトウェアです。これにより、デバイスは情報源に近いデータを収集および分析して、ローカルイベントに自動的に反応し、ローカルネットワークで互いに安全に通信することができます。

AWS IoT Greengrass では、パスワード、トークン、またはその他のシークレットをハードコーディングすることなく、 AWS IoT Greengrass デバイスからサービスやアプリケーションで認証できます。 AWS Secrets Manager を使用して、シークレットをクラウドに安全に保存および管理できます。 は Secrets Manager を AWS IoT Greengrass コアデバイスに AWS IoT Greengrass 拡張するため、コネクタと Lambda 関数はローカルシークレットを使用してサービスやアプリケーションとやり取りできます。

シークレットを AWS IoT Greengrass グループに統合するには、Secrets Manager シークレットを参照するグループリソースを作成します。このシークレットリソースは、関連付けられた ARN を使用してクラウドシークレットを参照します。シークレットリソースを作成、管理、使用する方法については、 AWS IoT デベロッパーガイドの「シークレットリソースの使用」を参照してください。

シークレットを AWS IoT Greengrass Core にデプロイするには、「シークレットを Core にデプロイする AWS IoT Greengrass 」を参照してください。

パラメータストアで AWS Secrets Manager シークレットを使用する

AWS Systems Manager パラメータストアは、設定データ管理とシークレット管理のための安全な階層型ストレージを提供します。パスワード、データベース文字列、およびライセンスコードなどのデータをパラメータ値として保存することができます。ただし、パラメータストアは保存されるシークレットの自動ローテーションサービスを提供していません。代わりに、パラメータストアを使用して、Secrets Manager にシークレットを保存することができ、パラメータストアのパラメータとしてそのシークレットを参照できます。

Secrets Manager を使用してパラメータストアを設定する場合、`secret-id` パラメータストアでは、名前文字列の前にスラッシュ (/) が必要です。

詳細については、「 AWS Systems Manager ユーザーガイド」の「パラメータストアパラメータからの AWS Secrets Manager シークレットの参照」を参照してください。

AWS Secrets Manager シークレットのローテーション

ローテーションとは、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、シークレットの自動ローテーションを設定できます。ローテーションには次の 2 つの形式があります。

- マネージドローテーション – ほとんどのマネージドシークレットでは、マネージドローテーションを使用します。このローテーションでは、サービスによってローテーションが構成および管理されます。マネージドローテーションは Lambda 関数を使用しません。
- Secrets Manager が管理する外部シークレットのローテーション – Secrets Manager パートナーが保持するシークレットの場合、マネージド外部シークレットローテーションを使用してパートナーのシステムのシークレットを更新します。これには Lambda 関数は必要ありません。
- the section called “Lambda 関数によるローテーション” – その他のタイプのシークレットの場合、Secrets Manager ローテーションは Lambda 関数を使用して、シークレットとデータベースまたはサービスを更新します。

AWS Secrets Manager シークレットのマネージドローテーション

一部のサービスは、ユーザーに代わってローテーションの設定と管理を行うマネージドローテーションを提供しています。マネージドローテーションでは、 AWS Lambda 関数を使用してデータベース内のシークレットと認証情報を更新しません。

次のサービスは、マネージドローテーションを提供しています。

- Amazon Aurora は、マスターユーザー認証情報のマネージドローテーションを提供します。詳細については、「Amazon Aurora ユーザーガイド」の「Amazon Aurora および AWS Secrets Managerでのパスワード管理」を参照してください。
- Amazon ECS Service Connect は、 AWS Private Certificate Authority TLS 証明書のマネージドローテーションを提供しています。詳細については、「Amazon Elastic Container Service Developer Guide」の「TLS with Service Connect」を参照してください。
- Amazon RDS は、マスターユーザー認証情報のマネージドローテーションを提供します。詳細については、「Amazon RDS ユーザーガイド」の「Amazon RDS および AWS Secrets Manager でのパスワード管理」を参照してください。

- Amazon DocumentDB は、マスターユーザー認証情報のマネージドローテーションを提供します。詳細については、Amazon DocumentDB ユーザーガイド」の「Amazon DocumentDB でのパスワード管理」および AWS Secrets Manager「」を参照してください。 Amazon DocumentDB
- Amazon Redshift は、管理者パスワードのマネージドローテーションを提供します。詳細については、「Amazon Redshift 管理ガイド」の「AWS Secrets Managerを使用して Amazon Redshift 管理者パスワードを管理する」を参照してください。
- マネージド外部シークレットは、Secrets Manager パートナーが保持するシークレットのマネージドローテーションを提供します。詳細については、「AWS Secrets Manager マネージド外部シークレットを使用したサードパーティーシークレットの管理」を参照してください。

> **Tip**
>
> 他のすべてのタイプのシークレットについては、「the section called “Lambda 関数によるローテーション”」を参照してください。

マネージドシークレットのローテーションは、通常 1 分以内に完了します。ローテーション中、シークレットを取得する新しい接続では、以前のバージョンの認証情報が取得される場合があります。アプリケーションでは、マスターユーザーを使用する代わりに、アプリケーションに必要な最小の特権で作成されたデータベースユーザーを使用するというベストプラクティスに従うことを強くお勧めします。アプリケーションユーザーの場合、可用性を最大限に高めるには、交代ユーザーローテーション戦略を使用できます。

Secrets Manager パートナーが保持するシークレットの場合、

マネージドローテーションのスケジュールを変更するには

1. Secrets Manager コンソールでマネージドシークレットを開きます。管理サービスからのリンクをたどるか、Secrets Manager コンソールでシークレットを検索できます。
2. [Rotation schedule] (ローテーションスケジュール) において、UTC タイムゾーンで [Schedule expression builder] (スケジュール式ビルダー) にスケジュールを入力するか、[Schedule expression] (スケジュール式) としてスケジュールを入力します。Secrets Manager は、スケジュールを `rate( )` 式または `cron( )` 式として保存します。[Start time] (開始時刻) を指定しない限り、ローテーションウィンドウは午前 0 時に自動的に開始されます。シークレットが 4 時間ごとにローテーションされるように設定できます。詳細については、「ローテーションスケジュール」を参照してください。

3. (オプション) [Window duration] (ウィンドウ期間) では、Secrets Manager がシークレットをローテーションするウィンドウの長さを選択します (3 時間のウィンドウの場合は **3h** など)。ウィンドウが次のローテーションウィンドウに重ならないようにしてください。時間単位のローテーションスケジュールでは、ウィンドウ期間を指定しない場合、ウィンドウは 1 時間後に自動的に終了します。日数単位のローテーションスケジュールの場合、ウィンドウは 1 日の終わりに自動的に終了します。
4. [保存] を選択します。

マネージドローテーションのスケジュールを変更するには (AWS CLI)

- `rotate-secret` を呼び出します。次の例では、月の 1 日と 15 日の 16 時から 18 時 (UTC) の間にシークレットがローテーションされます。詳細については、「ローテーションスケジュール」を参照してください。

```
aws secretsmanager rotate-secret \
    --secret-id MySecret \
    --rotation-rules \
        "{\"ScheduleExpression\": \"cron(0 16 1,15 * ? *)\", \"Duration\": \"2h\"}"
```

Secrets Manager が管理する外部シークレットのローテーション

Secrets Manager は、一部のソフトウェアベンダーと提携して、マネージド外部シークレットを提供しています。この機能は、ローテーションを自動的に処理することで、お客様がシークレットのライフサイクルを管理するのに役立ちます。マネージド外部シークレットを使用すると、お客様は異なるパートナーに保存されているシークレットごとに特定のローテーションロジックを維持する必要がなくなります。これは Secrets Manager によって処理されます。

Secrets Manager でオンボーディングされているパートナーのリストを表示するには、「マネージド外部シークレットパートナー」を参照してください。

コンソールでローテーションを設定する

各統合パートナーによって指定されたシークレットタイプと値を指定して作成された既存のマネージド外部シークレットのローテーションを設定するには、次の手順を実行します。

1. Secrets Manager コンソールを開きます。
2. リストからマネージド外部シークレットを選択します。

3. [設定] タブを選択します。
4. ローテーション設定セクションで、ローテーションの編集を選択します。
5. [Automatic rotation] (自動ローテーション) を有効化します。
6. ローテーションメタデータで、ローテーションに必要なパートナー固有のメタデータを追加します。

   他の必要なメタデータについては、統合パートナーが提供するガイドラインに従ってください。
7. シークレットローテーションのサービスアクセス許可で、ローテーション用の IAM ロールを選択または作成します。
   - 新しいロールの作成を選択して、必要なアクセス許可を持つロールを自動的に作成します
   - または、パートナーに適切なアクセス許可を持つ既存のロールを選択する

   デフォルトでは、アクセス許可はシークレットが作成されたリージョン内の個々のパートナーに限定されます。
8. ローテーションスケジュールを設定します (30 日ごとに自動的にローテーションするなど) 。
9. 保存を選択してローテーション設定を適用します。

このプロセス中に設定される 2 つの主要なメタデータフィールドは次のとおりです。

フィールド	説明
ExternalSecretRotationMetadata	Salesforce の API バージョンなど、ローテーションに必要なパートナー固有のメタデータ
ExternalSecretRotationRoleArn	ローテーションに使用される IAM ロールの ARN と、統合パートナーにスコープされたアクセス許可

これらのフィールドの詳細については、「Secrets Manager マネージド外部シークレットを使用してサードパーティーシークレットを管理する」を参照してください。

CLI を使用してローテーションを設定する

次のコマンドを実行して、Salesforce シークレットのローテーションを設定します。このコマンドは、シークレット ID、ローテーション用の IAM ロール ARN、ローテーションスケジュール、ローテーションプロセスに必要なパートナー固有のメタデータを指定します。

```
aws secretsmanager rotate-secret \
            --secret-id SampleSecret \
            --external-secret-rotation-role-arn arn:aws:iam::123412341234:role/xyz \
            --rotation-rules AutomaticallyAfterDays=1 \
            --external-secret-rotation-metadata
 '[{"Key":"apiVersion","Value":"v65.0"}]'
```

Lambda 関数によるローテーション

多くのタイプのシークレットの場合、Secrets Manager は AWS Lambda 関数を使用してシークレットとデータベースまたはサービスを更新します。Lambda 関数を使用する場合のコストについては、「料金」を参照してください。

他のサービスによって管理されるシークレット の場合、マネージドローテーションを使用します。マネージドローテーション を使用するには、最初に管理サービスを通じてシークレットを作成します。

ローテーション中、Secrets Manager はローテーションの状態を示すイベントをログに記録します。詳細については、「the section called “でログ記録する AWS CloudTrail ”」を参照してください。

シークレットをローテーションするために、Secrets Manager は、設定したスケジュールに従って Lambda 関数を呼び出します。自動ローテーションの設定中にシークレット値も手動で更新した場合、Secrets Manager は次のローテーション日を計算するときにそれを有効なローテーションと見なします。

ローテーション中、Secrets Manager は毎回異なるパラメータを使用して、同じ関数を複数回呼び出します。Secrets Manager は、次の JSON リクエスト構造のパラメータを使用して 関数を呼び出します。

```
{
    "Step" : "request.type",
    "SecretId" : "string",
    "ClientRequestToken" : "string",
    "RotationToken" : "string"
}
```

パラメータ :

- ステップ – ローテーションステップ: `create_secret`、`set_secret`、`test_secret`、または `finish_secret`。詳細については、「the section called “ローテーション関数の 4 つのステップ”」を参照してください。
- SecretId – ローテーションするシークレットの ARN。
- ClientRequestToken – シークレットの新しいバージョンの一意の ID。この値は、べき等性を確保するのに役立ちます。詳細については、「AWS Secrets Manager API Reference」の「PutSecretValue: ClientRequestToken」を参照してください。
- RotationToken – リクエストのソースを示す一意の ID。引き受けたロールまたはクロスアカウントローテーションを使用したシークレットローテーションに必要です。このローテーションでは、あるアカウントのシークレットを、別のアカウントの Lambda ローテーション関数を使用してローテーションします。いずれの場合も、ローテーション関数は Secrets Manager を呼び出す IAM ロールを引き受け、Secrets Manager はローテーショントークンを使用して IAM ロール ID を検証します。

ローテーションの失敗は、Lambda 関数または Secrets Manager の問題が原因で発生する可能性があります。ローテーションステップが失敗した場合、Secrets Manager はシークレットの開いているローテーションウィンドウ中にローテーションプロセス全体を複数回再試行します。

トピック

- Amazon RDS、Amazon Aurora、Amazon DocumentDB、Amazon Redshift のシークレットで自動ローテーションを設定にする
- データベース以外の AWS Secrets Manager シークレットの自動ローテーションを設定する
- を使用して自動ローテーションを設定する AWS CLI
- Lambda 関数のローテーション戦略
- Lambda ローテーション関数
- AWS Secrets Manager ローテーション関数テンプレート
- の Lambda ローテーション関数実行ロールのアクセス許可 AWS Secrets Manager
- AWS Lambda ローテーション関数のネットワークアクセス
- AWS Secrets Managerローテーションのトラブルシューティング

Amazon RDS、Amazon Aurora、Amazon DocumentDB、Amazon Redshift のシークレットで自動ローテーションを設定にする

このチュートリアルでは、データベースシークレットに the section called “Lambda 関数によるローテーション” を設定する方法について説明します。ローテーションとは、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、シークレットとデータベースの両方で認証情報が更新されます。Secrets Manager では、データベースシークレットの自動ローテーションを設定できます。

コンソールを使用してローテーションを設定するには、まずローテーション戦略を選択する必要があります。次に、ローテーションのシークレットを設定します。これにより、もしまだ未作成の場合は Lambda ローテーション関数が作成されます。コンソールは Lambda 関数実行ロールのアクセス権限も設定します。最後のステップとして、Lambda ローテーション関数が Secrets Manager とデータベースの両方にネットワーク経由でアクセスできることを確認する必要があります。

> **Warning**
>
> 自動ローテーションを有効にするには、Lambda ローテーション関数用に IAM 実行ロールを作成し、そのロールにアクセス権限ポリシーをアタッチするアクセス許可が必要です。`iam:CreateRole` 許可と `iam:AttachRolePolicy` 許可の両方が必要です。これらのアクセス許可を付与すると、アイデンティティは自身に任意の許可を付与できるようになります。

手順:

- ステップ 1: ローテーション戦略を選択し、オプションでスーパーユーザーシークレットを作成する
- ステップ 2: ローテーションの設定とローテーション関数の作成
- ステップ 3: (オプション) ローテーション関数に追加のアクセス許可条件を設定する
- ステップ 4: ローテーション関数用のネットワークアクセスを設定する
- 次の手順

ステップ 1: ローテーション戦略を選択し、オプションでスーパーユーザーシークレットを作成する

Secrets Manager が提供する戦略の詳細については、「the section called “Lambda 関数のローテーション戦略”」を参照してください。

交代ユーザー戦略を選択する場合は、シークレットを作成する を行い、データベースのスーパーユーザーの認証情報をそこに保存する必要があります。ローテーションでは最初のユーザーのクローンが作成されますが、ほとんどのユーザーにはその権限がないため、スーパーユーザーの認証情報を含むシークレットが必要です。Amazon RDS Proxy は、ユーザー交代戦略をサポートしていないことに注意してください。

ステップ 2: ローテーションの設定とローテーション関数の作成

Amazon RDS、Amazon DocumentDB、または Amazon Redshift のシークレットでローテーションを有効にするには

1. Secrets Manager コンソール (https://console.aws.amazon.com/secretsmanager/) を開きます。
2. [Secrets] (シークレット) ページで、自分のシークレットを選択します。
3. シークレットの詳細ページで、[Rotation configuration] (ローテーション設定) セクションの [Edit rotation] (ローテーションの編集) を選択します。
4. [Edit rotation configuration] (ローテーション設定の編集) ダイアログボックスで、次の操作を行います。

    a. [Automatic rotation] (自動ローテーション) を有効化します。

    b. [Rotation schedule] (ローテーションスケジュール) において、UTC タイムゾーンで [Schedule expression builder] (スケジュール式ビルダー) にスケジュールを入力するか、[Schedule expression] (スケジュール式) としてスケジュールを入力します。Secrets Manager は、スケジュールを `rate( )` 式または `cron( )` 式として保存します。[Start time] (開始時刻) を指定しない限り、ローテーションウィンドウは午前 0 時に自動的に開始されます。シークレットが 4 時間ごとにローテーションされるように設定できます。詳細については、「ローテーションスケジュール」を参照してください。

    c. (オプション) [Window duration] (ウィンドウ期間) では、Secrets Manager がシークレットをローテーションするウィンドウの長さを選択します (3 時間のウィンドウの場合は **3h** など)。ウィンドウが次のローテーションウィンドウに重ならないようにしてください。時間単位のローテーションスケジュールでは、ウィンドウ期間を指定しない場合、ウィンドウは

1 時間後に自動的に終了します。日数単位のローテーションスケジュールの場合、ウィンドウは 1 日の終わりに自動的に終了します。

d. (オプション) 変更を保存したときにシークレットをローテーションするには、[Rotate immediately when the secret is stored] (シークレットが保存されたときにすぐにローテーションする) を選択します。チェックボックスをオフにすると、最初のローテーションは設定したスケジュールから開始されます。

   手順 3 と 4 がまだ完了していないなどの理由でローテーションが失敗した場合、Secrets Manager はローテーションプロセスを複数回再試行します。

e. [Rotation function] (ローテーション関数) で、次のいずれかを行います。

   - [Create a new Lambda function] (新しい Lambda 関数の作成) を選択し、新しい関数の名前を 1 つ入力します。Secrets Manager は、関数名の先頭に「`SecretsManager`」を追加します。Secrets Manager は適切なテンプレートに基づいて関数を作成し、Lambda 実行ロールに必要なアクセス権限を設定します。
   - [Use an existing Lambda function] (既存の Lambda 関数を使用する) を選択して、別のシークレットに使用したローテーション関数を再利用します。推奨 VPC 設定に記載されているローテーション関数は、データベースと同じ VPC とセキュリティグループを持つため、関数がデータベースにアクセスしやすくなっています。

f. [ローテーション戦略] では、[単一ユーザー] または [代替ユーザー] 戦略を選択します。詳細については、「the section called “ステップ 1: ローテーション戦略を選択し、オプションでスーパーユーザーシークレットを作成する”」を参照してください。

5. [保存] を選択します。

ステップ 3: (オプション) ローテーション関数に追加のアクセス許可条件を設定する

ローテーション関数のリソースポリシーには、Lambda が混乱した代理プログラムとして使用されることを防ぐために、コンテキストキー「`aws:SourceAccount`」を含めることを推奨します。一部の AWS サービスでは、混乱した代理シナリオを避けるために、 AWS は `aws:SourceArn`および `aws:SourceAccount` グローバル条件キーの両方を使用することをお勧めします。ただし、ローテーション関数のポリシーに `aws:SourceArn` の条件を含めると、その ARN で指定されたシークレットだけをローテーションさせるためにローテーション関数を使用することができます。コンテキストキーのみを含めることをお勧めしますaws:SourceAccount複数のシークレットに対して回転関数を使用できるようにする。

ローテーション関数のリソースポリシーを更新するには

1. Secrets Manager のコンソールでシークレットを選択し、詳細ページの [Rotation configuration] (ローテーション設定) で、Lambda ローテーション関数を選択します。Lambda コンソールが開きます。
2. 「Lambda のリソースベースのポリシーを使用する」 の指示に従って `aws:sourceAccount` の条件を追加します。

```
"Condition": {
    "StringEquals": {
        "AWS:SourceAccount": "123456789012"
    }
},
```

AWS マネージドキー `aws/secretsmanager` 以外の KMS キーを使用しシークレットを暗号化する場合、Secrets Manager は Lambda の実行ロールに対し、そのキーの使用に関するアクセス許可を付与します。SecretArn 暗号化コンテキストを使用して復号化関数の使用を制限できます。この場合、ローテーション関数ロールには、ローテーションに使用するシークレットを復号化するアクセスのみが許可されます。

ローテーション関数の実行ロールを更新するには

1. Lambda ローテーション関数で [設定] を選択し、次に [実行ロール] で [ロール名] を選択します。
2. 「ロールのアクセス許可ポリシーの変更」の指示に従って、`kms:EncryptionContext:SecretARN` 条件を追加します。

```
"Condition": {
    "StringEquals": {
        "kms:EncryptionContext:SecretARN": "SecretARN"
    }
},
```

ステップ 4: ローテーション関数用のネットワークアクセスを設定する

詳細については、「the section called “AWS Lambda ローテーション関数のネットワークアクセス”」を参照してください。

次の手順

「the section called “ローテーションのトラブルシューティング”」を参照してください。

データベース以外の AWS Secrets Manager シークレットの自動ローテーションを設定する

このチュートリアルでは、非データベースシークレットに the section called “Lambda 関数によるローテーション” を設定する方法について説明します。ローテーションとは、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、そのシークレットの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。

データベースシークレットについては、「データベースシークレットの自動ローテーション (コンソール)」を参照してください。

> **Warning**
>
> 自動ローテーションを有効にするには、Lambda ローテーション関数用に IAM 実行ロールを作成し、そのロールにアクセス権限ポリシーをアタッチするアクセス許可が必要です。`iam:CreateRole` 許可と `iam:AttachRolePolicy` 許可の両方が必要です。これらのアクセス許可を付与すると、アイデンティティは自身に任意の許可を付与できるようになります。

手順:

- ステップ 1: 汎用ローテーション関数を作成する
- ステップ 2: ローテーション関数コードを記述する
- ステップ 3: シークレットのローテーションを設定する
- ステップ 4: ローテーション関数が Secrets Manager とデータベースまたはサービスにアクセスすることを許可する
- ステップ 5: Secrets Manager がローテーション関数を呼び出すことを許可する
- ステップ 6: ローテーション関数用のネットワークアクセスを設定する
- 次の手順

ステップ 1: 汎用ローテーション関数を作成する

最初に、Lambda ローテーション関数を作成します。シークレットをローテーションするためのコードが含まれないため、後のステップで記述します。ローテーション関数の機能の詳細については、「the section called “Lambda ローテーション関数”」を参照してください。

サポートされているリージョンでは、 AWS Serverless Application Repository を使用してテンプレートから関数を作成できます。サポートされているリージョンの一覧は、「AWS Serverless Application Repository FAQs」を参照してください。他のリージョンでは、関数を最初から作成し、テンプレートコードを関数にコピーします。

汎用ローテーション関数を作成するには

1. が リージョンで AWS Serverless Application Repository サポートされているかどうかを確認するには、 AWS 全般のリファレンスのAWS Serverless Application Repository 「エンドポイントとクォータ」を参照してください。
2. 次のいずれかを行います。
   - AWS Serverless Application Repository リージョンで がサポートされている場合:
     a. Lambda コンソールで、[アプリケーション]、[アプリケーション作成] の順に選択します。
     b. [アプリケーションの作成] ページで、[サーバーレスアプリケーション] タブを選択します。
     c. [パブリックアプリケーション] の検索ボックスに、「**SecretsManagerRotationTemplate**」と入力します。
     d. [カスタム IAM ロールまたはリソースポリシーを作成するアプリを表示する] を選択します。
     e. 「SecretsManagerRotationTemplate」タイルを選択します。
     f. [確認、設定、デプロイ] ページの [アプリケーション設定] タイルで、必須フィールドに入力します。
        - [エンドポイント] には、**https://** を含むリージョンのエンドポイントを入力します。; エンドポイントのリストについては、「the section called “Secrets Manager エンドポイント”」を参照してください。
        - Lambda 関数を VPC に配置するには、[vpcSecurityGroupIds] と [vpcSubnetIds] を含めます。
     g. [デプロイ] をクリックします。

- がリージョンでサポート AWS Serverless Application Repository されていない場合:

  a. Lambda コンソールから、[関数]、[関数の作成] の順に選択します。

  b. [関数の作成] ページで、次の操作を実行します。

     i. Author from scratch (製作者を最初から) を選択します。

     ii. [Function name] (関数名) には、関数の名前を入力します。

     iii. [ランタイム] には、[Python 3.12] を選択します。

     iv. [関数の作成] を選択してください。

ステップ 2: ローテーション関数コードを記述する

このステップでは、シークレットを更新するコードと、シークレットの対象となるサービスまたはデータベースを記述します。独自のローテーション関数を記述するためのヒントなど、ローテーション関数の動作については、「the section called “Lambda ローテーション関数”」を参照してください。次の ローテーション関数のテンプレート をリファレンスとして使用できます。

ステップ 3: シークレットのローテーションを設定する

このステップでは、シークレットのローテーションスケジュールを設定し、ローテーション関数をシークレットに接続します。

ローテーションを設定して空のローテーション関数を作成するには

1. Secrets Manager コンソール (https://console.aws.amazon.com/secretsmanager/) を開きます。
2. [Secrets] (シークレット) ページで、自分のシークレットを選択します。
3. シークレットの詳細ページで、[Rotation configuration] (ローテーション設定) セクションの [Edit rotation] (ローテーションの編集) を選択します。[Edit rotation configuration] (ローテーション設定の編集) ダイアログボックスで、次の操作を行います。

   a. [Automatic rotation] (自動ローテーション) を有効化します。

   b. [Rotation schedule] (ローテーションスケジュール) において、UTC タイムゾーンで [Schedule expression builder] (スケジュール式ビルダー) にスケジュールを入力するか、[Schedule expression] (スケジュール式) としてスケジュールを入力します。Secrets Manager は、スケジュールを `rate( )` 式または `cron( )` 式として保存します。[Start time] (開始時刻) を指定しない限り、ローテーションウィンドウは午前 0 時に自動的に開始され

ます。シークレットが 4 時間ごとにローテーションされるように設定できます。詳細については、「ローテーションスケジュール」を参照してください。

c. (オプション) [Window duration] (ウィンドウ期間) では、Secrets Manager がシークレットをローテーションするウィンドウの長さを選択します (3 時間のウィンドウの場合は **3h** など)。ウィンドウが次のローテーションウィンドウに重ならないようにしてください。時間単位のローテーションスケジュールでは、ウィンドウ期間を指定しない場合、ウィンドウは 1 時間後に自動的に終了します。日数単位のローテーションスケジュールの場合、ウィンドウは 1 日の終わりに自動的に終了します。

d. (オプション) 変更を保存したときにシークレットをローテーションするには、[Rotate immediately when the secret is stored] (シークレットが保存されたときにすぐにローテーションする) を選択します。チェックボックスをオフにすると、最初のローテーションは設定したスケジュールから開始されます。

e. [ローテーション関数] で、ステップ 1 で作成した [Lambda 関数] を選択します。

f. [保存] を選択します。

ステップ 4: ローテーション関数が Secrets Manager とデータベースまたはサービスにアクセスすることを許可する

Lambda ローテーション関数には、Secrets Manager のシークレットにアクセスする権限と、データベースまたはサービスにアクセスする権限が必要です。このステップでは、これらのアクセス権限を Lambda 実行ロールに付与します。 AWS マネージドキー `aws/secretsmanager` 以外の KMS キーを使用しシークレットを暗号化する場合は、そのキーの使用に関するアクセス許可を、Lambda の実行ロールに付与する必要があります。SecretArn 暗号化コンテキストを使用して復号化関数の使用を制限できます。この場合、ローテーション関数ロールには、ローテーションに使用するシークレットを復号化するアクセスのみが許可されます。IAM ポリシーの例については、「ローテーションへのアクセス許可」を参照してください。

手順については、「AWS Lambda 開発者ガイド」の「Lambda 実行ロール」を参照してください。

ステップ 5: Secrets Manager がローテーション関数を呼び出すことを許可する

Secrets Manager が設定したローテーションスケジュールで、ローテーション関数を呼び出すことができるようにするには、Lambda 関数のリソースポリシーで Secrets Manager サービスプリンシパルに `lambda:InvokeFunction` アクセス許可を付与する必要があります。

ローテーション関数のリソースポリシーには、Lambda が混乱した代理プログラムとして使用されることを防ぐために、コンテキストキー「`aws:SourceAccount`」を含めることを推奨します。ー

部の AWS サービスでは、混乱した代理シナリオを避けるために、 AWS は aws:SourceArnおよび aws:SourceAccount グローバル条件キーの両方を使用することをお勧めします。ただし、ローテーション関数のポリシーに aws:SourceArn の条件を含めると、その ARN で指定されたシークレットだけをローテーションさせるためにローテーション関数を使用することができます。コンテキストキーのみを含めることをお勧めしますaws:SourceAccount複数のシークレットに対して回転関数を使用できるようにする。

Lambda 関数にリソースポリシーをアタッチするには、「リソースベースのポリシーを使用する」を参照してください。

以下のポリシーによって、Secrets Manager は Lambda 関数を呼び出すことが可能になります。

JSON

```
{
    "Version":"2012-10-17",
    "Id": "default",
    "Statement": [
    {
        "Effect": "Allow",
        "Principal": {
            "Service": "secretsmanager.amazonaws.com"
            },
        "Action": "lambda:InvokeFunction",
        "Condition": {
            "StringEquals": {
                "AWS:SourceAccount": "123456789012"
            }
        },
        "Resource": "arn:aws:lambda:us-east-1:123456789012:function:function-
name"
    }
    ]
}
```

ステップ 6: ローテーション関数用のネットワークアクセスを設定する

このステップでは、ローテーション関数が Secrets Manager とそのシークレットの対象となるサービスまたはデータベースの両方に接続できるようにします。シークレットをローテーションする

ために、ローテーション関数が両方にアクセスできる必要があります。「the section called “AWS Lambda ローテーション関数のネットワークアクセス”」を参照してください。

次の手順

ステップ 3 でローテーションを設定する際に、シークレットをローテーションするスケジュールを設定します。スケジュール時にローテーションが失敗した場合、Secrets Manager はローテーションを複数回試行します。また、すぐにシークレットをローテーションする の手順に従って、直ちにローテーションを開始することもできます。

ローテーションが失敗した場合は、「ローテーションのトラブルシューティング」を参照してください。

を使用して自動ローテーションを設定する AWS CLI

このチュートリアルでは、 the section called “Lambda 関数によるローテーション”を使用して を設定する方法について説明します AWS CLI。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、そのシークレットの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。

また、コンソールを使用してローテーションを設定することもできます。データベースシークレットについては、「データベースシークレットの自動ローテーション (コンソール)」を参照してください。他のすべてのタイプのシークレットについては、「非データベースシークレットの自動ローテーション (コンソール)」を参照してください。

を使用してローテーションを設定するには AWS CLI、データベースシークレットをローテーションする場合は、まずローテーション戦略を選択する必要があります。交代ユーザー戦略を選択する場合は、データベースのスーパーユーザーの認証情報を含むシークレットを別途保存する必要があります。次に、ローテーション関数コードを記述します。Secrets Manager には、関数のベースとなるテンプレートが用意されています。コードを使用して Lambda 関数を作成し、Lambda 関数と Lambda 実行ロールの両方にアクセス許可を設定します。次のステップとして、Lambda 関数がネットワーク経由で Secrets Manager とデータベースまたはサービスの両方にアクセスできることを確認する必要があります。最後に、ローテーションのシークレットを設定します。

手順:

- データベースシークレットの前提条件: ローテーション戦略を選択する
- ステップ 1: ローテーション関数コードを記述する
- ステップ 2 : Lambda 関数を作成する
- ステップ 3: ネットワークアクセスを設定する
- ステップ 4: シークレットのローテーションを設定する

- 次の手順

データベースシークレットの前提条件: ローテーション戦略を選択する

Secrets Manager が提供する戦略の詳細については、「the section called “Lambda 関数のローテーション戦略”」を参照してください。

オプション 1: 単一ユーザー戦略

単一ユーザー戦略を選択した場合は、ステップ 1 に進むことができます。

オプション 2: ユーザー交代戦略

ユーザー交代戦略を選択した場合は、以下を行う必要があります。

- シークレットを作成し、データベースのスーパーユーザーの認証情報をその中に保存します。ユーザー交代ローテーションでは、最初のユーザーのクローンが作成されますが、ほとんどのユーザーにはその権限がないため、スーパーユーザーの認証情報を含むシークレットが必要です。
- スーパーユーザーシークレットの ARN を元のシークレットに追加します。詳細については、「the section called “シークレットの JSON 構造”」を参照してください。

Amazon RDS Proxy は、ユーザー交代戦略をサポートしていないことに注意してください。

ステップ 1: ローテーション関数コードを記述する

シークレットをローテーションするには、ローテーション関数が必要です。ローテーション関数は、Secrets Manager がシークレットをローテーションさせるために呼び出す Lambda 関数です。詳細については、「the section called “Lambda 関数によるローテーション”」を参照してください。このステップでは、シークレットを更新するコードと、シークレットの対象となるサービスまたはデータベースを記述します。

Secrets Manager は、ローテーション関数のテンプレート で Amazon RDS、Amazon Aurora、Amazon Redshift、Amazon DocumentDB データベースシークレットのテンプレートを提供します。

ローテーション関数コードを記述する

1. 次のいずれかを行います。
    - ローテーション関数テンプレートのリストを確認します。サービスとローテーション戦略に一致するものがある場合は、コードをコピーします。

- その他のタイプのシークレットについては、独自のローテーション関数を記述します。手順については、「the section called “Lambda ローテーション関数”」を参照してください。

2. 必要な依存関係と共にファイルを ZIP ファイル (*my-function.zip*) として保存します。

ステップ 2 : Lambda 関数を作成する

このステップでは、ステップ 1 で作成した ZIP ファイルを使用して Lambda 関数を作成します。関数が呼び出されたときに Lambda が引き受けるロール (Lambda 実行ロール) も設定します。

Lambda ローテーション関数と実行ロールを作成するには

1. Lambda 実行ロールの信頼ポリシーを作成した後に JSON ファイルとして保存します。詳細情報と例については、「the section called “ローテーションへのアクセス許可”」を参照してください。ポリシーは次の条件を満たす必要があります。

   - ロールがシークレットの Secrets Manager オペレーションを呼び出すことを許可します。
   - 例えば、新しいパスワードを作成するなど、ロールがシークレットが使用するサービスを呼び出すことを許可します。

2. Lambda 実行ロールを作成し、`iam create-role` を呼び出して、以前のステップで作成した信頼ポリシーを適用します。

```
aws iam create-role \
    --role-name rotation-lambda-role \
    --assume-role-policy-document file://trust-policy.json
```

3. `lambda create-function` を呼び出して、ZIP ファイルから Lambda 関数を作成します。

```
aws lambda create-function \
  --function-name my-rotation-function \
  --runtime python3.12 \
  --zip-file fileb://my-function.zip \
  --handler lambda_function.lambda_handler \
  --role arn:aws:iam::123456789012:role/service-role/rotation-lambda-role
```

4. Lambda 関数にリソースポリシーを設定し、`lambda add-permission` を呼び出すことで Secrets Manager がそれを呼び出せるようにします。

```
aws lambda add-permission \
  --function-name my-rotation-function \
```

```
  --action lambda:InvokeFunction \
  --statement-id SecretsManager \
  --principal secretsmanager.amazonaws.com \
  --source-account 123456789012
```

ステップ 3: ネットワークアクセスを設定する

詳細については、「the section called “AWS Lambda ローテーション関数のネットワークアクセス”」を参照してください。

ステップ 4: シークレットのローテーションを設定する

シークレットの自動ローテーションをオンにするには、`rotate-secret` を呼び出します。ローテーションスケジュールは `cron( )` または `rate( )` のスケジュール式で設定でき、ローテーション期間を設定できます。詳細については、「the section called “ローテーションスケジュール”」を参照してください。

```
aws secretsmanager rotate-secret \
    --secret-id MySecret \
    --rotation-lambda-arn arn:aws:lambda:aws-region:123456789012:function:my-rotation-function \
    --rotation-rules "{\"ScheduleExpression\": \"cron(0 16 1,15 * ? *)\", \"Duration\": \"2h\"}"
```

次の手順

「the section called “ローテーションのトラブルシューティング”」を参照してください。

Lambda 関数のローテーション戦略

the section called “Lambda 関数によるローテーション” では、Secrets Manager はデータベースシークレットに対して、2 つのローテーション戦略を提供します。

ローテーション戦略: シングルユーザー

この戦略は、1 つのシークレット内で 1 人のユーザーの認証情報を更新します。Amazon RDS Db2 インスタンスの場合、ユーザーは自分のパスワードを変更できないため、管理者の認証情報を別のシークレットで提供する必要があります。これは最も簡単なローテーション戦略であり、ほとんどのユースケースに使用することができます。特に、1 回限りの (アドホック) ユーザーまたはインタラクティブユーザーの認証情報には、この方法を使用することを推奨します。

シークレットがローテーションしても、開いているデータベース接続は切断されません。ローテーションの実行中、データベース内のパスワードが変更されてからシークレットが更新されるまでには少し時間がかかります。その間に、ローテーションされた認証情報を使用する呼び出しがデータベースによって拒否されるリスクは低く抑えられています。このリスクは、適切な再試行戦略を適用することで低減することが可能です。ローテーション後、新しい接続は新しい認証情報を使用します。

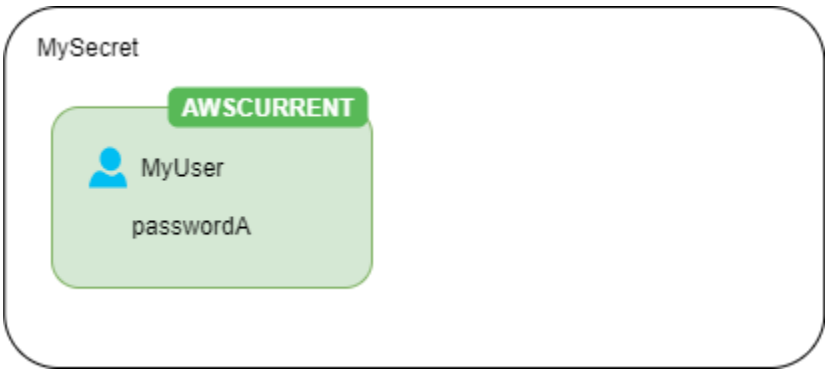
ローテーション戦略: 交代ユーザー

この戦略は、1 つのシークレット内で 2 人のユーザーの認証情報を更新します。最初のユーザーを作成し、その最初のローテーション中に、ローテーション関数がユーザーのクローンを作成して 2 人目のユーザーを作成します。シークレットがローテーションされるたびに、ローテーション関数はパスワードを更新するユーザーを切り替えます。ほとんどのユーザーにはクローン作成権限がないため、`superuser` の認証情報を別のシークレット内で用意する必要があります。データベース内のクローンユーザーがオリジナルユーザーと同じ権限を持っていない場合や、1 回限り (アドホック) またはインタラクティブなユーザーの認証情報には、シングルユーザーローテーション方法を使用することを推奨します。

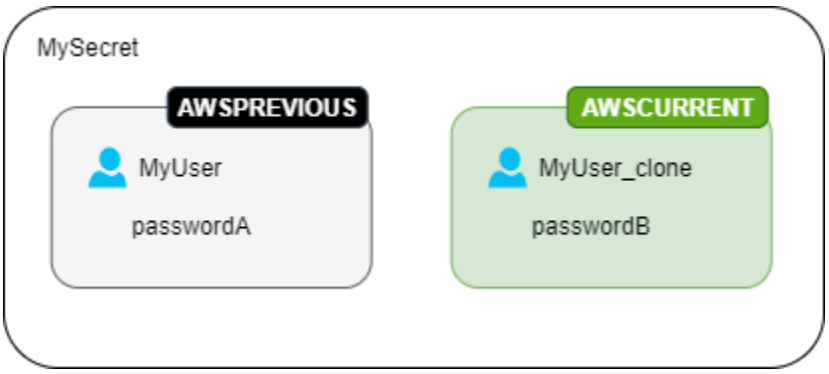
この戦略は、1 つ目のロールがデータベーステーブルを所有し、2 つ目のロールにそのデータベーステーブルへのアクセス許可を付与するといった権限モデルのデータベースに適しています。また、高可用性を必要とするアプリケーションにも適しています。アプリケーションは、ローテーション中にシークレットを取得しても、引き続き有効な認証情報セットを取得します。ローテーション後、`user` と `user_clone` の両方の認証情報が有効になります。このタイプのローテーション中にアプリケーションが拒否される可能性は、シングルユーザーのローテーションを比較すると一段と低くなります。データベースをホストしているサーバーファームでパスワードの変更がサーバー全体に伝播するまでに時間がかかる場合には、新しい認証情報を使用する呼び出しがデータベースによって拒否されるおそれがあります。このリスクは、適切な再試行戦略を適用することで低減することが可能です。

Secrets Manager は、元のユーザーと同じアクセス許可を持つクローンユーザーを作成します。クローンユーザーが作成された後に元のユーザーの権限を変更する場合は、クローンユーザー側の権限も変更する必要があります。

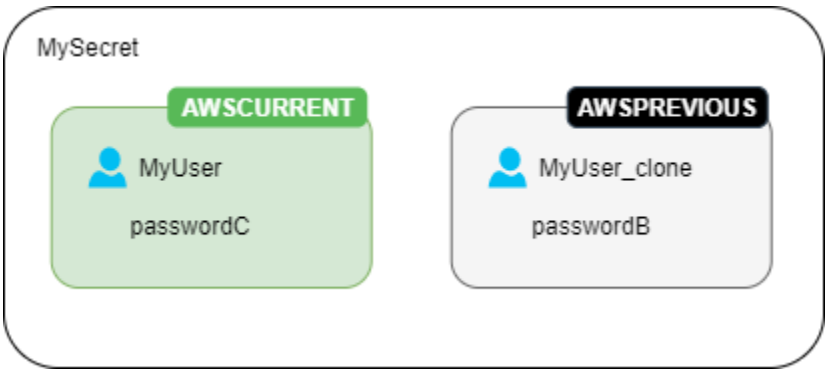
例えば、データベースユーザーの認証情報を使用してシークレットを作成した場合、そのシークレットには、使用した認証情報によるバージョンが 1 つ含まれます。



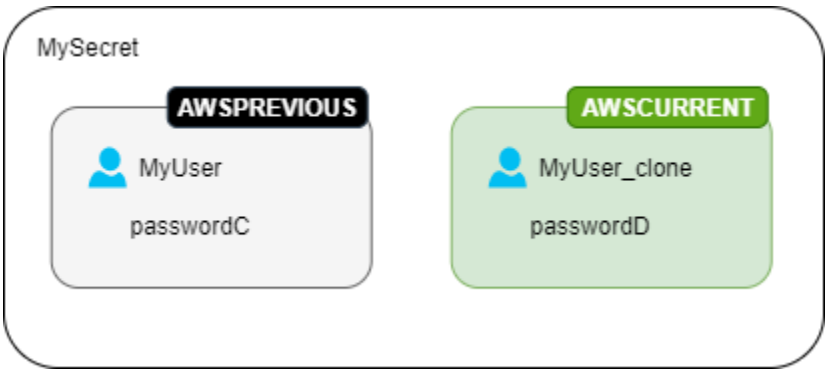
1 回目のローテーション – ローテーション関数がパスワードを生成し、それを使用してユーザーのクローンを作成します。それらの認証情報は、現在のシークレットバージョンになります。



2 回目のローテーション – ローテーション関数は、元のユーザーのパスワードを更新します。



3 回目のローテーション – ローテーション関数は、クローンされたユーザーのパスワードを更新します。



Lambda ローテーション関数

では[the section called “Lambda 関数によるローテーション”](#)、 AWS Lambda 関数はシークレットをローテーションします。 AWS Secrets Manager は[ステージングラベル](#)を使用してローテーション中にシークレットバージョンを識別します。

AWS Secrets Manager がシークレットタイプの[ローテーション関数テンプレート](#)を提供していない場合は、カスタムローテーション関数を作成できます。ローテーション関数を記述するときは、次のガイドラインに従ってください。

カスタムローテーション関数のベストプラクティス

- 「[汎用ローテーションテンプレート](#)」を開始点として使用します。
- デバッグまたはログ記録ステートメントには注意してください。これらは、Amazon CloudWatch Logs に情報を書き込む場合があります。ログに機密情報が含まれていないことを確認します。

  ログステートメントの例については、[the section called “ローテーション関数のテンプレート”](#) のソースコードを参照してください。
- セキュリティのため、 は Lambda ローテーション関数 AWS Secrets Manager のみがシークレットを直接ローテーションできるようにします。ローテーション関数では、別の Lambda 関数を呼び出してシークレットをローテーションすることはできません。
- デバッグのガイダンスについては、「[サーバーレスアプリケーションのテストとデバッグ](#)」を参照してください。
- 例えば、外部バイナリとライブラリを使用してリソースに接続する場合、それらにパッチを適用して更新する責任があります。
- ローテーション関数と *my-function.zip* などの依存関係を ZIP ファイルにパッケージ化します。

> **Warning**
>
> プロビジョニングされた同時実行パラメータを 10 未満の値に設定すると、Lambda 関数の実行スレッドが不十分であるためにスロットリングが発生する可能性があります。詳細については、「 AWS Lambda AWS Lambda デベロッパーガイド」の「予約された同時実行とプロビジョニングされた同時実行について」を参照してください。

ローテーション関数の 4 つのステップ

トピック

- createSecret: シークレットの新しいバージョンを作成する
- setSecret: データベースまたはサービスの認証情報を変更する
- testSecret: 新しいシークレットバージョンをテストする
- finishSecret: ローテーションを終了する

createSecret: シークレットの新しいバージョンを作成する

メソッド `createSecret` は、まず渡された `ClientRequestToken` で `get_secret_value` を呼び出して、シークレットが存在するかどうかを確認します。シークレットがない場合は、`create_secret` とトークンを `VersionId` として新しいシークレットを作成します。その後、`get_random_password` を使用して新しいシークレット値を生成します。次に `put_secret_value` を呼び出し、ステージングラベル `AWSPENDING` で保存します。新しいシークレット値を `AWSPENDING` に格納することで、冪等性を確保することができます。何らかの理由でローテーションが失敗した場合は、その後の呼び出しでそのシークレット値を参照できます。詳細については、「Lambda 関数を冪等にするにはどうすればよいですか?」を参照してください。

独自のローテーション関数を記述するためのヒント

- 新しいシークレット値には、データベースまたはサービスで有効な文字のみが含まれていることを確認する必要があります。`ExcludeCharacters` のパラメータを使用して文字を除外します。
- 関数をテストするときは、 を使用してバージョンステージを確認します。 AWS CLI を呼び出し`describe-secret`て を確認しますVersionIdsToStages。
- Amazon RDS MySQL の場合、交代ユーザーローテーションでは Secrets Manager が 16 文字以下の名前のクローンユーザーを作成します。ローテーション関数を変更して、長いユーザー名を許可することができます。MySQL バージョン 5.7 以降では最大 32 文字のユーザー名がサポートさ

れていますが、Secrets Manager ではユーザー名の末尾に「_clone」(6 文字) が追加されるため、ユーザー名は最大 26 文字にする必要があります。

setSecret: データベースまたはサービスの認証情報を変更する

メソッド `setSecret` は、データベースやサービス内の認証情報を、`AWSPENDING` のバージョンの新しいシークレット値と一致するように変更します。

独自のローテーション関数を記述するためのヒント

- データベースなど、ステートメントを解釈するサービスにステートメントを渡す場合は、クエリパラメータ化を使用します。詳細については、OWASP ウェブサイトの「Query Parameterization Cheat Sheet」を参照してください。
- ローテーション機能は、Secrets Manager のシークレットとターゲットリソースの両方にある顧客認証情報にアクセスして変更する権限を持つ特権的な代理プログラムです。混乱した代理攻撃を防ぐには、攻撃者がこの関数を使用して他のリソースにアクセスできないようにする必要があります。認証情報を更新する前に:
  - シークレットの `AWSCURRENT` バージョンの認証情報が有効であることを確認してください。`AWSCURRENT` の認証情報が有効でない場合は、ローテーションの試行を中止してください。
  - `AWSCURRENT` と `AWSPENDING` のシークレット値が同じリソース用であることを確認してください。ユーザー名とパスワードについては、`AWSCURRENT` と `AWSPENDING` のユーザー名が同じであることを確認してください。
  - 送信先のサービスリソースが同じであることを確認してください。データベースの場合、`AWSCURRENT` と `AWSPENDING` のホスト名が同じであることを確認してください。
- まれに、データベースの既存のローテーション関数のカスタマイズが必要な場合があります。例えば、交代ユーザーローテーションの場合、Secrets Manager は最初のユーザーのランタイム設定パラメータをコピーしてクローンユーザーを作成します。さらに属性を追加したり、クローンユーザーに付与する属性を変更したりする場合は、`set_secret` 関数のコードを更新する必要があります。

testSecret: 新しいシークレットバージョンをテストする

次に、Lambda ローテーション関数は、データベースまたはサービスにアクセスすることで、シークレットの `AWSPENDING` バージョンをテストします。ローテーション関数のテンプレート に基づくローテーション関数では、読み取りアクセスを使用して、新しいシークレットをテストします。

finishSecret: ローテーションを終了する

最後に、Lambda ローテーション関数はラベルを以前のシークレットバージョンAWSCURRENTからこのバージョンに移動します。Secrets Manager は、以前のバージョンに対しステージングラベル AWSPREVIOUS を付加します。これにより、シークレットの最後の有効なバージョンが保持されます。

メソッド finish_secret は、update_secret_version_stage を使用して、ステージングラベル AWSCURRENT を以前のシークレットバージョンから新しいシークレットバージョンに移動します。Secrets Manager は、以前のバージョンに対しステージングラベル AWSPREVIOUS を自動的に付加します。これにより、シークレットの最後の有効なバージョンが保持されます。

独自のローテーション関数を記述するためのヒント

- この時点より前に AWSPENDING を削除したり、別の API コールを使用して削除したりしないでください。これは、Secrets Manager にとっては、ローテーションが正常に完了しなかったことを意味する可能性があるからです。Secrets Manager は、以前のバージョンに対しステージングラベル AWSPREVIOUS を付加します。これにより、シークレットの最後の有効なバージョンが保持されます。

ローテーションが成功すると、AWSPENDING ステージングラベルは AWSCURRENT バージョンと同じバージョンにアタッチされるか、どのバージョンにもアタッチされない可能性があります。AWSPENDING ステージングラベルは存在するが、AWSCURRENT と同じバージョンにアタッチされていない場合、それ以降に呼び出されたローテーションでは、以前のローテーションリクエストがまだ進行中であるとみなされ、エラーが返されます。ローテーションに失敗すると、AWSPENDING ステージングラベルはバージョンが空のシークレットにアタッチされる可能性があります。詳細については、「ローテーションのトラブルシューティング」を参照してください。

AWS Secrets Manager ローテーション関数テンプレート

AWS Secrets Manager には、さまざまなデータベースシステムやサービスの認証情報の安全な管理を自動化するのに役立つ一連のローテーション関数テンプレートが用意されています。テンプレートはすぐに使用可能な Lambda 関数であり、認証情報のローテーションに関するベストプラクティスを実装し、手動による介入なしでセキュリティ体制を維持するのに役立ちます。

テンプレートは 2 つの主要なローテーション戦略をサポートしています。

- シングルユーザーローテーション: シングルユーザーの認証情報を更新します。

- 代替ユーザーのローテーション: 認証情報の変更中のダウンタイムを排除するために、2 人の個別のユーザーを維持します。

Secrets Manager には、あらゆるタイプのシークレットの開始点となる汎用テンプレートも用意されています。

テンプレートの使用方法については、以下を参照してください。

- データベースシークレットの自動ローテーション (コンソール)
- 非データベースシークレットの自動ローテーション (コンソール)

独自のローテーション関数を記述するには、「Write a rotation function」を参照してください。

テンプレート

- Amazon RDS と Amazon Aurora
  - Amazon RDS Db2 シングルユーザー
  - Amazon RDS Db2 交代ユーザー
  - Amazon RDS MariaDB シングルユーザー
  - Amazon RDS MariaDB 交代ユーザー
  - Amazon RDS および Amazon Aurora MySQL シングルユーザー
  - Amazon RDS および Amazon Aurora MySQL 交代ユーザー
  - Amazon RDS Oracle シングルユーザー
  - Amazon RDS Oracle 交代ユーザー
  - Amazon RDS および Amazon Aurora PostgreSQL シングルユーザー
  - Amazon RDS と Amazon Aurora PostgreSQL 交代ユーザー
  - Amazon RDS Microsoft SQLServer シングルユーザー
  - Amazon RDS Microsoft SQLServer 交代ユーザー
- Amazon DocumentDB (MongoDB 互換性)
  - Amazon DocumentDB シングルユーザー
  - Amazon DocumentDB 交代ユーザー
- Amazon Redshift
  - Amazon Redshift シングルユーザー
  - Amazon Redshift 交代ユーザー

- Amazon Timestream for InfluxDB
  - Amazon Timestream for InfluxDB シングルユーザー
  - Amazon Timestream for InfluxDB のユーザー交代
- Amazon ElastiCache
- Active Directory
  - Active Directory 認証情報
  - Active Directory キータブ
- その他のタイプのシークレット

Amazon RDS と Amazon Aurora

Amazon RDS Db2 シングルユーザー

- テンプレートの名前: SecretsManagerRDSDb2RotationSingleUser
- ローテーション戦略: ローテーション戦略: シングルユーザー
- **`SecretString`** 構造: the section called “Amazon RDS と Aurora の認証情報”
- ソースコード: https://github.com/aws-samples/aws-secrets-manager-rotation-lambdas/tree/master/SecretsManagerRDSDb2RotationSingleUser/lambda_function.py
- 依存関係: python-ibmdb 3.2.8

Amazon RDS Db2 交代ユーザー

- テンプレートの名前: SecretsManagerRDSDb2RotationMultiUser
- ローテーション戦略: the section called “交代ユーザー”
- **`SecretString`** 構造: the section called “Amazon RDS と Aurora の認証情報”
- ソースコード: https://github.com/aws-samples/aws-secrets-manager-rotation-lambdas/tree/master/SecretsManagerRDSDb2RotationMultiUser/lambda_function.py
- 依存関係: python-ibmdb 3.2.8

Amazon RDS MariaDB シングルユーザー

- テンプレートの名前: SecretsManagerRDSMariaDBRotationSingleUser
- ローテーション戦略: ローテーション戦略: シングルユーザー

- **`SecretString`** 構造: the section called “Amazon RDS と Aurora の認証情報”
- ソースコード: https://github.com/aws-samples/aws-secrets-manager-rotation-lambdas/tree/master/SecretsManagerRDSMariaDBRotationSingleUser/lambda_function.py
- 依存関係: PyMySQL 1.1.1。認証に sha256 パスワードを使用する場合は、PyMySQL[rsa] を使用します。Lambda ランタイムでコンパイルされたコードでパッケージを使用する方法については、「AWS 情報センター」の「コンパイル済みバイナリを含む Python パッケージをデプロイパッケージに追加して、そのパッケージに Lambda との互換性を持たせるには、どうすればよいですか?」を参照してください。

Amazon RDS MariaDB 交代ユーザー

- テンプレートの名前: SecretsManagerRDSMariaDBRotationMultiUser
- ローテーション戦略: the section called “交代ユーザー”
- **`SecretString`** 構造: the section called “Amazon RDS と Aurora の認証情報”
- ソースコード: https://github.com/aws-samples/aws-secrets-manager-rotation-lambdas/tree/master/SecretsManagerRDSMariaDBRotationMultiUser/lambda_function.py
- 依存関係: PyMySQL 1.1.1。認証に sha256 パスワードを使用する場合は、PyMySQL[rsa] を使用します。Lambda ランタイムでコンパイルされたコードでパッケージを使用する方法については、「AWS 情報センター」の「コンパイル済みバイナリを含む Python パッケージをデプロイパッケージに追加して、そのパッケージに Lambda との互換性を持たせるには、どうすればよいですか?」を参照してください。

Amazon RDS および Amazon Aurora MySQL シングルユーザー

- テンプレートの名前: SecretsManagerRDSMySQLRotationSingleUser
- ローテーション戦略: the section called “シングルユーザー”
- 期待される **`SecretString`**構造体: the section called “Amazon RDS と Aurora の認証情報”
- ソースコード: https://github.com/aws-samples/aws-secrets-manager-rotation-lambdas/tree/master/SecretsManagerRDSMySQLRotationSingleUser/lambda_function.py
- 依存関係: PyMySQL 1.1.1。認証に sha256 パスワードを使用する場合は、PyMySQL[rsa] を使用します。Lambda ランタイムでコンパイルされたコードでパッケージを使用する方法については、「AWS 情報センター」の「コンパイル済みバイナリを含む Python パッケージをデプロイパッケージに追加して、そのパッケージに Lambda との互換性を持たせるには、どうすればよいですか?」を参照してください。

Amazon RDS および Amazon Aurora MySQL 交代ユーザー

- テンプレートの名前: SecretsManagerRDSMySQLRotationMultiUser
- ローテーション戦略: the section called “交代ユーザー”
- 期待される **`SecretString`**構造体: the section called “Amazon RDS と Aurora の認証情報”
- ソースコード: https://github.com/aws-samples/aws-secrets-manager-rotation-lambdas/tree/master/SecretsManagerRDSMySQLRotationMultiUser/lambda_function.py
- 依存関係: PyMySQL 1.1.1。認証に sha256 パスワードを使用する場合は、PyMySQL[rsa] を使用します。Lambda ランタイムでコンパイルされたコードでパッケージを使用する方法については、「AWS 情報センター」の「コンパイル済みバイナリを含む Python パッケージをデプロイパッケージに追加して、そのパッケージに Lambda との互換性を持たせるには、どうすればよいですか?」を参照してください。

Amazon RDS Oracle シングルユーザー

- テンプレートの名前: SecretsManagerRDSOracleRotationSingleUser
- ローテーション戦略: the section called “シングルユーザー”
- 予想される **`SecretString`** 構造体 the section called “Amazon RDS と Aurora の認証情報”
- ソースコード:https://github.com/aws-samples/aws-secrets-manager-rotation-lambdas/tree/master/SecretsManagerRDSOracleRotationSingleUser/lambda_function.py
- 依存関係: python-oracledb 2.4.1

Amazon RDS Oracle 交代ユーザー

- テンプレートの名前: SecretsManagerRDSOracleRotationMultiUser
- ローテーション戦略: the section called “交代ユーザー”
- 予想される **`SecretString`** 構造体 the section called “Amazon RDS と Aurora の認証情報”
- ソースコードhttps://github.com/aws-samples/aws-secrets-manager-rotation-lambdas/tree/master/SecretsManagerRDSOracleRotationMultiUser/lambda_function.py
- 依存関係: python-oracledb 2.4.1

Amazon RDS および Amazon Aurora PostgreSQL シングルユーザー

- テンプレートの名前 SecretsManagerRDSPostgreSQLRotationSingleUser

- ローテーション戦略: ローテーション戦略: シングルユーザー
- 予想される **`SecretString`** 構造体: the section called “Amazon RDS と Aurora の認証情報”
- ソースコード: https://github.com/aws-samples/aws-secrets-manager-rotation-lambdas/tree/master/SecretsManagerRDSPostgreSQLRotationSingleUser/lambda_function.py
- 依存関係: PyGreSQL 6.1.0

Amazon RDS と Amazon Aurora PostgreSQL 交代ユーザー

- テンプレートの名前: SecretsManagerRDSPostgreSQLRotationMultiUser
- ローテーション戦略: the section called “交代ユーザー”
- 予想される**`SecretString`** 構造体: the section called “Amazon RDS と Aurora の認証情報”
- ソースコードhttps://github.com/aws-samples/aws-secrets-manager-rotation-lambdas/tree/master/SecretsManagerRDSPostgreSQLRotationMultiUser/lambda_function.py
- 依存関係: PyGreSQL 6.1.0

Amazon RDS Microsoft SQLServer シングルユーザー

- テンプレートの名前: SecretsManagerRDSSQLServerRotationSingleUser
- ローテーション戦略: the section called “シングルユーザー”
- 予想される **`SecretString`** 構造体: the section called “Amazon RDS と Aurora の認証情報”
- ソースコード: https://github.com/aws-samples/aws-secrets-manager-rotation-lambdas/tree/master/SecretsManagerRDSSQLServerRotationSingleUser/lambda_function.py
- 依存関係: Pymssql 2.3.11

Amazon RDS Microsoft SQLServer 交代ユーザー

- テンプレートの名前: SecretsManagerRDSSQLServerRotationMultiUs
- ローテーション戦略: the section called “交代ユーザー”
- 期待される **`SecretString`** 構造体: the section called “Amazon RDS と Aurora の認証情報”
- ソースコード: https://github.com/aws-samples/aws-secrets-manager-rotation-lambdas/tree/master/SecretsManagerRDSSQLServerRotationMultiUser/lambda_function.py
- 依存関係: Pymssql 2.3.11

Amazon DocumentDB (MongoDB 互換性)

Amazon DocumentDB シングルユーザー

- テンプレートの名前: SecretsManagerMongoDBRotationSingleUser
- ローテーション戦略: the section called “シングルユーザー”
- 期待される **`SecretString`** 構造体: the section called “Amazon DocumentDB 認証情報”
- ソースコード: https://github.com/aws-samples/aws-secrets-manager-rotation-lambdas/tree/master/SecretsManagerMongoDBRotationSingleUser/lambda_function.py
- 依存関係: PyMongo 4.7.3

Amazon DocumentDB 交代ユーザー

- テンプレートの名前: SecretsManagerMongoDBRotationMultiUser
- ローテーション戦略: the section called “交代ユーザー”
- 期待される **`SecretString`** 構造体: the section called “Amazon DocumentDB 認証情報”
- ソースコード: https://github.com/aws-samples/aws-secrets-manager-rotation-lambdas/tree/master/SecretsManagerMongoDBRotationMultiUser/lambda_function.py
- 依存関係: PyMongo 4.7.3

Amazon Redshift

Amazon Redshift シングルユーザー

- テンプレートの名前: SecretsManagerRedshiftRotationSingleUser
- ローテーション戦略: the section called “シングルユーザー”
- 期待される **`SecretString`** 構造体: the section called “Amazon Redshift 認証情報”
- ソースコード: https://github.com/aws-samples/aws-secrets-manager-rotation-lambdas/tree/master/SecretsManagerRedshiftRotationSingleUser/lambda_function.py
- 依存関係: redshift-connector 2.1.13

Amazon Redshift 交代ユーザー

- テンプレートの名前: SecretsManagerRedshiftRotationMultiUser

- ローテーション戦略: the section called “交代ユーザー”
- 期待される **SecretString** 構造体: the section called “Amazon Redshift 認証情報”
- ソースコード: https://github.com/aws-samples/aws-secrets-manager-rotation-lambdas/tree/master/SecretsManagerRedshiftRotationMultiUser/lambda_function.py
- 依存関係: redshift-connector 2.1.13

Amazon Timestream for InfluxDB

これらのテンプレートを使用するには、「Amazon Timestream Developer Guide」の「How Amazon Timestream for InfluxDB uses secrets」を参照してください。

Amazon Timestream for InfluxDB シングルユーザー

- テンプレート名: SecretsManagerInfluxDBRotationSingleUser
- 期待される **SecretString**構造体: the section called “Amazon Timestream for InfluxDB のシークレット構造”
- ソースコード: https://github.com/aws-samples/aws-secrets-manager-rotation-lambdas/tree/master/SecretsManagerInfluxDBRotationSingleUser/lambda_function.py
- 依存関係: InfluxDB 2.0 Python クライアント

Amazon Timestream for InfluxDB のユーザー交代

- テンプレート名: SecretsManagerInfluxDBRotationMultiUser
- 期待される **SecretString**構造体: the section called “Amazon Timestream for InfluxDB のシークレット構造”
- ソースコード: https://github.com/aws-samples/aws-secrets-manager-rotation-lambdas/tree/master/SecretsManagerInfluxDBRotationMultiUser/lambda_function.py
- 依存関係: InfluxDB 2.0 Python クライアント

Amazon ElastiCache

このテンプレートを使用するには、「Amazon ElastiCache ユーザーガイド」の「Automatically rotating passwords for users」(ユーザーのパスワードの自動ローテーション) を参照してください。

- テンプレート名: SecretsManagerElasticacheUserRotation

- 期待される **`SecretString`**構造体: the section called “Amazon ElastiCache の認証情報”
- ソースコード: https://github.com/aws-samples/aws-secrets-manager-rotation-lambdas/tree/master/SecretsManagerElasticacheUserRotation/lambda_function.py

Active Directory

Active Directory 認証情報

- テンプレート名: SecretsManagerActiveDirectoryRotationSingleUser
- 期待される **`SecretString`**構造体: the section called “Active Directory 認証情報”
- ソースコード: https://github.com/aws-samples/aws-secrets-manager-rotation-lambdas/tree/master/SecretsManagerActiveDirectoryRotationSingleUser/lambda_function.py

Active Directory キータブ

- テンプレート名: SecretsManagerActiveDirectoryAndKeytabRotationSingleUser
- 期待される **`SecretString`**構造体: the section called “Active Directory 認証情報”
- ソースコード: https://github.com/aws-samples/aws-secrets-manager-rotation-lambdas/tree/master/SecretsManagerActiveDirectoryAndKeytabRotationSingleUser/lambda_function.py
- 依存関係: msktutil

その他のタイプのシークレット

Secrets Manager は、任意のタイプのシークレットのローテーション関数を作成するための開始点として、このテンプレートを提供します。

- テンプレートの名前: SecretsManagerRotationTemplate
- ソースコード: https://github.com/aws-samples/aws-secrets-manager-rotation-lambdas/tree/master/SecretsManagerRotationTemplate/lambda_function.py

の Lambda ローテーション関数実行ロールのアクセス許可 AWS Secrets Manager

the section called “Lambda 関数によるローテーション” では、Secrets Manager が Lambda 関数を使用してシークレットをローテーションすると、Lambda は IAM 実行ロールを想定し、これらの認証

情報を Lambda 関数のコードに提供します。自動ローテーションの設定方法については、以下を参照してください。

- データベースシークレットの自動ローテーション (コンソール)
- 非データベースシークレットの自動ローテーション (コンソール)
- 自動ローテーション (AWS CLI)

次の例は、Lambda ローテーション関数の実行ロールのインラインポリシーを示しています。実行ロールを作成し、アクセス権限ポリシーをアタッチするには、を参照してください。AWS Lambda 実行ロール。

例:

- Lambda ローテーション関数の実行ロールのポリシー
- カスタマーマネージドキーのポリシーステートメント
- 交代ユーザー戦略のポリシーステートメント

Lambda ローテーション関数の実行ロールのポリシー

次のポリシーの例では、ローテーション関数が次の操作を許可します。

- *SecretARN* の Secrets Manager 操作を実行します。
- 新しいパスワードを作成します。
- データベースまたはサービスが VPC で実行されている場合、必要な設定のセットアップを行います。VPC 内のリソースにアクセスするように Lambda 関数を設定する、を参照してください。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:DescribeSecret",
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutSecretValue",
```

```
                "secretsmanager:UpdateSecretVersionStage"
            ],
            "Resource": "arn:aws:secretsmanager:us-
east-1:123456789012:secret:secretName-AbCdEf"
    },
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetRandomPassword"
            ],
            "Resource": "*"
        },
        {
            "Action": [
                "ec2:CreateNetworkInterface",
                "ec2:DeleteNetworkInterface",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DetachNetworkInterface"
            ],
            "Resource": "*",
            "Effect": "Allow"
        }
    ]
}
```

カスタマーマネージドキーのポリシーステートメント

AWS マネージドキー aws/secretsmanager 以外の KMS キーを使用しシークレットを暗号化する場合は、そのキーの使用に関するアクセス許可を、Lambda の実行ロールに付与する必要があります。SecretArn 暗号化コンテキストを使用して復号化関数の使用を制限できます。この場合、ローテーション関数ロールには、ローテーションに使用するシークレットを復号化するアクセスのみが許可されます。次に、実行ロールポリシーに追加してKMS キーを使用してシークレットを復号化するステートメントの例を示します。

```
    {
        "Effect": "Allow",
        "Action": [
            "kms:Decrypt",
            "kms:DescribeKey",
            "kms:GenerateDataKey"
        ],
```

```
            "Resource": "KMSKeyARN",
            "Condition": {
                "StringEquals": {
                    "kms:EncryptionContext:SecretARN": "SecretARN"
                }
            }
        }
```

カスタマーマネージドキーで暗号化された複数のシークレットに対してローテーション機能を使用するには、以下の例のようなステートメントを追加して、実行ロールがシークレットを復号化できるようにします。

```
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "kms:DescribeKey",
                "kms:GenerateDataKey"
            ],
            "Resource": "KMSKeyARN",
            "Condition": {
                "StringEquals": {
                    "kms:EncryptionContext:SecretARN": [
                        "arn1",
                        "arn2"
                    ]
                }
            }
        }
```

交代ユーザー戦略のポリシーステートメント

交代ユーザーローテーション戦略については、「the section called “Lambda 関数のローテーション戦略”」を参照してください。

Amazon RDS 認証情報を含むシークレットで、代替ユーザー戦略を使用しており、スーパーユーザーシークレットが Amazon RDS によって管理されている場合、ローテーション関数が Amazon RDS の読み取り専用 API を呼び出して、データベースの接続情報を取得できるようにする必要もあります。 AWS 管理ポリシー AmazonRDSReadOnlyAccess をアタッチすることをお勧めします。

次のポリシーの例では、関数が次の操作を許可します。

- *SecretARN* の Secrets Manager 操作を実行します。
- スーパーユーザーシークレットで認証情報を取得します。Secrets Manager は、スーパーユーザーシークレットの認証情報を使用し、ローテーションされたシークレットの認証情報を更新します。
- 新しいパスワードを作成します。
- データベースまたはサービスが VPC で実行される場合、必要な設定のセットアップを行います。詳細については、「VPC 内のリソースにアクセスするように Lambda 関数を設定する」を参照してください。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:DescribeSecret",
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutSecretValue",
                "secretsmanager:UpdateSecretVersionStage"
            ],
            "Resource": "arn:aws:secretsmanager:us-
east-1:123456789012:secret:secretName-AbCdEf"
    },
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue"
            ],
            "Resource": "arn:aws:secretsmanager:us-
east-1:123456789012:secret:secretName-AbCdEf"
    },
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetRandomPassword"
            ],
            "Resource": "*"
        },
        {
```

```
            "Action": [
                "ec2:CreateNetworkInterface",
                "ec2:DeleteNetworkInterface",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DetachNetworkInterface"
            ],
            "Resource": "*",
            "Effect": "Allow"
        }
    ]
}
```

AWS Lambda ローテーション関数のネットワークアクセス

the section called “Lambda 関数によるローテーション” では、Secrets Manager が Lambda 関数を使用してシークレットをローテーションする場合、Lambda ローテーション関数がシークレットにアクセスできる必要があります。シークレットに認証情報が含まれている場合、Lambda 関数はそれらの認証情報のソース (データベースやサービスなど) にもアクセスできる必要があります。

シークレットにアクセスするには

ローテーション用の Lambda 関数は、Secrets Manager のエンドポイントにアクセスできる必要があります。Lambda 関数がインターネットにアクセスできる場合は、パブリックなエンドポイントを使用できます。エンドポイントを見つけるには、「the section called “Secrets Manager エンドポイント”」を参照してください。

インターネットにアクセスできない VPC で Lambda 関数を実行する場合は、Secrets Manager サービスのプライベートなエンドポイントを、VPC 内に設定することをお勧めします。VPC は、リージョンのパブリックなエンドポイントに向けられたリクエストを傍受し、それらをプライベートエンドポイントにリダイレクトします。(詳しくは、「VPC エンドポイント (AWS PrivateLink)」を参照してください。)

別の方法としては、NAT ゲートウェイまたはインターネットゲートウェイを VPC に追加して (これで VPC のトラフィックはパブリックエンドポイントに到達できます)、Lambda 関数から Secrets Manager のパブリックなエンドポイントへのアクセスを許可することも考えられます。この方法では、VPC がある程度のリスクにさらされることになります。ゲートウェイ向けの IP アドレスには、パブリックなインターネットから攻撃が可能なためです。

(オプション) データベースまたはサービスにアクセスするには

API キーなどのシークレットについては、シークレットと一緒に更新する必要のあるソースデータベースやサービスはありません。

データベースまたはサービスを VPC の Amazon EC2 インスタンスで実行している場合は、同じ VPC で Lambda 関数を設定することをお勧めします。こうすることで、ローテーション関数はサービスと直接通信できるようになります。詳細については、Configuring VPC access を参照してください。

Lambda 関数からデータベースまたはサービスへのアクセスを可能にするには、ローテーション用の Lambda 関数にアタッチされたセキュリティグループによって、そのデータベースまたはサービスに対するアウトバウンド接続が許可されている必要があります。同時に、データベースまたはサービスにアタッチされているセキュリティグループでは、ローテーション用 Lambda 関数からのインバウンド接続を許可する必要もあります。

AWS Secrets Managerローテーションのトラブルシューティング

多くのサービスでは、Secrets Manager は、Lambda 関数を使用してシークレットをローテーションします。詳細については、「the section called “Lambda 関数によるローテーション”」を参照してください。Lambda ローテーション関数は、シークレットの対象となるデータベースまたはサービス、および Secrets Manager とやり取りします。ローテーションが想定通りに動作していないときは、まず CloudWatch Logs を確認します。

> **Note**
>
> 一部のサービスは、ユーザーのためにシークレットを管理できます (自動ローテーションの管理など)。詳細については、「the section called “マネージドローテーション”」を参照してください。

トピック

- AWS Lambda関数のシークレットローテーションの失敗をトラブルシューティングする方法
- 「環境変数に認証情報が見つかりました」の後にアクティビティがない
- 「CreateSecret」の後にアクティビティがない
- エラー:「KMS へのアクセスは許可されていません」

- エラー:「シークレット JSON にキーがありません」
- エラー:「setSecret: データベースにログインできません」
- エラー:「モジュール 'lambda_function' をインポートできません」
- 既存のローテーション関数を Python 3.12 にアップグレードする
- AWS Lambdaシークレットのローテーションが PutSecretValue失敗しました
- エラー: <a rotation> ステップ中に Lambda <arn> を実行するときのエラー

AWS Lambda関数のシークレットローテーションの失敗をトラブルシューティングする方法

Lambda 関数でシークレットローテーションが失敗した場合は、次の手順を使用して問題のトラブルシューティングと解決を行います。

考えられる原因

- Lambda 関数の同時実行が不十分である
- ローテーション中の複数の API コールによる競合状態
- Lambda 関数ロジックが正しくない
- Lambda 関数とデータベース間のネットワークの問題

一般的なトラブルシューティングステップ

1. CloudWatch ログを分析します:
   - Lambda 関数ログで特定のエラーメッセージまたは予期しない動作を検索する
   - すべてのローテーションステップ (CreateSecret、SetSecret、TestSecret、FinishSecret) が試行されていることを確認する
2. ローテーション中に API コールを確認します:
   - Lambda ローテーション中にシークレットで API コールが変更されないようにする
   - RotateSecret と PutSecretValue の呼び出しの間に競合状態がないことを確認する
3. Lambda 関数ロジックを確認します:
   - シークレットローテーションに最新のAWSサンプルコードを使用していることを確認します。

- カスタムコードを使用する場合は、すべてのローテーションステップが適切に処理されることを確認する

4. ネットワーク構成を確認します:

   - セキュリティグループルールで Lambda 関数がデータベースにアクセスできることを確認する
   - Secrets Manager の適切な VPC エンドポイントまたはパブリックエンドポイントアクセスを確保する

5. シークレットバージョンをテストします:

   - シークレットの AWSCURRENT バージョンがデータベースアクセスを許可していることを確認する
   - AWSPREVIOUS または AWSPENDING のバージョンが有効かどうかを確認する

6. 保留中のローテーションをクリアします:

   - ローテーションが一貫して失敗する場合は、AWSPENDING ステージングラベルをクリアしてローテーションを再試行する

7. Lambda の同時実行設定を確認します:

   - 同時実行設定がワークロードに適していることを確認する
   - 同時実行の問題が疑われる場合は、「同時実行関連のローテーション失敗のトラブルシューティング」に関するセクションを参照する

「環境変数に認証情報が見つかりました」の後にアクティビティがない

「環境変数に認証情報が見つかりました」の後にアクティビティがなく、タスクの所要時間が長い (例: デフォルトの Lambda タイムアウトは 30000 ms) 場合は、Secrets Manager エンドポイントへのアクセス時に Lambda 関数がタイムアウトしている可能性があります。

ローテーション用の Lambda 関数は、Secrets Manager のエンドポイントにアクセスできる必要があります。Lambda 関数がインターネットにアクセスできる場合は、パブリックなエンドポイントを使用できます。エンドポイントを見つけるには、「the section called “Secrets Manager エンドポイント”」を参照してください。

インターネットにアクセスできない VPC で Lambda 関数を実行する場合は、Secrets Manager サービスのプライベートなエンドポイントを、VPC 内に設定することをお勧めします。VPC は、リージョンのパブリックなエンドポイントに向けられたリクエストを傍受し、それらをプライベートエン

ドポイントにリダイレクトします。(詳しくは、「VPC エンドポイント (AWS PrivateLink)」を参照してください。)

別の方法としては、NAT ゲートウェイまたはインターネットゲートウェイを VPC に追加して (これで VPC のトラフィックはパブリックエンドポイントに到達できます)、Lambda 関数から Secrets Manager のパブリックなエンドポイントへのアクセスを許可することも考えられます。この方法では、VPC がある程度のリスクにさらされることになります。ゲートウェイ向けの IP アドレスには、パブリックなインターネットから攻撃が可能なためです。

「CreateSecret」の後にアクティビティがない

CreateSecret の実行後にローテーションが停止する原因となる問題は次のとおりです。

VPC ネットワーク ACL では、HTTPS トラフィックの送受信が許可されません。

> 詳細については、「Amazon VPC ユーザーガイド」の「ネットワーク ACL を使用してサブネットへのトラフィックを制御する」を参照してください。

Lambda 関数のタイムアウト設定が短すぎてタスクを実行できません。

> 詳細については、「AWS Lambdaデベロッパーガイド」の「Lambda 関数オプションの設定」を参照してください。

Secrets Manager VPC エンドポイントは、割り当てられたセキュリティグループへの進入時に VPC CIDR を許可しません。

> 詳細については、「Amazon Virtual Private Cloud ユーザーガイド」の「Control traffic to resources using security groups」(セキュリティグループを使用してリソースへのトラフィックを制御する) を参照してください。

Secrets Manager VPC エンドポイントポリシーでは、Lambda が VPC エンドポイントを使用することを許可していません。

> 詳細については、「the section called “VPC エンドポイント (AWS PrivateLink)”」を参照してください。

シークレットは交代ユーザーローテーションを使用し、スーパーユーザーシークレットは Amazon RDS によって管理され、Lambda 関数は RDS API にアクセスできません。

> スーパーユーザーシークレットが他のAWSサービスによって管理されている交代ユーザーローテーションでは、Lambda ローテーション関数がサービスエンドポイントを呼び出してデータベース接続情報を取得できる必要があります。データベースサービスに VPC エンドポイントを設定することを推奨します。詳細については、以下を参照してください。

- 「Amazon RDS ユーザーガイド」の「Amazon RDS API およびインターフェース VPC エンドポイント」
- 「Amazon Redshift 管理ガイド」の「VPC エンドポイントの操作」

エラー:「KMS へのアクセスは許可されていません」

`ClientError: An error occurred (AccessDeniedException) when calling the GetSecretValue operation: Access to KMS is not allowed` が表示されている場合、シークレットの暗号化に使用された KMS キーを使用してシークレットを復号化するアクセス許可が、ローテーション関数に付与されていません。暗号化コンテキストを特定のシークレットに制限する条件が、アクセス許可ポリシーに含まれている可能性があります。必要なアクセス許可の詳細については、「the section called “カスタマーマネージドキーのポリシーステートメント”」を参照してください。

エラー:「シークレット JSON にキーがありません」

Lambda ローテーション関数では、シークレット値が特定の JSON 構造になっている必要があります。このエラーが表示される場合は、ローテーション関数がアクセスしようとしたキーが JSON にない可能性があります。各タイプのシークレットの JSON 構造については、「the section called “シークレットの JSON 構造”」を参照してください。

エラー:「setSecret: データベースにログインできません」

このエラーを引き起こす可能性のある問題は次のとおりです。

ローテーション関数はデータベースにアクセスできません。

タスクの所要時間が長い (例: 5000 ミリ秒以上) 場合、Lambda ローテーション関数はネットワーク経由でデータベースにアクセスできない可能性があります。

データベースまたはサービスを VPC の Amazon EC2 インスタンスで実行している場合は、同じ VPC で Lambda 関数を設定することをお勧めします。こうすることで、ローテーション関数はサービスと直接通信できるようになります。詳細については、Configuring VPC access を参照してください。

Lambda 関数からデータベースまたはサービスへのアクセスを可能にするには、ローテーション用の Lambda 関数にアタッチされたセキュリティグループによって、そのデータベースまたはサービスに対するアウトバウンド接続が許可されている必要があります。同時に、データベース

またはサービスにアタッチされているセキュリティグループでは、ローテーション用 Lambda 関数からのインバウンド接続を許可する必要もあります。

シークレットの認証情報が正しくありません。

タスクの所要時間が短い場合、Lambda ローテーション関数がシークレット内の認証情報を使用しても認証できない可能性があります。コマンド を使用して、 AWSCURRENTおよびAWSPREVIOUSバージョンのシークレットの情報を使用して手動でログインし、認証情報を確認しますAWS CLIget-secret-value。

データベースは **scram-sha-256** を使用してパスワードを暗号化します。

Aurora PostgreSQL バージョン 13 以降のデータベースで、パスワードの暗号化に scram-sha-256 を使用しているが、ローテーション関数が scram-sha-256 をサポートしていない libpq バージョン 9 以前を使用している場合、ローテーション関数はデータベースに接続できません。

scram-sha-256 暗号化を使用するデータベースユーザーを判別するには

- ブログ「SCRAM Authentication in RDS for PostgreSQL 13」(RDS for PostgreSQL 13 での SCRAM 認証) の「Checking for users with non-SCRAM passwords」(SCRAM 以外のパスワードを持つユーザーの確認) を参照してください。

ローテーション関数が使用する **libpq** のバージョンを判別するには

1. Linux ベースのコンピュータの Lambda コンソールで、ローテーション関数に移動し、デプロイバンドルをダウンロードします。zip ファイルを作業ディレクトリに解凍します。
2. コマンドラインの作業ディレクトリで、以下を実行します。

   ```
   readelf -a libpq.so.5 | grep RUNPATH
   ```
3. 文字列 *PostgreSQL-9.4.x*、または 10 未満のメジャーバージョンが表示されている場合、ローテーション関数は scram-sha-256 をサポートしていません。
   - scram-sha-256 をサポートしていないローテーション関数の出力を次に示します。

     ```
     0x000000000000001d (RUNPATH) Library runpath: [/
     local/p4clients/pkgbuild-a1b2c/workspace/build/
     PostgreSQL/PostgreSQL-9.4.x_client_only.123456.0/AL2_x86_64/
     DEV.STD.PTHREAD/build/private/tmp/brazil-path/build.libfarm/lib:/
     local/p4clients/pkgbuild-a1b2c/workspace/src/PostgreSQL/build/
     private/install/lib]
     ```

- scram-sha-256 をサポートしているローテーション関数の出力を次に示します。

  ```
  0x000000000000001d (RUNPATH) Library runpath: [/
  local/p4clients/pkgbuild-a1b2c/workspace/build/
  PostgreSQL/PostgreSQL-10.x_client_only.123456.0/AL2_x86_64/
  DEV.STD.PTHREAD/build/private/tmp/brazil-path/build.libfarm/lib:/
  local/p4clients/pkgbuild-a1b2c/workspace/src/PostgreSQL/build/
  private/install/lib]
  ```

- scram-sha-256 をサポートしているローテーション関数の出力を次に示します。

  ```
  0x000000000000001d (RUNPATH) Library runpath: [/local/
  p4clients/pkgbuild- a1b2c /workspace/build/PostgreSQL/
  PostgreSQL-14.x_client_only. 123456 .0/AL2_x86_64/
  DEV.STD.PTHREAD/build/private/tmp/brazil-path/build.libfarm/lib:/
  local/p4clients/pkgbuild- a1b2c /workspace/src/PostgreSQL/build/
  private/install/lib]
  ```

- scram-sha-256 をサポートしているローテーション関数の出力を次に示します。

  ```
  0x000000000000001d (RUNPATH) Library runpath: [/local/p4clients/
  pkgbuild- a1b2c/workspace/build/PostgreSQL/PostgreSQL-
  14.x_client_only.123456.0/AL2_x86_64/DEV.STD.PTHREAD/build/
  private/tmp/brazil- path/build.libfarm/lib:/local/p4clients/
  pkgbuild- a1b2c/workspace/src/PostgreSQL/build/private/install/
  lib]
  ```

> **Note**
>
> 2021 年 12 月 30 日より前に自動シークレットローテーションを設定した場合、ローテーション関数には scram-sha-256 をサポートしていない前のバージョンの libpq がバンドルされています。scram-sha-256 をサポートするには、ローテーション関数を再作成する必要があります。

データベースには SSL/TLS アクセスが必要です。

SSL/TLS 接続が必要なデータベースを使用しているが、ローテーション関数が暗号化されていない接続を使用する場合、ローテーション関数はデータベースに接続できません。Amazon RDS

(Oracle と Db2 を除く) および Amazon DocumentDB のローテーション関数では、使用可能な場合、データベースへの接続に Secure Sockets Layer (SSL) または Transport Layer Security (TLS) が使用されます。使用できない場合は、暗号化されていない接続が使用されます。

> **Note**
>
> 2021 年 12 月 20 日より前に自動シークレットローテーションを設定した場合は、ローテーション関数が SSL/TLS をサポートしていない前のテンプレートに基づいている可能性があります。SSL/TLS を使用する接続をサポートするには、ローテーション関数を再作成する必要があります。

ローテーション関数がいつ作成されたかを特定するには

1. Secrets Manager コンソール (https://console.aws.amazon.com/secretsmanager/) で、シークレットを開きます。[Rotation configuration] (ローテーション構成) セクションの [Lambda rotation function] (Lambda ローテーション関数) の下に、[Lambda function ARN] (Lambda 関数 ARN) が表示されます (`arn:aws:lambda:`*`aws-region`*`:`*`123456789012`*`:function:`*`SecretsManagerMyRotationFunction`* など)。ARN の末尾から関数名をコピーします (この例では *`SecretsManagerMyRotationFunction`*)。
2. AWS Lambdaコンソール https://console.aws.amazon.com/lambda/ の Functions で、検索ボックスに Lambda 関数名を貼り付け、Enter を選択し、Lambda 関数を選択します。
3. 関数の詳細ページで、[Configuration] (設定) タブの [Tags] (タグ) で、aws:cloudformation:stack-name キーの横にある値をコピーします。
4. AWS CloudFormationコンソール https://console.aws.amazon.com/cloudformation の スタックで、検索ボックスにキー値を貼り付け、Enter を選択します。
5. スタックのリストがフィルタリングされ、Lambda ローテーション関数を作成したスタックだけが表示されます。[Created date] (作成日) 列に、スタックが作成された日付が表示されます。これが、Lambda ローテーション関数が作成された日付です。

エラー:「モジュール 'lambda_function' をインポートできません」

古い (Python 3.7 から新しいバージョンの Python に自動的にアップグレードされた) Lambda 関数を実行している場合に、このエラーが表示されることがあります。このエラーを解決するには、Lambda 関数のバージョンを Python 3.7 に戻してから、the section called “既存のローテーショ

ン関数を Python 3.7 から Python 3.12 にアップグレードする" を実行します。詳細については、「AWS re:Post」の「Secrets Manager Lambda 関数のローテーションが「pg モジュールが見つかりません」というエラーで失敗したのはなぜですか?」を参照してください。

既存のローテーション関数を Python 3.12 にアップグレードする

Secrets Manager は Lambda ローテーション関数の Python 3.12 に移行しています。Python 3.12 を使用する新しいローテーション関数に切り替えるには、デプロイ方法と現在のローテーション Lambda Python バージョンに基づいてアップグレードパスに従う必要があります。次の手順を使用して、Python バージョンと基盤となる依存関係の両方をアップグレードします。

既存のローテーション関数を Python 3.7 から Python 3.12 にアップグレードする

2022 年 11 月よりも前に作成された一部のローテーション関数では、Python 3.7 が使用されていました。AWS SDK for Python は、2023 年 12 月に Python 3.7 のサポートを停止しました。詳細については、AWS SDKs」を参照してください。Python 3.12 を使用する新しいローテーション関数に切り替えるには、既存のローテーション関数にランタイムプロパティを追加するか、ローテーション関数を再作成します。

Python 3.7 を使用する Lambda ローテーション関数を見つけるには

1. にサインインAWS マネジメントコンソールし、https://console.aws.amazon.com/lambda/ で AWS Lambdaコンソールを開きます。
2. [関数] のリストで、**`SecretsManager`** をフィルタリングします。
3. フィルタリングされた関数のリストの [ランタイム] で、Python 3.7 を見つけます。

Python 3.12 にアップグレードするには:

- オプション 1: を使用してローテーション関数を再作成するCloudFormation
- オプション 2: を使用して既存のローテーション関数のランタイムを更新するCloudFormation
- オプション 3:AWS CDKユーザーの場合は、CDK ライブラリをアップグレードする

オプション 1: を使用してローテーション関数を再作成するCloudFormation

Secrets Manager コンソールを使用してローテーションを有効にすると、Secrets Manager は CloudFormationを使用して Lambda ローテーション関数を含む必要なリソースを作成します。コンソールを使用してローテーションを有効にした場合、またはCloudFormationスタックを使用してローテーション関数を作成した場合は、同じCloudFormationスタックを使用してローテーション関数を新しい名前で再作成できます。新しい関数は、より新しいバージョンの Python を使用します。

ローテーション関数を作成したCloudFormationスタックを検索するには

- Lambda 関数の詳細ページの [設定] タブで、[タグ] を選択します。aws:cloudformation:stack-id の横にある ARN を表示します。

  次の例に示すように、スタック名は ARN に埋め込まれます。

  - ARN: `arn:aws:cloudformation:`*`aws-region`*`:123456789012:stack/`**`SecretsManagerRDSMySQLRotationSingleUser5c2-SecretRotationScheduleHostedRotationLambda`**`-3CUDHZMDMBO8/79fc9050-2eef-11ed-`
  - スタック名: **`SecretsManagerRDSMySQLRotationSingleUser5c2-SecretRotationScheduleHostedRotationLambda`**

ローテーション関数を再作成するには (CloudFormation)

1. でCloudFormationスタックを名前で検索し、更新を選択します。

   ルートスタックの更新を推奨するダイアログボックスが表示された場合は、[ルートスタックに移動] を選択し、[更新] を選択します。
2. [スタックの更新] ページで、[テンプレートの準備] から [Application Composer で編集] を選択し、次に [Application Composer でテンプレートを編集] から [Application Composer で編集] を選択します。
3. Application Composer で、次の操作を行います。

   a. テンプレートコードの `SecretRotationScheduleHostedRotationLambda` で、`"functionName": "SecretsManagerTestRotationRDS"` の値を新しい関数名 (JSON の **`"functionName": "SecretsManagerTestRotationRDSupdated"`** など) に置き換えます

   b. [テンプレートの更新] を選択します。

   c. [CloudFormationに進む] ダイアログボックスで、[確認してCloudFormationに進む] を選択します。
4. CloudFormationスタックワークフローを続行し、送信を選択します。

オプション 2: を使用して既存のローテーション関数のランタイムを更新するCloudFormation

Secrets Manager コンソールを使用してローテーションを有効にすると、Secrets Manager は CloudFormationを使用して Lambda ローテーション関数を含む必要なリソースを作成します。コ

ンソールを使用してローテーションを有効にした場合、またはCloudFormationスタックを使用してローテーション関数を作成した場合は、同じCloudFormationスタックを使用してローテーション関数のランタイムを更新できます。

ローテーション関数を作成したCloudFormationスタックを検索するには

- Lambda 関数の詳細ページの [設定] タブで、[タグ] を選択します。aws:cloudformation:stack-id の横にある ARN を表示します。

  次の例に示すように、スタック名は ARN に埋め込まれます。

  - ARN: `arn:aws:cloudformation:`*`aws-region`*`:123456789012:stack/`**`SecretsManagerRDSMySQLRotationSingleUser5c2-SecretRotationScheduleHostedRotationLambda`**`-3CUDHZMDMBO8/79fc9050-2eef-11ed-`
  - スタック名: **`SecretsManagerRDSMySQLRotationSingleUser5c2-SecretRotationScheduleHostedRotationLambda`**

ローテーション関数のランタイムを更新するには (CloudFormation)

1. でCloudFormation、名前でスタックを検索し、更新を選択します。

   ルートスタックの更新を推奨するダイアログボックスが表示された場合は、[ルートスタックに移動] を選択し、[更新] を選択します。
2. [スタックの更新] ページで、[テンプレートの準備] から [Application Composer で編集] を選択し、次に [Application Composer でテンプレートを編集] から [Application Composer で編集] を選択します。
3. Application Composer で、次の操作を行います。

   a. テンプレート JSON で、`SecretRotationScheduleHostedRotationLambda` の `Properties` の `Parameters` で **`"runtime": "python3.12"`** を追加します。

   b. [テンプレートの更新] を選択します。

   c. [CloudFormationに進む] ダイアログボックスで、[確認してCloudFormationに進む] を選択します。
4. CloudFormationスタックワークフローを続行し、送信を選択します。

オプション 3:AWS CDKユーザーの場合は、CDK ライブラリをアップグレードする

バージョン v2.258.0 AWS CDKより前の を使用してシークレットのローテーションを設定した場合は、v2.258.0 以降にアップグレードすることで Lambda 関数を Python 3.12 AWS CDKに更新できます。詳細については、「AWS Cloud Development Kit (AWS CDK) v2 デベロッパーガイド」を参照してください。

既存のローテーション関数を Python 3.9 以降から Python 3.12 にアップグレードする

古いバージョンの Python を使用している Lambda ローテーション関数を見つけるには

1. にサインインAWS マネジメントコンソールし、https://console.aws.amazon.com/lambda/ で AWS Lambdaコンソールを開きます。
2. [関数] のリストで、**`SecretsManager`** をフィルタリングします。
3. フィルタリングされた関数のリストのランタイムで、 より前の Python バージョンを探しますPython 3.12。

デプロイ方法によるパスの更新

このリストで識別される Lambda ローテーション関数は、Secrets Manager コンソール、AWS Serverless Application Repositoryアプリケーション、またはCloudFormation変換を使用してデプロイできます。これらのデプロイ戦略ごとに個別の更新パスがあります。

関数のデプロイ方法に応じて、次のいずれかの手順を使用して Lambda ローテーション関数を更新します。

AWS Secrets Manager console-deployed functions

既存の Lambda 関数の依存関係を手動で更新することはできないため、新しい Lambda 関数は AWS Secrets Managerコンソールを介してデプロイする必要があります。

AWS Secrets Managerコンソールにデプロイされた関数をアップグレードするには、次の手順に従います。

1. https://console.aws.amazon.com/secretsmanager/ から Secrets Manager コンソールを開きます。
2. AWS Secrets Manager で、[シークレット] を選択します。更新する Lambda 関数を使用するシークレットを選択します。
3. [ローテーション] タブに移動し、[ローテーション設定の更新] オプションを選択します。

4. [ローテーション関数] で [新しい関数を作成する] を選択し、Lambda ローテーション関数の名前を入力します。

   a. (オプション) 更新が完了したら、更新された Lambda 関数をテストして、正常に動作することを確認できます。[ローテーション] タブで、[シークレットの即時ローテーション] を選択して、即時ローテーションを開始します。

   b. (オプション) 関数ログと、実行時に Amazon CloudWatch で使用される Python バージョンを表示できます。詳細については、「AWS Lambda開発者ガイド」の「Lambda 関数の CloudWatch Logs を表示する」を参照してください。

5. 新しいローテーション関数を設定したら、古いローテーション関数を削除できます。

AWS Serverless Application Repository deployments

次の手順は、AWS Serverless Application Repositoryデプロイをアップグレードする方法を示しています。を介してデプロイされた Lambda 関数AWS Serverless Application Repositoryには、関数が属する Lambda アプリケーションへのリンクThis function belongs to an application. Click here to manage it.を含むバナーがあります。

> **Important**
>
> AWS Serverless Application Repositoryの可用性はAWS リージョン、 によって異なります。

デプロイAWS Serverless Application Repositoryされた関数を更新するには、次の手順に従います。

1. https://console.aws.amazon.com/lambda/ でAWS Lambdaコンソールを開きます。
2. 更新する必要がある Lambda 関数の [設定] タブに移動します。

   - デプロイされたAWS Serverless Application Repositoryアプリケーションを更新するときは、関数に関する次の情報が必要になります。この情報は、Lambda コンソールで確認できます。

     - Lambda アプリケーションの名前
       - Lambda アプリケーション名は、バナーのリンクを使用して確認できます。たとえば、バナーには次の `serverlessrepo-SecretsManagerRedshiftRotationSingleUser` が表示さ

れます。この例では、名前は SecretsManagerRedshiftRotationSingleUser です。

- Lambda ローテーション関数名
- Secrets Manager エンドポイント
  - エンドポイントは、SECRETS_MANAGER_ENDPOINT 変数に割り当てられた [設定] と [環境変数] タブにあります。

3. Python をアップグレードするには、サーバーレスアプリケーションのセマンティックバージョンを更新する必要があります。「AWS Serverless Application Repositoryデベロッパーガイド」の「アプリケーションの更新」を参照してください。

Custom Lambda rotation functions

カスタム Lambda ローテーション関数を作成した場合は、これらの関数の各パッケージの依存関係とランタイムをアップグレードする必要があります。詳細については、「Lambda 関数ランタイムを最新バージョンにアップグレードする」を参照してください。

AWS::SecretsManager-2024-09-16 transform macro

Lambda 関数がこの変換を通じてデプロイされている場合、既存のテンプレートを使用してスタックを更新すると、更新された Lambda ランタイムを使用できます。

既存のテンプレートを使用してCloudFormationスタックを更新するには、次の手順に従います。

1. https://console.aws.amazon.com/cloudformation でCloudFormationコンソールを開きます。
2. [スタック] ページで、更新する実行中のスタックを選択します。
3. スタックの詳細ペインで、[更新] を選択します。
4. [テンプレートの更新方法を選択する] で、[直接更新] を選択します。
5. [テンプレートの指定] ページで、[既存のテンプレートを使用する] を選択します。
6. 他のすべてのオプションをデフォルト値のままにして、[スタックの更新] を選択します。

スタックの更新で問題が発生した場合は、「CloudFormationユーザーガイド」の「スタックが失敗する原因を判断する」を参照してください。

AWS::SecretsManager-2020-07-23 transform macro

AWS::SecretsManager-2020-07-23 を使用している場合は、新しい変換バージョンに移行することをお勧めします。詳細については、 AWSセキュリティブログのAWS Secrets Manager「変換の拡張バージョンの導入:AWS:SecretsManager-2024-09-16」を参照し

てください。AWS::SecretsManager-2020-07-23 を引き続き使用する場合、ランタイムバージョンと Lambda 関数コードアーティファクトの間に不一致エラーが発生する可能性があります。詳細については、「CloudFormationテンプレートリファレンス」の「AWS「::SecretsManager::RotationSchedule HostedRotationLambda」を参照してください。

スタックの更新で問題が発生した場合は、「CloudFormationユーザーガイド」の「スタックが失敗する原因を判断する」を参照してください。

Python アップグレードの検証

Python のアップグレードを確認するには、Lambda コンソール (https://console.aws.amazon.com/lambda/) を開き、[関数] ページにアクセスします。更新した関数を選択します。[コードソース] セクションで、ディレクトリに含まれているファイルを確認し、Python .so ファイルがバージョン 3.12 であることを確認します。

AWS Lambdaシークレットのローテーションが**PutSecretValue**失敗しました

Secrets Manager で引き受けたロールまたはクロスアカウントローテーションを使用していて、CloudTrail で次のメッセージを含む RotationFailed イベントが見つかった場合: 「Pending secret version *VERSION_ID* for Secret *SECRET_ARN* was not created by Lambda *LAMBDA_ARN.*」。AWSPENDING ステージングラベルを削除してローテーションを再起動し、RotationToken パラメータを使用するように Lambda 関数を更新する必要があります。

Lambda ローテーション関数を更新して **RotationToken** を含める

1. Lambda 関数コードをダウンロードする
   - Lambda のコンソールを開く
   - ナビゲーションペインで、[関数] を選択する
   - 関数名の Lambda シークレットローテーション関数を選択する
   - ダウンロードでは、関数コード .zip、AWS SAMファイル、両方のいずれかを選択します。
   - [OK] を選択して、ローカルマシンに関数を保存します。
2. Lambda_handler を編集する

   クロスアカウントローテーションの create_secret ステップに rotation_token パラメータを含めます:

```
def lambda_handler(event, context):
    """Secrets Manager Rotation Template

    This is a template for creating an AWS Secrets Manager rotation lambda

    Args:
        event (dict): Lambda dictionary of event parameters. These keys must
 include the following:
            - SecretId: The secret ARN or identifier
            - ClientRequestToken: The ClientRequestToken of the secret version
            - Step: The rotation step (one of createSecret, setSecret, testSecret,
 or finishSecret)
            - RotationToken: the rotation token to put as parameter for
 PutSecretValue call

        context (LambdaContext): The Lambda runtime information

    Raises:
        ResourceNotFoundException: If the secret with the specified arn and stage
 does not exist

        ValueError: If the secret is not properly configured for rotation

        KeyError: If the event parameters do not contain the expected keys

    """
    arn = event['SecretId']
    token = event['ClientRequestToken']
    step = event['Step']
    # Add the rotation token
    rotation_token = event['RotationToken']

    # Setup the client
    service_client = boto3.client('secretsmanager',
 endpoint_url=os.environ['SECRETS_MANAGER_ENDPOINT'])

    # Make sure the version is staged correctly
    metadata = service_client.describe_secret(SecretId=arn)
    if not metadata['RotationEnabled']:
        logger.error("Secret %s is not enabled for rotation" % arn)
        raise ValueError("Secret %s is not enabled for rotation" % arn)
    versions = metadata['VersionIdsToStages']
    if token not in versions:
```

```
        logger.error("Secret version %s has no stage for rotation of secret %s." %
 (token, arn))
        raise ValueError("Secret version %s has no stage for rotation of secret
 %s." % (token, arn))
    if "AWSCURRENT" in versions[token]:
        logger.info("Secret version %s already set as AWSCURRENT for secret %s." %
 (token, arn))
        return
    elif "AWSPENDING" not in versions[token]:
        logger.error("Secret version %s not set as AWSPENDING for rotation of
 secret %s." % (token, arn))
        raise ValueError("Secret version %s not set as AWSPENDING for rotation of
 secret %s." % (token, arn))
    # Use rotation_token
    if step == "createSecret":
        create_secret(service_client, arn, token, rotation_token)

    elif step == "setSecret":
        set_secret(service_client, arn, token)

    elif step == "testSecret":
        test_secret(service_client, arn, token)

    elif step == "finishSecret":
        finish_secret(service_client, arn, token)

    else:
        raise ValueError("Invalid step parameter")
```

3. create_secret コードの編集

   rotation_token パラメータを受け入れて使用するように create_secret 関数を改訂します:

```
# Add rotation_token to the function
def create_secret(service_client, arn, token, rotation_token):
"""Create the secret

This method first checks for the existence of a secret for the passed in token. If
 one does not exist, it will generate a
new secret and put it with the passed in token.
```

```
Args:
service_client (client): The secrets manager service client

arn (string): The secret ARN or other identifier

token (string): The ClientRequestToken associated with the secret version

rotation_token (string): the rotation token to put as parameter for PutSecretValue
 call

Raises:
ResourceNotFoundException: If the secret with the specified arn and stage does not
 exist

"""
# Make sure the current secret exists
service_client.get_secret_value(SecretId=arn, VersionStage="AWSCURRENT")

# Now try to get the secret version, if that fails, put a new secret
try:
service_client.get_secret_value(SecretId=arn, VersionId=token,
 VersionStage="AWSPENDING")
logger.info("createSecret: Successfully retrieved secret for %s." % arn)
except service_client.exceptions.ResourceNotFoundException:
# Get exclude characters from environment variable
exclude_characters = os.environ['EXCLUDE_CHARACTERS'] if 'EXCLUDE_CHARACTERS' in
 os.environ else '/@"\'\\'
# Generate a random password
passwd = service_client.get_random_password(ExcludeCharacters=exclude_characters)

# Put the secret, using rotation_token
service_client.put_secret_value(SecretId=arn, ClientRequestToken=token,
 SecretString=passwd['RandomPassword'], VersionStages=['AWSPENDING'],
 RotationToken=rotation_token)
logger.info("createSecret: Successfully put secret for ARN %s and version %s." %
 (arn, token))
```

4. 更新された Lambda 関数コードをアップロードする

   Lambda 関数コードを更新したら、アップロードしてシークレットをローテーションします。

エラー: *<a rotation>* ステップ中に Lambda *<arn>* を実行するときのエラー

CreateSecret と SetSecret の間など、Lambda 関数がセットのループで停止して、シークレットローテーションが断続的に失敗する場合、問題は同時実行設定に関連している可能性があります。

同時実行のトラブルシューティング手順

> **Warning**
>
> プロビジョニングされた同時実行パラメータを 10 未満の値に設定すると、Lambda 関数の実行スレッドが不十分であるためにスロットリングが発生する可能性があります。詳細については、「AWS LambdaAWS Lambdaデベロッパーガイド」の「予約済み同時実行数とプロビジョニングされた同時実行数について」を参照してください。

1. Lambda の同時実行設定を確認して調整します:
   - `reserved_concurrent_executions` の値がが低すぎないことを確認する (例: 1)
   - 予約された同時実行を使用する場合は、少なくとも 10 に設定する
   - 柔軟性を高めるために、予約されていない同時実行の使用を検討する
2. プロビジョニングされた同時実行の場合:
   - プロビジョニングされた同時実行パラメータを明示的に設定しないでください (Terraform など)。
   - 設定する必要がある場合は、10 以上の値を使用します。
   - 選択した値がユースケースで機能することを徹底的にテストします。
3. 同時実行をモニタリングして調整します:
   - 次の式を使用して同時実行数を計算します。同時実行数 = (1 秒あたりの平均リクエスト数) * (秒単位の平均リクエスト時間)。詳細については、「予約済み同時実行数の見積もり」(Estimating reserved concurrency) を参照してください。
   - ローテーション中に値を観察して記録し、適切な同時実行設定を決定します。
   - 低い同時実行値を設定するときは注意してください。使用可能な実行スレッドが足りない場合、スロットリングが発生する可能性があります。

Lambda 同時実行の設定の詳細については、「AWS Lambdaデベロッパーガイド」の「予約済み同時実行の設定」および「プロビジョニング済み同時実行の設定」を参照してください。

ローテーションスケジュール

Secrets Manager は、設定したローテーションウィンドウ中にスケジュールに従ってシークレットをローテーションします。スケジュールとウィンドウを設定するには、ウィンドウ期間とともに [cron()] または [rate()] 式を使用します。Secrets Manager は、ローテーションウィンドウ中の任意の時刻にシークレットをローテーションします。シークレットは 4 時間ごとに、1 時間単位でローテーションさせることができます。

ローテーションの有効化については、以下を参照してください。

- the section called “マネージドローテーション”
- the section called “データベースシークレットの自動ローテーション (コンソール)”
- the section called “非データベースシークレットの自動ローテーション (コンソール)”

Secrets Manager のローテーションスケジュールでは、UTC タイムゾーンが使用されます。

ローテーションウィンドウ

Secrets Manager のローテーションウィンドウは、メンテナンスウィンドウに似ています。シークレットをローテーションするタイミングをローテーションウィンドウに設定し、Secrets Manager はローテーションウィンドウ中にシークレットをローテーションします。

Secrets Manager のローテーションウィンドウは、常に正時に開始されます。日数で `rate( )` 式を使用するローテーションスケジュールの場合、ローテーションウィンドウは午前 0 時に開始されます。`cron( )` 式を使用して、ローテーションウィンドウの開始時刻を設定できます。例については、the section called “cron 式” を参照してください。

デフォルトでは、ローテーションウィンドウは、時間単位のローテーションスケジュールの場合は 1 時間後に閉じ、日単位のローテーションスケジュールの場合は 1 日の最後に閉じます。

[ウィンドウ期間] を設定して、ローテーションウィンドウの長さを変更できます。ローテーションウィンドウは最低 1 時間で設定できます。ローテーションウィンドウが次のローテーションウィンドウに重ならないようにしてください。つまり、時間単位のローテーションスケジュールの場合、ローテーションウィンドウがローテーション間の時間数以下であることを確認します。日単位のローテーションスケジュールの場合、開始時刻とウィンドウ期間の合計時間が 24 時間以下であることを確認します。

rate 式

Secrets Manager の Rate 式は、次のような形式です。*Value* は正の整数で、*Unit* は hour、hours、day、または days にできます。

```
rate(Value Unit)
```

シークレットが 4 時間ごとにローテーションされるように設定できます。最大ローテーション期間は 999 日です。例:

- `rate(4 hours)` は、シークレットが 4 時間ごとにローテーションされることを意味します。
- `rate(1 day)` は、シークレットが毎日ローテーションされることを意味します。
- `rate(10 days)` は、シークレットが 10 日ごとにローテーションされることを意味します。

cron 式

Secrets Manager の Cron 式の形式は次のようになります。

```
cron(Minutes Hours Day-of-month Month Day-of-week Year)
```

時間の増分を含む cron 式は、毎日リセットされます。例えば、`cron(0 4/12 * * ? *)` は午前 4 時、および午後 4 時、次いで翌日の午前 4 時、および午後 4 時を意味します。Secrets Manager のローテーションスケジュールでは、UTC タイムゾーンが使用されます。

スケジュールの例	式
8 時間ごと、午前 0 時から開始。	`cron(0 /8 * * ? *)`
8 時間ごと、午前 8 時から開始。	`cron(0 8/8 * * ? *)`
10 時間ごと、午前 2 時から開始。 ローテーションウィンドウは 2:00、12:00、および 22:00、次いで翌日の 2:00、12:00、および 22:00 に開始されます。	`cron(0 2/10 * * ? *)`
毎日午前 10:00。	`cron(0 10 * * ? *)`

スケジュールの例	式
毎週土曜日の午後 6:00。	`cron(0 18 ? * SAT *)`
毎月 1 日の午前 8:00。	`cron(0 8 1 * ? *)`
3 か月ごとに第 1 日曜日の午前 1:00。	`cron(0 1 ? 1/3 SUN#1 *)`
毎月最終日の午後 5:00。	`cron(0 17 L * ? *)`
月曜日から金曜日までの午前 8:00。	`cron(0 8 ? * MON-FRI *)`
毎月 1 日と 15 日の午後 4:00。	`cron(0 16 1,15 * ? *)`
毎月第 1 日曜日の午前 0:00。	`cron(0 0 ? * SUN#1 *)`
1 月に開始され、最初の月曜日の午前 0 時から 11 か月ごと。	`cron(0 0 ? 1/11 2#1 *)`

Secrets Manager の cron 式要件

Secrets Manager では、cron 式に使用できる設定値にはいくつかの制限があります。Secrets Manager の cron 式では、分フィールドに 0 が必要です。これは、Secrets Manager のローテーションウィンドウが正時に開始されるためです。年フィールドには、* が必要です。これは、Secrets Manager では 1 年以上離れているローテーションスケジュールがサポートされていないためです。次の表に、使用できるオプションを示します。

フィールド	値	ワイルドカード
分	0 を指定	なし
時間	0 ~ 23	/ (フォワードスラッシュ) を使用して増分を指定します。例えば、2/10 は午前 2 時から 10 時間ごとを意味します。シークレットが 4 時間ごとにローテーションされるように設定できます。

フィールド	値	ワイルドカード
日	1 ~ 31	, (カンマ) を使用して追加の値を含めます。例えば、1,15 は月の 1 日と 15 日を意味します。 – (ダッシュ) を使用して範囲を指定します。例えば、1–15 は月の 1 日から 15 日までの日を意味します。 * (アスタリスク) を使用してフィールド内のすべての値を含めます。例えば、* は月のすべての日を意味します。 [?] (疑問符) のワイルドカードは、任意を意味します。Cron 式の Day-of-month フィールドと Day-of-week フィールドを同時に指定することはできません。一方のフィールドに値を指定する場合、もう一方のフィールドで [?] (疑問符) を使用する必要があります。 / (フォワードスラッシュ) を使用して増分を指定します。例えば、1/2 は 1 日目から 2 日おき、つまり 1 日目、3 日目、5 日目などを意味します。 L を使用して月の最終日を指定します。

フィールド	値	ワイルドカード
		DAYL を使用して、月の最終曜日を指定します。例えば、SUNL は月の最終日曜日を意味します。
月	1 ~ 12 または JAN ~ DEC	, (カンマ) を使用して追加の値を含めます。例えば、JAN,APR,JUL,OCT は 1 月、4 月、7 月、および 10 月を意味します。 – (ダッシュ) を使用して範囲を指定します。例えば、1–3 とは1 年の 1 か月目から 3 か月目までを意味します。 * (アスタリスク) を使用してフィールド内のすべての値を含めます。例えば、* はすべての月を意味します。 / (フォワードスラッシュ) を使用して増分を指定します。例えば、1/3 は、1 か月目から 3 か月おき、つまり 1 か月目、4 か月目、7 か月目、10 か月目などを意味します。

フィールド	値	ワイルドカード
曜日	1 ~ 7 または SUN ~ SAT	# を使用して月内の曜日を指定します。例えば、TUE#3 は月の第 3 火曜日を意味します。 , (カンマ) を使用して追加の値を含めます。例えば、1,4 は 1 日目の曜日と 4 日目の曜日を意味します。 – (ダッシュ) を使用して範囲を指定します。例えば、1–4 は 1 日目から 4 日目までの曜日を意味します。 * (アスタリスク) を使用してフィールド内のすべての値を含めます。例えば、* はすべての曜日を意味します。 [?] (疑問符) のワイルドカードは、任意を意味します。Cron 式の Day-of-month フィールドと Day-of-week フィールドを同時に指定することはできません。一方のフィールドに値を指定する場合、もう一方のフィールドで [?] (疑問符) を使用する必要があります。 / (フォワードスラッシュ) を使用して増分を指定します。例えば、1/2 は 1 日目の曜日から 2 日おき、つまり 1 日

フィールド	値	ワイルドカード
		目、3 日目、5 日目、7 日目などの曜日を意味します。 L を使用して最終曜日を指定します。
年	* を指定してください	なし

シー AWS Secrets Manager クレットをすぐにローテーションする

ローテーションできるのは、ローテーションが設定されているシークレットのみです。シークレットにローテーションが設定されているかどうかを確認するには、コンソールでシークレットを表示し、[Rotation configuration] (ローテーション設定) セクションまでスクロールします。[Rotation status] (ローテーションステータス) が [Enabled] (有効) の場合、シークレットにローテーションが設定されます。そうでない場合は、「シークレットのローテーション」を参照してください。

すぐにシークレットをローテーションするには (コンソール)

1. https://console.aws.amazon.com/secretsmanager/ から Secrets Manager コンソールを開きます。
2. シークレットを選択します。
3. [Secret Details] (シークレットの詳細) ページの、[Rotation configuration] (ローテーション設定) で、[Rotate secret immediately] (すぐにシークレットをローテーションさせる) をクリックします。
4. [Rotate secret] (シークレットのローテーション) ダイアログボックスで、[Rotate] (ローテーション) をクリックします。

AWS CLI

Exampleすぐにシークレットをローテーションする

次の rotate-secret の例では、すぐにローテーションが開始されます。シークレットのローテーションは、すでに設定されている必要があります。

```
$ aws secretsmanager rotate-secret \
```

```
    --secret-id MyTestSecret
```

ローテーションされていないシークレットを検索する

AWS Config を使用してシークレットを評価し、標準に従ってシークレットがローテーションしているかどうかを確認できます。 AWS Config ルールを使用して、シークレットの内部セキュリティおよびコンプライアンス要件を定義します。その後 AWS Config 、 はルールに準拠していないシークレットを識別できます。また、シークレットメタデータ、ローテーション設定、シークレット暗号化に使用される KMS キー、Lambda ローテーション関数、シークレットに関連付けられたタグの変更を追跡することもできます。

複数の AWS アカウント と組織 AWS リージョン 内にシークレットがある場合は、その設定とコンプライアンスデータを集約できます。詳細については、「Multi-account Multi-Region data aggregation」を参照してください。

シークレットがローテーションしているかどうかを評価するには

1. AWS Config ルールを使用してリソースを評価する手順に従い、次のルールから選択します。
   - secretsmanager-rotation-enabled-check — Secrets Manager に保存されているシークレットに対してローテーションが設定されているかどうかを確認します。
   - secretsmanager-scheduled-rotation-success-check – 最後に成功したローテーションが、設定されたローテーション頻度の範囲内であるかどうかをチェックします。チェックの最小頻度は日次です。
   - secretsmanager-secret-periodic-rotation — 指定した日数内にシークレットがローテーションされたかどうかを確認します。
2. 必要に応じて、シークレットが準拠していない場合に通知する AWS Config ように を設定します。詳細については、「 が Amazon SNS トピック AWS Config に送信する通知」を参照してください。

Secrets Manager で自動ローテーションをキャンセルする

シークレットの自動ローテーションを構成している場合に、ローテーションを停止するには、ローテーションをキャンセルできます。

自動ローテーションをキャンセルするには

1. https://console.aws.amazon.com/secretsmanager/ から Secrets Manager コンソールを開きます。
2. シークレットを選択します。
3. シークレットの詳細ページで、[ローテーション設定] から [ローテーションの編集] を選択します。
4. [ローテーション設定の編集] ダイアログボックスで、[自動ローテーション] をオフにして、[保存] を選択します。

   Secrets Manager はローテーション設定情報を保持するため、将来ローテーションを再びオンにする場合に、後から使用できます。

AWS Secrets Manager 他の AWS サービスによって管理されるシークレット

多くの AWS サービスは、 にシークレットを保存して使用します AWS Secrets Manager。これらのシークレットは、マネージドシークレットである場合もあります。これは、シークレットを作成したサービスが、シークレットの管理をサポートしていることを意味します。例えば、いくつかのマネージドシークレットにはマネージドローテーションが含まれているため、ローテーションを自分で設定する必要はありません。また、管理サービスでは、復旧期間なしでシークレットを更新または削除することが制限されています。管理サービスはシークレットに依存するため、この制限は機能停止を防ぐのに役立ちます。

> **Note**
>
> マネージドシークレットは、それらを管理する AWS サービスによってのみ作成できます。

マネージドシークレットは、識別しやすいように管理サービス ID を含む命名規則を使用しています。

```
Secret name: ServiceID!MySecret
Secret ARN : arn:aws:us-east-1:ServiceID!MySecret-a1b2c3
```

シークレットを管理するサービスの ID

- `appflow` – the section called “Amazon AppFlow”
- `databrew` – the section called “AWS Glue DataBrew”
- `datasync` – the section called “AWS DataSync”
- `directconnect` – the section called “Direct Connect”
- `ecs-sc` – the section called “Amazon Elastic Container Service”
- `events` – the section called “Amazon EventBridge”
- `marketplace-deployment` – the section called “AWS Marketplace”
- `opsworks-cm` – the section called “AWS OpsWorks for Chef Automate”
- `pcs` – the section called “AWS 並列コンピューティングサービス”
- `rds` – the section called “Amazon RDS”

- `redshift` – the section called “Amazon Redshift”
- `sqlworkbench` – the section called “Amazon Redshift Query Editor V2”

他の AWS のサービスによって管理されているシークレットを検索するには、「マネージドシークレットの検索」を参照してください。

AWS のサービス シー AWS Secrets Manager クレットを使用する

以下の各 AWS のサービス が Secrets Manager とどのように統合されるかについて説明します。

- AWS App Runner での の使用方法 AWS Secrets Manager
- AWS App2Container が を使用する方法 AWS Secrets Manager
- AWS AppConfig での の使用方法 AWS Secrets Manager
- Amazon AppFlow が を使用する方法 AWS Secrets Manager
- AWS AppSync での の使用方法 AWS Secrets Manager
- Amazon Athena が を使用する方法 AWS Secrets Manager
- Amazon Aurora が を使用する方法 AWS Secrets Manager
- AWS CodeBuild での の使用方法 AWS Secrets Manager
- Amazon Data Firehose が を使用する方法 AWS Secrets Manager
- AWS DataSync での の使用方法 AWS Secrets Manager
- Amazon DataZone が を使用する方法 AWS Secrets Manager
- AWS Direct Connect での の使用方法 AWS Secrets Manager
- AWS Directory Service での の使用方法 AWS Secrets Manager
- Amazon DocumentDB (MongoDB 互換) が を使用する方法 AWS Secrets Manager
- AWS Elastic Beanstalk での の使用方法 AWS Secrets Manager
- Amazon Elastic Container Registry が を使用する方法 AWS Secrets Manager
- Amazon Elastic Container Service
- Amazon ElastiCache が を使用する方法 AWS Secrets Manager
- AWS Elemental Live での の使用方法 AWS Secrets Manager
- AWS Elemental MediaConnect での の使用方法 AWS Secrets Manager
- AWS Elemental MediaConvert での の使用方法 AWS Secrets Manager

- AWS Elemental MediaLive での の使用方法 AWS Secrets Manager
- AWS Elemental MediaPackage での の使用方法 AWS Secrets Manager
- AWS Elemental MediaTailor での の使用方法 AWS Secrets Manager
- Amazon EMR で Secrets Manager を使用する方法
- Amazon EventBridge が を使用する方法 AWS Secrets Manager
- Amazon FSx が AWS Secrets Manager シークレットを使用する方法
- AWS Glue DataBrew での の使用方法 AWS Secrets Manager
- AWS Glue Studio が を使用する方法 AWS Secrets Manager
- AWS IoT SiteWise での の使用方法 AWS Secrets Manager
- Amazon Kendra が を使用する方法 AWS Secrets Manager
- Amazon Kinesis Video Streams が を使用する方法 AWS Secrets Manager
- AWS Launch Wizard での の使用方法 AWS Secrets Manager
- Amazon Lookout for Metrics の使用方法 AWS Secrets Manager
- Amazon Managed Grafana が を使用する方法 AWS Secrets Manager
- AWS Managed Services での の使用方法 AWS Secrets Manager
- Amazon Managed Streaming for Apache Kafka の使用方法 AWS Secrets Manager
- Amazon Managed Workflows for Apache Airflow が を使用する方法 AWS Secrets Manager
- AWS Marketplace
- AWS Migration Hub での の使用方法 AWS Secrets Manager
- AWS Parallel Computing Service が を使用する方法 AWS Secrets Manager
- AWS ParallelCluster での の使用方法 AWS Secrets Manager
- Amazon Q で Secrets Manager を使用する方法
- Amazon OpenSearch Ingestion が Secrets Manager を使用する方法
- AWS OpsWorks for Chef Automate での の使用方法 AWS Secrets Manager
- Amazon Quick が を使用する方法 AWS Secrets Manager
- Amazon RDS が を使用する方法 AWS Secrets Manager
- Amazon Redshift が を使用する方法 AWS Secrets Manager
- Amazon Redshift Query Editor V2
- Amazon SageMaker AI が を使用する方法 AWS Secrets Manager

- AWS Schema Conversion Tool での の使用方法 AWS Secrets Manager
- Amazon Timestream for InfluxDB が を使用する方法 AWS Secrets Manager
- AWS Toolkit for JetBrains での の使用方法 AWS Secrets Manager
- がシー AWS Secrets Manager クレット AWS Transfer Family を使用する方法
- がシー AWS Secrets Manager クレット AWS Wickrを使用する方法

AWS App Runner での の使用方法 AWS Secrets Manager

AWS App Runner は、ソースコードまたはコンテナイメージから AWS クラウド内のスケーラブルで安全なウェブアプリケーションに直接デプロイするための、高速でシンプルで費用対効果の高い方法を提供する AWS サービスです。新しいテクノロジーを学習したり、使用するコンピューティングサービスを決定したり、 AWS リソースをプロビジョニングして設定する方法を知っている必要はありません。

App Runner を使用すると、サービスを作成、もしくはその設定を更新した際に、シークレットや設定内容をサービスの環境変数として参照できるようになります。詳細については、「AWS App Runner 開発者ガイド」の「Referencing environment variables」(環境変数の参照) と「Managing environment variables」(環境変数の管理) を参照してください。

AWS App2Container が を使用する方法 AWS Secrets Manager

AWS App2Container は、オンプレミスのデータセンターまたは仮想マシンで実行されるアプリケーションをリフトアンドシフトし、Amazon ECS、Amazon EKS、または によって管理されるコンテナで実行できるようにするコマンドラインツールです AWS App Runner。

App2Container は Secrets Manager を使用して、リモートコマンドを実行するためにワーカーマシンをアプリケーションサーバーに接続するための資格を管理します。詳細については、AWS App2Container ユーザーガイドの「App2Container のシークレットを管理する」を参照してください。 AWS App2Container

AWS AppConfig での の使用方法 AWS Secrets Manager

AWS AppConfig は、アプリケーション設定の作成、管理、迅速なデプロイ AWS Systems Manager に使用できる の機能です。設定には、Secrets Manager に保存されている認証情報データまたはその他の機密情報を含めることができます。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。詳細については、「AWS AppConfig ユーザーガイド」の「自由形式の設定プロファイルの作成」を参照してください。自動ローテーションが有効

になっているシークレットを AWS AppConfig がどのように処理するかについては、AWS AppConfig 「 ユーザーガイド」の「Secrets Manager のキーローテーション」を参照してください。

Amazon AppFlow が を使用する方法 AWS Secrets Manager

Amazon AppFlow は、Salesforce、 AWS のサービス、Amazon Simple Storage Service (Amazon S3)、Amazon Redshift などの Software as a Service (SaaS) アプリケーション間で安全にデータを交換できるフルマネージド型の統合サービスです。

Amazon AppFlow では、SaaS アプリケーションをソースまたはデスティネーションとして設定すると、接続が作成されます。これには、認証トークン、ユーザー名、およびパスワードなど、SaaS アプリケーションへの接続に必要な情報が含まれます。Amazon AppFlow は、プレフィックス `appflow` を持つ Secrets Manager マネージドシークレットに接続データを保存します。シークレットを保存する費用は、Amazon AppFlow の料金に含まれています。詳細については、「Amazon AppFlow ユーザーガイド」の「Data protection in Amazon AppFlow」(Amazon AppFlow でのデータ保護) を参照してください。

AWS AppSync での の使用方法 AWS Secrets Manager

AWS AppSync は、アプリケーション開発者が Amazon DynamoDB、、HTTP APIs などの複数のソースからのデータを組み合わせるための AWS Lambda堅牢でスケーラブルな GraphQL インターフェイスを提供します。

AWS AppSync は Secrets Manager シークレットの認証情報を使用して Amazon RDS と Aurora に接続します。詳細については、「AWS AppSync デベロッパーガイド」の「Tutorial: Aurora Serverless」(チュートリアル: Aurora Serverless) を参照してください。

Amazon Athena が を使用する方法 AWS Secrets Manager

Amazon Athena は、標準的な SQL を使用して Amazon Simple Storage Service (Amazon S3) 内のデータを直接分析することを容易にする、インタラクティブなクエリサービスです。

Amazon Athena データソースコネクタは、Secrets Manager シークレットと Athena フェデレーションクエリ機能を使用して、データをクエリできます。詳細については、「Amazon Athena ユーザーガイド」の「Amazon Athena 横串検索の使用」を参照してください。

Amazon Aurora が を使用する方法 AWS Secrets Manager

Amazon Aurora はフルマネージド型のリレーショナルデータベースエンジンで、MySQL および PostgreSQL との互換性があります。

Aurora のマスターユーザー認証情報を管理するために、Aurora はマネージドシークレットを作成できます。そのシークレットの料金が請求されます。Aurora は、これらの認証情報のローテーションも管理します。詳細については、「Amazon Aurora ユーザーガイド」の「Amazon Aurora および AWS Secrets Managerでのパスワード管理」を参照してください。

その他の Aurora 認証情報については、「シークレットを作成する」を参照してください。

Amazon RDS の Data API を呼び出すと、Secrets Manager を使用してデータベースの認証情報を渡すことができます。詳細については、「Amazon Aurora ユーザーガイド」の「Aurora Serverless のデータ API の使用」を参照してください。

Amazon RDS クエリエディタを使用してデータベースに接続する場合、データベースの認証情報を Secrets Manager に保存できます。詳細については、「Amazon RDS ユーザーガイド」の「クエリエディタの使用」を参照してください。

AWS CodeBuild での の使用方法 AWS Secrets Manager

AWS CodeBuild は、クラウド内のフルマネージド型のビルドサービスです。CodeBuild はソースコードをコンパイルし、単体テストを実行して、すぐにデプロイできるアーティファクトを生成します。

Secrets Manager を使用して、プライベートレジストリの認証情報を保存できます。詳細については、「AWS CodeBuild ユーザーガイド」の「CodeBuild 用の AWS Secrets Manager を使用したプライベートレジストリのサンプル」を参照してください。

Amazon Data Firehose が を使用する方法 AWS Secrets Manager

Amazon Data Firehose を使用して、さまざまなストリーミング送信先にリアルタイムのストリーミングデータを配信できます。送信先が認証情報またはキーを要求する場合、Firehose はランタイムに Secrets Manager からシークレットを取得し、送信先に接続します。詳細については、「Amazon Data Firehose デベロッパーガイド」の「Amazon Data Firehose AWS Secrets Manager で を認証する」を参照してください。

AWS DataSync での の使用方法 AWS Secrets Manager

AWS DataSync は、ストレージシステムとサービス間のデータの移動を簡素化、自動化、高速化するオンラインデータ転送サービスです。

DataSync でサポートされているストレージシステムの一部では、データの読み取りと書き込みに認証情報が必要です。DataSync は Secrets Manager を使用してストレージ認証情報に対する保存ま

たはアクセスを行います。DataSync を設定して、ユーザーに代わってシークレットを作成することも、カスタムシークレットを指定することもできます。サービスマネージドシークレットは、プレフィックス `aws-datasync` で始まります。DataSync の外部で作成したシークレットの使用に対してのみ課金されます。「AWS DataSync ユーザーガイド」の「ストレージロケーションの認証情報の提供」を参照してください。

Amazon DataZone が を使用する方法 AWS Secrets Manager

Amazon DataZone は、データのカタログ化、検出、管理、共有、分析を可能にするデータ管理サービスです。 AWS Glue クローラー ジョブを使用してクロールされた Amazon Redshift クラスターのテーブルとビューのデータアセットを使用できます。Amazon Redshift に接続するには、Secrets Manager シークレットに Amazon DataZone 認証情報を指定します。詳細については、「Amazon DataZone ユーザーガイド」の「新しい AWS Glue 接続を使用して Amazon Redshift データベースのデータソースを作成する」を参照してください。 DataZone

AWS Direct Connect での の使用方法 AWS Secrets Manager

Direct Connect は、標準イーサネット光ファイバケーブルを介して内部ネットワークを Direct Connect ロケーションにリンクします。この接続を使用すると、仮想インターフェイスをパブリックに直接作成できます AWS のサービス。

Direct Connect は、接続関連付けキー名と接続関連付けキーペア (CKN/CAK ペア) を、プレフィックス を持つマネージドシークレットに保存しますdirectconnect。シークレットのコストは料金に含まれています Direct Connect。シークレットを更新するには、Secrets Manager Direct Connect ではなく を使用する必要があります。詳細については、「Direct Connect ユーザーガイド」の「Associate a MACsec CKN/CAK with a LAG 」(MACsec CKN/CAK を LAG に関連付ける) を参照してください。

AWS Directory Service での の使用方法 AWS Secrets Manager

Directory Service には、Microsoft Active Directory (AD) を他の AWS サービスで使用する複数の方法が用意されています。認証情報にシークレットを使用すると、Amazon EC2 インスタンスをディレクトリに参加させることができます。詳細については、「Direct Connect ユーザーガイド」で以下を参照してください。

- Linux EC2 インスタンスを AWS Managed Microsoft AD ディレクトリにシームレスに結合する
- Linux EC2 インスタンスを AD Connector ディレクトリにシームレスに結合する
- LinuxEC2スタンスを Simple AD ディレクトリにシームレスに結合する

Amazon DocumentDB (MongoDB 互換) が を使用する方法 AWS Secrets Manager

Amazon DocumentDB (MongoDB 互換) は、MongoDB ワークロードをサポートする、完全管理のドキュメントデータベース・サービスです。Amazon DocumentDB は Secrets Manager と統合してクラスターのプライマリユーザーパスワードを管理し、セキュリティを強化し、認証情報管理を簡素化します。

Amazon DocumentDB はパスワードを生成し、Secrets Manager に保存して、シークレット設定を管理します。デフォルトでは、Amazon DocumentDB は 7 日ごとにシークレットをローテーションしますが、必要に応じてローテーションスケジュールを変更できます。Amazon DocumentDB クラスターを作成または変更するときに、Secrets Manager でプライマリユーザーパスワードを管理するように指定できます。詳細については、「Amazon DocumentDB デベロッパーガイド」で、「Password management with Amazon DocumentDB and Secrets Manager」を参照してください。

AWS Elastic Beanstalk での の使用方法 AWS Secrets Manager

を使用すると AWS Elastic Beanstalk、アプリケーションを実行するインフラストラクチャについて知ることなく、 AWS クラウドにアプリケーションをすばやくデプロイおよび管理できます。Elastic Beanstalk は、 Dockerfile に記述されたイメージを構築することによって、またはリモートの Docker イメージを取り込むことによって、Docker 環境を起動できます。プライベートリポジトリをホストするオンラインレジストリで認証するために、Elastic Beanstalk は Secrets Manager シークレットを使用します。詳細については、「AWS Elastic Beanstalk デベロッパーガイド」の「Docker の設定」を参照してください。

Amazon Elastic Container Registry が を使用する方法 AWS Secrets Manager

Amazon Elastic Container Registry (Amazon ECR) は、安全でスケーラブル、信頼性の高い AWS マネージドコンテナイメージレジストリサービスです。Docker CLI またはその他のクライアントを使用して、リポジトリからイメージをプッシュおよびプルできます。Amazon ECR プライベートレジストリにキャッシュするイメージを含むアップストリームレジストリごとに、プルスルーキャッシュルールを作成する必要があります。認証が必要なアップストリームレジストリでは、認証情報を Secrets Manager シークレットに保存する必要があります。Secrets Manager シークレットは、Amazon ECR コンソールまたは Secrets Manager コンソールのいずれかで作成できます。詳細については、「Amazon ECR User Guide」の「Creating a pull through cache rule」を参照してください。

Amazon Elastic Container Service

Amazon Elastic Container Service (Amazon ECS) は、コンテナ化されたアプリケーションを簡単にデプロイ、管理、スケーリングできる、完全マネージド型のコンテナオーケストレーションサービスです。Secrets Manager のシークレットを参照することによって、機密データをコンテナに挿入できます。詳細については、「Amazon Elastic Container Service デベロッパーガイド」の次のページを参照してください:

- チュートリアル: Secrets Manager のシークレットを使用して機密データを指定する
- アプリケーションを介してプログラムによりシークレットを取得する
- 環境変数経由でシークレットを取得する
- ログ記録設定のシークレットを取得する

Amazon ECS は、コンテナ用に FSx for Windows File Server ボリュームをサポートします。Amazon ECS は、Secrets Manager のシークレットに保存されている認証情報を使用して、Active Directory にドメイン参加したり、FSx for Windows File Server ファイルシステムをアタッチしたりします。詳細については、「Amazon Elastic Container Service デベロッパーガイド」の「チュートリアル: Amazon ECS で FSx for Windows File Server ファイルシステムを使用」と「FSx for Windows File Server ボリューム」を参照してください。

レジストリ認証情報で Secrets Manager シークレットを使用することで、認証 AWS を必要とする 外のプライベートレジストリでコンテナイメージを参照できます。詳細については、「Amazon Elastic Container Service デベロッパーガイド」の「タスクのプライベートレジストリの認証」を参照してください。

Amazon ECS Service Connect を使用する場合、Amazon ECS は Secrets Manager マネージドシークレットを使用して TLS AWS Private Certificate Authority 証明書を保存します。シークレットの保存にかかるコストは Amazon ECS の料金に含まれています。シークレットを更新するには、Secrets Manager ではなく Amazon ECS を使用する必要があります。詳細については、「Amazon Elastic Container Service Developer Guide」の「TLS with Service Connect」を参照してください。

Amazon ElastiCache が を使用する方法 AWS Secrets Manager

ElastiCache では、ロールベースのアクセス制御 (RBAC) と呼ばれる機能を使用して、クラスターを保護できます。これらの認証情報は Secrets Manager に保存します。Secrets Manager は、このタイプのシークレットのために、ローテーションテンプレート を提供します。詳細については、

「Amazon ElastiCache ユーザーガイド」の「Automatically rotating passwords for users」(ユーザーのパスワードの自動ローテーション) を参照してください。

AWS Elemental Live での の使用方法 AWS Secrets Manager

AWS Elemental Live は、ブロードキャストおよびストリーミング配信用のライブ出力を作成できるリアルタイムビデオサービスです。

AWS Elemental Live はシークレット ARN を使用して、Secrets Manager から暗号化キーを含むシークレットを取得します。Elemental Live は、暗号化キーを使用してビデオを暗号化/復号化します。詳細については、「 Elemental Live User Guide」の「How delivery from AWS Elemental Live to MediaConnect works at runtime」を参照してください。

AWS Elemental MediaConnect での の使用方法 AWS Secrets Manager

AWS Elemental MediaConnect は、ブロードキャスターやその他のプレミアムビデオプロバイダーがにライブビデオを確実に取り込み AWS クラウド 、 内外の複数の宛先に配信できるようにするサービスです AWS クラウド。

静的キー暗号化を使用してソース、出力、およびエンタイトルメントを保護し、暗号化キーを AWS Secrets Managerに蓄積します。詳しくは、「AWS Elemental MediaConnect ユーザーガイド」の「AWS Elemental MediaConnectでのスタティックキーの暗号化」を参照してください。

AWS Elemental MediaConvert での の使用方法 AWS Secrets Manager

AWS Elemental MediaConvert はファイルベースの動画処理サービスで、あらゆるサイズのメディアライブラリを使用して、コンテンツ所有者やディストリビューターにスケーラブルな動画処理を提供します。MediaConvert を使用して Kantar のウォーターマークをエンコードするには、Kantar の認証情報をSecrets Managerで保存してください。詳細については、「 AWS Elemental MediaConvert ユーザーガイド」のAWS Elemental MediaConvert 「出力でのオーディオウォーターマークに Kantar を使用する」を参照してください。

AWS Elemental MediaLive での の使用方法 AWS Secrets Manager

AWS Elemental MediaLive は、ブロードキャストおよびストリーミング配信用のライブ出力を作成できるリアルタイムビデオサービスです。組織が AWS Elemental MediaLive または で AWS Elemental Link デバイスを使用している場合は AWS Elemental MediaConnect、デバイスをデプロイしてデバイスを設定する必要があります。詳細については、「MediaLive User Guide」の「Setting up MediaLive as a trusted entity」を参照してください。

AWS Elemental MediaPackage での の使用方法 AWS Secrets Manager

AWS Elemental MediaPackage は、 just-in-timeのビデオパッケージングおよび発信サービスです AWS クラウド。MediaPackage を使用すると、安全性、スケーラビリティ、および信頼性に優れたビデオストリームをさまざまな再生デバイスおよびコンテンツ配信ネットワーク (CDN) に配信できます。詳細については、「AWS Elemental MediaPackage User Guide」の「Secrets Manager access for CDN authorization」を参照してください。

AWS Elemental MediaTailor での の使用方法 AWS Secrets Manager

AWS Elemental MediaTailor は、 で実行されるスケーラブルな広告挿入およびチャネルアセンブリサービスです AWS クラウド。

MediaTailor は、Secrets Manager アクセストークン認証を元の位置にサポートします。Secrets Manager アクセストークン認証では、MediaTailor はSecrets Manager シークレットを使用して、出所のリクエストを認証します。詳細については、「 AWS Elemental MediaTailor ユーザーガイド」のAWS Secrets Manager 「アクセストークン認証の設定」を参照してください。

Amazon EMR で Secrets Manager を使用する方法

Amazon EMR は、Apache Hadoop や Apache Spark などのビッグデータフレームワークを で簡単に実行 AWS し、大量のデータを処理および分析するプラットフォームです。これらのフレームワークおよび関連するオープンソースプロジェクト (Apache Hive や Apache Pig など) 、分析用のデータおよびビジネスインテリジェンスワークロードを処理できます。Amazon EMR を使用して、Amazon S3 や Amazon DynamoDB などの他の AWS データストアやデータベースとの間で大量のデータを変換および移動することもできます。

Amazon EC2 上で実行する Amazon EMR で Secrets Manager を使用する方法

Amazon EMR でクラスターを作成する場合、Secrets Manager のシークレットでアプリケーション設定データをクラスターに提供できます。詳細については、「Amazon EMR 管理ガイド」の「機密性の高い設定データを Secrets Manager に保存する」を参照してください。

さらに、EMR ノートブックを作成するときに、Secrets Manager を使用してプライベート Git ベースのレジストリ認証情報を保存できます。詳細については、「Amazon EMR 管理ガイド」の「Add a Git-based Repository to Amazon EMR」(Git ベースのリポジトリを Amazon EMR に追加する) を参照してください。

EMR Serverless で Secrets Manager を使用する方法

EMR Serverless は、分析アプリケーションの運用を簡素化するサーバーレスのランタイム環境を提供するため、クラスターを設定、最適化、保護、および運用する必要がありません。

データを に保存し AWS Secrets Manager 、EMR Serverless 設定でシークレット ID を使用できます。この方法を使用すると、機密性の高い設定データをプレーンテキストで渡して外部 API に公開する必要がなくなります。

詳細については、「Amazon EMR Serverless ユーザーガイド」の「EMR Serverless でのデータ保護に Secrets Manager を使用する」を参照してください。

Amazon EventBridge が を使用する方法 AWS Secrets Manager

Amazon EventBridge は、アプリケーションをさまざまなソースからのデータに接続するために使用できるサーバーレスのイベントバスサービスです。

Amazon EventBridge API 送信先を作成すると、EventBridge は、プレフィックス `events` を持つ Secrets Manager マネージドシークレットにその接続を保存します。シークレットを保存する費用は、API 送信先を使用する料金に含まれています。シークレットを更新するには、Secrets Manager ではなく EventBridge を使用する必要があります。詳細については、「Amazon EventBridge ユーザーガイド」の「API 送信先」を参照してください。

Amazon FSx が AWS Secrets Manager シークレットを使用する方法

Amazon FSx for Windows File Server は、フルマネージドの Microsoft Windows ファイルサーバーで、完全にネイティブの Windows ファイルシステムでバックアップされています。ファイル共有を作成または管理するときは、 AWS Secrets Manager シークレットから認証情報を渡すことができます。詳細については、「Amazon FSx for Windows File Server ユーザーガイド」の「ファイル共有」および「ファイル共有設定を Amazon FSx に移行する」を参照してください。

AWS Glue DataBrew での の使用方法 AWS Secrets Manager

AWS Glue DataBrew は、コードを記述せずにデータをクリーンアップおよび正規化するために使用できるビジュアルデータ準備ツールです。DataBrew では、一連のデータ変換ステップを recipe と呼びます。 は`DETERMINISTIC_DECRYPT`、データセット内の個人を特定できる情報 (PII) の変換を実行するための `DETERMINISTIC_ENCRYPT`、、および `CRYPTOGRAPHIC_HASH` recipe ステップ AWS Glue DataBrew を提供します。これは Secrets Manager シークレットに保存されている暗号化キーを使用します。DataBrew のデフォルトシークレットを使用して暗号化キーを保存する場

合、DataBrew はプレフィックス `databrew` を持つマネージドシークレットを作成します。シークレットを保存する費用は、DataBrew を使用する料金に含まれています。新しいシークレットを作成して暗号化キーを保存する場合、DataBrew はプレフィックス `AwsGlueDataBrew` のシークレットを作成します。そのシークレットの料金が請求されます。

AWS Glue Studio が を使用する方法 AWS Secrets Manager

AWS Glue Studio は、 で抽出、変換、ロード (ETL) ジョブを簡単に作成、実行、モニタリングできるグラフィカルインターフェイスです AWS Glue。 AWS Glue Studio内で Elasticsearch Spark コネクタを設定することで、Amazon OpenSearch Service を抽出、変換、ロード (ETL) ジョブのデータストアとして使用できます。OpenSearch クラスターに接続するには、Secrets Manager のシークレットを使用します。詳細については、「AWS Glue 開発者ガイド」の「チュートリアル: Elasticsearch 用の AWS Glue コネクタの使用」を参照してください。

AWS IoT SiteWise での の使用方法 AWS Secrets Manager

AWS IoT SiteWise は、産業機器から大規模にデータを収集、モデル化、分析、視覚化できるマネージドサービスです。 AWS IoT SiteWise コンソールを使用してゲートウェイを作成できます。そしてデータソースは、ゲートウェイに接続されたローカルサーバーまたは産業機器です。ソースで認証が必要な場合は、シークレットを使用して認証します。詳細については、「AWS IoT SiteWise ユーザーガイド」の「Configuring data source authentication」(データソース認証を設定する) を参照してください。

Amazon Kendra が を使用する方法 AWS Secrets Manager

Amazon Kendra は、ユーザーが自然言語を使用して高度な検索アルゴリズムデータを検索できるようにする、高精度のインテリジェント検索サービスです。

データベースの認証情報を含むシークレットを指定して、データベースに保存されているドキュメントにインデックスを作成することができます。詳細については、「Amazon Kendra ユーザーガイド」の「Using a database data source」(データベースデータソースの使用) を参照してください。

Amazon Kinesis Video Streams が を使用する方法 AWS Secrets Manager

Amazon Kinesis Video Streams を使用して、顧客構内の IP カメラに接続し、カメラからの動画をローカルで記録および保存し、動画をクラウドにストリーミングして長期保存、再生、分析処理を行うことができます。IP カメラからメディアを録画してアップロードするには、Kinesis Video Streams Edge Agent を AWS IoT Greengrass にデプロイします。カメラにストリーミングされるメディアファイルにアクセスするために必要な認証情報は、Secrets Manager シークレットに保存しま

す。詳細については、「Amazon Kinesis Video Streams デベロッパーガイド」の「Amazon Kinesis Video Streams Edge Agent を AWS IoT Greengrassにデプロイする」を参照してください。

AWS Launch Wizard での の使用方法 AWS Secrets Manager

AWS Launch Wizard for Active Directory は、 AWS クラウド アプリケーションのベストプラクティスを適用して、新しい Active Directory インフラストラクチャの設定や、 AWS クラウド またはオンプレミスの既存のインフラストラクチャへのドメインコントローラーの追加をガイドするサービスです。

AWS Launch Wizard では、ドメインコントローラーを Active Directory に参加させるために、ドメイン管理者の認証情報を Secrets Manager に追加する必要があります。詳細については、「AWS Launch Wizard ユーザーガイド」の「Set up for AWS Launch Wizard for Active Directory」(Active Directory 用にセットアップする) を参照してください。

Amazon Lookout for Metrics の使用方法 AWS Secrets Manager

Amazon Lookout for Metrics は、データ内の異常を検出し、その根本原因を特定し、迅速な対応を可能にします。Amazon Redshift または Amazon RDS をLookout for Metrics ディテクタのデータソースとして使用できます。データソースを設定するには、データベースパスワードを含むシークレットを使用します。詳細については、「Amazon Lookout for Metrics Developer Guide」(Amazon Lookout for Metrics デベロッパーガイド) の「Using Amazon RDS with Lookout for Metrics」(Amazon RDS での Lookout for Metrics の使用)、および「Using Amazon Redshift with Lookout for Metrics」(Amazon Redshift での Lookout for Metrics の使用) を参照してください。

Amazon Managed Grafana が を使用する方法 AWS Secrets Manager

Amazon Managed Grafana は、フルマネージドで安全なデータ可視化サービスであり、複数のソースから取得した運用メトリクス、ログ、トレースを直ちにクエリ、相関分析、可視化できます。Amazon Redshift をデータソースとして使用する場合は、 AWS Secrets Manager シークレットを使用して Amazon Redshift 認証情報を指定できます。詳細については、「Amazon Managed Grafana ガイド」の「Amazon Redshift の設定」を参照してください。

AWS Managed Services での の使用方法 AWS Secrets Manager

AWS Managed Services は、インフラストラクチャの継続的な管理を提供するエンタープライズサービスです AWS 。AMS セルフサービスプロビジョニング (SSP) モードは、AMS マネージドアカウントのネイティブ AWS のサービス および API 機能へのフルアクセスを提供します。AMS で Secrets Manager に対するアクセスをリクエストする方法については、「AMS アドバンストユー

ザーガイド」の「AWS Secrets Manager (AMS セルフサービスプロビジョニング)」を参照してください。

Amazon Managed Streaming for Apache Kafka の使用方法 AWS Secrets Manager

Amazon Managed Streaming for Apache Kafka (Amazon MSK) は、Apache Kafka を使ってストリーミングデータを処理するアプリケーションの構築および実行を可能にする、フルマネージドサービスです。 AWS Secrets Managerを使用して保存および保護されているユーザーネームとパスワードを使用して、Amazon MSK クラスターへのアクセスを制御できます。詳細については、Amazon Managed Streaming for Apache Kafka デベロッパーガイドの Username and password authentication with AWS Secrets Manager を参照してください。

Amazon Managed Workflows for Apache Airflow が を使用する方法 AWS Secrets Manager

Amazon Managed Workflows for Apache Airflow は、Apache Airflow のマネージドオーケストレーションサービスです。これにより、クラウド内のエンドツーエンドのデータパイプラインを大規模にセットアップして運用することが容易になります。

Apache Airflow 接続は、Secrets Manager シークレットを使用して設定できます。詳細については、「Amazon Managed Workflows for Apache Airflow ユーザーガイド」の「Secrets Manager シークレットを使用した Apache Airflow 接続の設定」および「Apache Airflow 変数 AWS Secrets Manager の でのシークレットキーの使用」を参照してください。

AWS Marketplace

AWS Marketplace クイック起動を使用すると、 はライセンスキーとともにソフトウェアを AWS Marketplace 配布します。 はライセンスキーを Secrets Manager マネージドシークレットとしてアカウントに AWS Marketplace 保存します。シークレットの保存コストは、 の料金に含まれています AWS Marketplace。シークレットを更新するには、Secrets Manager AWS Marketplace ではなくを使用する必要があります。詳細については、「AWS Marketplace 販売者ガイド」の「Configure Quick Launch」を参照してください。

AWS Migration Hub での の使用方法 AWS Secrets Manager

AWS Migration Hub は、複数の AWS ツールやパートナーソリューション間で移行タスクを追跡するための単一の場所を提供します。

AWS Migration Hub オーケストレーターは、サーバーとエンタープライズアプリケーションの への移行を簡素化および自動化します AWS。Migration Hub Orchestrator は、ソースサーバーへの接続情報にシークレットを使用します。詳細については、「AWS Migration Hub Orchestrator ユーザーガイド」で、以下を参照してください。

- SAP NetWeaver アプリケーションを に移行する AWS
- Amazon EC2 上のアプリケーションのリホスト

Migration Hub 戦略の推奨事項は、アプリケーションの実行可能なトランスフォーメーションパスに関する移行およびモダナイゼーション戦略の推奨事項を提供します。戦略の推奨事項で、接続情報のシークレットを使用して SQL Server データベースを分析できます。詳細については、「戦略推奨データベース分析」を参照してください。

AWS Parallel Computing Service が を使用する方法 AWS Secrets Manager

AWS Parallel Computing Service (AWS PCS) は、ハイパフォーマンスコンピューティング (HPC) と分散機械学習ワークロードの実行とスケーリングを容易にするマネージドサービスです AWS。

クラスタージョブスケジューラに接続するために、 AWS PCS はスケジューラキー`pcs`を保存するプレフィックスを持つマネージドシークレットを作成します。シークレットの保存コストは、PCS AWS の料金に含まれています。 AWS PCS クラスターを削除すると、PCS AWS は自動的にシークレットを削除します。詳細については、「PCS ユーザーガイド」の「PCS AWS でのクラスターシークレットの使用」を参照してください。 AWS

> **Important**
>
> AWS PCS クラスターシークレットを変更または削除しないでください。

AWS ParallelCluster での の使用方法 AWS Secrets Manager

AWS ParallelCluster は、 でハイパフォーマンスコンピューティング (HPC) クラスターをデプロイおよび管理するために使用できるオープンソースのクラスター管理ツールです AWS クラウド。 AWS Managed Microsoft AD (Active Directory) と AWS ParallelCluster 統合された を含む複数のユーザー環境を作成できます。 AWS ParallelCluster は、Secrets Manager シークレットを使用して Active Directory へのログインを検証します。詳細については、「AWS ParallelCluster ユーザーガイド」の「Active Directory の統合」を参照してください。

Amazon Q で Secrets Manager を使用する方法

Amazon Q を認証してデータソースにアクセスできるようにするには、Secrets Manager シークレットを使用して、Amazon Q にデータソースアクセス認証情報を提供します。コンソールを使用する場合は、新しいシークレットを作成するか、既存のシークレットを使用するかを選択できます。詳細については、「Amazon Q Developer ガイド」の「Concepts - Authentication」を参照してください。

Amazon OpenSearch Ingestion が Secrets Manager を使用する方法

Amazon OpenSearch Ingestion は、フルマネージド型のサーバーレスデータコレクターです。Amazon OpenSearch Service ドメインと OpenSearch Serverless コレクションにリアルタイムログ、メトリクス、トレースデータを配信します。Secrets Manager で OpenSearch Ingestion パイプラインを使用して、認証情報を安全に管理できます。詳細については、以下を参照してください。

- Using an OpenSearch Ingestion pipeline with Atlassian Services
- Using an OpenSearch Ingestion pipeline with Amazon DocumentDB
- Using an OpenSearch Ingestion pipeline with Confluent Cloud Kafka
- Using an OpenSearch Ingestion pipeline with Kafka
- Amazon OpenSearch Ingestion を使用したセルフマネージド OpenSearch クラスターからのデータの移行

AWS OpsWorks for Chef Automate での の使用方法 AWS Secrets Manager

OpsWorks は、OpsWorks for Puppet Enterprise または を使用してクラウドエンタープライズでアプリケーションを設定および運用するのに役立つ設定管理サービスです AWS OpsWorks for Chef Automate。

で新しいサーバーを作成すると AWS OpsWorks CM、OpsWorks CM はプレフィックス が付いた Secrets Manager マネージドシークレットにサーバーの情報を保存しますopsworks-cm。シークレットの費用は OpsWorksの料金に含まれています。詳細については、「OpsWorks ユーザーガイド」の「AWS Secrets Managerとの統合」を参照してください。

Amazon Quick が を使用する方法 AWS Secrets Manager

Amazon Quick は、分析、データの可視化、レポート作成に使用できるクラウド規模のビジネスインテリジェンス (BI) サービスです。Quick では、さまざまなデータソースを使用できます。データベース認証情報を Secrets Manager シークレットに保存する場合、Quick はそれらのシークレットを使用してデータベースに接続できます。詳細については、「Amazon Quick AWS Secrets Manager ユー

ザーガイド」の「Amazon Quick でのデータベース認証情報の代わりにシークレットを使用する」を参照してください。

Amazon RDS が を使用する方法 AWS Secrets Manager

Amazon Relational Database Service (Amazon RDS) は、 AWS クラウドでリレーショナルデータベースを簡単にセットアップし、運用し、スケーリングすることのできるウェブサービスです。

Aurora を含む Amazon Relational Database Service (Amazon RDS) のマスターユーザー認証情報を管理するために、Amazon RDS ではマネージドシークレットを作成できます。そのシークレットの料金が請求されます。Amazon RDS は、これらの認証情報のローテーションも管理します。詳細については、「Amazon RDS ユーザーガイド」の「Amazon RDS および AWS Secrets Managerでのパスワード管理」を参照してください。

Amazon RDS の他の認証情報については「シークレットを作成する」を参照してください。

Amazon RDS クエリエディタを使用してデータベースに接続する場合、データベースの認証情報を Secrets Manager に保存できます。詳細については、「Amazon RDS ユーザーガイド」の「クエリエディタの使用」を参照してください。

Amazon Redshift が を使用する方法 AWS Secrets Manager

Amazon Redshift は、 クラウド内でのフルマネージド型、ペタバイト規模のデータウェアハウスサービスです。

Amazon Redshift の管理者認証情報を管理するために、Amazon Redshift は、ユーザーのためにマネージドシークレットを作成できます。そのシークレットの料金が請求されます。Amazon Redshift は、これらの認証情報のローテーションも管理します。詳細については、「Amazon Redshift 管理ガイド」の「AWS Secrets Managerを使用して Amazon Redshift 管理者パスワードを管理する」を参照してください。

その他の Amazon Redshift 認証情報については、「シークレットを作成する]を参照してください。

Amazon Redshift Data API を呼び出すと、Secrets Manager のシークレットを使用してクラスターの認証情報を渡すことができます。詳細については、「Using the Amazon Redshift Data API」(Amazon Redshift Data API の使用) を参照してください。

Amazon Redshift クエリエディタを使用してデータベースに接続すると、Amazon Redshift は、プレフィックス `redshiftqueryeditor` の Secrets Manager シークレットに認証情報を保存できます。そのシークレットの料金が請求されます。詳細については、「Amazon Redshift 管理ガイド」の「クエリエディタを使用してデータベースのクエリを実行する」を参照してください。

クエリエディタ v2 については、「the section called “Amazon Redshift Query Editor V2”」を参照してください。

Amazon Redshift Query Editor V2

Amazon Redshift クエリエディタ v2 は、Amazon Redshift データウェアハウスでクエリを作成および実行するために使用できるウェブベースの SQL クライアントアプリケーションです。Amazon Redshift query editor v2 を使用してデータベースに接続すると、Amazon Redshift では、プレフィックス `sqlworkbench` を持つ Secrets Manager マネージドシークレットに認証情報を保存できます。シークレットを保存する費用は、Amazon Redshift の使用料金に含まれています。シークレットを更新するには、Secrets Manager ではなく Amazon Redshift を使用する必要があります。詳細については、「Amazon Redshift 管理ガイド」の「クエリエディタ v2 の使用」を参照してください。

以前のクエリエディタについては、「the section called “Amazon Redshift”」を参照してください。

Amazon SageMaker AI が を使用する方法 AWS Secrets Manager

SageMaker AI は、フルマネージド型の機械学習サービスです。SageMaker AI では、データサイエンティストやデベロッパーが迅速かつ簡単に機械学習モデルの構築とトレーニングを行うことができ、それらを稼働準備が整ったホストされている環境に直接デプロイできます。統合された Jupyter オーサリングノートブックインスタンスから、調査および分析用のデータソースに簡単にアクセスできるため、サーバーを管理する必要がありません。

Jupyter Notebook インスタンスを Git リポジトリに関連付けると、ノートブックインスタンスを停止または削除しても持続するソース管理環境にノートブックを保存できます。Secrets Manager を使用して、プライベートリポジトリの認証情報を管理できます。詳細については、「Amazon SageMaker AI デベロッパーガイド」の「Associate Git Repositories with Amazon SageMaker Notebook Instances」(Git リポジトリを Amazon SageMaker ノートブックインスタンスに関連付ける) を参照してください。

Databricks からデータをインポートするために、Data Wrangler は JDBC URL をSecrets Manager に保存します。詳細については、「Databricks (JDBC) からデータをインポートする」を参照してください。

Snowflake からデータをインポートするために、Data Wrangler は Secrets Manager のシークレットに資格情報を格納します。詳細については、「Snowflake からデータをインポートする」を参照してください。

AWS Schema Conversion Tool での の使用方法 AWS Secrets Manager

AWS Schema Conversion Tool (AWS SCT) を使用して、既存のデータベーススキーマをあるデータベースエンジンから別のデータベースエンジンに変換できます。リレーショナル OLTP スキーマやデータウェアハウススキーマを変換できます。変換されたスキーマは、Amazon Relational Database Service (Amazon RDS) MySQL、MariaDB、Oracle、SQL Server、PostgreSQL DB、Amazon Aurora DB クラスター、または Amazon Redshift クラスターに適しています。変換されたスキーマは、Amazon Elastic Compute Cloud インスタンス上のデータベースで使用するか、S3 バケットにデータとして保存することもできます。

データベーススキーマを変換すると、 AWS SCT は保存したデータベース認証情報を使用できます AWS Secrets Manager。詳細については、「 AWS SCT ユーザーガイド」の「ユーザーインターフェイス AWS Secrets Manager での の使用」を参照してください。 AWS Schema Conversion Tool

Amazon Timestream for InfluxDB が を使用する方法 AWS Secrets Manager

Timestream for InfluxDB は、オープンソース API を使用してリアルタイムの時系列アプリケーション AWS 用に InfluxDB データベースを で簡単に実行できるマネージド時系列データベースエンジンです。 APIs Timestream for InfluxDB を使用して、1 桁ミリ秒のクエリ応答時間でクエリに応答できる時系列ワークロードを設定、操作、スケーリングできます。

Timestream for InfluxDB データベースを作成すると、Timestream は管理者認証情報を格納するシークレットを自動的に作成します。詳細については、「Timestream Developer Guide」の「How Timestream for InfluxDB uses secrets」を参照してください。

AWS Toolkit for JetBrains での の使用方法 AWS Secrets Manager

AWS Toolkit for JetBrains は、JetBrains の統合開発環境 (IDEs) 用のオープンソースプラグインです。 JetBrains このツールキットにより、開発者は AWSを使用するサーバーレスアプリケーションの開発、デバッグ、およびデプロイが簡単になります。ツールキットを使用して Amazon Redshift クラスターに接続する場合、Secrets Manager シークレット を使用して認証できます。詳細については、「AWS Toolkit for JetBrains ユーザーガイド」の「Accessing Amazon Redshift clusters」(Amazon Redshift へのアクセス) を参照してください。

がシー AWS Secrets Manager クレット AWS Transfer Family を使用する方法

AWS Transfer Family は、 AWS ストレージサービスとの間でファイルを転送できる安全な転送サービスです。

Transfer Family は、Applicability Statement 2 (AS2) プロトコルを使用するサーバー向けに基本認証の使用をサポートするようになりました。認証情報のために Secrets Manager の新しいシークレットを作成することも、既存のシークレットを選択することもできます。詳細については、「AWS Transfer Family ユーザーガイド」の「AS2 コネクタの基本認証」を参照してください。

Transfer Family ユーザーを認証するには、 を ID プロバイダー AWS Secrets Manager として使用できます。詳細については、「 ユーザーガイド」の「カスタム ID プロバイダーの使用」およびブログ記事「 AWS Transfer Family のパスワード認証を有効にする AWS Secrets Manager」を参照してください。 AWS Transfer Family

Transfer Family がワークフローで処理するファイルで、Pretty Good Privacy (PGP) 復号化を使用できます。ワークフローステップで復号化を使用するには、Secrets Manager で管理する PGP キーを提供します。詳細については、「AWS Transfer Family ユーザーガイド」の「PGP キーの生成と管理」を参照してください。

がシー AWS Secrets Manager クレット AWS Wickrを使用する方法

AWS Wickr は、組織や政府機関が one-to-one のグループメッセージング、音声およびビデオ通話、ファイル共有、画面共有などを通じて安全に通信できるようにするend-to-endの暗号化サービスです。Wickr データ保持ボットを使用すると、ワークフローを自動化できます。ボットが にアクセスできる場合は AWS のサービス、Secrets Manager シークレットを作成してボットの認証情報を保存する必要があります。詳細については、「AWS Wickr 管理ガイド」の「データ保持ボットの開始」を参照してください。

AWS Secrets Manager マネージド外部シークレットを使用したサードパーティーシークレットの管理

マネージド外部シークレットは、 の新しいシークレットタイプ AWS Secrets Manager であり、統合パートナーからの認証情報を保存して自動的に更新できます。この機能により、統合パートナーのシークレットをローテーションするためのカスタム AWS Lambda 関数を作成および維持する必要がなくなります。オンボーディングされたすべてのパートナーの完全なリストについては、「 統合パートナー」を参照してください。

でアプリケーションを構築する場合 AWS、多くの場合、ワークロードは API キー、OAuth トークン、認証情報ペアなどの安全な認証情報を介してサードパーティーアプリケーションとやり取りする必要があります。以前は、各アプリケーションに固有の複雑なローテーション Lambda 関数の構築や継続的なメンテナンスなど、これらの認証情報を保護および管理するためのカスタムアプローチを開発する必要がありました。

マネージド外部シークレットは、サードパーティーの認証情報を各パートナーが定義した事前定義された形式で保存するための標準化されたアプローチを提供します。この機能には、シークレットの作成中に有効になる自動ローテーション (デフォルトではコンソール) 、シークレット管理ワークフローの完全な透明性とユーザーコントロール、きめ細かなアクセス許可管理、オブザーバビリティ、ガバナンス、コンプライアンス、ディザスタリカバリ、モニタリングコントロールなど、Secrets Manager が提供する完全な機能セットが含まれます。

コンソールまたは を使用して、マネージド外部シークレットの自動ローテーションを設定する方法については、 AWS CLI「」を参照してくださいSecrets Manager が管理する外部シークレットのローテーション。マネージド外部シークレットを作成し、ローテーションを有効にするstep-by-stepの例については、「」を参照してくださいSalesforce クライアントシークレットを作成する。

主な特徴

マネージド外部シークレットには、サードパーティーの認証情報管理を簡素化するいくつかの主要な機能があります。

- Lambda フリーのマネージドローテーションは、カスタムローテーション関数の作成と管理のオーバーヘッドを排除します。外部 を作成すると、ローテーションは自動的に有効になり、アカウントにデプロイされた Lambda 関数はありません。

- 事前定義されたシークレット形式により、シークレットを統合パートナーに適切に関連付け、ローテーションに必要なメタデータを含めることができます。各パートナーは必要な形式を定義します。
- 統合されたパートナーエコシステムは、標準化されたオンボーディングプロセスを通じて複数のパートナーをサポートします。パートナーは Secrets Manager と直接統合して、シークレットの作成とマネージドローテーション機能に関するプログラムによるガイダンスを提供します。
- 完全な監査可能性は、すべてのローテーションアクティビティ、シークレット値の更新、管理オペレーションの AWS CloudTrail ログ記録を通じて、完全な透明性を維持します。

マネージド外部シークレットパートナー

Secrets Manager は、サードパーティーのアプリケーションとネイティブに統合して、パートナーが保持するシークレットをローテーションします。各パートナーは、シークレットのローテーションに必要なメタデータとシークレット値フィールドを定義します。

シークレット値には、サードパーティークライアントとの接続に必要なフィールドが含まれており、CreateSecret 呼び出し中に保存されます。ローテーションメタデータは、ローテーション中にシークレットを更新するために使用されるフィールドを保持し、RotateSecret 呼び出しで使用されます。これらのフィールドは、統合パートナーによって定義され、マネージドローテーションフローを許可します。

ローテーションを適切に機能させるには、シークレットライフサイクルを管理するための特定のアクセス許可を Secrets Manager に提供する必要があります。詳細については、「セキュリティとアクセス許可」を参照してください。

以下のトピックでは、シークレットのローテーションに必要な各メタデータフィールドの説明と、ローテーションに必要な Secrets Manager シークレットの各フィールドの説明について説明します。

トピック

統合パートナー	シークレットのタイプ
BigID	BigIDClientSecret
Cisco	CiscoSecurityPlatformApiKey
Confluent Cloud	ConfluentCloudApiKey

統合パートナー	シークレットのタイプ
Databricks	DatabricksPAT
Databricks	DatabricksSPSecret
Datadog	DatadogApiKey
Datadog	DatadogApplicationKey
Datadog	DatadogAdminKey
GitLab	GitLabAccessToken
Jenkins	JenkinsApiToken
MongoDB アトラス	MongoDBAtlasServiceAccount
MongoDB アトラス	MongoDBAtlasDatabaseUser
Netskope	NetskopeApiToken
アイドル	PaddleApiKey
Salesforce	SalesforceClientSecret
Snowflake	SnowflakeKeyPairAuthentication
Snowflake	SnowflakePat
SonarQube	SonarQubeToken
Twilio	TwilioApiKey
Twilio	TwilioAuthToken
Twilio	TwilioMainApiKey

ビッグ ID 更新トークン

シークレット値フィールド

以下は、Secrets Manager シークレットに含める必要があるフィールドです。

```
{
  "hostname": "Host Name",
  "refreshToken": "Refresh Token"
}
```

hostname

これは、BigID インスタンスがホストされているホスト名です。インスタンスの完全修飾ドメイン名を入力する必要があります。

refreshToken

管理 → アクセス管理 → ユーザーの選択 → トークンの生成 → 保存を介して BigID コンソールで生成された JWT ユーザー更新トークン

使用フロー

上記のフィールドと CreateSecret 呼び出しを使用してシークレットを作成できます。BigIDClientSecret ローテーション設定は、RotateSecret 呼び出しを使用して設定できます。また、RotateSecret 呼び出しでロール ARN を指定する必要があります。これにより、シークレットをローテーションするために必要なアクセス許可がサービスに付与されます。アクセス許可ポリシーの例については、「セキュリティとアクセス許可」を参照してください。このパートナーでは、ローテーションメタデータフィールドを空のままにできることに注意してください。

Cisco セキュリティプラットフォーム API キー

シークレット値フィールド

以下は、Secrets Manager シークレットに含める必要があるフィールドです。

```
{
  "refreshToken": "refresh token value",
  "organizationId": "organization ID",
  "apiKeyId": "API key ID",
```

```
  "apiBaseUrl": "https://api.security.cisco.com"
}
```

refreshToken

Cisco API キー。60 日間の有効期間を持つ OAuth 更新トークンです。ローテーションプロセスはこのフィールドを更新します。アプリケーションはこのトークンを有効期間の短いアクセストークンと交換します。

organizationId

API キーを所有する Cisco Security Cloud Control 組織。この値は、API キーページの URL `enterpriseId`に表示される です。トークンに埋め込まれた組織 ID を使用しないでください。

apiKeyId

ローテーションする API キーの識別子。この値は、組織 ID と同じ API キーページで確認できます。

apiBaseUrl

Cisco API ベース URL。例: `https://api.security.cisco.com`。値は、パス、ポート、クエリ文字列、またはユーザー情報を持たない HTTPS オリジンである必要があります。

また、ローテーションはシークレット値に `expiresDate`フィールドを書き込みます。このフィールドには、現在の更新トークンの有効期限が ISO 8601 形式で記録されます。指定したフィールドではありません。

Secrets Manager はアクセストークンを保存しません。アクセストークンの有効期間は約 18 時間であるため、保存されているコピーはほとんどの場合無効になります。アプリケーションは、必要なときに更新トークンからアクセストークンを書き込みます。

シークレットメタデータフィールド

このシークレットタイプでは、ローテーションメタデータフィールドは必要ありません。Secrets Manager はシークレット自体からルーティング値を読み取り、このシークレットタイプは管理者シークレットを使用しません。

使用フロー

このローテーションでは、単一シークレットアーキテクチャを使用します。管理者シークレットは必要ありません。保存された更新トークンは、独自のローテーション呼び出しを認証します。

シークレットを作成するには、CreateSecret API コールを使用します。シークレット値を上記のフィールドに設定し、シークレットタイプを CiscoSecurityPlatformApiKey に設定します。ローテーションを設定するには、RotateSecret API コールを使用します。RotateSecret 呼び出しで、シークレットをローテーションするアクセス許可をサービスに付与するロール ARN を指定します。アクセス許可ポリシーの例については、「セキュリティとアクセス許可」を参照してください。

ローテーション中、Secrets Manager は現在の更新Ciscoトークンを使用してトークン更新エンドポイントを呼び出します。 はアクセストークンCiscoを返し、60 日サイクルの 45 日後に新しい更新トークンも返します。Secrets Manager は、最新の更新トークンを保留中のバージョンとして保存し、アクセストークンを破棄します。次に、保留中のトークンを Cisco API に対して検証し、現在のトークンに昇格します。は更新トークンを無効にする API Ciscoを提供しないため、取り消しステップはありません。

アプリケーションはシークレットから更新トークンを読み取り、アクセストークンと交換します。次に、そのアクセストークンを 18 時間キャッシュします。更新トークンは、ハードコーディングするのではなく、シークレットから毎回読み取ります。HTTP 401 レスポンスを処理するには、シークレットを再度読み取ります。これは、前の更新トークンが置換Ciscoを発行した後に機能しなくなる可能性があるためです。

このシークレットを少なくとも 15 日ごとにローテーションします。Secrets Manager は 50 日を超えるローテーション間隔を拒否します。これは、少なくともその頻度で更新するためのCiscoガイダンスに従います。間隔を長くすると、 が代替更新トークンCiscoを発行する 45 日目から 60 日目までのウィンドウをスキップすることもできます。トークンの残りの有効期間がローテーション間隔よりも短い場合、トークンの有効期限が切れるのではなく、ローテーションはエラーで失敗します。

有効期限が切れた更新トークンは更新できません。その場合は、CiscoSecurity Cloud Control コンソールで新しい API キーを発行し、シークレットを更新します。長時間の停止がその状態にならないように、ローテーションの失敗をモニタリングします。

Confluent Cloud API キー

シークレット値フィールド

以下は、Secrets Manager シークレットに含める必要があるフィールドです。

```
{
  "apiKey": "API Key ID",
  "apiSecret": "API Secret",
  "serviceAccountId": "Service Account ID",
```

```
  "resourceId": "Resource ID",
  "environmentId": "Environment ID"
}
```

apiKey

認証に使用される Confluent Cloud API キー ID。

apiSecret

認証に使用される Confluent Cloud API シークレット。

serviceAccountId

この API キーが表すサービスアカウントプリンシパルの ID。例: `sa-abc123`。ローテーションロジックはこれを使用して、正しいプリンシパルの新しいキーを作成します。

resourceId

(オプション) API キーをスコープするためのリソース ID。Kafka クラスター (`lkc-xxxxx`)、ksqlDB クラスター (`lksqlc-xxxxx`)、スキーマレジストリ (`lsrc-xxxxx`)、Flink リージョン (`aws.us-west-2`、`azure.centralus`、`gcp.us-central1`)、または を使用できますTableflow。クラウドリソース管理 API キーの場合は、このフィールドを省略します。

environmentId

(オプション) Confluent クラウド環境 ID。例: `env-abcde`。クラスタースコープのキーを作成するときに使用されます。

シークレットメタデータフィールド

Confluent Cloud API キーのメタデータフィールドは次のとおりです。

```
{
  "adminSecretArn": "arn:aws:secretsmanager:us-
east-1:111122223333:secret:ConfluentCloudApiKey"
}
```

adminSecretArn

(オプション) このシークレットのローテーションに使用される管理 Confluent Cloud API キー認証情報を含むシークレットの Amazon リソースネーム (ARN)。サービスアカウントの API キーを

作成および削除するには、管理者 API キーに CloudClusterAdmin または OrganizationAdmin ロールが必要です。省略すると、ユーザーシークレットの独自の認証情報が自己ローテーションに使用されます。

使用フロー

ローテーションは 2 つのモードをサポートしています。セルフローテーションモード (デフォルト) では、ユーザーシークレットの独自の `apiKey/apiSecret` を使用して、Confluent API コールを認証し、キーの作成と削除を行います。ユーザーシークレットの API キーには、独自のサービスアカウントのキーを管理するのに十分なアクセス許可が必要です。admin-secret モードでは、代わりに管理者権限を持つ `apiKey/apiSecret` を含む別の管理者シークレットが使用されます。

上記のフィールドと、ConfluentCloudApiKey としてシークレットタイプを含むシークレット値を使用してCreateSecret 呼び出しを使用してシークレットを作成できます。ローテーション設定は、RotateSecret 呼び出しを使用して設定できます。セルフローテーションを選択した場合は、オプションの `adminSecretArn`フィールドを省略できます。RotateSecret 呼び出しでロール ARN を指定する必要があります。これにより、シークレットをローテーションするために必要なアクセス許可がサービスに付与されます。アクセス許可ポリシーの例については、「セキュリティとアクセス許可」を参照してください。

別の管理者認証情報セットを使用してシークレットをローテーションすることを選択した場合は、管理者`apiKey`と AWS Secrets Manager を含む で管理者シークレットを作成します`apiSecret`。API キーシークレットの RotateSecret 呼び出しのローテーションメタデータで、この管理者シークレットの ARN を指定する必要があります。

ローテーション中、ドライバーは Confluent Cloud API を介してターゲットサービスアカウントの新しい API キーを作成し、新しいキーを検証し、新しい認証情報でシークレットを更新し、古い API キーを削除します。

Databricks 個人用アクセストークン

シークレット値フィールド

以下は、Secrets Manager シークレットに含める必要があるフィールドです。

```
{
  "token": "dapi followed by 32 hexadecimal characters",
  "tokenId": "64-character hexadecimal token ID",
```

```
  "workspaceUrl": "https://dbc-xxxxxxxx-xxxx.cloud.databricks.com",
  "comment": "Comment shown in the token list"
}
```

トークン

Databricks 個人用アクセストークン。Secrets Manager はこのフィールドをローテーションします。

tokenId

トークン ID。Secrets Manager はこれを使用して現在のトークンの範囲を読み取り、置き換えるトークンを取り消します。

workspaceUrl

Databricks ワークスペース URL。例: `https://dbc-xxxxxxxx-xxxx.cloud.databricks.com`。ベア HTTPS オリジンを指定します。Secrets Manager は、ユーザー情報、ポート、パス、クエリ文字列、フラグメント、または末尾のスラッシュを含む URL を拒否します。

コメント

(オプション) トークンリスト内のDatabricksトークンの横に表示されるコメント。最大長は 255 文字です。

シークレットメタデータフィールド

以下は、Databricks個人用アクセストークンのメタデータフィールドです。

```
{
  "adminSecretArn": "arn:aws:secretsmanager:us-
east-1:111122223333:secret:DatabricksRotatorToken",
  "tokenLifetimeDays": "90"
}
```

adminSecretArn

(オプション) このシークレットをローテーションする DatabricksPAT タイプの別のシークレットの Amazon リソースネーム (ARN)。トークン自体をローテーションするには、このフィールドを省略します。

tokenLifetimeDays

(オプション) 新しい各トークンの有効期間。日数。値はローテーション間隔よりも大きくする必要があります。デフォルト: 90。

ワークスペース管理者は、ここで選択した値よりも低いトークンの最大有効期間を設定できます。ワークスペースの最大数が低い場合、トークンの作成は失敗し、ローテーションを完了できません。値を選択する前にワークスペース設定を確認してください。

使用フロー

このローテーションタイプは 2 つのモードをサポートしています。セルフローテーションでは、トークン自体がローテーションされます。アシストローテーションでは、同じタイプの 2 番目のトークンがローテーションします。いずれの場合も、置換トークンは置き換えるトークンのスコープを継承します。Secrets Manager がそれらのスコープを読み取れない場合、異なる権限を持つトークンを作成する代わりにローテーションは失敗します。

上記のフィールドと DatabricksPAT としてシークレットタイプを含むシークレット値を持つ CreateSecret 呼び出しを使用してシークレットを作成できます。ローテーション設定は、RotateSecret 呼び出しを使用して設定できます。セルフローテーションを選択した場合は、オプションの `adminSecretArn`フィールドを省略できます。RotateSecret 呼び出しでロール ARN を指定する必要があります。これにより、シークレットをローテーションするために必要なアクセス許可がサービスに付与されます。アクセス許可ポリシーの例については、「セキュリティとアクセス許可」を参照してください。

ローテーション中、Secrets Manager は現在のトークンを検証し、そのスコープを読み取ります。AWSPENDING と同じスコープを持つ置換トークンを作成し、置換を保存します。新しいトークンが認証されることを確認します。また、新しいトークン`workspaceUrl`のスコープ、所有者、および が、置き換えるトークンと一致することも確認します。次に、前のトークンを取り消し、AWSPENDING を AWSCURRENT に昇格します。

両方のトークンは、Secrets Manager が置換を作成する瞬間から、前のトークンを取り消すまで認証されます。そのウィンドウ中にシークレットを読み取るアプリケーションは中断されません。は認可の決定をDatabricksキャッシュするため、失効自体は結果整合性があります。前のトークンは、Secrets Manager が取り消した後、短期間受け入れられ続ける場合があります。認証情報の侵害の疑いに応じてローテーションする場合は、トークンが受け入れられなくなったDatabricksことを を通じて確認します。

Databricks は、90 日間使用されない個人用アクセストークンを取り消します。ローテーションは、実行時にのみローテータートークンを使用するため、ローテーション間隔を 90 日より短く設定します。そうしないと、 は非アクティブ時のローテータートークンをDatabricks取り消し、それ以降のローテーションは失敗します。ローテーション間隔は 30 日、デフォルトの有効期間は 90 日で、両方の要件を満たします。

Secrets Manager は、ループバック、リンクローカル、またはプライベートアドレスに解決`workspaceUrl`される を拒否します。したがって、パブリックアクセスが無効になっているフロントエンド AWS PrivateLink経由でのみ到達するワークスペースは、個人用アクセストークンのローテーションではサポートされていません。

Databricks サービスプリンシパル OAuth クライアントシークレット

シークレット値フィールド

以下は、Secrets Manager シークレットに含める必要があるフィールドです。

```
{
  "accountId": "Databricks account UUID",
  "servicePrincipalId": "Numeric service principal ID",
  "clientId": "Service principal application UUID",
  "clientSecret": "dose followed by the secret value",
  "oauthSecretId": "Hexadecimal OAuth secret ID"
}
```

accountId

Databricks アカウント ID。これは UUID です。

servicePrincipalId

サービスプリンシパルの数値内部 ID。この値はアプリケーション ID とは異なります。

clientId

サービスプリンシパルのアプリケーション ID。Secrets Manager は、OAuth リクエストで`client_id` として送信します。

clientSecret

現在の OAuth クライアントシークレット。Secrets Manager はこのフィールドをローテーションします。

oauthSecretId

現在の OAuth シークレットの Databricks ID。Secrets Manager はこれを使用して、現在のシークレットの範囲を読み取り、置き換えるシークレットを削除します。

シークレットメタデータフィールド

以下は、Databricksサービスプリンシパル OAuth クライアントシークレットのメタデータフィールドです。

```
{
  "adminSecretArn": "arn:aws:secretsmanager:us-
east-1:111122223333:secret:DatabricksAdminServicePrincipal",
  "secretLifetimeDays": "730"
}
```

adminSecretArn

(オプション) このシークレットをローテーションする DatabricksSPSecret タイプの別のシークレットの Amazon リソースネーム (ARN)。このフィールドを省略して、サービスプリンシパルが独自のシークレットをローテーションできるようにします。

secretLifetimeDays

(オプション) 新しい各シークレットの有効期間。日数。値はローテーション間隔よりも大きくする必要があります。最大数は 730 です。デフォルト: 730。

使用フロー

このローテーションタイプは 2 つのモードをサポートしています。自己ローテーションでは、サービスプリンシパルは独自のシークレットをローテーションします。アシストローテーションでは、2 番目のサービスプリンシパルがローテーションします。OAuth シークレットエンドポイントはアカウントレベルであるため、ローテーションを実行するサービスプリンシパルには`account_admin`ロールが必要です。

上記のフィールドと CreateSecret 呼び出しを使用してシークレットを作成できます。DatabricksSPSecret ローテーション設定は、RotateSecret 呼び出しを使用して設定できます。セルフローテーションを選択した場合は、オプションの `adminSecretArn`フィールドを省略できます。RotateSecret 呼び出しでロール ARN を指定する必要があります。これにより、シークレットを

ローテーションするために必要なアクセス許可がサービスに付与されます。アクセス許可ポリシーの例については、「セキュリティとアクセス許可」を参照してください。

ローテーション中、Secrets Manager はクライアント認証情報フローを使用してアクセストークンをリクエストします。置き換えるシークレットのスコープを格納する置き換えシークレットを作成し、置き換えを AWSPENDING として保存します。これにより、新しいシークレットが認証されることを確認します。次に、前のシークレットを削除し、AWSPENDING を AWSCURRENT に昇格します。

サービスプリンシパルは、最大 5 つの OAuth シークレットをサポートします。サービスプリンシパルが既に 5 つを保持している場合、 はリクエストDatabricksを拒否し、ローテーションは失敗します。再度ローテーションする前に、未使用のシークレットを削除します。

このタイプに必要なaccount_adminロールは、Databricks個人用アクセストークンのローテーションに必要なワークスペースアクセスとは異なります。単一の許可は両方のフローをカバーしません。

Datadog 管理キー

シークレット値フィールド

以下は、Secrets Manager シークレットに含める必要があるフィールドです。

```
{
  "adminApiKey": "32-character hex API key",
  "adminApiKeyId": "API key UUID",
  "adminAppKey": "Application key starting with ddapp_",
  "adminAppKeyId": "Application key UUID",
  "serviceAccountId": "Service Account UUID",
  "site": "datadoghq.com"
}
```

adminApiKey

Datadog 管理 API キー (32 文字の 16 進数文字列) 。

adminApiKeyId

管理者 API キーの一意の識別子 (UUID)。

adminAppKey

Datadog 管理者アプリケーションキー。サービスアカウントが所有し、スコープが `api_keys_write`、`api_keys_delete`、、`org_app_keys_read`org_app_keys_write、である必要がありますservice_account_write。

adminAppKeyId

管理者アプリケーションキーの一意の識別子 (UUID)。

serviceAccountId

管理者アプリケーションキーを所有する Datadog サービスアカウント ID (UUID)。

site

Datadog サイト (、、 など`datadoghq.com`datadoghq.eu`us5.datadoghq.com`)。

シークレットメタデータフィールド

Datadog 管理キーのメタデータフィールドは次のとおりです。

```
{
  "adminSecretArn": "arn:aws:secretsmanager:us-
east-1:111122223333:secret:DatadogAdminKey"
}
```

adminSecretArn

(オプション) 認証に使用される別の管理者シークレットの Amazon リソースネーム (ARN)。指定しない場合、このシークレットは独自の認証情報 (自己ローテーション) を使用して自身をローテーションします。

使用フロー

このローテーションタイプは、API キーとアプリケーションキーの両方をペアとしてローテーションします。シークレットが独自の認証情報を使用して置き換えを作成するセルフローテーション (デフォルト) 、または別の管理者シークレットを使用した管理者アシストローテーションをサポートします。

上記のフィールドと DatadogAdminKey としてのシークレットタイプを含むシークレット値を使用してCreateSecret 呼び出しを使用してシークレットを作成できます。ローテーション設定

は、RotateSecret 呼び出しを使用して設定できます。セルフローテーションを選択した場合は、オプションの adminSecretArnフィールドを省略できます。RotateSecret 呼び出しでロール ARN を指定する必要があります。これにより、シークレットをローテーションするために必要なアクセス許可がサービスに付与されます。アクセス許可ポリシーの例については、「セキュリティとアクセス許可」を参照してください。

ローテーション中、ドライバーは現在の API キーを検証し、新しい API キーと新しいアプリケーションキー (現在のキーからスコープを継承) を作成し、両方の新しいキーを検証し、新しい認証情報を使用して古いペアを削除し、新しいシークレットバージョンを AWSCURRENT に昇格します。

Datadog API キー

シークレット値フィールド

以下は、Secrets Manager シークレットに含める必要があるフィールドです。

```
{
  "apiKey": "32-character hex API key",
  "apiKeyId": "API key UUID"
}
```

apiKey

現在の Datadog API キー。Datadog にメトリクス、ログ、トレースを送信するために使用される 32 文字の 16 進文字列。

apiKeyId

API キーの一意の識別子 (UUID)。Datadog API または組織設定を介して見つかります。

シークレットメタデータフィールド

Datadog API キーのメタデータフィールドは次のとおりです。

```
{
  "adminSecretArn": "arn:aws:secretsmanager:us-
east-1:111122223333:secret:DatadogAdminKey"
}
```

adminSecretArn

このシークレットのローテーションに使用される管理 Datadog 認証情報 (API キーとアプリケーションキー) を含む DatadogAdminKey タイプのシークレットの Amazon リソースネーム (ARN)。アプリケーションキーにはスコープ `api_keys_write`、 が必要です`api_keys_delete`。

使用フロー

このローテーションでは、2 つのシークレットアーキテクチャを使用します。DatadogAdminKey タイプの管理者シークレットは、Datadog キー管理 API コールを認証するために必要な API キーとアプリケーションキーを提供します。

上記のフィールドと DatadogApiKey としてのシークレットタイプを含むシークレット値を使用してCreateSecret 呼び出しを使用してシークレットを作成できます。ローテーション設定は、RotateSecret 呼び出しを使用して設定できます。ローテーションメタデータ`adminSecretArn`で を指定する必要があります。また、RotateSecret 呼び出しでロール ARN を指定し、シークレットをローテーションするために必要なアクセス許可をサービスに付与する必要があります。アクセス許可ポリシーの例については、「セキュリティとアクセス許可」を参照してください。

管理者シークレットタイプ (`DatadogAdminKey`) は、ユーザーシークレットタイプ () とは異なります`DatadogApiKey`。この違いにより、 によってスコープされるデフォルトのローテーションロールポリシー`secretsmanager:resource/Type`は、管理者シークレットへのアクセスを許可しません。ローテーションロールに管理者シークレットへのアクセスを明示的に指定する必要があります。これを行うには、 `DatadogAdminKey`タイプにスコープされたステートメントを追加します。または、ロールポリシーで管理者シークレット`ARN`を直接指定します。

ローテーション中、ドライバーは Datadog Key Management API v2 を介して新しい API キーを作成し、検証エンドポイントを使用して新しいキーを検証し、新しいキーを AWSCURRENT に昇格させ、Datadog から置き換えられたキー (2 つの古いローテーション) を削除します。これにより、2 キーの交代パターンが維持され、ダウンタイムのないローテーションが保証されます。

Datadog アプリケーションキー

シークレット値フィールド

以下は、Secrets Manager シークレットに含める必要があるフィールドです。

```
{
  "appKey": "Application key starting with ddapp_",
  "appKeyId": "Application key UUID",
  "serviceAccountId": "Service Account UUID"
}
```

appKey

サービスアカウントが所有する Datadog アプリケーションキー。で始まり、ddapp_その後に 34 文字の英数字が続きます。

appKeyId

アプリケーションキーの一意の識別子 (UUID)。

serviceAccountId

このアプリケーションキーを所有する Datadog サービスアカウント ID (UUID)。ローテーションできるのは、サービスアカウント所有のアプリケーションキーのみです。

シークレットメタデータフィールド

Datadog アプリケーションキーのメタデータフィールドは次のとおりです。

```
{
  "adminSecretArn": "arn:aws:secretsmanager:us-
east-1:111122223333:secret:DatadogAdminKey"
}
```

adminSecretArn

このシークレットのローテーションに使用される管理 Datadog 認証情報 (API キーとアプリケーションキー) を含む DatadogAdminKey タイプのシークレットの Amazon リソースネーム (ARN)。管理者シークレットは、このアプリケーションキーと同じサービスアカウントに属している必要があります。

使用フロー

このローテーションでは、2 つのシークレットアーキテクチャを使用します。DatadogAdminKey タイプの管理者シークレットは、認証情報を提供します。権限のエスカレーションを防

ぐserviceAccountIdには、管理者シークレットの がユーザーシークレットの と一致するserviceAccountId必要があります。

上記のフィールドと DatadogApplicationKey としてのシークレットタイプを含むシークレット値を使用してCreateSecret 呼び出しを使用してシークレットを作成できます。ローテーション設定は、RotateSecret 呼び出しを使用して設定できます。ローテーションメタデータadminSecretArnで を指定する必要があります。また、RotateSecret 呼び出しでロール ARN を指定する必要があります。これにより、シークレットをローテーションするために必要なアクセス許可がサービスに付与されます。アクセス許可ポリシーの例については、「セキュリティとアクセス許可」を参照してください。

管理者シークレットタイプ (DatadogAdminKey) は、ユーザーシークレットタイプ () とは異なりますDatadogApplicationKey。この違いにより、 によってスコープされるデフォルトのローテーションロールポリシーsecretsmanager:resource/Typeは、管理者シークレットへのアクセスを許可しません。ローテーションロールに管理者シークレットへのアクセスを明示的に指定する必要があります。これを行うには、 DatadogAdminKeyタイプにスコープされたステートメントを追加します。または、ロールポリシーで管理者シークレットARNを直接指定します。

ローテーション中、ドライバーは現在のキーの所有権を検証し、Datadog サービスアカウント API を介して新しいアプリケーションキーを作成し、新しいキーを検証して AWSCURRENT に昇格し、古いキーを削除します。

GitLab アクセストークン

シークレット値フィールド

以下は、Secrets Manager シークレットに含める必要があるフィールドです。

```
{
  "token": "GitLab access token value",
  "tokenId": "numeric token ID",
  "gitlabUrl": "GitLab instance URL",
  "projectId": "project ID (optional)",
  "groupId": "group ID (optional)"
}
```

トークン

GitLab アクセストークン値 (で始まりますglpat-)。これはローテーションされるフィールドです。

tokenId

数値トークン ID。各ローテーションを新しいトークンの ID で更新しました。

gitlabUrl

GitLab インスタンス URL (例: `https://gitlab.com`)。HTTPS を使用する必要があります。

projectId

(オプション) 数値プロジェクト ID。プロジェクトアクセストークンにのみ を指定します。

groupId

(オプション) 数値グループ ID。グループアクセストークンにのみ を指定します。

シークレットメタデータフィールド

GitLab アクセストークンのメタデータフィールドは次のとおりです。

```
{
  "adminSecretArn": "arn:aws:secretsmanager:us-east-1:111122223333:secret:GitLabAdmin",
  "daysToExpiry": "30 (optional)"
}
```

adminSecretArn

(オプション) このシークレットのローテーションに使用される API スコープの GitLab アクセストークンを含む GitLabAccessToken タイプのシークレットの Amazon リソースネーム (ARN)。GitLab 省略すると、トークン自体がローテーションされます (`api`または `self_rotate`スコープが必要) 。プロジェクトトークンの場合、管理者トークンにはプロジェクトで メンテナンスロールが必要です。グループトークンには、グループの所有者ロールが必要です。

daysToExpiry

(オプション) 新しいトークンの有効期限が切れるまでの日数 (1 ~ 365)。GitLab ローテーション API の `expires_at`フィールドにマッピングします。省略すると、新しいトークンはインスタンスのデフォルトの有効期限を継承します。

使用フロー

このローテーションは、単一シークレット (セルフローテーション) アーキテクチャと 2 シークレット (管理者支援) アーキテクチャの両方をサポートしています。トークンの範囲は、オプションの

フィールドprojectIdと groupIdフィールドによって決まります。どちらのフィールドも存在しない場合、トークンは個人用アクセストークンです。projectId が存在する場合、トークンはプロジェクトアクセストークンです。groupId が存在する場合、トークンはグループアクセストークンです。

CreateSecret 呼び出しを使用してシークレットを作成します。シークレット値を上記のフィールドに設定し、シークレットタイプを GitLabAccessToken に設定します。ローテーションを設定するには、RotateSecret 呼び出しを使用します。シークレットをローテーションするために必要なアクセス許可をサービスに付与するロール ARN を指定します。アクセス許可ポリシーの例については、「セキュリティとアクセス許可」を参照してください。

管理者支援ローテーションを使用する場合、管理者シークレットもタイプ ですGitLabAccessToken。ローテーションロールに管理者シークレットへのアクセスを明示的に指定する必要があります。これを行うには、ロールポリシーで管理者シークレット ARN にスコープされたステートメントを直接追加します。

ローテーション中、ドライバーは現在のトークンがアクティブであることを検証します。次に GitLab ローテーションエンドポイントを呼び出します。これにより、新しいトークンがアトミックに作成され、古いトークンが取り消されます。Secrets Manager は、新しいトークン値と ID を AWSPENDING として保存し、GitLab API を介して検証し、AWSCURRENT に昇格します。Secrets Manager キャッシュライブラリを使用するアプリケーションは、次の更新時に新しいトークンを自動的に取得します。

Jenkins API トークン

シークレット値フィールド

以下は、Secrets Manager シークレットに含める必要があるフィールドです。

```
{
  "apiToken": "API token value",
  "tokenUuid": "token UUID",
  "username": "user that owns the token",
  "jenkinsUrl": "https://jenkins.example.com"
}
```

apiToken

Jenkins API トークン値 (32 ~ 34 文字の 16 進数文字列) 。ローテーションプロセスはこのフィールドを更新します。

tokenUuid

API トークンのユニバーサル一意識別子 (UUID)。ローテーションはこの UUID を使用して古いトークンを取り消し、ローテーションのたびに新しいトークンの UUID でこのフィールドを更新します。

ユーザー名

トークンを所有するJenkinsユーザー。

jenkinsUrl

Jenkins インスタンス URL (例: `https://jenkins.example.com`)。URL は HTTPS を使用する必要があり、末尾にスラッシュを付けることはできません。

シークレットメタデータフィールド

Jenkins API トークンのメタデータフィールドは次のとおりです。

```
{
  "adminSecretArn": "arn:aws:secretsmanager:us-east-1:111122223333:secret:JenkinsAdmin"
}
```

adminSecretArn

(オプション) 管理者トークンを含む JenkinsApiToken タイプのシークレットの Amazon リソースネーム (ARN)。管理者トークンは、ターゲットユーザーの API トークンを生成および取り消すことができます。省略すると、トークン自体がローテーションされます (自己ローテーション) 。

使用フロー

このローテーションは、単一シークレット (セルフローテーション) アーキテクチャと 2 シークレット (管理者支援) アーキテクチャの両方をサポートしています。を指定するとadminSecretArn、管理者トークンはターゲットユーザーのトークンを生成して取り消します。これを省略すると、現在のトークン自体がローテーションします。

シークレットを作成するには、CreateSecret API コールを使用します。シークレット値を上記のフィールドに設定し、シークレットタイプを JenkinsApiToken に設定します。ローテーションを設定するには、RotateSecret API コールを使用します。RotateSecret 呼び出しで、シークレットをローテーションするアクセス許可をサービスに付与するロール ARN を指定します。アクセス許可ポ

リシーの例については、「セキュリティとアクセス許可」を参照してください。管理者支援ローテーションを使用する場合は、ローテーションロールに管理者シークレット ARN へのアクセス権を明示的に付与します。

ローテーション中、ドライバーは現在のトークンを検証し、 から Cross-Site Request Forgery (CSRF) crumb を取得しますJenkins。次に、新しい API トークンを生成し、保留中のバージョンとして保存します。API に対して新しいトークンを検証しJenkins、現在のトークンに昇格してから、によって古いトークンを取り消しますtokenUuid。Secrets Manager キャッシュライブラリを使用するアプリケーションは、次回の更新時に新しいトークンを取得します。

Jenkins はカスタマーホスト型であるため、Jenkinsインスタンスは Secrets Manager ローテーションサービスから HTTPS 経由で到達可能である必要があります。ローテーション呼び出しは AWS、マネージドプレフィックスリスト から発信されますcom.amazonaws.*region*.secretsmanager-managed-external-secrets。インスタンスのセキュリティグループまたはファイアウォールでこのプレフィックスリストからのインバウンドアクセスを許可し、インスタンスがパブリックに信頼された TLS 証明書を提示していることを確認します。詳細については、「Amazon VPC ユーザーガイド」の「AWSマネージドプレフィックスリスト」を参照してください。パブリックインターネット経由で到達できないインスタンスはサポートされていません。

MongoDB Atlas データベースユーザー

シークレット値フィールド

以下は、Secrets Manager シークレットに含める必要があるフィールドです。

```
{
  "username": "database username",
  "password": "database password",
  "clusterUrl": "cluster hostname",
  "databaseName": "authentication database",
  "groupId": "Atlas Project ID"
}
```

ユーザー名

MongoDB データベースユーザー名 (SCRAM 認証) 。SCRAM 認証を受け入れるには、このユーザーを MongoDB Atlas で設定する必要があります。

password

MongoDB Atlas データベースユーザーの現在のパスワード。

clusterUrl

MongoDB Atlas クラスターのホスト名。例: `cluster0.abc123.mongodb.net`。`mongodb+srv://` プレフィックスは含めないでください。これは、ローテーション中に新しいパスワードを検証するために使用されます。

databaseName

ユーザーの認証情報が保存されている認証データベース。通常`admin`、SCRAM ユーザーまたは X.509/LDAP `$external`用です。

groupId

24 文字の 16 進数の Atlas プロジェクト ID (グループ ID とも呼ばれます) 。これは Atlas プロジェクト設定で確認できます。

シークレットメタデータフィールド

以下は、MongoDB Atlas データベースユーザーのメタデータフィールドです。

```
{
  "adminSecretArn": "arn:aws:secretsmanager:us-
east-1:111122223333:secret:MongoDBAtlasServiceAccount",
  "apiVersion": "2025-03-12"
}
```

adminSecretArn

プロジェクトデータベースアクセス管理者のアクセス許可を持つ Atlas サービスアカウントの OAuth 認証情報 (タイプ: MongoDBAtlasServiceAccount) を含むシークレットの Amazon リソースネーム (ARN)。この管理者シークレットは、パスワードの更新のために Atlas Admin API を認証するために使用されます。

apiVersion

(オプション) `yyyy-mm-dd`形式の Atlas Admin API バージョン日付。この値は、 `Accept` ヘッダーで として使用されます`application/vnd.atlas.{apiVersion}+json`。指定されない場合のデフォルト値は `2025-03-12` です。

使用フロー

このローテーションタイプでは、2 シークレットアーキテクチャを使用します。Atlas Admin API への認証には、Atlas サービスアカウント OAuth 認証情報 (`clientId`、`clientSecret`、`serviceAccountId`) を含む管理者シークレットが必要です。管理者シークレットのタイプは MongoDBAtlasServiceAccount である必要があります。

上記のフィールドと MongoDBAtlasDatabaseUser というシークレットタイプを含むシークレット値を使用してCreateSecret 呼び出しを使用してシークレットを作成できます。ローテーション設定は、RotateSecret 呼び出しを使用して設定できます。ローテーションメタデータ`adminSecretArn`で を指定する必要があります。また、RotateSecret 呼び出しでロール ARN を指定する必要があります。これにより、シークレットをローテーションするために必要なアクセス許可がサービスに付与されます。アクセス許可ポリシーの例については、「セキュリティとアクセス許可」を参照してください。

管理者シークレットはユーザーシークレット (MongoDBAtlasDatabaseUser) とは異なるタイプ (MongoDBAtlasServiceAccount) であるため、 によってスコープされるデフォルトのローテーションロールポリシー`secretsmanager:resource/Type`は、管理者シークレットへのアクセスを許可しません。 MongoDBAtlasDatabaseUser MongoDBAtlasServiceAccount タイプを対象とするステートメントを追加するか、ロールポリシーで管理者シークレット ARN を直接指定することで、管理者シークレットへのアクセスをローテーションロールに明示的に提供する必要があります。

ローテーション中、ドライバーは新しいパスワードを生成し、Atlas Admin API を呼び出してデータベースユーザーのパスワードを更新し、クラスターへの実際の MongoDB 接続を開いて新しいパスワードを検証します。クラスターの認証レイヤーが新しいパスワードを受け入れるまでに、パスワードの更新後 5～10 秒の伝播遅延があることに注意してください。

MongoDB Atlas サービスアカウントの認証情報

シークレット値フィールド

以下は、Secrets Manager シークレットに含める必要があるフィールドです。

```
{
  "clientId": "service account OAuth client ID",
  "clientSecret": "service account OAuth client secret",
  "orgId": "Atlas Organization ID"
}
```

clientId

MongoDB Atlas サービスアカウント OAuth クライアント ID。これは で始まりmdb_sa_id_、その後に 24 文字の 16 進数文字列が続く必要があります。

clientSecret

認証に使用される MongoDB Atlas サービスアカウント OAuth クライアントシークレット。

orgId

24 文字の 16 進数の Atlas Organization ID。これは Atlas Organization Settings にあります。

シークレットメタデータフィールド

以下は、MongoDB Atlas サービスアカウントのメタデータフィールドです。

```
{
  "adminSecretArn": "arn:aws:secretsmanager:us-
east-1:111122223333:secret:MongoDBAtlasServiceAccount",
  "apiVersion": "2025-03-12"
}
```

adminSecretArn

(オプション) このサービスアカウントのシークレットのローテーションに使用される管理サービスアカウントの OAuth 認証情報を含むシークレットの Amazon リソースネーム (ARN)。管理者シークレットには、シークレット構造内に clientIdと clientSecretの値が含まれている必要があります。省略すると、サービスアカウントは自己ローテーションに独自の認証情報を使用します。

apiVersion

(オプション) yyyy-mm-dd形式の Atlas Admin API バージョン日付。この値は、 Accept ヘッダーで として使用されますapplication/vnd.atlas.{apiVersion}+json。指定されない場合のデフォルト値は 2025-03-12 です。

使用フロー

ローテーションは 2 つの認証モードをサポートしています。自己ローテーションモード (デフォルト) では、サービスアカウントは独自の認証情報を使用してシークレットを作成および削除します。

これには、サービスアカウントに独自のシークレットを管理するアクセス許可が必要です。管理者支援ローテーションモードでは、別のシークレットに保存されている別の管理者サービスアカウントの認証情報が使用されます。これは、サービスアカウントに自己管理アクセス許可がない場合に必要です。

上記のフィールドと MongoDBAtlasServiceAccount というシークレットタイプを含むシークレット値を使用してCreateSecret 呼び出しを使用してシークレットを作成できます。ローテーション設定は、RotateSecret 呼び出しを使用して設定できます。セルフローテーションを選択した場合は、オプションの `adminSecretArn`フィールドを省略できます。RotateSecret 呼び出しでロール ARN を指定する必要があります。これにより、シークレットをローテーションするために必要なアクセス許可がサービスに付与されます。アクセス許可ポリシーの例については、「セキュリティとアクセス許可」を参照してください。

別の認証情報セット (管理者シークレットに保存) を使用してシークレットをローテーションすることを選択した場合は、管理者サービスアカウントの `clientId`と AWS Secrets Manager を含む に管理者シークレットを作成します`clientSecret`。この管理者シークレットの ARN は、サービスアカウントシークレットの RotateSecret 呼び出しのローテーションメタデータで指定する必要があります。

ローテーション中、ドライバーは Atlas Admin API を介してサービスアカウントの新しいシークレットを作成し、OAuth トークンを生成して新しいシークレットを検証し、新しい認証情報でシークレットを更新し、古いシークレットを削除します。

Netskope API トークン

シークレット値フィールド

以下は、Secrets Manager シークレットに含める必要があるフィールドです。

```
{
  "apiToken": "API token value",
  "tenantUrl": "https://example.goskope.com",
  "serviceAccountName": "service account name"
}
```

apiToken

サービスアカウントの Netskope REST API ベアラートークン。ローテーションプロセスはこのフィールドを更新します。

tenantUrl

Netskope テナント URL。例: `https://example.goskope.com`。URL は HTTPS を使用する必要があり、 で終わる必要があります。末尾にスラッシュを付けるgoskope.comことはできません。などのリージョンホストhttps://example.eu.goskope.comも有効です。

serviceAccountName

トークンを所有するNetskopeサービスアカウントの名前。Secrets Manager は、ログ記録と監査のためにこの名前を記録します。

また、ローテーションはシークレット値expiresDateに createdDateと を書き込みます。これらのフィールドは、現在のトークンの問題時間と有効期限を ISO 8601 形式で記録します。指定したフィールドではありません。

サービスアカウントロールはシークレットに保存されません。ロールはトークンではなくサービスアカウントに属しているため、ローテーションによって変更されることはありません。

シークレットメタデータフィールド

このシークレットタイプでは、ローテーションメタデータフィールドは必要ありません。管理者シークレットは使用されず、Secrets Manager はトークンの有効期間を 365 日間に設定します。

使用フロー

このローテーションでは、単一シークレットアーキテクチャを使用します。管理者シークレットは必要ありません。サービスアカウントは、現在のトークンを使用して独自の置き換えを生成します。

シークレットを作成するには、CreateSecret API コールを使用します。シークレット値を上記のフィールドに設定し、シークレットタイプを NetskopeApiToken に設定します。ローテーションを設定するには、RotateSecret API コールを使用します。RotateSecret 呼び出しで、シークレットをローテーションするアクセス許可をサービスに付与するロール ARN を指定します。アクセス許可ポリシーの例については、「セキュリティとアクセス許可」を参照してください。

ローテーション中、Secrets Manager は現在のトークンを使用してNetskopeサービスアカウントエンドポイントを呼び出し、置き換えを生成します。新しいトークンを保留中のバージョンとして保存し、NetskopeAPI に対して検証してから現在のトークンに昇格します。Secrets Manager は 350 日を超えるローテーション間隔を拒否し、すべての間隔を 365 日間のトークン有効期間内に保持します。

Netskope は、新しいトークンを生成するとすぐに古いトークンを無効にするため、2 つのトークンが重複することはありません。ローテーション中に約 30 秒間、現在のバージョンのシークレットは機能しなくなったトークンを保持します。アプリケーションは、シークレットを再度読み取って再試行することで、HTTP 401 レスポンスを処理する必要があります。このパターンなしでトークンをキャッシュするアプリケーションは、ローテーションのたびに失敗します。

サービスアカウントロールが IP 許可リストを使用している場合は、 AWSマネージドプレフィックスリストcom.amazonaws.*region*.secretsmanager-managed-external-secretsをその許可リストに追加します。ローテーション呼び出しはこのプレフィックスリストから発信され、許可リストで除外されている場合は HTTP 403 Netskopeを返します。詳細については、「Amazon VPC ユーザーガイド」の「AWSマネージドプレフィックスリスト」を参照してください。

Paddle API キー

シークレット値フィールド

以下は、Secrets Manager シークレットに含める必要があるフィールドです。

```
{
  "api_key": "Paddle API key value"
}
```

api_key

Paddle API キー値 (pdl_live_、、または で始まりpdl_test_、キーコンテンツapikey_がpdl_sdbx_続きます) 。これは、ローテーションされるフィールドです。

シークレットメタデータフィールド

Paddle API キーのメタデータフィールドは次のとおりです。

```
{
  "gracePeriodSeconds": "300 (optional)"
}
```

gracePeriodSeconds

(オプション) ローテーション後に古い API キーが有効なままになる秒数 (0 ~ 86400)。デフォルト: 300。古いキーをすぐに期限切れにするには、 を 0 に設定します。猶予期間のカウントダウンは、新しいキーが最初に正常に使用されたときに開始されます。

使用フロー

このローテーションでは、単一シークレットアーキテクチャを使用します。管理者シークレットは必要ありません。API キーは、Paddle ローテーションエンドポイントを介して自身をローテーションします。

CreateSecret 呼び出しを使用してシークレットを作成します。シークレット値を上記の フィールドに設定し、シークレットタイプを PaddleApiKey に設定します。ローテーションを設定するには、RotateSecret 呼び出しを使用します。シークレットをローテーションするために必要なアクセス許可をサービスに付与するロール ARN を指定します。アクセス許可ポリシーの例については、「セキュリティとアクセス許可」を参照してください。

ローテーション中、ドライバーはキー ID を抽出し、キープレフィックスから Paddle API ホストを解決します。次に、設定可能な猶予期間で Paddle ローテーションエンドポイントを呼び出します。Secrets Manager は新しいキーを AWSPENDING として保存し、変更しない API コールを実行して検証し、AWSCURRENT に昇格します。古いキーは、自動期限切れになる前の猶予期間 (デフォルトは 300 秒) の間有効です。

API キーは、Paddle Dashboard で「回転可能」オプションを有効にして作成する必要があります。Secrets Manager は、エラーのあるローテーション不可能なキーを拒否します。

Salesforce クライアントシークレット

シークレット値フィールド

以下は、Secrets Manager シークレットに含める必要があるフィールドです。

```
{
  "consumerKey": "client ID",
  "consumerSecret": "client secret",
  "baseUri": "https://domain.my.salesforce.com",
  "appId": "app ID",
  "consumerId": "consumer ID"
}
```

consumerKey

コンシューマーキーは、クライアント ID とも呼ばれ、OAuth 2.0 認証情報の認証情報識別子です。コンシューマーキーは、Salesforce External Client App Manager OAuth 設定から直接取得できます。

consumerSecret

コンシューマーシークレットは、クライアントシークレットとも呼ばれ、OAuth 2.0 クライアント認証情報フローを使用して認証するためにコンシューマーキーで使用されるプライベートパスワードです。コンシューマーシークレットは、Salesforce External Client App Manager OAuth 設定から直接取得できます。

baseUri

ベース URI は、Salesforce API とのやり取りに使用される Salesforce APIs。これは、次の例の形式になります: `https://domainName.my.salesforce.com`。

appId

アプリケーション ID は、Salesforce External Client Application (ECA) の識別子です。これは、Salesforce OAuth 使用状況エンドポイントを呼び出すことで取得できます。先頭は `0x`、英数字のみを使用する必要があります。このフィールドは、Salesforce ローテーションガイドの external_client_app_identifier を参照します。

consumerId

コンシューマー ID は、Salesforce External Client Application (ECA) コンシューマーの識別子です。これは、アプリケーション ID エンドポイントで Salesforce OAuth 認証情報を呼び出すことで取得できます。このフィールドは、Salesforce ローテーションガイドの consumer_id を参照します。

シークレットメタデータフィールド

Salesforce が保持するシークレットをローテーションするために必要なメタデータフィールドを次に示します。

```
{
  "apiVersion": "v65.0",
  "adminSecretArn": "arn:aws:secretsmanager:us-
east-1:111122223333:secret:SalesforceClientSecret"
}
```

apiVersion

Salesforce API バージョンは、Salesforce 組織の API バージョンです。バージョンは v65.0 以上である必要があります。*vXX.X X* が数値である形式である必要があります。

adminSecretArn

(オプション) 管理者シークレット ARN は、この Salesforce クライアントシークレットのローテーションに使用される管理 OAuth 認証情報を含むシークレットの Amazon リソースネーム (ARN) です。少なくとも、管理者シークレットには、シークレット構造内に consumerKey と consumerSecret の値を含める必要があります。これはオプションのフィールドであり、省略した場合、ローテーション中に Secrets Manager はこのシークレット内の OAuth 認証情報を使用して Salesforce で認証します。

使用フロー

では AWS Secrets Manager、同じシークレットの認証情報を使用して Salesforce シークレットをローテーションすることも、ローテーションのために管理者シークレットの認証情報を使用することもできます。上記のフィールドと CreateSecret 呼び出しを使用してシークレットを作成できます。SalesforceClientSecret ローテーション設定は、RotateSecret 呼び出しを使用して設定できます。この呼び出しでは、上記の例のようにメタデータフィールドの指定が必要です - 同じシークレットの認証情報を使用してローテーションを選択した場合は、adminSecretArn フィールドをスキップできます。さらに、お客様は RotateSecret 呼び出しでロール ARN を指定する必要があります。これにより、シークレットをローテーションするために必要なアクセス許可がサービスに付与されます。アクセス許可ポリシーの例については、「セキュリティとアクセス許可」を参照してください。

管理者シークレットに保存されている別の認証情報セットを使用してシークレットを更新するには、コンシューマーシークレットと同じステップ AWS Secrets Manager に従って で管理者シークレットを作成します。この管理者シークレットの ARN は、コンシューマーシークレットの RotateSecret 呼び出しのローテーションメタデータで指定する必要があります。

ローテーションロジックは、Salesforce が提供するガイダンスに従います。

Snowflake キーペア

シークレット値フィールド

以下は、Secrets Manager シークレットに含める必要があるフィールドです。

```
{
  "account": "Your Account Identifier",
  "user": "Your user name",
  "privateKey": "Your private Key",
  "publicKey": "Your public Key",
```

```
  "passphrase": "Your Passphrase"
}
```

ユーザー

このキーペア認証に関連付けられた Snowflake ユーザー名。このユーザーは、キーペア認証を受け入れるように Snowflake で設定する必要があり、パブリックキーはこのユーザーのプロファイルに割り当てる必要があります。

アカウント

接続の確立に使用される Snowflake アカウント識別子。これは Snowflake URL (.snowflakecomputing.com より前の部分) から抽出できます。

privateKey

認証に使用される PEM 形式の RSA プライベートキー。BEGIN/END マーカーはオプションです。

publicKey

プライベートキーに対応する PEM 形式のパブリックキー。BEGIN/END マーカーはオプションです。

パスフレーズ

(オプション) このフィールドは、暗号化されたプライベートキーの復号に使用されるパスフレーズを参照します。

シークレットメタデータフィールド

Snowflake のメタデータフィールドは次のとおりです。

```
{
  "cryptographicAlgorithm": "Your Cryptographic algorithm",
  "encryptPrivateKey": "True/False"
}
```

cryptographicAlgorithm

(オプション) これは、キー生成に使用されるアルゴリズムを指します。次の 3 つのアルゴリズムを選択できます: RS256|RS384|RS512 。このフィールドはオプションで、選択したデフォルトのアルゴリズムは RS256 です。

encryptPrivateKey

(オプション) このフィールドを使用して、プライベートキーを暗号化するかどうかを選択できます。これはデフォルトでは false です。暗号化のパスフレーズはランダムに生成されます。

使用フロー

上記のフィールドと SnowflakeKeyPairAuthentication としてシークレットタイプを含むシークレット値を使用してCreateSecret 呼び出しを使用してシークレットを作成できます。ローテーション設定は、RotateSecret 呼び出しを使用して設定できます。必要に応じて、要件に基づいてシークレットメタデータフィールド (複数可) を指定できます。また、RotateSecret 呼び出しでロール ARN を指定する必要があります。これにより、シークレットをローテーションするために必要なアクセス許可がサービスに付与されます。アクセス許可ポリシーの例については、「セキュリティとアクセス許可」を参照してください。このパートナーでは、ローテーションメタデータフィールドを空のままにできることに注意してください。

Snowflake プログラムによるアクセストークン

シークレット値フィールド

以下は、Secrets Manager シークレットに含める必要があるフィールドです。

```
{
  "account": "Snowflake account identifier",
  "user": "Snowflake username",
  "privateKey": "PEM-encoded private key",
  "passphrase": "private key passphrase (optional)",
  "patTokenName": "PAT name",
  "patTokenValue": "PAT secret value"
}
```

アカウント

Snowflake アカウント識別子 (例: `myorg-myaccount`)。これは Snowflake URL `.snowflakecomputing.com`の より前の部分です。

ユーザー

PAT を所有する Snowflake ユーザー名。このユーザーには、キーペア認証が設定されている必要があります。

privateKey

キーペア認証用の PEM エンコードされたプライベートキー。このキーはローテーションされません。これは ROTATE PAT コマンドの認証に使用されます (PAT 自体はローテーションできません)。

パスフレーズ

(オプション) 暗号化されたプライベートキーのパスフレーズ。プライベートキーが暗号化されていない場合は空のままにします。

patTokenName

ローテーションするプログラムによるアクセストークンの名前。Snowflake のトークン名と一致する必要があります。

patTokenValue

プログラムによるアクセストークンのシークレット値。これはローテーションされるフィールドです。

シークレットメタデータフィールド

以下は、Snowflake プログラムによるアクセストークンのメタデータフィールドです。

```
{
  "daysToExpiry": "15",
  "expireOldTokenAfterHours": "24"
}
```

daysToExpiry

(オプション) 作成時 (1～365) に設定された PAT の DAYS_TO_EXPIRY 値。デフォルト: 15。Snowflake 設定と一致する必要があります。ローテーションスケジュールがトークンの TTL よりも短いことを検証するために使用されます。

expireOldTokenAfterHours

(オプション) 前のトークンがローテーション後に期限切れになるまでの時間 (0～720)。デフォルト: 24。古いトークンをすぐに期限切れにするには、 を 0 に設定します。

使用フロー

このローテーションでは、単一シークレットアーキテクチャを使用します。シークレットには、キーペア認証情報 (ローテーションコマンドの認証用) と PAT 値 (ローテーションされた認証情報) の両方が含まれます。

上記のフィールドと SnowflakePat としてのシークレットタイプを含むシークレット値を使用してCreateSecret 呼び出しを使用してシークレットを作成できます。ローテーション設定は、RotateSecret 呼び出しを使用して設定できます。デフォルト値を使用するには、ローテーションメタデータフィールドを空のままにすることができます。RotateSecret 呼び出しでロール ARN を指定する必要があります。これにより、シークレットをローテーションするために必要なアクセス許可がサービスに付与されます。アクセス許可ポリシーの例については、「セキュリティとアクセス許可」を参照してください。

ローテーション中、ドライバーはキーペア認証を介して Snowflake に接続し、 `ALTER USER ... ROTATE PAT` コマンドを実行します。これにより、新しいトークンがアトミックに生成され、設定された猶予期間で古いトークンが期限切れになります。次に、新しいトークンをパスワードとして接続して検証します。

SonarQube トークン

シークレット値フィールド

以下は、Secrets Manager シークレットに含める必要があるフィールドです。

```
{
  "token": "token value",
  "tokenName": "token name",
  "login": "user that owns the token",
  "sonarqubeUrl": "https://sonarqube.example.com",
  "tokenType": "USER_TOKEN",
  "projectKey": "project key"
}
```

トークン

SonarQube トークン値 (squ_、、sqa_または で始まりますsqp_)。ローテーションプロセスはこのフィールドを更新します。

tokenName

トークン名 (ユーザーごとに一意) 。ローテーションは、この名前を使用して古いトークンを取り消します。

login (ログイン)

トークンを所有するSonarQubeユーザー。

sonarqubeUrl

SonarQube インスタンス URL (例: `https://sonarqube.example.com`)。URL は HTTPS を使用する必要があり、末尾にスラッシュを付けることはできません。

tokenType

(オプション) トークンのタイプ: `USER_TOKEN` (デフォルト)`GLOBAL_ANALYSIS_TOKEN`、、または `PROJECT_ANALYSIS_TOKEN`。

projectKey

(オプション) SonarQubeプロジェクトキー。`tokenType` が の場合にのみ必要です`PROJECT_ANALYSIS_TOKEN`。

シークレットメタデータフィールド

SonarQube トークンのメタデータフィールドは次のとおりです。

```
{
  "adminSecretArn": "arn:aws:secretsmanager:us-
east-1:111122223333:secret:SonarQubeAdmin"
}
```

adminSecretArn

(オプション) 管理者トークンを含む SonarQubeToken タイプのシークレットの Amazon リソースネーム (ARN)。このフィールドは、グローバル分析トークンとプロジェクト分析トークンに必要です。ユーザートークンではオプションです。これを省略してセルフローテーションを有効にします。この場合、現在のトークンは独自の置換を生成して取り消します。

使用フロー

このローテーションは、ユーザートークンの単一シークレット (セルフローテーション) と、グローバル分析トークンとプロジェクト分析トークンの 2 シークレット (管理者アシスト) ローテーションをサポートします。tokenType 値はトークンの範囲を決定します。

シークレットを作成するには、CreateSecret API コールを使用します。シークレット値を上記のフィールドに設定し、シークレットタイプを SonarQubeToken に設定します。ローテーションを設定するには、RotateSecret API コールを使用します。RotateSecret 呼び出しで、シークレットをローテーションするアクセス許可をサービスに付与するロール ARN を指定します。アクセス許可ポリシーの例については、「セキュリティとアクセス許可」を参照してください。管理者アシストローテーションを使用する場合は、ローテーションロールに管理者シークレット ARN へのアクセス権を明示的に付与します。

ローテーション中、ドライバーは新しいトークンを生成し、保留中のバージョンとして保存し、SonarQubeAPI に対して検証し、最新に昇格してから、古いトークンを によって取り消しますtokenName。Secrets Manager キャッシュライブラリを使用するアプリケーションは、次の更新時に新しいトークンを取得します。

SonarQube サーバーはカスタマーホストであるため、SonarQubeインスタンスは Secrets Manager ローテーションサービスから HTTPS 経由で到達可能である必要があります。ローテーション呼び出しは、 AWSマネージドプレフィックスリスト から発信されますcom.amazonaws.*region*.secretsmanager-managed-external-secrets。インスタンスのセキュリティグループまたはファイアウォールでこのプレフィックスリストからのインバウンドアクセスを許可し、インスタンスがパブリックに信頼された TLS 証明書を提示していることを確認します。詳細については、「Amazon VPC ユーザーガイド」の「AWSマネージドプレフィックスリスト」を参照してください。パブリックインターネット経由で到達できないインスタンスはサポートされていません。

Twilio API キー

シークレット値フィールド

以下は、Secrets Manager シークレットに含める必要があるフィールドです。

```
{
  "accountSid": "AC followed by 32 hexadecimal characters",
  "apiKeySid": "SK followed by 32 hexadecimal characters",
  "apiKeySecret": "API key secret",
```

```
  "validationUrl": "https://api.twilio.com/2010-04-01/Accounts/ACxxxx/Messages.json?
PageSize=1"
}
```

accountSid

API キーを所有するTwilioアカウントまたはサブアカウントの SID。

apiKeySid

現在の API キーの SID。標準キーと制限付きキーの両方がサポートされています。

apiKeySecret

現在の API キー SID とペアになっているシークレット。Secrets Manager は、このフィールドを`apiKeySid`と一緒にローテーションします。

validationUrl

(オプション) Secrets Manager が新しいキーが機能することを確認するために呼び出す読み取り専用 Twilio API URL。URL は HTTPS と`twilio.com`ホストを使用する必要があります。

このフィールドは、制限付きキーに必要です。制限付きキーは、ポリシーで許可されているエンドポイントのみを呼び出すことができるため、ポリシーで許可されていることがわかっているエンドポイントを指定します。標準キーの場合、Secrets Manager はデフォルトのエンドポイントを取得します。

シークレットメタデータフィールド

Twilio API キーのメタデータフィールドは次のとおりです。

```
{
  "adminSecretArn": "arn:aws:secretsmanager:us-
east-1:111122223333:secret:TwilioMainApiKey",
  "retainPreviousKey": "false",
  "friendlyNamePrefix": "aws-sm-api-key"
}
```

adminSecretArn

管理者認証情報を保持するシークレットの Amazon リソースネーム (ARN)。TwilioMainApiKey タイプのシークレットまたは TwilioAuthToken タイプのシークレットを使用します。

retainPreviousKey

(オプション) 昇格後も以前のキーがアクティブのままであるかどうか。デフォルト: false。これにより、前のキーがすぐに削除されます。このフィールドを true に設定すると、アプリケーションがキャッシュされた認証情報を更新できるように、前のキーを 1 つのローテーションサイクルで使用できます。

friendlyNamePrefix

(オプション) 各置換キーのフレンドリ名のプレフィックス。デフォルト: `aws-sm-api-key`。

使用フロー

このローテーションでは、2 つのシークレットアーキテクチャを使用します。キー管理エンドポイントは Twilio API キー認証を拒否するため、API Twilio キー自体をローテーションすることはできません。したがって、管理者シークレットは、 がキーを作成および削除する認証情報を提供します。

上記のフィールドと TwilioApiKey としてのシークレットタイプを含むシークレット値を使用してCreateSecret 呼び出しを使用してシークレットを作成できます。ローテーション設定は、RotateSecret 呼び出しを使用して設定できます。ローテーションメタデータadminSecretArnで を指定する必要があります。また、RotateSecret 呼び出しでロール ARN を指定する必要があります。これにより、シークレットをローテーションするために必要なアクセス許可がサービスに付与されます。アクセス許可ポリシーの例については、「セキュリティとアクセス許可」を参照してください。

管理者シークレットタイプ (TwilioMainApiKey または TwilioAuthToken) は、ユーザーシークレットタイプ () とは異なりますTwilioApiKey。この違いにより、 によってスコープされるデフォルトのローテーションロールポリシーsecretsmanager:resource/Typeは、管理者シークレットへのアクセスを許可しません。ローテーションロールに管理者シークレットへのアクセス権を明示的に付与する必要があります。これを行うには、管理者シークレットタイプに限定されたステートメントを追加します。または、ロールポリシーで管理者シークレットARNを直接指定します。

ローテーション中、Secrets Manager は現在のキーのタイプとポリシーを読み取ります。同じタイプとポリシーを持つ置換キーを作成し、AWSPENDING と同じ置換キーを保存します。これにより、新しいキーが検証エンドポイントに対して認証されることを確認します。次に、前のキーを削除し、AWSPENDING を AWSCURRENT に昇格します。true retainPreviousKeyに設定すると、Secrets Manager は前のキーを保持し、次のローテーション中に削除します。

各置換キーは、置き換えるキーのタイプとポリシーを継承するため、制限付きキーは同じアクセス許可で制限されたままになります。Secrets Manager が現在のキーのタイプまたはポリシーを読み取れない場合、異なる権限を持つキーを作成する代わりにローテーションは失敗します。

Twilio 認証トークン

シークレット値フィールド

以下は、Secrets Manager シークレットに含める必要があるフィールドです。

```
{
  "accountSid": "AC followed by 32 hexadecimal characters",
  "authToken": "32-character hexadecimal Auth Token",
  "validationUrl": "https://api.twilio.com/2010-04-01/Accounts/ACxxxx/Messages.json?
PageSize=1"
}
```

accountSid

Auth Token Secrets Manager がローテーションするTwilioアカウント SID。

authToken

アカウントの現在のプライマリ認証トークン。Secrets Manager はこのフィールドをローテーションします。

validationUrl

(オプション) Secrets Manager が昇格されたトークンが機能することを確認するために呼び出す読み取り専用 Twilio API URL。URL は HTTPS とtwilio.comホストを使用する必要があります。Secrets Manager は、このフィールドを省略するとデフォルトのエンドポイントを取得します。

シークレットメタデータフィールド

Twilio 認証トークンのメタデータフィールドは次のとおりです。

```
{
  "propagationDelaySeconds": "60"
}
```

propagationDelaySeconds

(オプション) Secrets Manager が昇格された認証トークンを検証するまで待機する秒数。最大数は 300 です。デフォルト: Twilio関数の伝播をカバーする 60。

遅延は、AWSCURRENT が前のトークンに名前を付けるウィンドウを拡張します。アプリケーションが Twilio Functions を使用しない場合は、このフィールドを 0 に設定して、そのウィンドウを短縮します。

使用フロー

このローテーションタイプは自己ローテーションします。Secrets Manager はアカウントの現在のプライマリ認証トークンで認証されるため、個別の管理者認証情報は必要なく、ローテーションメタデータには必須フィールドがありません。

上記のフィールドと TwilioAuthToken としてシークレットタイプを含むシークレット値を使用してCreateSecret 呼び出しを使用してシークレットを作成できます。ローテーション設定は、RotateSecret 呼び出しを使用して設定できます。RotateSecret 呼び出しでロール ARN を指定する必要があります。これにより、シークレットをローテーションするために必要なアクセス許可がサービスに付与されます。アクセス許可ポリシーの例については、「セキュリティとアクセス許可」を参照してください。

ローテーション中、Secrets Manager はセカンダリ認証トークンを作成し、AWSPENDING として保存します。これにより、セカンダリトークンがプライマリに昇格され、以前のプライマリトークンがすぐに無効になります。を待機しpropagationDelaySeconds、昇格されたトークンが認証されることを確認し、AWSPENDING を AWSCURRENT に昇格します。

プロモーションは、新しいトークンを使用可能にし、以前のトークンを同時に無効にします。したがって、このタイプには、両方のトークンが認証される重複ウィンドウはありません。認証トークンをキャッシュするアプリケーションは、接続を開いたときに取得するか、AWSCURRENT を再読み込みして認証の失敗後に再試行する必要があります。重複ウィンドウの場合は、 で Twilio API キーを使用しますretainPreviousKey。

Twilio メイン API キー

シークレット値フィールド

以下は、Secrets Manager シークレットに含める必要があるフィールドです。

```
{
  "accountSid": "AC followed by 32 hexadecimal characters",
  "adminKeySid": "SK followed by 32 hexadecimal characters",
  "adminKeySecret": "Main API key secret"
}
```

accountSid

メイン API キーを所有するTwilioアカウント SID。

adminKeySid

Secrets Manager が管理者認証情報として使用するメイン API キーの SID。

adminKeySecret

メイン API キー SID とペアになったシークレット。

シークレットメタデータフィールド

Secrets Manager はローテーションしないため、このシークレットタイプにはメタデータフィールドがありません。

使用フロー

このシークレットタイプは、TwilioApiKey タイプのシークレットをローテーションするときに、Secrets Manager が認証情報の作成と削除に使用するTwilioメイン API キーを保存します。Secrets Manager は、このタイプのシークレットをローテーションしません。

Twilio は、メイン API キーを作成する API オペレーションを提供しないため、Twilioコンソールでこの認証情報のライフサイクルを管理します。

上記のフィールドと TwilioMainApiKey としてシークレットタイプを含むシークレット値を使用してCreateSecret 呼び出しを使用してシークレットを作成できます。このシークレットのローテーションを設定しないでください。ローadminSecretArnテーションメタデータフィールドを通じてTwilioApiKey タイプのシークレットから参照します。

手動で管理される管理者認証情報を回避するには、代わりに TwilioAuthToken タイプのシークレットadminSecretArnを指定します。このタイプはそれ自体をローテーションし、それ自体の管理者認証情報は必要ありません。

セキュリティおよびアクセス許可

マネージド外部シークレットでは、サードパーティーアプリケーションアカウントの管理者レベルの権限を と共有する必要はありません AWS。代わりに、ローテーションプロセスでは、提供された認証情報とメタデータを使用して、認証情報の更新と検証のためにサードパーティーアプリケーションへの認可された API コールを行います。

マネージド外部シークレットは、他の Secrets Manager シークレットタイプと同じセキュリティ標準を維持します。シークレット値は、KMS キーを使用して保管中に暗号化され、TLS を使用して転送中に暗号化されます。シークレットへのアクセスは、IAM ポリシーとリソースベースのポリシーによって制御されます。カスタマーマネージドキーを使用してシークレットを暗号化する場合は、ローテーションロールの IAM ポリシーと CMK 信頼ポリシーを更新して、ローテーションを成功させるために必要なアクセス許可を付与する必要があります。

ローテーションを適切に機能させるには、シークレットライフサイクルを管理するための特定のアクセス許可を Secrets Manager に提供する必要があります。これらのアクセス許可は、個々のシークレットに限定し、最小特権の原則に従うことができます。指定したローテーションロールはセットアップ時に検証され、ローテーションオペレーションにのみ使用されます。

IP イングレスを外部リソースに制限するには、シークレットが存在するリージョンで Secrets Manager が管理する外部シークレット AWSマネージドプレフィックスリストのみを許可します。プレフィックスリストの名前は でcom.amazonaws.*region*.secretsmanager-managed-external-secrets、*region* はシークレットの AWS リージョンです。マネージドプレフィックスリストを使用すると、基になる IP 範囲が変更されても進入ルールが自動的に最新の状態になります。

プログラムでプレフィックスリストを参照する場合は、GetManagedPrefixListEntries API を使用して、許可リストに登録する必要がある IPsを一覧表示できます。このプレフィックスリストを使用して進入ルールを定期的に更新して、最新の状態に保つ必要があります。

AWS Secrets Manager には、Secrets Manager コンソールを使用してシークレットを作成するときにシークレットを管理するために必要なアクセス許可を持つ IAM ポリシーを作成するためのシングルタッチソリューションも用意されています。このロールのアクセス許可は、各リージョンの統合パートナーごとにスコープダウンされます。

アクセス許可ポリシーの例:

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

```
  "Statement": [
    {
      "Sid": "AllowRotationAccess",
      "Action": [
        "secretsmanager:DescribeSecret",
        "secretsmanager:GetSecretValue",
        "secretsmanager:PutSecretValue",
        "secretsmanager:UpdateSecretVersionStage"
      ],
      "Resource": "*",
      "Effect": "Allow",
      "Condition": {
        "StringEquals": {
          "secretsmanager:resource/Type": "SalesforceClientSecret"
        }
      }
    },
    {
      "Sid": "AllowPasswordGenerationAccess",
      "Action": [
        "secretsmanager:GetRandomPassword"
      ],
      "Resource": "*",
      "Effect": "Allow"
    }
  ]
}
```

注: secretsmanager:resource/Type で使用できるシークレットタイプのリストは、統合パートナーにあります。

信頼ポリシーの例:

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

```
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "111122223333"
        },
        "ArnLike": {
          "aws:SourceArn": "arn:aws:secretsmanager:us-east-1:111122223333:secret:*"
        }
      }
    }
  ]
}
```

マネージド外部シークレットのモニタリングとトラブルシューティング

マネージド外部シークレットは、 AWS CloudTrail ログと Amazon CloudWatch メトリクスを通じて包括的なモニタリング機能を提供します。すべてのローテーションアクティビティは、成功、失敗、およびプロセス中に発生したエラーに関する詳細情報とともにログに記録されます。

ローテーションワークフローの一般的な問題には、ロールのアクセス許可またはシークレット値の誤った設定が含まれます。これらのフィールドを設定しないと、サービスがシークレットにアクセスしたり、統合パートナークライアントに接続してシークレットを更新したりできなくなるため、統合パートナーによって指定された形式でローテーションが失敗する可能性があります。その他の問題には、ネットワーク接続の問題、認証情報の有効期限、またはパートナーサービスの可用性などがあります。マネージドローテーションサービスには、信頼性を最大化するための再試行ロジックとエラー処理が含まれています。

Amazon CloudWatch を使用して、ローテーションスケジュール、成功率、パフォーマンスメトリクスをモニタリングできます。イベントブリッジを介してカスタムアラームを設定して、ローテーションの失敗や注意が必要なその他の問題を警告できます。

既存のシークレットの移行

既存のパートナーシークレットを管理外部シークレットに移行するオプションがあります。これは UpdateSecret 呼び出しで実行できます。ガイドで説明されているように、シークレット値とメタデータを更新する必要があります。これらのシークレットにカスタムローテーションロジックがすでに設定されている場合は、まず CancelRotateSecret 呼び出しを使用してローテーションをキャンセルする必要があります。

制約事項と考慮事項

マネージド外部シークレットは、有効期間が 4 時間未満のエフェメラルシークレットをサポートしていません。パブリックキーインフラストラクチャ証明書に関連付けられたシークレットもサポートされていません。

マネージド外部シークレットは、 をオンボーディングしたパートナーでのみサポートされます AWS Secrets Manager。詳細なリストについては、「 統合パートナー」を参照してください。パートナーがリストに表示されない場合 にオンボードするように指示する AWS Secrets Manager

Secrets Manager ローテーションエンジンの外部にあるパートナークライアントサービスから直接シークレット値を更新またはローテーションすると、システム間の同期が中断される可能性があります。Secrets Manager は、手動シークレット値更新のコンソール警告とプログラムによる防止を提供しますが、サードパーティーアプリケーションで直接値を変更することはできます。out-of-band 更新後に同期を再確立するには、正しいシークレットを反映するようにシークレット値を更新してからRotateSecret API を呼び出して、ローテーションが引き続き成功するようにする必要があります。

で AWS Secrets Manager シークレットを作成する AWS CloudFormation

シークレットを作成する に示すように、CloudFormation テンプレートで AWS::SecretsManager::Secret リソースを使用して、CloudFormation スタックでシークレットを作成できます。

Amazon RDS または Aurora の管理者シークレットを作成するには、AWS::RDS::DBCluster で ManageMasterUserPassword を使用することをお勧めします。次に、Amazon RDS がシークレットを作成し、ローテーションを管理します。詳細については、「マネージドローテーション」を参照してください。

Amazon Redshift および Amazon DocumentDB の認証情報については、最初に Secrets Manager によって生成されたパスワードでシークレットを作成し、動的参照を使用してシークレットからユーザー名とパスワードを取得し、新しいデータベースの認証情報として使用します。次に、 AWS::SecretsManager::SecretTargetAttachment リソースを使用して、Secrets Manager がシークレットをローテーションするために必要なシークレットに、データベースに関する詳細を追加します。最後に、自動ローテーションを有効にするには、AWS::SecretsManager::RotationSchedule リソースを使用して、ローテーション関数 と スケジュール を提供します。以下の例を参照してください。

- Amazon Redshift 認証情報を使用してシークレットを作成する
- Amazon DocumentDB 認証情報を使用してシークレットを作成する

シークレットにリソースポリシーをアタッチするには、AWS::SecretsManager::ResourcePolicy リソースを使用します。

でリソースを作成する方法については CloudFormation、「 CloudFormation ユーザーガイド」の「テンプレートの基本を学ぶ」を参照してください。また、 AWS Cloud Development Kit (AWS CDK)を使用することもできます 詳細については、AWS Secrets Manager Construct ライブラリを参照してください。

を使用して AWS Secrets Manager シークレットを作成する CloudFormation

この例では、**CloudFormationCreatedSecret-*a1b2c3d4e5f6*** という名前のシークレットが作成されます。シークレット値は次の JSON で、シークレットの作成時に生成された 32 文字のパスワードが使用されます。

```
{
    "password": "EXAMPLE-PASSWORD",
    "username": "saanvi"
}
```

この例では、次の CloudFormation リソースが使用されています。

- AWS::SecretsManager::Secret

でリソースを作成する方法については CloudFormation、「 CloudFormation ユーザーガイド」の「テンプレートの基本を学ぶ」を参照してください。

JSON

```
{
    "Resources": {
        "CloudFormationCreatedSecret": {
            "Type": "AWS::SecretsManager::Secret",
            "Properties": {
                "Description": "Simple secret created by CloudFormation.",
                "GenerateSecretString": {
                    "SecretStringTemplate": "{\"username\": \"saanvi\"}",
                    "GenerateStringKey": "password",
                    "PasswordLength": 32
                }
            }
        }
    }
}
```

YAML

```
Resources:
  CloudFormationCreatedSecret:
    Type: 'AWS::SecretsManager::Secret'
    Properties:
      Description: Simple secret created by CloudFormation.
      GenerateSecretString:
        SecretStringTemplate: '{"username": "saanvi"}'
        GenerateStringKey: password
        PasswordLength: 32
```

自動ローテーションを使用して AWS Secrets Manager シークレットを作成し、 を使用して Amazon RDS MySQL DB インスタンスを作成する CloudFormation

Amazon RDS または Aurora の管理者シークレットを作成するには、`AWS::RDS::DBCluster` でマスター パスワードの Secrets Manager シークレットを作成する例に示すように、`ManageMasterUserPassword` を使用することをお勧めします。次に、Amazon RDS がシークレットを作成し、ローテーションを管理します。詳細については、マネージドローテーション を参照してください。

を使用して AWS Secrets Manager シークレットと Amazon Redshift クラスターを作成する CloudFormation

Amazon Redshift の管理者シークレットを作成するには、「`AWS::Redshift::Cluster`」や「`AWS::RedshiftServerless::Namespace`」に記載されている例を使用することをお勧めします。

を使用して AWS Secrets Manager シークレットと Amazon DocumentDB インスタンスを作成する CloudFormation

次の例では、シークレットと、そのシークレット内の認証情報をユーザーおよびパスワードとして使用する Amazon DocumentDB インスタンスを作成します。シークレットには、シークレットにアクセスできるユーザーを定義するリソースベースのポリシーがアタッチされています。また、このテンプレートでは、ローテーション関数のテンプレート から Lambda ローテーション関数を作成し、こ

のシークレットが毎月の最初の日の午前 8 時から午前 10 時 (UTC) に自動的にローテーションされるように設定します。セキュリティのベストプラクティスとして、インスタンスは Amazon VPC に置かれています。

この例では、Secrets Manager に次の CloudFormation リソースが使用されています。

- AWS::SecretsManager::Secret
- AWS::SecretsManager::SecretTargetAttachment
- AWS::SecretsManager::RotationSchedule

でリソースを作成する方法については CloudFormation、「 CloudFormation ユーザーガイド」の「テンプレートの基本を学ぶ」を参照してください。

JSON

```
{
   "AWSTemplateFormatVersion":"2010-09-09",
   "Transform":"AWS::SecretsManager-2020-07-23",
   "Resources":{
      "TestVPC":{
         "Type":"AWS::EC2::VPC",
         "Properties":{
            "CidrBlock":"10.0.0.0/16",
            "EnableDnsHostnames":true,
            "EnableDnsSupport":true
         }
      },
      "TestSubnet01":{
         "Type":"AWS::EC2::Subnet",
         "Properties":{
            "CidrBlock":"10.0.96.0/19",
            "AvailabilityZone":{
               "Fn::Select":[
                  "0",
                  {
                     "Fn::GetAZs":{
                        "Ref":"AWS::Region"
                     }
                  }
               ]
            },
```

```
            "VpcId":{
               "Ref":"TestVPC"
            }
         }
      },
      "TestSubnet02":{
         "Type":"AWS::EC2::Subnet",
         "Properties":{
            "CidrBlock":"10.0.128.0/19",
            "AvailabilityZone":{
               "Fn::Select":[
                  "1",
                  {
                     "Fn::GetAZs":{
                        "Ref":"AWS::Region"
                     }
                  }
               ]
            },
            "VpcId":{
               "Ref":"TestVPC"
            }
         }
      },
      "SecretsManagerVPCEndpoint":{
         "Type":"AWS::EC2::VPCEndpoint",
         "Properties":{
            "SubnetIds":[
               {
                  "Ref":"TestSubnet01"
               },
               {
                  "Ref":"TestSubnet02"
               }
            ],
            "SecurityGroupIds":[
               {
                  "Fn::GetAtt":[
                     "TestVPC",
                     "DefaultSecurityGroup"
                  ]
               }
            ],
            "VpcEndpointType":"Interface",
```

```
            "ServiceName":{
               "Fn::Sub":"com.amazonaws.${AWS::Region}.secretsmanager"
            },
            "PrivateDnsEnabled":true,
            "VpcId":{
               "Ref":"TestVPC"
            }
         }
      },
      "MyDocDBClusterRotationSecret":{
         "Type":"AWS::SecretsManager::Secret",
         "Properties":{
            "GenerateSecretString":{
               "SecretStringTemplate":"{\"username\": \"someadmin\",\"ssl\": true}",
               "GenerateStringKey":"password",
               "PasswordLength":16,
               "ExcludeCharacters":"\"@/\\"
            },
            "Tags":[
               {
                  "Key":"AppName",
                  "Value":"MyApp"
               }
            ]
         }
      },
      "MyDocDBCluster":{
         "Type":"AWS::DocDB::DBCluster",
         "Properties":{
            "DBSubnetGroupName":{
               "Ref":"MyDBSubnetGroup"
            },
            "MasterUsername":{
               "Fn::Sub":"{{resolve:secretsmanager:
${MyDocDBClusterRotationSecret}::username}}"
            },
            "MasterUserPassword":{
               "Fn::Sub":"{{resolve:secretsmanager:
${MyDocDBClusterRotationSecret}::password}}"
            },
            "VpcSecurityGroupIds":[
               {
                  "Fn::GetAtt":[
                     "TestVPC",
```

```
                    "DefaultSecurityGroup"
                ]
            }
        ]
    }
},
"DocDBInstance":{
    "Type":"AWS::DocDB::DBInstance",
    "Properties":{
        "DBClusterIdentifier":{
            "Ref":"MyDocDBCluster"
        },
        "DBInstanceClass":"db.r5.large"
    }
},
"MyDBSubnetGroup":{
    "Type":"AWS::DocDB::DBSubnetGroup",
    "Properties":{
        "DBSubnetGroupDescription":"",
        "SubnetIds":[
            {
                "Ref":"TestSubnet01"
            },
            {
                "Ref":"TestSubnet02"
            }
        ]
    }
},
"SecretDocDBClusterAttachment":{
    "Type":"AWS::SecretsManager::SecretTargetAttachment",
    "Properties":{
        "SecretId":{
            "Ref":"MyDocDBClusterRotationSecret"
        },
        "TargetId":{
            "Ref":"MyDocDBCluster"
        },
        "TargetType":"AWS::DocDB::DBCluster"
    }
},
"MySecretRotationSchedule":{
    "Type":"AWS::SecretsManager::RotationSchedule",
    "DependsOn":"SecretDocDBClusterAttachment",
```

```
         "Properties":{
            "SecretId":{
               "Ref":"MyDocDBClusterRotationSecret"
            },
            "HostedRotationLambda":{
               "RotationType":"MongoDBSingleUser",
               "RotationLambdaName":"MongoDBSingleUser",
               "VpcSecurityGroupIds":{
                  "Fn::GetAtt":[
                     "TestVPC",
                     "DefaultSecurityGroup"
                  ]
               },
               "VpcSubnetIds":{
                  "Fn::Join":[
                     ",",
                     [
                        {
                           "Ref":"TestSubnet01"
                        },
                        {
                           "Ref":"TestSubnet02"
                        }
                     ]
                  ]
               }
            },
            "RotationRules":{
              "Duration": "2h",
              "ScheduleExpression": "cron(0 8 1 * ? *)"
            }
         }
      }
   }
}
```

YAML

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::SecretsManager-2020-07-23
Resources:
  TestVPC:
    Type: AWS::EC2::VPC
```

```
    Properties:
      CidrBlock: 10.0.0.0/16
      EnableDnsHostnames: true
      EnableDnsSupport: true
  TestSubnet01:
    Type: AWS::EC2::Subnet
    Properties:
      CidrBlock: 10.0.96.0/19
      AvailabilityZone: !Select
        - '0'
        - !GetAZs
          Ref: AWS::Region
      VpcId: !Ref TestVPC
  TestSubnet02:
    Type: AWS::EC2::Subnet
    Properties:
      CidrBlock: 10.0.128.0/19
      AvailabilityZone: !Select
        - '1'
        - !GetAZs
          Ref: AWS::Region
      VpcId: !Ref TestVPC
  SecretsManagerVPCEndpoint:
    Type: AWS::EC2::VPCEndpoint
    Properties:
      SubnetIds:
        - !Ref TestSubnet01
        - !Ref TestSubnet02
      SecurityGroupIds:
        - !GetAtt TestVPC.DefaultSecurityGroup
      VpcEndpointType: Interface
      ServiceName: !Sub com.amazonaws.${AWS::Region}.secretsmanager
      PrivateDnsEnabled: true
      VpcId: !Ref TestVPC
  MyDocDBClusterRotationSecret:
    Type: AWS::SecretsManager::Secret
    Properties:
      GenerateSecretString:
        SecretStringTemplate: '{"username": "someadmin","ssl": true}'
        GenerateStringKey: password
        PasswordLength: 16
        ExcludeCharacters: '"@/\'
      Tags:
        - Key: AppName
```

```
          Value: MyApp
  MyDocDBCluster:
    Type: AWS::DocDB::DBCluster
    Properties:
      DBSubnetGroupName: !Ref MyDBSubnetGroup
      MasterUsername: !Sub '{{resolve:secretsmanager:
${MyDocDBClusterRotationSecret}::username}}'
      MasterUserPassword: !Sub '{{resolve:secretsmanager:
${MyDocDBClusterRotationSecret}::password}}'
      VpcSecurityGroupIds:
        - !GetAtt TestVPC.DefaultSecurityGroup
  DocDBInstance:
    Type: AWS::DocDB::DBInstance
    Properties:
      DBClusterIdentifier: !Ref MyDocDBCluster
      DBInstanceClass: db.r5.large
  MyDBSubnetGroup:
    Type: AWS::DocDB::DBSubnetGroup
    Properties:
      DBSubnetGroupDescription: ''
      SubnetIds:
        - !Ref TestSubnet01
        - !Ref TestSubnet02
  SecretDocDBClusterAttachment:
    Type: AWS::SecretsManager::SecretTargetAttachment
    Properties:
      SecretId: !Ref MyDocDBClusterRotationSecret
      TargetId: !Ref MyDocDBCluster
      TargetType: AWS::DocDB::DBCluster
  MySecretRotationSchedule:
    Type: AWS::SecretsManager::RotationSchedule
    DependsOn: SecretDocDBClusterAttachment
    Properties:
      SecretId: !Ref MyDocDBClusterRotationSecret
      HostedRotationLambda:
        RotationType: MongoDBSingleUser
        RotationLambdaName: MongoDBSingleUser
        VpcSecurityGroupIds: !GetAtt TestVPC.DefaultSecurityGroup
        VpcSubnetIds: !Join
          - ','
          - - !Ref TestSubnet01
            - !Ref TestSubnet02
      RotationRules:
        Duration: 2h
```

```
        ScheduleExpression: cron(0 8 1 * ? *)
```

Secrets Manager が を使用する方法 AWS CloudFormation

コンソールを使用してローテーションを有効にすると、Secrets Manager は AWS CloudFormation を使用してローテーション用のリソースを作成します。そのプロセス中に新しいローテーション関数を作成すると、 は適切な `AWS::Serverless::Function`に基づいて CloudFormation を作成しますローテーション関数のテンプレート。次に`RotationSchedule`、シークレットのローテーション関数とローテーションルールを設定する CloudFormation を設定します。自動ローテーションをオンにした後、バナーでスタックの表示 CloudFormation を選択すると、スタックを表示できます。

自動ローテーションを有効にする方法については、 シークレットのローテーション を参照してください。

で AWS Secrets Manager シークレットを作成する AWS Cloud Development Kit (AWS CDK)

CDK アプリでシークレットを作成、管理、および取得するには、AWS Secrets Manager Construct Library を使用します。これには、`ResourcePolicy`、`RotationSchedule`、`Secret`、`SecretRotation`、および `SecretTargetAttachment` のコンストラクトが含まれています。

CDK アプリケーションでシークレットを使用するには、最初にコンソールまたは CLI を使用してシークレットを作成し、次にシークレットを CDK アプリケーションにインポートするのが良いでしょう。

例については以下を参照してください。

- シークレットを作成する
- シークレットをインポートする
- シークレットを取得する
- シークレットの使用許可を付与する
- シークレットをローテーションする
- データベースシークレットを作成する
- シークレットを他のリージョンにレプリケートする

CDK の詳細については、「AWS Cloud Development Kit (AWS CDK) v2 デベロッパーガイド」を参照してください。

シークレットイベント通知

Secrets Manager は、シークレットのステージングラベルが変更されると、Amazon EventBridge にイベントを発行します。これらのイベントに一致する EventBridge ルールを作成し、ターゲットにルーティングしてイベント駆動型ワークフローを構築できます。

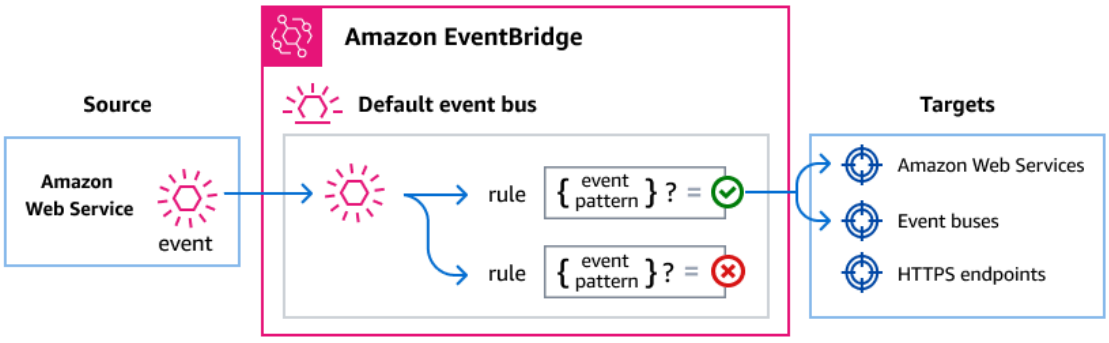
EventBridge が Secrets Manager イベントをルーティングする方法

以下は、EventBridge が Secrets Manager イベントと連携する方法について説明します。

多くの AWS サービスと同様に、Secrets Manager はイベントを生成して EventBridge のデフォルトイベントバスに送信します。イベントバスはルーターであり、イベントを受信して指定された送信先やターゲットに配信します。ターゲットには、他の AWS サービス、カスタムアプリケーション、SaaS パートナーアプリケーションを含めることができます。

EventBridge は、イベントバスで作成されたルールに従ってイベントを配信します。ルールごとに、フィルターまたはイベントパターンを指定して、必要なイベントのみを選択します。イベントがイベントバスに送信されるたびに、EventBridge はイベントを各ルールと比較します。イベントがルールと一致する場合、EventBridge は指定されたターゲットにイベントをルーティングします。

次の図は、EventBridge が Secrets Manager イベントをデフォルトのイベントバスから指定したターゲットにルーティングする方法を示しています。



詳細については、EventBridge ユーザーガイド」の「ルールの作成」を参照してください。

Secrets Manager の イベント

Secrets Manager は、 AWSPENDINGと を除くステージングラベルが変更されるたびに、デフォルトの EventBridge イベントバスにSecret Label Updatedイベントを発行しますAWSPREVIOUS。

これにはカスタムラベルが含まれます。イベントは、すべてのシークレットに対してデフォルトで有効になっています。これらのイベントを使用するには、イベントと一致する EventBridge ルールを作成し、ターゲットにルーティングします。ターゲットとして使用できるサービスの完全なリストについては、EventBridge ユーザーガイド」の「EventBridge ターゲット」を参照してください。EventBridge

Secrets Manager は、タグ、説明、ローテーション設定、リソースポリシーの更新などのメタデータ変更のイベントを発行しません。

イベントの詳細のタイプ	説明
シークレットラベルが更新されました	Secrets Manager は、ステージングラベルを新しいバージョンのシークレットに移動しました。

イベントの構造

AWS サービスからのすべてのイベントには、次の 2 種類のデータが含まれます。

- イベントのソースである AWS サービス、イベントが生成された時刻、イベントが発生したアカウントとリージョンなど、イベントに関するメタデータを含む一般的なフィールドのセット。これらの一般的なフィールドの定義については、「Amazon EventBridge イベントリファレンス」の「イベントの構造」を参照してください。
- `detail` フィールドには、その特定のサービスイベントに固有のデータが含まれます。

Secrets Manager イベントに一致するイベントパターンの作成

イベントパターンは、選択するイベントに含めるデータを指定するフィルターです。

各イベントパターンは JSON 形式のオブジェクトで、以下が含まれています。

- イベントを送信するサービスを識別する `source` 属性。Secrets Manager イベントの場合、ソースは ですaws.secretsmanager。
- (オプション): 照合するイベント名の配列を含む `detail-type` 属性。
- (オプション): 照合対象となるその他のイベントデータを含む `detail` 属性。

次のイベントパターンは、AWSCURRENTラベルが新しいバージョンに移動したときにイベントに一致します。これは、アクティブなシークレット値が変更されたことを示します。

```
{
  "source": ["aws.secretsmanager"],
  "detail-type": ["Secret Label Updated"],
  "detail": {
    "labelUpdated": ["AWSCURRENT"]
  }
}
```

次のイベントパターンは、名前が で始まるシークレットのラベル変更のみに一致しますMyTestSecret。

```
{
  "source": ["aws.secretsmanager"],
  "detail-type": ["Secret Label Updated"],
  "detail": {
    "name": [{"prefix": "MyTestSecret"}]
  }
}
```

次のイベントパターンは、 で始まる名前のシークレットで AWSCURRENTが変更された場合にのみ一致するようにフィルターを組み合わせますprod/。

```
{
  "source": ["aws.secretsmanager"],
  "detail-type": ["Secret Label Updated"],
  "detail": {
    "name": [{"prefix": "prod/"}],
    "labelUpdated": ["AWSCURRENT"]
  }
}
```

ワークフローでカスタムステージングラベルを使用する場合は、それらのラベルのイベントを一致させることができます。次の例は、 という名前のラベルMyLabelが移動されたときに一致します。

```
{
  "source": ["aws.secretsmanager"],
  "detail-type": ["Secret Label Updated"],
```

```
  "detail": {
    "labelUpdated": ["MyLabel"]
  }
}
```

イベントパターンの記述の詳細については、「EventBridge ユーザーガイド」の「イベントパターン」を参照してください。

イベントパターンの作成は、ルールを作成する大規模なプロセスのステップです。詳細については、EventBridge ユーザーガイド」の「ルールの作成」を参照してください。

シークレットラベルの更新イベント

Secrets Manager は、 AWSPENDINGと を除くすべてのラベルについて、ステージングラベルが新しいバージョンのシークレットに移動したときにSecret Label Updatedイベントを送信しますAWSPREVIOUS。

次の例は、 Secret Label Updatedイベントの構造を示しています。各フィールドの説明については、次のリストを参照してください。

Secrets Manager イベントの特定の値が含まれているため、 フィールドsourceと detail-typeフィールドは以下のとおりです。すべてのイベントに含まれる他のメタデータフィールドの定義については、「Amazon EventBridge イベントリファレンス」の「イベントの構造」を参照してください。

```
{
  "detail-type": "Secret Label Updated",
  "source": "aws.secretsmanager",
  "detail": {
    "name" : "string",
    "labelUpdated" : "string",
    "versionId" : "string"
  }
}
```

detail-type

イベントのタイプを示します。

このイベントの場合、この値は Secret Label Updated です。

`source`

イベントを発生させたサービスを識別します。Secrets Manager イベントの場合、この値は `aws.secretsmanager`です。

`detail`

イベントに関する情報を含む JSON オブジェクト。このフィールドの内容は、イベントを生成するサービスによって決まります。

このイベントの場合、このデータには以下が含まれます。

`name`

シークレットのわかりやすい名前。

`labelUpdated`

新しいバージョンにアタッチされたステージングラベル。

`versionId`

シークレットの新しいバージョンの一意の識別子。

Exampleシークレットラベルの更新イベント

次の例は、AWSCURRENTラベルが新しいバージョンに移動したシークレットの`Secret Label Updated`イベントを示しています。

```
{
  "version": "0",
  "id": "6a7e8feb-b491-4cf7-a9f1-bf3703467718",
  "detail-type": "Secret Label Updated",
  "source": "aws.secretsmanager",
  "account": "012345678901",
  "time": "2024-02-06T16:43:48Z",
  "region": "us-west-2",
  "resources": [
    "arn:aws:secretsmanager:us-west-2:012345678901:secret:mySecret-a1b2c3"
  ],
  "detail": {
    "name": "mySecret",
    "labelUpdated": "AWSCURRENT",
    "versionId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
  }
```

```
}
```

AWS Secrets Manager シークレットのモニタリング

AWS には、Secrets Manager のシークレットを監視し、問題が発生したときに報告し、必要に応じて自動アクションを実行するためのモニタリングツールが用意されています。ログは予期しない使用や変更を調査する場合に使用することができ、不要な変更をロールバックできます。シークレットの不適切な使用や、シークレットを削除しようとする試みに対する自動チェックを設定できます。

トピック

- で AWS Secrets Manager イベントを ログに記録する AWS CloudTrail
- Amazon CloudWatch AWS Secrets Manager によるモニタリング
- Amazon EventBridge と AWS Secrets Manager イベントを照合する
- 削除が予定されている AWS Secrets Manager シークレットがいつアクセスされるかをモニタリングする
- を使用してシー AWS Secrets Manager クレットのコンプライアンスをモニタリングする AWS Config
- Secrets Manager のコストを監視する
- Amazon GuardDuty で脅威を検出する

で AWS Secrets Manager イベントを ログに記録する AWS CloudTrail

AWS CloudTrail は Secrets Manager のすべての API コールをイベントとして記録します。これには、Secrets Manager コンソールからのコールや、ローテーションおよびシークレットバージョン削除のための他のいくつかのイベントが含まれます。Secrets Manager レコードのログエントリのリストについては、「CloudTrail エントリ」を参照してください。

CloudTrail コンソールを使用して、過去 90 日間に記録された イベントを表示できます。Secrets Manager のイベントなど、 AWS アカウント内のイベントの継続的な記録については、CloudTrail がログファイルを Amazon S3 バケットに配信するように証跡を作成します。AWS 「アカウントの証跡の作成」を参照してください。CloudTrail を CloudTrail ログファイルを複数の AWS アカウントおよび AWS リージョン から受信するように設定することもできます。

CloudTrail ログで収集されたデータをさらに分析して処理するように、他の AWS サービスを設定できます。「CloudTrail ログとのAWS サービス統合」を参照してください。CloudTrail が、新

しいログファイルを Amazon S3 バケットに発行するときにも通知を受け取ることができます。「CloudTrail の Amazon SNS 通知の設定」を参照してください。

CloudTrail ログ (コンソール) からSecrets Manager のイベントを取得するには

1. CloudTrail コンソールの https://console.aws.amazon.com/cloudtrail/ を開いてください。
2. コンソールが、イベントの発生したリージョンを示してしていることを確認します。コンソールには、選択したリージョンで発生したイベントのみが表示されます。コンソール右上のドロップダウンリストからリージョンを選択してください。
3. 左のナビゲーションペインで [Event history] (イベント履歴)を選択します。
4. [Filter] (フィルター) 条件、および [Time range] (時間範囲) (またはその両方) を選択すると探しているイベントを見つけるのに役立ちます。例えば、次のようになります。

   a. すべての Secrets Manager イベントを表示するには、[ルックアップ属性] で [イベントソース] を選択します。次に、[Enter event source] (イベントソースの入力) で、**`secretsmanager.amazonaws.com`** を選択します。

   b. シークレットのすべてのイベントを表示するには、[ルックアップ属性] で [リソース名] を選択します。次に、[リソース名を入力] にシークレットの名前を入力します。
5. 追加の詳細を表示するには、イベントの横にある展開矢印を選択します。利用可能なすべての情報を表示するには、[View event] (イベントの表示) を選択します。

AWS CLI

Example CloudTrail ログから Secrets Manager のイベントを取得する

次の `lookup-events` の例では、Secrets Manager のイベントが検索されます。

```
aws cloudtrail lookup-events \
    --region us-east-1 \
    --lookup-attributes
 AttributeKey=EventSource,AttributeValue=secretsmanager.amazonaws.com
```

AWS CloudTrail Secrets Manager の エントリ

AWS Secrets Manager は、すべての Secrets Manager オペレーションと、ローテーションと削除に関連するその他のイベントのエントリを AWS CloudTrail ログに書き込みます。これらのイベントに対するアクションの詳細については、「EventBridge で Secrets Manager のイベントをマッチさせる」を参照してください。

ログエントリの種類

- Secrets Manager の操作ログエントリ
- 削除用のログエントリ
- レプリケーションのログエントリ
- ローテーションのログエントリ

Secrets Manager の操作ログエントリ

Secrets Manager の操作の呼び出しによって発生するイベントには `"detail-type": ["AWS API Call via CloudTrail"]` があります。

> **Note**
>
> 2024 年 2 月以前は、一部の Secrets Manager 操作で、シークレット ARN に「arn」ではなく「aRN」が含まれるイベントが報告されていました。詳細については、「AWS re:Post」を参照してください。

以下は、ユーザーまたはサービスが API、SDK、または CLI を通じて Secrets Manager オペレーションを呼び出す際に生成される CloudTrail エントリです。

BatchGetSecretValue

BatchGetSecretValue オペレーションによって生成。シークレットを取得する方法の詳細については、「シークレットの取得」を参照してください。

CancelRotateSecret

CancelRotateSecret オペレーションによって生成。ローテーションの詳細は、「シークレットのローテーション」を参照してください。

CreateSecret

CreateSecret オペレーションによって生成。シークレットを作成する方法の詳細については、「シークレットの管理」を参照してください。

DeleteResourcePolicy

DeleteResourcePolicy オペレーションによって生成。許可の詳細については、「the section called “認証とアクセスコントロール”」を参照してください。

DeleteSecret

DeleteSecret オペレーションによって生成。シークレットの削除については、「the section called “シークレットの削除”」を参照してください。

DescribeSecret

DescribeSecret オペレーションによって生成。

GetRandomPassword

GetRandomPassword オペレーションによって生成。

GetResourcePolicy

GetResourcePolicy オペレーションによって生成。許可の詳細については、「the section called “認証とアクセスコントロール”」を参照してください。

GetSecretValue

GetSecretValue と BatchGetSecretValue オペレーションによって生成。シークレットを取得する方法の詳細については、「シークレットの取得」を参照してください。

ListSecrets

ListSecrets オペレーションによって生成。シークレットを一覧する方法の詳細については、「the section called “シークレットを検索する”」を参照してください。

ListSecretVersionIds

ListSecretVersionIds オペレーションによって生成。

PutResourcePolicy

PutResourcePolicy オペレーションによって生成。許可の詳細については、「the section called “認証とアクセスコントロール”」を参照してください。

PutSecretValue

PutSecretValue オペレーションによって生成。シークレットを更新する方法の詳細については、「the section called “シークレットの変更”」を参照してください。

RemoveRegionsFromReplication

RemoveRegionsFromReplication オペレーションによって生成。シークレットのレプリケーションの詳細については、「マルチリージョンレプリケーション」を参照してください。

ReplicateSecretToRegions

ReplicateSecretToRegions オペレーションによって生成。シークレットのレプリケーションの詳細については、「マルチリージョンレプリケーション」を参照してください。

RestoreSecret

RestoreSecret オペレーションによって生成。削除されたシークレットを復元する方法については、「the section called “シークレットを復元する”」を参照してください。

RotateSecret

RotateSecret オペレーションによって生成。ローテーションの詳細は、「 シークレットのローテーション」を参照してください。

StopReplicationToReplica

StopReplicationToReplica オペレーションによって生成。シークレットのレプリケーションの詳細については、「マルチリージョンレプリケーション」を参照してください。

TagResource

TagResource オペレーションによって生成。シークレットのタグ付けの詳細については、「the section called “ シークレットにタグ付けする”」を参照してください。

UntagResource

UntagResource オペレーションによって生成。シークレットのタグ解除の詳細については、「the section called “ シークレットにタグ付けする”」を参照してください。

UpdateSecret

UpdateSecret オペレーションによって生成。シークレットを更新する方法の詳細については、「the section called “シークレットの変更”」を参照してください。

UpdateSecretVersionStage

UpdateSecretVersionStage オペレーションによって生成。バージョンステージの詳細については、「the section called “シークレットバージョン”」を参照してください。

ValidateResourcePolicy

ValidateResourcePolicy オペレーションによって生成。許可の詳細については、「the section called “認証とアクセスコントロール”」を参照してください。

削除用のログエントリ

Secrets Manager 操作のイベントに加えて、Secrets Manager は削除に関連する次のイベントを生成します。これらのイベントには `"detail-type": ["AWS Service Event via CloudTrail"]` があります。

CancelSecretVersionDelete

Secrets Manager サービスによって生成されるもの。バージョンを持つシークレットで `DeleteSecret` を呼び出し、後で `RestoreSecret` を呼び出すと、Secrets Manager は、復元されたシークレットバージョンごとにこのイベントをログに記録します。削除されたシークレットを復元する方法については、「the section called “シークレットを復元する”」を参照してください。

EndSecretVersionDelete

シークレットバージョンが削除されたときに、Secrets Manager サービスによって生成されるもの。詳細については、「the section called “シークレットの削除”」を参照してください。

StartSecretVersionDelete

Secrets Manager がシークレットバージョンの削除を開始する際に、Secrets Manager サービスによって生成されるもの。シークレットの削除については、「the section called “シークレットの削除”」を参照してください。

SecretVersionDeletion

Secrets Manager が廃止されたシークレットバージョンを削除ときに Secrets Manager サービスによって生成。詳細については、「Secret versions」(シークレットバージョン) を参照してください。

レプリケーションのログエントリ

Secrets Manager 操作のイベントに加えて、Secrets Manager はレプリケーションに関連する次のイベントを生成します。これらのイベントには `"detail-type": ["AWS Service Event via CloudTrail"]` があります。

ReplicationFailed

レプリケーションが失敗したときに Secrets Manager サービスによって生成されます。シークレットのレプリケーションの詳細については、「マルチリージョンレプリケーション」を参照してください。

ReplicationStarted

Secrets Manager がシークレットのレプリケーションを開始する際に、Secrets Manager サービスによって生成されます。シークレットのレプリケーションの詳細については、「マルチリージョンレプリケーション」を参照してください。

ReplicationSucceeded

Secrets Manager サービスが正常にレプリケートされたときに Secrets Manager サービスによって生成されます。シークレットのレプリケーションの詳細については、「マルチリージョンレプリケーション」を参照してください。

ローテーションのログエントリ

Secrets Manager 操作のイベントに加えて、Secrets Manager はローテーションに関連する次のイベントを生成します。これらのイベントには `"detail-type": ["AWS Service Event via CloudTrail"]` があります。

RotationStarted

Secrets Manager がシークレットのローテーションを開始する際に、Secrets Manager サービスによって生成されるもの。ローテーションの詳細は、「 シークレットのローテーション」を参照してください。

RotationAbandoned

Secrets Manager がローテーションの試みを放棄し、シークレットの既存のバージョンから `AWSPENDING` ラベルを削除するときに、Secrets Manager サービスによって生成されるもの。Secrets Manager は、ローテーション中にシークレットの新しいバージョンを作成すると、ローテーションを中止します。ローテーションの詳細は、「 シークレットのローテーション」を参照してください。

RotationFailed

ローテーションに失敗したときに、Secrets Manager サービスによって生成されるもの。ローテーションの詳細は、「the section called “ローテーションのトラブルシューティング”」を参照してください。

RotationSucceeded

Secrets Manager サービスが正常にローテーションされたときに Secrets Manager サービスによって生成されるもの。ローテーションの詳細は、「 シークレットのローテーション」を参照してください。

TestRotationStarted

Secrets Manager が、即時ローテーションが予定されていないシークレットのローテーションテストを開始したときに Secrets Manager サービスによって生成されるもの。ローテーションの詳細は、「シークレットのローテーション」を参照してください。

TestRotationSucceeded

Secrets Manager が、即時ローテーションが予定されていないシークレットのローテーションテストに成功したときに生成されるもの。ローテーションの詳細は、「シークレットのローテーション」を参照してください。

TestRotationFailed

Secrets Manager が、即時ローテーションが予定されていないシークレットのローテーションをテストし、ローテーションが失敗したときに、Secrets Manager サービスによって生成されるもの。ローテーションの詳細は、「the section called “ローテーションのトラブルシューティング”」を参照してください。

Amazon CloudWatch AWS Secrets Manager によるモニタリング

Amazon CloudWatch を使用すると、 AWS サービスをモニタリングし、アラームを作成して、メトリクスが変更されたときに通知できます。CloudWatch はこれらの統計を 15 か月間保持するため、履歴情報にアクセスし、Web アプリケーションやサービスパフォーマンスをより適切に把握できます。では AWS Secrets Manager、削除対象としてマークされたシークレットを含むアカウント内のシークレットの数、およびコンソールを介した呼び出しを含む Secrets Manager への API コールをモニタリングできます。メトリクスの監視方法の詳細については、「CloudWatch ユーザーガイド」の「CloudWatch メトリクスの使用」を参照してください。

Secrets Manager メトリクスを検索するには

1. CloudWatch コンソールで [メトリクス] から [すべてのメトリクス] を選択します。
2. [メトリックス] 検索で、ボックスに「`secret`」と入力します。
3. 以下の操作を実行します。
   - アカウントのシークレット数を監視するには、[AWS/SecretsManager] を選択し、[SecretCount] を選択します。このメトリクスは 1 時間ごとにパブリッシュされます。
   - コンソールを介した呼び出しを含む Secrets Manager への API コールをモニタリングするには、Usage > By AWS Resource を選択し、モニタリングする API コールを選択しま

す。Secrets Manager API リストについては、「Secrets Manager operations」を参照してください。

4. 以下の操作を実行します。
   - メトリクスのグラフを作成するには、「Amazon CloudWatch ユーザーガイド」の「メトリクスのグラフ化」を参照してください。
   - 異常を検出するには、「Amazon CloudWatch ユーザーガイド」の「CloudWatch 異常検出の使用」を参照してください。
   - メトリクスの統計情報を取得するには、「Amazon CloudWatch ユーザーガイド」の「メトリクスの統計を取得する」を参照してください。

CloudWatch アラーム

メトリックスの値が変化し、アラームの状態が変化したときに Amazon SNS メッセージを送信する CloudWatch アラームを作成できます。アカウントのシークレット数である Secrets Manager メトリクス `ResourceCount` にアラームを設定できます。アラームは、指定期間にわたって単一のメトリクスを監視し、指定したしきい値に対応したメトリクスの値に基づいて、数期間にわたって アクションを実行します。アラームは、持続している状態変化に対してのみアクションを呼び出します。CloudWatch のアラームは、メトリクスが特定の状態にあるだけではアクションを呼び出しません。アクションを呼び出すには、指定した期間継続している必要があります。

詳細については、「CloudWatch ユーザーガイド」の「Amazon CloudWatch でのアラームの使用」と「異常検出に基づいて CloudWatch アラームを作成する」を参照してください。

また、特定のしきい値を監視するアラームを設定し、これらのしきい値に達したときに通知を送信したりアクションを実行したりできます。詳細については、「Amazon CloudWatch ユーザーガイド」を参照してください。

Amazon EventBridge と AWS Secrets Manager イベントを照合する

Amazon EventBridge では、CloudTrail ログエントリのSecrets Manager イベントを照合できます。これらのイベントを探す EventBridge ルールを設定し、新たに生成されたイベントをターゲットに送信してアクションを起こすことができます。Secrets Manager が ログする CloudTrail エントリのリストについては、「CloudTrail エントリ」を参照してください。EventBridge の設定方法については、「EventBridge ユーザーガイド」の「EventBridge を使い始める」を参照してください。

> **Note**
>
> このトピックでは、EventBridge が CloudTrail ログエントリから派生する一致する Secrets Manager イベントについて説明します。Secrets Manager は、CloudTrail を必要としないイベントなど、ネイティブ`Secret Label Updated`イベントを EventBridge に直接発行します。ネイティブイベントの詳細については、「」を参照してくださいシークレットイベント通知。

指定したシークレットに対するすべての変更にマッチさせる

> **Note**
>
> Secrets Manager イベントの中には、シークレットの ARN を異なる大文字で返すものもあるため、複数のアクションにマッチするイベントパターンでは、ARN でシークレットを指定するために、`arn` キーと `aRN` の両方を含める必要がある場合があります。詳細については、「AWS re:Post」を参照してください。

次の例では、シークレットへの変更のログエントリを照合する EventBridge イベントパターンを示しています。

```
{
    "source": ["aws.secretsmanager"],
    "detail-type": ["AWS API Call via CloudTrail"],
    "detail": {
        "eventSource": ["secretsmanager.amazonaws.com"],
        "eventName": ["DeleteResourcePolicy", "PutResourcePolicy", "RotateSecret",
 "TagResource", "UntagResource", "UpdateSecret"],
        "responseElements": {
            "arn": ["arn:aws:secretsmanager:us-west-2:012345678901:secret:mySecret-
a1b2c3"]
        }
    }
}
```

シークレット値がローテーションされたらイベントをマッチさせる

シークレットのアクティブな値がいつ変化するかを検出するには、AWSCURRENTステージングラベルのネイティブSecret Label Updatedイベントと一致することをお勧めします。Secrets Managerは、手動更新か自動ローテーションかにかかわらず、アクティブなシークレット値が変更されるたびにAWSCURRENTラベルを新しいバージョンに移動します。このイベントは、すべてのシークレットでデフォルトで有効になっており、デフォルトの EventBridge イベントバスにルーティングされます。これを使用するには、次の例に示すように、それに一致するイベントパターンを追加します。

```
{
    "source": ["aws.secretsmanager"],
    "detail-type": ["Secret Label Updated"],
    "detail": {
        "labelUpdated": ["AWSCURRENT"]
    }
}
```

Secret Label Updated イベントの詳細については、「」を参照してくださいシークレットイベント通知。

または、シークレット値を変更するオペレーションの CloudTrail ログエントリを照合することもできます。これらのイベントの一部は Secrets Manager オペレーションからのもので、一部は Secrets Manager サービスによって生成されます。両方detail-typeに を含める必要があります。次の例は、これらの CloudTrail ログエントリに一致する EventBridge イベントパターンを示しています。

```
{
    "source": ["aws.secretsmanager"],
    "detail-type": [
        "AWS API Call via CloudTrail",
        "AWS Service Event via CloudTrail"
    ],
    "detail": {
        "eventSource": ["secretsmanager.amazonaws.com"],
        "eventName": ["PutSecretValue", "UpdateSecret", "RotationSucceeded"]
    }
}
```

削除が予定されている AWS Secrets Manager シークレットがいつアクセスされるかをモニタリングする

Amazon CloudWatch Logs AWS CloudTrailと Amazon Simple Notification Service (Amazon SNS) の組み合わせを使用して、削除保留中のシークレットへのアクセス試行を通知するアラームを作成できます。アラームから通知を受け取ると、削除が本当に必要かどうかを時間をかけて判断するために、シークレットの削除をキャンセルしなければならない場合があります。調査の結果、シークレットが実際にまだ必要であるため、シークレットが保存されたままになる可能性があります。または、使用する新しいシークレットの詳細を使用して、ユーザーを更新する必要がある場合があります。

次の手順は、特定のエラーメッセージを生成する `GetSecretValue` オペレーションのリクエストが CloudTrail ログファイルに書き込まれた場合に、通知を受け取る方法を説明します。他の API オペレーションは、アラームをトリガーせずにシークレットで実行できます。この CloudWatch アラームは、古い認証情報を使用しているユーザーまたはアプリケーションを示す可能性のある使用を検出します。

これらの手順を開始する前に、 AWS Secrets Manager API リクエストをモニタリングする予定の AWS リージョン および アカウントで CloudTrail を有効にする必要があります。手順については、「AWS CloudTrail ユーザーガイド」の「Creating a trail for the first time」を参照してください。

ステップ 1: CloudTrail ログファイルの CloudWatch ログへの配信を設定します

CloudTrail ログファイルの CloudWatch Logs への配信を設定します。これにより、CloudWatch Logs は削除が保留されているシークレットを取得する、Secrets Manager API リクエストを監視できます。

CloudTrail ログファイルの CloudWatch Logs への配信を設定するには

1. CloudTrail コンソールの https://console.aws.amazon.com/cloudtrail/ を開いてください。
2. 上部のナビゲーションバーで、シークレットをモニタリング AWS リージョン する を選択します。
3. 左のナビゲーションペインで [Trails](証跡) を選択し、CloudWatch 向けに設定する証跡の名前を選択します。
4. [Trails Configuration](証跡の設定) ページで、[CloudWatch Logs] のセクションまでスクロールダウンし、編集アイコン

(✏)
を選択します。

5. [New or existing log group] (新規または既存のロググループ)にロググループの名前 (**`CloudTrail/MyCloudWatchLogGroup`**) を入力します。
6. [IAM role] (IAM ロール) には、CloudTrail_CloudWatchLogs_Role というデフォルトのロールを使用できます。このロールには、CloudTrail イベントをロググループに配信するために必要なアクセス許可を持つ、デフォルトロールポリシーがあります。
7. [Continue] (続行) を選択して設定を保存します。
8. [AWS CloudTrail will deliver CloudTrail events associated with API activity in your account to your CloudWatch Logs log group AWS CloudTrail] (がアカウントでの API アクティビティに関連する CloudTrail イベントを CloudWatch Logs のロググループに配信する) ページで、[Allow] (許可) を選択します。

ステップ 2: CloudWatch アラームを作成する

Secrets Manager の `GetSecretValue` API オペレーションリクエストが、削除保留中のシークレットにアクセスした場合に通知を受け取るには、CloudWatch アラームを作成し、通知を設定する必要があります。

CloudWatch アラームを作成するには

1. 次の場所から CloudWatch コンソールにサインインします (https://console.aws.amazon.com/cloudwatch/)。
2. 上部のナビゲーションバーで、シークレットをモニタリングする AWS リージョンを選択します。
3. 左のナビゲーションペインで [Logs] (ログ) を選択します。
4. [Log Groups] (ロググループ) のリストで、以前の手順で作成したロググループ (CloudTrail/MyCloudWatchLogGroupなど) の横にあるチェックボックスを選択します。その後、[Create Metric Filter] (メトリクスフィルタの作成) を選択します。
5. [Filter Pattern] (フィルタパターン) に、以下を入力するか貼り付けます。

```
{ $.eventName = "GetSecretValue" && $.errorMessage = "*secret because it was marked
 for deletion*" }
```

[Assign Metric] (メトリクスの割り当て) を選択します。

6. [Create Metric Filter and Assign a Metric] (メトリクスフィルタの作成とメトリクスの割り当て) ページで、次の操作を実行します。

   a. [Metric Namespace] (メトリクス名前空間) に「**CloudTrailLogMetrics**」と入力します。

   b. [Metric Name] (メトリクス名) に「**AttemptsToAccessDeletedSecrets**」と入力します。

   c. [Show advanced metric settings] (メトリクスの詳細設定の表示) を選択した後、必要に応じて [Metric Value] (メトリクス値) に「**1**」と入力します。

   d. [Create Filter] (フィルタの作成) を選択します。

7. フィルタボックスで、[Create Alarm] (アラームの作成) を選択します。

8. [Create Alarm] (アラームの作成) ウィンドウで、以下の操作を行います。

   a. [Name] (名前) に、「**AttemptsToAccessDeletedSecretsAlarm**」と入力します。

   b. [Whenever:] (次の時:) の [is:] (が:) で [>=] を選択し、「**1**」と入力します。

   c. [Send notification to:] (通知の送信:) の横で、次のいずれかを実行します。

      - 新しい Amazon SNS トピックを作成し使用するには、[New list] (新しいリスト) を選択し、新しいトピック名を入力します。[Email list:] (E メールリスト:) に、E メールアドレスを少なくとも 1 つ入力します。カンマで区切って、複数の E メールアドレスを入力できます。
      - 既存の Amazon SNS トピックを使用するには、使用するトピックの名前を選択します。リストが存在しない場合は、[Select list] (リストの選択) を選択します。

   d. [Create Alarm] (アラームの作成) を選択します。

ステップ 3: CloudWatch アラームをテストする

アラームをテストするには、シークレットを作成し、それを削除用にスケジュールします。次に、シークレット値の取得を試みます。アラームで設定したアドレスに間もなく E メールが届きます。これは、削除予定のシークレットの使用について警告します。

を使用してシー AWS Secrets Manager クレットのコンプライアンスをモニタリングする AWS Config

AWS Config を使用してシークレットを評価し、標準に準拠しているかどうかを確認できます。AWS Config ルールを使用して、シークレットの内部セキュリティおよびコンプライアンス要件を定義します。その後 AWS Config 、 はルールに準拠していないシークレットを識別できます。また、シークレットメタデータ、ローテーション設定、シークレット暗号化に使用される KMS キー、Lambda ローテーション関数、シークレットに関連付けられたタグの変更を追跡することもできます。

変更を通知する AWS Config ように を設定できます。詳細については、「 が Amazon SNS トピック AWS Config に送信する通知」を参照してください。

複数の AWS アカウント と組織 AWS リージョン 内にシークレットがある場合は、その設定とコンプライアンスデータを集約できます。詳細については、「Multi-account Multi-Region data aggregation」を参照してください。

シークレットが準拠しているかどうかを評価するには

- AWS Config ルールを使用してリソースを評価する手順に従い、次のいずれかのルールを選択します。
  - `secretsmanager-secret-unused` — 指定した日数内にシークレットがアクセスされたかどうかを確認します。
  - `secretsmanager-using-cmk` — シークレットが AWS マネージドキー `aws/secretsmanager`または作成したカスタマーマネージドキーを使用して暗号化されているかどうかを確認します AWS KMS。
  - `secretsmanager-rotation-enabled-check` — Secrets Manager に保存されているシークレットに対してローテーションが設定されているかどうかを確認します。
  - `secretsmanager-scheduled-rotation-success-check` – 最後に成功したローテーションが、設定されたローテーション頻度の範囲内であるかどうかをチェックします。チェックの最小頻度は日次です。
  - `secretsmanager-secret-periodic-rotation` — 指定した日数内にシークレットがローテーションされたかどうかを確認します。

Secrets Manager のコストを監視する

Amazon CloudWatch を使用して、推定 AWS Secrets Manager 請求額をモニタリングできます。詳細については、CloudWatch ユーザーガイド」の「推定 AWS 請求額をモニタリングするための請求アラームの作成」を参照してください。

コストをモニタリングするもう 1 つのオプションは、 AWS コスト異常検出です。詳細情報については、「AWS Cost Management User Guide」の「Detecting unusual spend with AWS Cost Anomaly Detection」を参照してください。

Session Manager の使用状況を監視する方法については、「the section called “CloudWatch を使用して監視する”」と「the section called “で口グ記録する AWS CloudTrail ”」を参照してください。

AWS Secrets Manager 料金の詳細については、「」を参照してくださいthe section called “料金”。

Amazon GuardDuty で脅威を検出する

Amazon GuardDuty は、 AWS 環境でアカウント、コンテナ、ワークロード、およびデータを保護するのに役立つ脅威検出サービスです。機械学習 (ML) モデルと異常および脅威検出機能を使用して、GuardDuty はさまざまな口グソースを継続的に監視し、環境内の潜在的なセキュリティリスクと悪意のあるアクティビティを特定して優先順位を付けます。例えば、GuardDuty は、インスタンス起動口ールを通じて Amazon EC2 インスタンス専用に作成された認証情報が、 AWS内の別のアカウントから使用されていることを検出した場合に、シークレットへの異常なアクセスや不審なアクセス、認証情報の漏洩などの潜在的な脅威を検出します。詳細については、「Amazon GuardDuty User Guide」を参照してください。

検出の別のユースケースの例は、異常な動作です。例えば、 AWS Secrets Manager 通常、 が Java SDK を使用してエンティティから `create-secret`、`get-secret-value`、`describe-secret`、 の`list-secrets`呼び出しを取得し、別のエンティティが VPN の外部 AWS CLI から の呼び出し`batch-get-secret-value`と`get-secret-value`使用を開始した場合、GuardDuty は 2 番目のエンティティが異常に APIs を呼び出しているという検出結果を報告できます。詳細については、「GuardDuty IAM finding type CredentialAccess:IAMUser/AnomalousBehavior」を参照してください。

のコンプライアンス検証 AWS Secrets Manager

Secrets Manager を使用する際のお客様のコンプライアンス責任は、お客様のデータの機密性、貴社のコンプライアンス目的、適用される法律および規制によって決まります。 は、コンプライアンスに役立つ以下のリソース AWS を提供します。

- 「セキュリティ＆コンプライアンスクイックリファレンスガイド」 – これらのデプロイガイドには、アーキテクチャ上の考慮事項の説明と、 AWSでセキュリティとコンプライアンスに重点を置いたベースライン環境をデプロイするための手順が記載されています。
- 「Architecting for HIPAA Security and Compliance」ホワイトペーパー – このホワイトペーパーでは、企業が AWS を使用して HIPAA 準拠のアプリケーションを作成する方法について説明します。
- AWS コンプライアンスリソース – このワークブックとガイドのコレクションは、お客様の業界や地域に適用される場合があります。
- AWS Config では、自社プラクティス、業界ガイドライン、および規制に対するリソースの設定の準拠状態が評価されます。詳細については、「the section called “シークレットのコンプライアンスを監視する”」を参照してください。
- AWS Security Hub CSPM では、 内のセキュリティ状態を包括的に把握 AWS できるため、セキュリティ業界標準とベストプラクティスへの準拠を確認できます。Security Hub CSPM を使用して Secrets Manager リソースを評価する方法については、AWS Security Hub CSPM 「 ユーザーガイド」のAWS Secrets Manager 「 コントロール」を参照してください。
- IAM アクセスアナライザーは、外部エンティティにシークレットへのアクセスを許可するポリシー (ポリシー内の条件ステートメントを含む) を分析します。詳細については、「アクセスアナライザーを使用したアクセスのプレビュー」を参照してください。
- AWS Systems Manager には、Secrets Manager の定義済みのランブックが用意されています。詳細については、「シークレットマネージャーの、Systems Manager Automation ランブックリファレンス」を参照してください。
- を使用して、サードパーティーの監査レポートをダウンロードできます AWS Artifact。詳細については、「Downloading Reports in AWS Artifact」を参照してください。

コンプライアンス標準

AWS Secrets Manager は、以下の規格の監査を受けており、コンプライアンス認定を取得する必要がある場合にソリューションの一部になる可能性があります。

- HIPAA – AWS は、医療保険の相互運用性と説明責任に関する法律 (HIPAA) コンプライアンスプログラムを拡張し、HIPAA 対象サービス AWS Secrets Manager として を含めました。と事業提携契約 (BAA) を締結している場合は AWS、Secrets Manager を使用して HIPAA 準拠のアプリケーションを構築できます。 は、医療情報の処理と保存 AWS に を活用する方法について詳しく知りたいお客様向けに HIPAA に焦点を当てたホワイトペーパー AWS を提供しています。詳細については、「HIPAA コンプライアンス」を参照してください。
- PCI 参加組織 – サービスプロバイダーレベル 1 で、PCI (Payment Card Industry) データセキュリティ標準 (DSS) バージョン 3.2 の準拠証明書 AWS Secrets Manager を取得しています。 AWS 製品やサービスを使用してカード所有者データを保存、処理、または送信するお客様は、独自の PCI DSS コンプライアンス証明書を管理する AWS Secrets Manager 際に を使用できます。PCI コンプライアンスパッケージのコピーをリクエストする方法など、 AWS PCI DSS の詳細については、「PCI DSS レベル 1」を参照してください。
- ISO – ISO/IEC 27001、ISO/IEC 27017、ISO/IEC 27018、および ISO 9001 のコンプライアンス認証を正常に完了 AWS Secrets Manager しました。詳細については、「ISO 27001」、「ISO 27017」、「ISO 27018」、および「ISO 9001」を参照してください。
- AICPA SOC – System and Organization Control (SOC) レポートとは、重要なコンプライアンスの統制および目標を Secrets Manager がどのように達成したかを実証する、サードパーティーによる独立した審査報告書です。これらのレポートの目的は、運用とコンプライアンスをサポートするために確立された AWS コントロールをユーザーと監査人が理解できるようにすることです。詳細については、「SOC Compliance」(SOC コンプライアンス) を参照してください。
- FedRAMP – Federal Risk and Authorization Management Program (FedRAMP) は政府全体のプログラムであり、クラウドの製品やサービスに対するセキュリティ評価、認可、および継続的なモニタリングに関する標準化されたアプローチを提供しています。また、FedRAMP プログラムは、East/West と GovCloud のサービスおよびリージョン向けに、政府のデータや規制されたデータを消費するための暫定認証も提供します。詳細については、「 FedRAMP への準拠」を参照してください。
- 国防総省 – 米国防総省 (DoD) クラウドコンピューティングセキュリティ要求事項ガイド (SRG) には、クラウドサービスプロバイダー (CSP) が DoD の暫定認可を取得して DoD ユーザーへのサービス提供を可能にする、標準化された評価と認可プロセスが規定されています。詳細については、DoD SRG リソースを参照してください。
- IRAP – オーストラリア政府のお客様は、情報セキュリティ登録評価プログラム (IRAP) を使用して、適切な制御が行われていることを検証し、オーストラリアサイバーセキュリティセンター (ACSC) が作成したオーストラリア政府情報セキュリティマニュアル (ISM) の要件に対応する適切な責任モデルを決定することができます。詳細については、IRAP リソースを参照してください。

- OSPAR – アマゾン ウェブ サービス (AWS) は、シンガポールの銀行協会 (ABS) の外部委託サービスプロバイダーの統制目標と手順に関するガイドライン (ABS ガイドライン) と AWS 連携し、シンガポールの金融サービス業界が設定したクラウドサービスプロバイダーの高い期待を満たすことを顧客に AWS 約束しました。リソースの詳細については、OSPAR リソースを参照してください。

のセキュリティ AWS Secrets Manager

のセキュリティが最優先事項 AWS です。お客様は AWS 、最もセキュリティの影響を受けやすい組織の要件を満たすように構築されたデータセンターとネットワークアーキテクチャを活用できます。

ユーザーと は、セキュリティの責任 AWS を共有します。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 クラウドで AWS サービスを実行するインフラストラクチャを保護する AWS 責任があります。 AWS また、 では、安全に使用できるサービスも提供しています。サードパーティーの監査人は、AWS コンプライアンスプログラム の一環として、セキュリティの有効性を定期的にテストおよび検証します。が適用されるコンプライアンスプログラムの詳細については AWS Secrets Manager、AWS 「コンプライアンスプログラムによる対象範囲内のサービス」を参照してください。
- クラウド内のセキュリティ — AWS サービスがお客様の責任を決定します。また、ユーザーは、データの機密性、会社の要件、適用される法律や規制など、その他の要因についても責任を負います。

その他のリソースについては、「セキュリティの柱 - AWS Well-Architected フレームワーク」を参照してください。

トピック

- を使用して AWS Secrets Manager シークレット AWS CLI を保存するリスクを軽減する
- の認証とアクセスコントロール AWS Secrets Manager
- でのデータ保護 AWS Secrets Manager
- でのシークレットの暗号化と復号 AWS Secrets Manager
- のインフラストラクチャセキュリティ AWS Secrets Manager
- AWS Secrets Manager VPC エンドポイントの使用
- IAM ポリシーによる API アクセスコントロール
- の耐障害性 AWS Secrets Manager
- ポスト量子 TLS

を使用して AWS Secrets Manager シークレット AWS CLI を保存するリスクを軽減する

AWS Command Line Interface (AWS CLI) を使用して AWS オペレーションを呼び出す場合は、これらのコマンドをコマンドシェルに入力します。たとえば、Windows コマンドプロンプトや Windows PowerShell、または Bash や Z シェルなどを使用できます。これらのコマンドシェルの多くには、生産性を向上させるために設計された機能が含まれています。ただし、この機能はシークレットを侵害するために使用される可能性があります。たとえば、ほとんどのシェルでは、上矢印キーを使用して、最後に入力されたコマンドを表示できます。コマンド履歴機能は、セキュリティ保護されていないセッションにアクセスされた場合に、悪用される可能性があります。また、バックグラウンドで動作する他のユーティリティは、タスクをより効率的に実行できるようにする目的でコマンドパラメータにアクセスできます。このようなリスクを軽減するには、次の手順を実行します。

- コンソールから離れるときは、常にコンピュータをロックします。
- 要らないまたは使用しなくなったコンソールユーティリティをアンインストールするか無効にします。
- シェルまたはリモートアクセスプログラムを使用している場合は、入力したコマンドのログを記録しないようにします。
- シェルのコマンド履歴によってキャプチャされないようにパラメータを渡す方法を使用します。次の例は、シークレットテキストをテキストファイルに入力し、そのファイルを AWS Secrets Manager コマンドに渡してすぐにファイルを破棄する方法を示しています。これは、典型的なシェル履歴がシークレットテキストをキャプチャしないことを意味します。

  次の例は、典型的な Linux コマンドを示しています (シェルには多少異なるコマンドが必要な場合があります)。

```
$ touch secret.txt
        # Creates an empty text file
$ chmod go-rx secret.txt
        # Restricts access to the file to only the user
$ cat > secret.txt
        # Redirects standard input (STDIN) to the text file
ThisIsMyTopSecretPassword^D
        # Everything the user types from this point up to the CTRL-D (^D) is saved in
 the file
$ aws secretsmanager create-secret --name TestSecret --secret-string file://
secret.txt       # The Secrets Manager command takes the --secret-string parameter
 from the contents of the file
```

```
$ shred -u secret.txt
        # The file is destroyed so it can no longer be accessed.
```

これらのコマンドを実行した後、上下の矢印を使用してコマンド履歴をスクロールし、シークレットテキストがどの行にも表示されないようにする必要があります。

> **Important**
>
> デフォルトでは、コマンドヒストリバッファのサイズを最初に [1] に減らさないかぎり、Windows で同様のテクニックを実行することはできません。

Windows コマンドプロンプトで 1 コマンドのコマンド履歴バッファのみを使用するように設定するには

1. 管理者コマンドプロンプトを開きます ([Run as administrator (管理者として実行)])。
2. 左上にあるアイコンを選択し、[プロパティ] を選択します。
3. [オプション] タブで、[バッファサイズ] と [Number of Buffers (バッファ数)] の両方を **1** に設定し、[OK] を選択します。
4. 履歴に入れたくないコマンドを入力する必要がある場合は、直後に次のようなコマンドを 1 つ続けます。

   ```
   echo.
   ```

   これにより、機密性の高いコマンドがフラッシュされます。

Windows Command Prompt シェルでは、SysInternals SDelete ツールをダウンロードして、次のようなコマンドを使用することができます。

```
C:\> echo. 2> secret.txt
        # Creates an empty file
C:\> icacls secret.txt /remove "BUILTIN\Administrators" "NT AUTHORITY/SYSTEM" /
inheritance:r   # Restricts access to the file to only the owner
C:\> copy con secret.txt /y
        # Redirects the keyboard to text file, suppressing prompt to overwrite
THIS IS MY TOP SECRET PASSWORD^Z
      # Everything the user types from this point up to the CTRL-Z (^Z) is saved in the
 file
```

```
C:\> aws secretsmanager create-secret --name TestSecret --secret-string file://
secret.txt      # The Secrets Manager command takes the --secret-string parameter from
 the contents of the file
C:\> sdelete secret.txt
        # The file is destroyed so it can no longer be accessed.
```

の認証とアクセスコントロール AWS Secrets Manager

Secrets Manager は AWS Identity and Access Management (IAM) を使用して、シークレットへのアクセスを保護します。IAM は認証とアクセスコントロールを提供します。認証は、リクエストを実行するアイデンティティを検証するものです。Secrets Manager は、パスワード、アクセスキー、および多要素認証 (MFA) トークンでのサインインプロセスを使用して、ユーザーのアイデンティティを検証します。「 にサインイン AWSする」を参照してください。アクセスコントロールは、シークレットなどの AWS リソースで、承認された個人のみがオペレーションを実行できるようにします。Secrets Manager は、ポリシーを使用して、リソースにアクセスできるユーザー、およびそのアイデンティティがそれらのリソースに対して実行できるアクションを定義します。IAM でのポリシーとアクセス許可を参照してください。

トピック

- のアクセス許可リファレンス AWS Secrets Manager
- Secrets Manager 管理者のアクセス許可
- シークレットへのアクセス許可
- Lambda ローテーション関数のアクセス許可
- 暗号化キーのアクセス許可
- レプリケーション権限
- アイデンティティベースのポリシー
- リソースベースのポリシー
- 属性ベースのアクセス制御 (ABAC) を使用してシークレットへのアクセスを制御する
- AWS の マネージドポリシー AWS Secrets Manager
- AWS Secrets Manager シークレットへのアクセス許可を持つユーザーを特定する
- 別のアカウントから AWS Secrets Manager シークレットにアクセスする
- オンプレミス環境からシークレットにアクセスする

のアクセス許可リファレンス AWS Secrets Manager

Secrets Manager のアクセス許可リファレンスは、サービス認証リファレンスの AWS Secrets Managerのアクション、リソース、条件キーで利用できます。

Secrets Manager 管理者のアクセス許可

Secrets Manager 管理者のアクセス許可を付与するには、[Adding and removing IAM identity permissions] (IAM アイデンティティのアクセス許可の追加および削除) をクリックし、次のポリシーをアタッチします。

- SecretsManagerReadWrite
- IAMFullAccess

エンドユーザーには管理者のアクセス許可を付与しないことをお勧めします。これを付与すると、ユーザーはシークレットを作成および管理できますが、ローテーションを有効にするために必要なアクセス許可 (IAMFullAccess) により、エンドユーザーには適切ではない重要なアクセス許可が付与されます。

シークレットへのアクセス許可

IAM アクセス許可ポリシーを使用すると、シークレットにアクセスできるユーザーまたはサービスを制御できます。アクセス許可のポリシーは、誰がどのアクションをどのリソースで実行できるかを示します。次のようにできます:

- the section called “アイデンティティベースのポリシー”
- the section called “リソースベースのポリシー”

Lambda ローテーション関数のアクセス許可

Secrets Manager は AWS Lambda 関数を使用してシークレットをローテーションします。Lambda 関数には、シークレットと、シークレットにその認証情報が含まれているデータベースまたはサービスへのアクセス許可が必要です。「ローテーションへのアクセス許可」を参照してください。

暗号化キーのアクセス許可

Secrets Manager は AWS Key Management Service (AWS KMS) キーを使用してシークレットを暗号化します。には正しいアクセス許可 AWS マネージドキー `aws/secretsmanager`が自動的に付与

されます。別の KMS キーを使用する場合、Secrets Manager にはそのキーに対するアクセス許可が必要です。「the section called “KMS キーのアクセス許可”」を参照してください。

レプリケーション権限

IAM アクセス許可ポリシーを使用すると、どのユーザーまたはサービスがシークレットを他のリージョンに複製できるかを制御できます。「the section called “レプリケーションを防止する”」を参照してください。

アイデンティティベースのポリシー

アクセス許可ポリシーは IAM アイデンティティ: ユーザー、ユーザーグループ、およびロールにアタッチすることができます。アイデンティティベースのポリシーで、アイデンティティがアクセスできるシークレットと、アイデンティティがそのシークレットで実行できるアクションを指定します。詳細については、「Adding and removing IAM identity permissions」(IAM アクセス許可の追加と削除) を参照してください。

別のサービスのアプリケーションまたはユーザーを表すロールに権限を付与できます。例えば、Amazon EC2 インスタンスで実行されているアプリケーションは、データベースにアクセスする必要がある場合があります。EC2 インスタンスのプロファイルに IAM ロールを作成し、権限ポリシーを使用して、データベースの資格情報を含むシークレットへのアクセスをロールに付与することができます。詳細については、「Using an IAM role to grant permissions to applications running on Amazon EC2 instances」(IAMロールを使用してAmazon EC2インスタンス上で動作するアプリケーションに権限を付与する) を参照してください。その他、Amazon Redshift、AWS Lambda、Amazon ECS などのサービスにもロールをアタッチすることができます。

また、IAM 以外のアイデンティティシステムによって認証されたユーザーにアクセス許可を付与することもできます。例えば、Amazon Cognito 使用してサインインするモバイルアプリケーションユーザーに IAM ロールを関連付けることができます。ロールは、ロールのアクセス許可ポリシーにあるアクセス許可を持つ、一時的な認証情報をアプリに付与します。次に、アクセス許可ポリシーを使用して、シークレットへのアクセス許可をロールに付与できます。詳細については、「Identity providers and federation」(ID プロバイダとフェデレーション) を参照してください。

アイデンティティベースのポリシーを使用して以下を行うことができます。

- 複数のシークレットへのアクセスをアイデンティティに許可します。
- 新しいシークレットを作成できるユーザーと、まだ作成されていないシークレットにアクセスできるユーザーを制御します。
- IAM グループにシークレットへのアクセス許可を付与します。

例:

- 例: 個別のシークレット値を取得するためのアクセス許可
- 例: 個々のシークレットを読み、記述するアクセス許可
- 例: バッチでシークレット値のグループを取得するためのアクセス許可
- 例: ワイルドカード
- 例: シークレットを作成するアクセス許可
- 例: シークレットを暗号化するための特定の AWS KMS キーを拒否する

例: 個別のシークレット値を取得するためのアクセス許可

シークレット値を取得するアクセス許可を付与するには、ポリシーをシークレットまたはアイデンティティにアタッチできます。使用するポリシーの種類を決定する方法については、「アイデンティティベースのポリシーおよびリソースベースのポリシー」を参照してください。ポリシーをアタッチする方法については、the section called “リソースベースのポリシー” および the section called “アイデンティティベースのポリシー” を参照してください。

この例は、IAM グループにアクセス許可を付与する場合に役立てることができます。バッチ API コールでシークレットのグループを取得するためのアクセス許可を付与するには、「the section called “例: バッチでシークレット値のグループを取得するためのアクセス許可”」を参照してください。

Exampleカスタマーマネージドキーを使用して暗号化されたシークレットを読み込む

シークレットがカスタマーマネージドキーを使用して暗号化されている場合、ID に次のポリシーをアタッチすることで、シークレットの読み取りを許可できます。\

JSON

```
{
"Version":"2012-10-17",
"Statement": [
{
  "Effect": "Allow",
  "Action": "secretsmanager:GetSecretValue",
  "Resource": "arn:aws:secretsmanager:us-east-1:123456789012:secret:secretName-
AbCdEf"
},
{
```

```
  "Effect": "Allow",
  "Action": "kms:Decrypt",
  "Resource": "arn:aws:kms:us-east-1:123456789012:key/key-id"
}
]
}
```

例: 個々のシークレットを読み、記述するアクセス許可

Example 1 つのシークレットを読み、記述する

次のポリシーをアイデンティティにアタッチすると、シークレットへのアクセスを許可することができます。

JSON

```
{
"Version":"2012-10-17",
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "secretsmanager:GetSecretValue",
      "secretsmanager:DescribeSecret"
      ],
      "Resource": "arn:aws:secretsmanager:us-
east-1:123456789012:secret:secretName-AbCdEf"
    }
]
}
```

例: バッチでシークレット値のグループを取得するためのアクセス許可

Exampleバッチ内のシークレットグループを読み取る

次のポリシーをアイデンティティにアタッチすると、バッチ API コールのシークレットのグループを取得するためのアクセス許可を付与できます。このポリシーは、バッチコールに他のシークレットが含まれていても、*SecretARN1*、*SecretARN2*、*SecretARN3* で指定されたシークレットのみを取得できるように呼び出し元を制限します。呼び出し元がバッチ API コールで他のシー

クレットもリクエストしたとしても、Secrets Manager はそれらを返しません。詳細については、「BatchGetSecretValue」を参照してください。

JSON

```
{
"Version":"2012-10-17",
"Statement": [
{
  "Effect": "Allow",
  "Action": [
    "secretsmanager:BatchGetSecretValue",
    "secretsmanager:ListSecrets"
  ],
  "Resource": "*"
},
{
  "Effect": "Allow",
  "Action": [
    "secretsmanager:GetSecretValue"
  ],
  "Resource": [
    "arn:aws:secretsmanager:us-east-1:123456789012:secret:secretName1-AbCdEf",
    "arn:aws:secretsmanager:us-east-1:123456789012:secret:secretName2-AbCdEf",
    "arn:aws:secretsmanager:us-east-1:123456789012:secret:secretName3-AbCdEf"
    ]
}
]
}
```

例: ワイルドカード

ワイルドカードを使用して、ポリシーの要素に値のセットを含めることができます。

Exampleパス内のすべてのシークレットにアクセスする

次のポリシーは、「*TestEnv/*」で始まる名前が付いているすべてのシークレットを取得するアクセス許可を付与します。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": "secretsmanager:GetSecretValue",
        "Resource": "arn:aws:secretsmanager:us-
east-1:123456789012:secret:TestEnv/*"
    }
}
```

Exampleすべてのシークレットのメタデータにアクセスする

以下のポリシーは、DescribeSecret および List で始まるアクセス許可 (ListSecrets および ListSecretVersionIds) を付与します。

JSON

```
{
"Version":"2012-10-17",
"Statement": {
"Effect": "Allow",
"Action": [
  "secretsmanager:DescribeSecret",
  "secretsmanager:List*"
],
"Resource": "*"
}
}
```

Exampleシークレット名を一致する

次のポリシーは、Secrets Manager のシークレット名へのすべてのアクセス許可を付与します。このポリシーを使用するには、「the section called “アイデンティティベースのポリシー”」を参照してください。

シークレット名を照合するには、リージョン、アカウント ID、シークレット名、ワイルドカード (?) を組み合わせてシークレットの ARN を作成し、個々のランダムな文字に一致させます。Secrets

Manager は、ARN の一部としてシークレット名に 6 つのランダムな文字を追加するため、このワイルドカードを使用してこれらの文字に一致させることができます。"another_secret_name-*" 構文を使用した場合、Secrets Manager は、意図した 6 つのランダムの文字があるシークレットだけでなく、"another_secret_name-<anything-here>a1b2c3" にも一致します。

シークレットの ARN の 6 つのランダムな文字以外のすべての部分を予測できるため、ワイルドカード文字の '??????' 構文を使用することで、まだ存在していないシークレットに安全にアクセス許可を付与することができます。ただし、シークレットを削除して同じ名前で再作成すると、6 つの文字が変更されたにもかかわらず、ユーザーが自動的に新しいシークレットへのアクセス許可を受け取ることに注意してください。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "secretsmanager:*",
            "Resource": [
                "arn:aws:secretsmanager:us-
east-1:123456789012:secret:a_specific_secret_name-a1b2c3",
                "arn:aws:secretsmanager:us-
east-1:123456789012:secret:another_secret_name-??????"
            ]
        }
    ]
}
```

例: シークレットを作成するアクセス許可

シークレットを作成するアクセス許可を付与するには、ユーザーが属する IAM グループにアクセス許可ポリシーをアタッチすることをお勧めします。「IAM ユーザーグループ」を参照してください。

Exampleシークレットを作成する

次のポリシーは、シークレットを作成してシークレットのリストを表示するアクセス許可を付与します。このポリシーを使用するには、「the section called “アイデンティティベースのポリシー”」を参照してください。

JSON

```
{
"Version":"2012-10-17",
"Statement": [
{
  "Effect": "Allow",
  "Action": [
    "secretsmanager:CreateSecret",
    "secretsmanager:ListSecrets"
  ],
  "Resource": "*"
}
]
}
```

例: シークレットを暗号化するための特定の AWS KMS キーを拒否する

> **Important**
>
> カスタマーマネージドキーを拒否するには、キーポリシーまたはキー付与を使用してアクセスを制限することをお勧めします。詳細については、「AWS Key Management Service Developer Guide」の「Authentication and access control for AWS KMS」を参照してください。

Example AWS マネージドキーを拒否する **aws/secretsmanager**

次のポリシーは、シークレットを作成または更新するための の使用 AWS マネージドキー `aws/secretsmanager`を拒否します。このポリシーでは、カスタマーマネージドキーを使用してシークレットを暗号化する必要があります。ポリシーには、2 つのステートメントが含まれています。

1. 最初のステートメント は`Sid: "RequireCustomerManagedKeysOnSecrets"`、 を使用してシークレットを作成または更新するリクエストを拒否します AWS マネージドキー `aws/secretsmanager`。
2. 2 番目のステートメント は`Sid: "RequireKmsKeyIdParameterOnCreate"`、Secrets Manager がデフォルトで を使用するため、KMS キーを含まないシークレットの作成リクエストを拒否します AWS マネージドキー `aws/secretsmanager`。

JSON

```
{
"Version":"2012-10-17",
"Statement": [
  {
      "Sid": "RequireCustomerManagedKeysOnSecrets",
      "Effect": "Deny",
      "Action": [
          "secretsmanager:CreateSecret",
          "secretsmanager:UpdateSecret"
      ],
      "Resource": "*",
      "Condition": {
          "StringLikeIfExists": {
              "secretsmanager:KmsKeyArn": "<key_ARN_of_the_AWS_managed_key>"
          }
      }
  },
  {
      "Sid": "RequireKmsKeyIdParameterOnCreate",
      "Effect": "Deny",
      "Action": "secretsmanager:CreateSecret",
      "Resource": "*",
      "Condition": {
          "Null": {
              "secretsmanager:KmsKeyArn": "true"
          }
      }
  }
]
}
```

リソースベースのポリシー

リソースベースのポリシーで、シークレットにアクセスできるユーザーと、ユーザーがそのシークレットで実行できるアクションを指定します。リソースベースのポリシーを使用して、以下を行うことができます。

- 単一のシークレットへのアクセスを複数のユーザーまたはロールに許可します。

- 他の AWS アカウントのユーザーまたはロールにアクセスを許可します。

コンソールでリソースベースのポリシーをシークレットにアタッチすると、Secrets Manager は自動化された推論エンジン Zelkova と API `ValidateResourcePolicy` を使用して、シークレットへのアクセス許可を幅広い範囲の IAM プリンシパルに付与しないようにします。または、CLI または SDK からの `BlockPublicPolicy` パラメータを含む `PutResourcePolicy` API を呼び出しします。

> **Important**
>
> リソースポリシーの検証と `BlockPublicPolicy` パラメータは、シークレットに直接アタッチされているリソースポリシーを通じてパブリックアクセスが付与されないようにすることで、リソースを保護するのに役立ちます。これらの機能を利用したうえでさらに、次のポリシーを注意深く調べて、パブリックアクセスが許可されていないことを確かめてください。
>
> - 関連付けられた AWS プリンシパルにアタッチされたアイデンティティベースのポリシー (IAM ロールなど)
> - 関連付けられたリソースにアタッチされた AWS リソースベースのポリシー (例: AWS Key Management Service (AWS KMS) キー)
>
> シークレットへのアクセス許可を確認するには、「シークレットへのアクセス許可を持つユーザーを特定する」を参照してください。

シークレットのリソースポリシーを表示、変更、または削除するには (コンソール)

1. Secrets Manager のコンソール (https://console.aws.amazon.com/secretsmanager/) を開きます。
2. シークレットのリストから、自分のシークレットを選択します。
3. シークレットの詳細ページの [概要] タブの [リソース許可] セクションで、[許可を編集] を選択します。
4. [Code] (コード) フィールドで、次のいずれかの操作を行い、[Save] (保存する) をクリックします。
   - リソースポリシーをアタッチまたは変更するには、ポリシーを入力します。
   - ポリシーを削除するには、[Code] (コード) フィールドをクリアします。

AWS CLI

Exampleリソースポリシーを取得する

次に、シークレットにアタッチされたリソースベースのポリシーを取得する、get-resource-policy の例を示します。

```
aws secretsmanager get-resource-policy \
    --secret-id MyTestSecret
```

Exampleリソースポリシーを削除する

次に、シークレットにアタッチされているリソースベースのアポリシーを削除する、delete-resource-policy の例を示します。

```
aws secretsmanager delete-resource-policy \
    --secret-id MyTestSecret
```

Exampleリソースポリシーを追加する

次の put-resource-policy の例では、ポリシーが広範なアクセスをシークレットに提供していないことを最初に確認しながら、シークレットに許可ポリシーを追加しています。このポリシーは、ファイルから読み込まれます。詳細については、「 AWS CLI ユーザーガイド」の「ファイルからの AWS CLI パラメータのロード」を参照してください。

```
aws secretsmanager put-resource-policy \
    --secret-id MyTestSecret \
    --resource-policy file://mypolicy.json \
    --block-public-policy
```

mypolicy.json の内容:

JSON

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

```
            "Principal": {
                "AWS": "arn:aws:iam::123456789012:role/MyRole"
            },
            "Action": "secretsmanager:GetSecretValue",
            "Resource": "*"
        }
    ]
}
```

AWS SDK

シークレットにアタッチされているポリシーを取得するには、GetResourcePolicy を使用します。

シークレットにアタッチされているポリシーを削除するには、DeleteResourcePolicy を使用します。

ポリシーをシークレットにアタッチするには、PutResourcePolicy を使用します。ポリシーがすでにアタッチされている場合、コマンドはそのポリシーを新しいポリシーに置き換えます。ポリシーは、JSON 構造化テキストとしてフォーマットする必要があります。JSON ポリシードキュメント構造を参照してください。

詳細については、「the section called “AWS SDKs”」を参照してください。

例

例:

- 例: 個別のシークレット値を取得するためのアクセス許可
- 例: アクセス許可と VPC
- 例:サービスプリンシパル

例: 個別のシークレット値を取得するためのアクセス許可

シークレット値を取得するアクセス許可を付与するには、ポリシーをシークレットまたはアイデンティティにアタッチできます。使用するポリシーの種類を決定する方法については、「アイデンティティベースのポリシーおよびリソースベースのポリシー」を参照してください。ポリシーをアタッチする方法については、the section called “リソースベースのポリシー” および the section called “アイデンティティベースのポリシー” を参照してください。

この例は、単一のシークレットへのアクセスを複数のユーザーまたはロールに付与する場合に役立てることができます。バッチ API コールでシークレットのグループを取得するためのアクセス許可を付与するには、「the section called “例: バッチでシークレット値のグループを取得するためのアクセス許可”」を参照してください。

Exampleシークレットを 1 つ読み取る

シークレットに次のポリシーをアタッチすると、シークレットへのアクセスを許可することができます。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:role/EC2RoleToAccessSecrets"
            },
            "Action": "secretsmanager:GetSecretValue",
            "Resource": "*"
        }
    ]
}
```

例: アクセス許可と VPC

VPC 内からSecrets Manager にアクセスする必要がある場合は、アクセス許可ポリシーに条件を含めることで、Secret Manager へのリクエストが VPC から確実に送信されるようにすることができます。詳細については、「VPC エンドポイント条件でリクエストを制限する」および「the section called “VPC エンドポイント (AWS PrivateLink)”」を参照してください。

他の AWS サービスからのシークレットへのアクセスリクエストも VPC からのものであることを確認してください。そうでない場合、このポリシーはアクセスを拒否します。

Exampleリクエストが VPC エンドポイント経由で送信されるように要求する

例えば、次のポリシーでは、リクエストが VPC エンドポイント *vpce-1234a5678b9012c* を通過して送信されている場合にのみ、ユーザーが Secrets Manager オペレーションを実行できます。

JSON

```
{
"Id": "example-policy-1",
"Version":"2012-10-17",
"Statement": [
{
  "Sid": "RestrictGetSecretValueoperation",
  "Effect": "Deny",
  "Principal": "*",
  "Action": "secretsmanager:GetSecretValue",
  "Resource": "*",
  "Condition": {
    "StringNotEquals": {
      "aws:sourceVpce": "vpce-12345678"
    }
  }
}
]
}
```

Exampleリクエストが VPC から送信されるように要求する

次のポリシーでは、シークレットを作成および管理するコマンドが許可されるのは、そのコマンドが *vpc-12345678* から送信されている場合のみです。また、このポリシーでは、リクエストが vpc-2b2b2b2b から届いた場合にのみ、シークレットの暗号化された値へのアクセスを使用するオペレーションを許可します。1 つの VPC でアプリケーションを実行する場合は、このようなポリシーを使用できますが、管理機能には 2 番目の切り離された VPC を使用します。

JSON

```
{
"Id": "example-policy-2",
"Version":"2012-10-17",
"Statement": [
{
  "Sid": "AllowAdministrativeActionsfromONLYvpc-12345678",
  "Effect": "Deny",
  "Principal": "*",
  "Action": [
```

```
      "secretsmanager:Create*",
      "secretsmanager:Put*",
      "secretsmanager:Update*",
      "secretsmanager:Delete*",
      "secretsmanager:Restore*",
      "secretsmanager:RotateSecret",
      "secretsmanager:CancelRotate*",
      "secretsmanager:TagResource",
      "secretsmanager:UntagResource"
    ],
    "Resource": "*",
    "Condition": {
      "StringNotEquals": {
        "aws:sourceVpc": "vpc-12345678"
      }
    }
  },
  {
    "Sid": "AllowSecretValueAccessfromONLYvpc-2b2b2b2b",
    "Effect": "Deny",
    "Principal": "*",
    "Action": [
      "secretsmanager:GetSecretValue"
    ],
    "Resource": "*",
    "Condition": {
      "StringNotEquals": {
        "aws:sourceVpc": "vpc-2b2b2b2b"
      }
    }
  }
  ]
  }
```

例:サービスプリンシパル

シークレットにアタッチされているリソースポリシーにAWS サービスプリンシパルが含まれている場合は、aws:SourceArn そして aws:SourceAccount グローバル条件キーの使用をお勧めします。ARN とアカウントの値は、別の AWS サービスから Secrets Manager にリクエストが来たときにのみ、承認コンテキストに含まれます。この条件の組み合わせにより、混同される可能性のある副次的なシナリオを回避することができます。

リソース ARN にリソースポリシーで許可されていない文字が含まれている場合、そのリソース ARN を `aws:SourceArn` 条件キーの値として使用することはできません。その代わりに `aws:SourceAccount` 条件キーを使用してください。その他の要件の詳細については、「IAM requirements」(IAMの要件) を参照してください。

サービスプリンシパルは通常、シークレットにアタッチされたポリシーではプリンシパルとして使用されませんが、一部の AWS サービスではそれが必要です。サービスがシークレットにアタッチする必要があるリソースポリシーの詳細については、サービスのドキュメントを参照してください。

Exampleサービスプリンシパルを使用して、サービスがシークレットにアクセスできるようにする

JSON

```
{
"Version":"2012-10-17",
"Statement": [
{
  "Effect": "Allow",
  "Principal": {
    "Service": [
      "s3.amazonaws.com"
    ]
  },
  "Action": "secretsmanager:GetSecretValue",
  "Resource": "*",
  "Condition": {
    "ArnLike": {
      "aws:sourceArn": "arn:aws:s3::123456789012:*"
    },
    "StringEquals": {
      "aws:sourceAccount": "123456789012"
    }
  }

}
]
}
```

属性ベースのアクセス制御 (ABAC) を使用してシークレットへのアクセスを制御する

属性ベースのアクセス制御 (ABAC) は、部門、ビジネスユニット、認証結果に影響を与える可能性のあるその他の要因など、ユーザー、データ、または環境の属性や特性に基づいてアクセス許可を定義する認証戦略です。では AWS、これらの属性はタグと呼ばれます。

タグを使用したアクセス許可の制御は、急速に成長している環境や、ポリシー管理が煩雑になっている状況で役に立ちます。ABAC ルールはランタイムに動的に評価され、ユーザーのアプリケーションとデータへのアクセスと許可される操作のタイプは、ポリシーのコンテキスト要因に基づいて自動的に変更されます。例えば、ユーザーが部門を変更すると、権限を更新したり、新しいロールをリクエストしたりすることなく、アクセスが自動的に調整されます。詳細については、「ABAC とは AWS」、「タグに基づいてシークレットにアクセスするためのアクセス許可を定義する」、「IAM Identity Center で ABAC を使用して Secrets Manager の認可ニーズをスケールする」を参照してください。

例: 特定のタグを持つシークレットへの ID アクセスを許可する

次のポリシーは、キーが「*ServerName*」、値が「*ServerABC*」のタグが付けられたシークレットの `DescribeSecret` アクセスを許可します。このポリシーを ID にアタッチすると、その ID にはアカウント内のそのタグを持つシークレットに対するアクセス権限が与えられます。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": {
  "Effect": "Allow",
  "Action": "secretsmanager:DescribeSecret",
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "secretsmanager:ResourceTag/ServerName": "ServerABC"
      }
    }
  }
}
```

例: シークレットのタグに一致するタグを持つ ID のみにアクセスを制限する

次のポリシーでは、アカウント GetSecretValue のすべての ID が、ID の *AccessProject* タグの値がシークレットの *AccessProject* タグと同じ値を持つアカウントのシークレットにアクセスすることを許可します。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": {
  "Effect": "Allow",
  "Principal": {
    "AWS": "123456789012"
  },
  "Condition": {
    "StringEquals": {
      "aws:ResourceTag/AccessProject": "${ aws:PrincipalTag/AccessProject }"
    }
  },
  "Action": "secretsmanager:GetSecretValue",
  "Resource": "*"
  }
}
```

AWS の マネージドポリシー AWS Secrets Manager

AWS 管理ポリシーは、 によって作成および管理されるスタンドアロンポリシーです AWS。 AWS 管理ポリシーは、ユーザー、グループ、ロールにアクセス許可の割り当てを開始できるように、多くの一般的なユースケースにアクセス許可を付与するように設計されています。

AWS 管理ポリシーは、すべての AWS お客様が使用できるため、特定のユースケースに対して最小特権のアクセス許可を付与しない場合があることに注意してください。ユースケースに固有のカスタマー管理ポリシーを定義して、アクセス許可を絞り込むことをお勧めします。

AWS 管理ポリシーで定義されているアクセス許可は変更できません。が AWS 管理ポリシーで定義されたアクセス許可 AWS を更新すると、ポリシーがアタッチされているすべてのプリンシパル ID (ユーザー、グループ、ロール) に影響します。 AWS は、新しい が起動されるか、新しい API オペ

レーション AWS のサービス が既存のサービスで使用できるようになったときに、 AWS 管理ポリシーを更新する可能性が高くなります。

詳細については、IAM ユーザーガイドの「AWS マネージドポリシー」を参照してください。

AWS マネージドポリシー: SecretsManagerReadWrite

このポリシーは、Amazon RDS AWS Secrets Manager、Amazon Redshift、Amazon DocumentDB リソースを記述するアクセス許可、およびシークレットの暗号化と復号 AWS KMS に使用するアクセス許可を含む、 への読み取り/書き込みアクセスを提供します。このポリシーは、 AWS CloudFormation 変更セットの作成、 によって管理される Amazon S3 バケットからのローテーションテンプレートの取得 AWS、 AWS Lambda 関数の一覧表示、Amazon EC2 VPCsの説明を行うアクセス許可も提供します。これらのアクセス許可は、コンソールが既存のローテーション機能を使用してローテーションを設定するために必要です。

新しいローテーション関数を作成するには、 AWS CloudFormation スタックと AWS Lambda 実行ロールを作成するアクセス許可も必要です。IAMFullAccess マネージドポリシーを割り当てることができます。「ローテーションへのアクセス許可」を参照してください。

アクセス許可の詳細

このポリシーには、以下のアクセス許可が含まれています。

- `secretsmanager` – プリンシパルに Secrets Manager の全アクションの実行を許可します。
- `cloudformation` – プリンシパルが CloudFormation スタックを作成できるようにします。これは、 コンソールを使用してローテーションを有効にするプリンシパルが CloudFormation スタックを介して Lambda ローテーション関数を作成できるようにするために必要です。詳細については、「the section called “Secrets Manager が を使用する方法 CloudFormation”」を参照してください。
- `ec2` – プリンシパルに Amazon EC2 VPC の記述を許可します。これは、コンソールを使用するプリンシパルが、シークレットに保存している認証情報のデータベースと同じ VPC 内に、ローテーション関数を作成できるようにするために必要です。
- `kms` – プリンシパルが暗号化オペレーションに AWS KMS キーを使用できるようにします。これは、Secrets Manager がシークレットを暗号化および復号できるようにするために必要です。詳細については、「the section called “シークレット暗号化と復号”」を参照してください。
- `lambda` – プリンシパルに Lambda ローテーション関数の一覧表示を許可します。これは、コンソールを使用するプリンシパルが、既存のローテーション関数を選択できるようにするために必要です。

- `rds` – プリンシパルに Amazon RDS DB のクラスターとインスタンスの記述を許可します。これは、コンソールを使用するプリンシパルが Amazon RDS クラスターまたはインスタンスを選択できるようにするために必要です。
- `redshift` – プリンシパルに Amazon Redshift でのクラスターの記述を許可します。これは、コンソールを使用するプリンシパルが Amazon Redshift クラスターを選択できるようにするために必要です。
- `redshift-serverless` – プリンシパルが Amazon Redshift Serverless の名前空間を記述できるようにします。これは、コンソールを使用するプリンシパルが Amazon Redshift Serverless 名前空間を選択できるようにするために必要です。
- `docdb-elastic` – プリンシパルに Amazon DocumentDB での Elastic クラスターの記述を許可します。これは、コンソールを使用するプリンシパルが、Amazon DocumentDB の Elastic クラスターを選択できるようにするために必要です。
- `tag` – プリンシパルに、アカウント内のタグ付けされた全リソースの取得を許可します。
- `serverlessrepo` – プリンシパルが CloudFormation 変更セットを作成できるようにします。これは、コンソールを使用するプリンシパルが Lambda ローテーション関数を作成できるようにするために必要です。詳細については、「the section called “Secrets Manager が を使用する方法 CloudFormation”」を参照してください。
- `s3` – プリンシパルが管理対象の Amazon S3 バケットからオブジェクトを取得できるようにしますAWS。このバケットには Lambda ローテーション関数のテンプレート が含まれます。このアクセス許可は、コンソールを使用するプリンシパルがバケット内のテンプレートに基づいて、Lambda ローテーション関数を作成できるようにするために必要です。詳細については、「the section called “Secrets Manager が を使用する方法 CloudFormation”」を参照してください。

ポリシーを表示するには、SecretsManagerReadWrite の JSON ポリシーのドキュメントを参照してください。

AWS マネージドポリシー: AWSSecretsManagerClientReadOnlyAccess

このポリシーは、クライアントアプリケーションの AWS Secrets Manager シークレットへの読み取り専用アクセスを提供します。これにより、プリンシパルはシークレット値を個別にまたはバッチで取得し、シークレットを一覧表示し、シークレットメタデータを記述できます。また、カスタマーマネージドキーで暗号化されたシークレットを復号するために必要な AWS KMS アクセス許可も付与されます。

アクセス許可の詳細

このポリシーには、以下のアクセス許可が含まれています。

- secretsmanager – プリンシパルがシークレット値を個別またはバッチで取得し、シークレットを一覧表示し、シークレットメタデータを記述できるようにします。
- kms – プリンシパルが AWS KMS キーを使用してシークレットを復号できるようにします。このアクセス許可は、サービス固有の条件を通じて Secrets Manager が使用するキーに限定されます。

JSON ポリシードキュメントの最新バージョンなど、ポリシーの詳細については、「 AWS マネージドポリシーリファレンスガイド」のAWSSecretsManagerClientReadOnlyAccess」を参照してください。

AWS マネージドポリシーに対する Secrets Manager の更新

Secrets Manager の AWS 管理ポリシーの更新に関する詳細を表示します。

変更	説明	日付	バージョン
SecretsManagerReadWrite – 既存のポリシーへの更新	このポリシーは、 cloudformation:TagResource および アクセスcloudformation:UntagResource 許可を追加するために更新されました。cloudformation:CreateAction 条件キーは、これらのアクセス許可の範囲を指定します。このポリシーで既に許可されている CreateChangeSet および ExecuteChangeSet アクションを通じてスタックを	2026 年 8 月 25 日	v6

変更	説明	日付	バージョン
	作成または更新するときにのみ適用されます。		
AWSSecretsManagerClientReadOnlyAccess – 既存のポリシーの更新	このポリシーが更新され、クライアントアプリケーションが 1 回の呼び出しで複数のシークレットを取得できるように `BatchGetSecretValue` および アクセス`ListSecrets` 許可が追加されました。	2026 年 6 月 2 日	v4
AWSSecretsManagerClientReadOnlyAccess – 新しい管理ポリシー	Secrets Manager は、クライアントアプリケーションのシークレットへの読み取り専用アクセスを提供する新しい管理ポリシーを作成しました。このポリシーでは、シークレット値を取得してシークレットメタデータを記述し、シークレットを復号するために必要な AWS KMS アクセス許可を付与します。	2025 年 11 月 5 日	v1

変更	説明	日付	バージョン
SecretsManagerReadWrite – 既存のポリシーへの更新	このポリシーは、Amazon Redshift Serverless への記述アクセスを許可するように更新されました。これにより、コンソールユーザーは Amazon Redshift シークレットを作成するときに Amazon Redshift Serverless 名前空間を選択できるようになります。	2024 年 3 月 12 日	v5
SecretsManagerReadWrite – 既存のポリシーへの更新	このポリシーは、Amazon DocumentDB の Elastic クラスターへの記述アクセスを許可するように更新されました。これにより、コンソールユーザーは Amazon DocumentDB シークレットを作成するときに Elastic クラスターを選択できます。	2023 年 9 月 12 日	v4

変更	説明	日付	バージョン
SecretsManagerReadWrite – 既存のポリシーへの更新	このポリシーは、Amazon Redshift への記述アクセスを許可するように更新されました。これにより、コンソールユーザーは Amazon Redshift シークレットを作成するときに Amazon Redshift クラスターを選択できます。この更新では、Lambda ローテーション関数テンプレートを保存する によって管理 AWS される Amazon S3 バケットへの読み取りアクセスを許可する新しいアクセス許可も追加されました。	2020 年 6 月 24 日	v3
SecretsManagerReadWrite – 既存のポリシーへの更新	このポリシーは、Amazon RDS クラスターへの記述アクセスを許可するように更新されました。これにより、コンソールユーザーは Amazon RDS シークレットを作成するときにクラスターを選択できます。	2018 年 5 月 3 日	v2

変更	説明	日付	バージョン
SecretsManagerReadWrite – 新しいポリシー	Secrets Manager は、Secrets Manager へのすべての読み取り/書き込みアクセスで、コンソールを使用するために必要なアクセス許可を付与するポリシーを作成しました。	2018 年 04 月 4 日	v1

AWS Secrets Manager シークレットへのアクセス許可を持つユーザーを特定する

デフォルトでは、IAM アイデンティティにはシークレットへのアクセス許可がありません。シークレットへのアクセスを承認する時、Secrets Manager はシークレットにアタッチされたリソースベースのポリシー、およびリクエストを送信している IAM ユーザーまたは IAM ロールにアタッチされたすべてのアイデンティティベースのポリシーを評価します。これを行うために、Secrets Manager は、IAM ユーザーガイドの [Determining whether a request is allowed or denied] (リクエストの許可または拒否を決定する) に記載されているものと類似したプロセスを使用します。

複数のポリシーが 1 つのリクエストに適用される場合、Secrets Manager は階層を使用してアクセス許可を制御します。

1. ポリシー内のステートメントに明示的な deny が含まれている場合、リクエストアクションとリソースに一致します。

   この明示的な deny によって、他のすべての内容が上書きされ、アクションがブロックされます。

2. 明示的な deny はないが、ステートメントに明示的な `allow` が含まれている場合、リクエストアクションとリソースに一致します。

   この明示的な `allow` によって、リクエスト内のアクションにステートメント内のリソースへのアクセスが付与されます。

ID とシークレットが 2 つの異なるアカウントにある場合、シークレットのリソースポリシーと ID にアタッチされたポリシーの両方allowに AWS が必要です。それ以外の場合、リクエストは拒否されます。詳細については、「クロスアカウントアクセス」を参照してください。

3. リクエストアクションとリソースに一致する明示的な allow が含まれているステートメントがない場合

   AWS はデフォルトでリクエストを拒否します。これは暗黙的な拒否と呼ばれます。

シークレットのリソースベースのポリシーを表示するには

- 次のいずれかを行います。
  - Secrets Manager のコンソール (https://console.aws.amazon.com/secretsmanager/) を開きます。シークレットの詳細ページの [Resource permissions] (リソースに対するアクセス許可) セクションで、[Edit permissions] (アクセス許可の編集) をクリックします。
  - を使用して AWS CLI を呼び出すget-resource-policyか、 AWS SDK を使用して を呼び出しますGetResourcePolicy。

アイデンティティベースのポリシーを使用してアクセスできるユーザーを特定するには

- IAM ポリシーシミュレーターを使用します。IAM Policy Simulator を使用した IAM ポリシーのテストを参照してください。

別のアカウントから AWS Secrets Manager シークレットにアクセスする

1 つのアカウントのユーザーに別のアカウントのシークレットへのアクセス (クロスアカウントアクセス) を許可するには、リソースポリシーとアイデンティティポリシーの両方でアクセスを許可する必要があります。これは、シークレットと同じアカウントのアイデンティティにアクセスを許可することとは異なります。

クロスアカウント許可は、以下のオペレーションに対してのみ有効です。

- CancelRotateSecret
- DeleteResourcePolicy
- DeleteSecret
- DescribeSecret

- GetRandomPassword
- GetResourcePolicy
- GetSecretValue
- ListSecretVersionIds
- PutResourcePolicy
- PutSecretValue
- RemoveRegionsFromReplication
- ReplicateSecretToRegions
- RestoreSecret
- RotateSecret
- StopReplicationToReplica
- TagResource
- UntagResource
- UpdateSecret
- UpdateSecretVersionStage
- ValidateResourcePolicy

PutResourcePolicy アクションを指定して `BlockPublicPolicy` パラメータを使用し、シークレットに直接アタッチされているリソースポリシーを通じてパブリックアクセスが付与されないようにすることで、リソースを保護できます。IAM Access Analyzer を使用してクロスアカウントアクセスを確認することもできます。

また、シークレットの暗号化に使用している KMS キーをアイデンティティで使用できるようにする必要があります。これは、クロスアカウントアクセスに AWS マネージドキー (`aws/secretsmanager`) を使用できないためです。代わりに、作成した KMS キーを使用してシークレットを暗号化し、キーポリシーをそれにアタッチする必要があります。KMS キーの作成には料金が発生します。シークレットの暗号化キーを変更するには、the section called “シークレットの変更” を参照してください。

> **Important**
>
> `secretsmanager:PutResourcePolicy` アクセス許可を付与するリソースベースのポリシーは、他のアカウントのプリンシパルであっても、リソースベースのポリシーを変更できるようにします。このアクセス許可により、プリンシパルはシークレットへの完全な管理

アクセスの取得など、既存のアクセス許可を エスカレートできます。最小特権アクセスの原則をポリシーに適用することをお勧めします。詳細については、「リソースベースのポリシー」を参照してください。

以下のポリシーの例では、Account1 にシークレットと暗号化キーがあり、Account2 にシークレット値へのアクセスを許可するアイデンティティがあると仮定します。

ステップ 1: リソースポリシーを Account1 のシークレットにアタッチする

- 次のポリシーは、*Account2* の *ApplicationRole* に *Account1* のシークレットへのアクセスを許可します。このポリシーを使用するには、the section called “リソースベースのポリシー” を参照してください。

  JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:role/ApplicationRole"
            },
            "Action": "secretsmanager:GetSecretValue",
            "Resource": "*"
        }
    ]
}
```

ステップ 2: Account1 の KMS キーのキーポリシーにステートメントを追加する

- 次のキーポリシーステートメントは、*Account2* の *ApplicationRole* が *Account1* の KMS キーを使用して *Account1* のシークレットを復号するのを許可します。このステートメントを使用するには、それを KMS キーのキーポリシーに追加します。詳細については、キーポリシーの変更を参照してください。プリンシパルに必要な KMS キーのアクセス許可は、シークレットで実行するオペレーションによって異なります。シークレット値を読み取るには、プリンシパルに が必要ですkms:Decrypt。プリンシパルがシークレット値 (、、 などCreateSecret) PutSecretValueも作成または変更する場合はUpdateSecret、 も付与し

ますkms:GenerateDataKey。各オペレーションに必要な KMS キーのアクセス許可については、「」を参照してくださいthe section called “KMS キーのアクセス許可”。

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::Account2:role/ApplicationRole"
  },
  "Action": [
    "kms:Decrypt",
    "kms:DescribeKey"
  ],
  "Resource": "*"
}
```

ステップ 3: Account2 のアイデンティティにアイデンティティポリシーをアタッチする

- 次のポリシーは、*Account2* の *ApplicationRole* が *Account1* にある暗号化キーを使用て、*Account1* のシークレットにアクセスし、シークレット値を復号するのを許可します。このポリシーを使用するには、the section called “アイデンティティベースのポリシー” を参照してください。シークレットの ARN は、Secrets Manager コンソールのシークレット詳細ページの [Secret ARN] (シークレット ARN) で確認できます。または、describe-secret を呼び出すこともできます。

  JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "secretsmanager:GetSecretValue",
            "Resource": "arn:aws:secretsmanager:us-
east-1:123456789012:secret:secretName-AbCdEf"
        },
        {
            "Effect": "Allow",
            "Action": "kms:Decrypt",
            "Resource": "arn:aws:kms:us-
east-1:123456789012:key/EncryptionKey"
```

```
        }
    ]
}
```

オンプレミス環境からシークレットにアクセスする

AWS Identity and Access Management Roles Anywhere を使用して、 の外部で実行されるサーバー、コンテナ、アプリケーションなどのワークロードの一時的なセキュリティ認証情報を IAM で取得できます AWS。ワークロードは、 AWS アプリケーションで使用するのと同じ IAM ポリシーと IAM ロールを使用して AWS リソースにアクセスできます。IAM Roles Anywhere を使用すると、AWS 内のリソースからアクセスが可能な認証情報だけでなく、アプリケーションサーバーなどのオンプレミスデバイスがアクセスする認証情報についても、Secrets Manager で保存および管理できるようになります。詳細については、「IAM Roles Anywhere ユーザーガイド」を参照してください。

でのデータ保護 AWS Secrets Manager

責任 AWS 共有モデル、 でのデータ保護に適用されます AWS Secrets Manager。このモデルで説明されているように、 AWS はすべての を実行するグローバルインフラストラクチャを保護する責任があります AWS クラウド。お客様は、このインフラストラクチャでホストされているコンテンツに対する管理を維持する責任があります。このコンテンツには、使用される AWS のサービス のセキュリティ設定と管理タスクが含まれます。データプライバシーの詳細については、「データプライバシーのよくある質問」を参照してください。欧州でのデータ保護の詳細については、AWS セキュリティブログに投稿された「AWS 責任共有モデルおよび GDPR」のブログ記事を参照してください。

データ保護の目的で、(AWS Identity and Access Management IAM) を使用して AWS アカウント 認証情報を保護し、個々のユーザーアカウントを設定することをお勧めします。この方法により、それぞれのジョブを遂行するために必要なアクセス許可のみを各ユーザーに付与できます。また、次の方法でデータを保護することをお勧めします。

- 各アカウントで多要素認証 (MFA) を使用します。
- SSL/TLS を使用して AWS リソースと通信します。Secrets Manager は、すべてのリージョンで TLS 1.2 および 1.3 をサポートしています。また、Secrets Manager は TLS (PQTLS) ネットワーク暗号化プロトコル用のハイブリッドポスト量子キー交換オプションもサポートしています。
- IAM プリンシパルに関連付けられているアクセスキー ID とシークレットアクセスキーを使用して、Secrets Manager へのプログラムリクエストに署名します。または、AWS Security Token

Service (AWS STS) を使用して、一時的なセキュリティ認証情報を生成し、リクエストに署名することもできます。

- で API とユーザーアクティビティのログ記録を設定します AWS CloudTrail。「the section called “で ログ記録する AWS CloudTrail ”」を参照してください。
- コマンドラインインターフェイスまたは API AWS を介して にアクセスするときに FIPS 140-2 検証済み暗号化モジュールが必要な場合は、FIPS エンドポイントを使用します。「the section called “Secrets Manager エンドポイント”」を参照してください。
- を使用して Secrets Manager AWS CLI にアクセスする場合は、the section called “を使用して AWS Secrets Manager シークレット AWS CLI を保存するリスクを軽減する”。

保管中の暗号化

Secrets Manager は AWS Key Management Service 、 (AWS KMS) による暗号化を使用して、保管中のデータの機密性を保護します。 は、多くのサービスで使用されるキーストレージと暗号化 AWS サービス AWS KMS を提供します。Secrets Manager のシークレットはすべて、一意のデータキーで暗号化されます。各データキーは、KMS キーで保護されます。Secrets Manager AWS マネージドキー でアカウントにデフォルトの暗号化を使用することも、 AWS KMSで独自のカスタマー管理キーを作成することもできます。カスタマー管理キーを使用すると、KMS キーアクティビティの認可をきめ細かく制御できます。詳細については、「the section called “シークレット暗号化と復号”」を参照してください。

転送中の暗号化

Secrets Manager は、転送中のデータを暗号化するための安全なプライベートエンドポイントを提供します。セキュアエンドポイントとプライベートエンドポイントにより AWS 、 は Secrets Manager への API リクエストの整合性を保護できます。 AWS では、X.509 証明書または Secrets Manager シークレットアクセスキーを使用して、呼び出し元が API コールに署名する必要があります。この要件は、署名バージョン 4 署名プロセス (Sigv4) に記載されています。

AWS Command Line Interface (AWS CLI) またはいずれかの AWS SDKs を使用して を呼び出す場合は AWS、使用するアクセスキーを設定します。その後、これらのツールは自動的にアクセスキーを使用してリクエストに署名します。「the section called “を使用して AWS Secrets Manager シークレット AWS CLI を保存するリスクを軽減する”」を参照してください。

ネットワーク間トラフィックのプライバシー

AWS には、既知のネットワークルートとプライベートネットワークルートを介してトラフィックをルーティングするときにプライバシーを維持するためのオプションが用意されています。

サービスとオンプレミスのクライアントおよびアプリケーションとの間のトラフィック

プライベートネットワークと の間には 2 つの接続オプションがあります AWS Secrets Manager。

- An AWS Site-to-Site VPN 接続。詳細については、AWS Site-to-Site VPN とは」を参照してください。
- AWS Direct Connect 接続。詳細については、「AWS Direct Connect とは」を参照してください。

同じリージョン内の AWS リソース間のトラフィック

で Secrets Manager と API クライアント間のトラフィックを保護する場合は AWS、Secrets Manager API エンドポイントにプライベートにアクセスするように AWS PrivateLink を設定します。

暗号化キーの管理

Secrets Manager が保護されたシークレットデータの新しいバージョンを暗号化する必要がある場合、Secrets Manager は にリクエストを送信 AWS KMS して KMS キーから新しいデータキーを生成します。Secrets Manager は、このデータキーをエンベロープ暗号化に使用します。Secrets Manager は、暗号化されたシークレットを使用して、暗号化されたデータキーを保存します。シークレットを復号化する必要がある場合、Secrets Manager はデータキーを復号 AWS KMS するように求めます。Secrets Manager は、復号されたデータキーを使用して、暗号化されたシークレットを復号します。Secrets Manager では、データキーは暗号化されていない形式で保存されることはなく、キーはメモリから速やかに削除されます。詳細については、「the section called “シークレット暗号化と復号”」を参照してください。

でのシークレットの暗号化と復号 AWS Secrets Manager

Secrets Manager は、エンベロープ暗号化を AWS KMS キーおよびデータキーと共に使用して各シークレット値を保護します。シークレットのシークレット値が変更されるたびに、Secrets Manager はそれを保護するために新しいデータキーを AWS KMS からリクエストします。データキーは、KMS キーの下で暗号化され、シークレットのメタデータに保存されます。シークレットを

復号するために、Secrets Manager はまず KMS キーを使用して暗号化されたデータキーを復号します AWS KMS。

Secrets Manager は、シークレット値を直接暗号化するとき、KMS キーを使用しません。代わりに、KMS キーを使用して 256 ビット Advanced Encryption Standard (AES) 対称型データキーを生成し、このデータキーを使用してシークレット値を暗号化します。Secrets Manager は、プレーンテキストのデータキーを使用して外部でシークレット値を暗号化し AWS KMS、メモリから削除します。また、データキーの暗号化されたコピーを、シークレットのメタデータに保存します。

トピック

- AWS KMS キーの選択
- 暗号化されるもの
- プロセスの暗号化と復号
- KMS キーのアクセス許可
- Secrets Manager による KMS キーの使用方法
- AWS マネージドキー (aws/secretsmanager) のキーポリシー
- Secrets Manager の暗号化コンテキスト
- Secrets Manager の とのやり取りをモニタリングする AWS KMS

AWS KMS キーの選択

シークレットを作成するときは、 AWS アカウント および リージョンで任意の対称暗号化カスタマーマネージドキーを選択するか、Secrets Manager () AWS マネージドキー の を使用できますaws/secretsmanager。を選択し AWS マネージドキー aws/secretsmanager、まだ存在しない場合は、Secrets Manager によって作成され、シークレットに関連付けられます。アカウントの各シークレットに、同じ KMS キーまたは異なる KMS キーを使用できます。異なる KMS キーを使用して、シークレットのグループのキーにカスタムアクセス許可を設定したり、それらのキーの特定の操作を監査したりする場合があります。Secrets Manager は、対称型 暗号化KMS キーのみをサポートします。外部キーストアで KMS キーを使用する場合、要求が AWSの外に移動する必要があるため、KMS キーの暗号化操作に時間がかかり、信頼性と耐久性が低下する可能性があります。

シークレットの暗号化キーの変更の詳細については、「the section called “シークレットの暗号化キーを変更する”」を参照してください。

暗号化キーを変更すると、Secrets Manager は新しいキーを使用して、AWSCURRENT、AWSPENDING、AWSPREVIOUS バージョンを再暗号化します。シークレットから

ロックアウトされないように、Secrets Manager は既存のすべてのバージョンを以前のキーで暗号化したままの状態にします。つまり、AWSCURRENT、AWSPENDING、AWSPREVIOUS のバージョンを以前のキーまたは新しいキーで復号化できます。以前のキーに対する kms:Decrypt アクセス許可がない場合、暗号化キーを変更すると、Secrets Manager はシークレットバージョンを復号して再暗号化することはできません。この場合、既存のバージョンは再暗号化されません。

AWSCURRENT を新しい暗号化キーでのみ復号できるようにするには、新しいキーを使用してシークレットの新しいバージョンを作成します。次に、AWSCURRENT シークレットのバージョンを復号するには、新しいキーに対するアクセス許可が必要です。

へのアクセス AWS マネージドキー aws/secretsmanager許可を拒否し、シークレットがカスタマーマネージドキーで暗号化されるように要求できます。詳細については、「the section called “例: シークレットを暗号化するための特定の AWS KMS キーを拒否する”」を参照してください。

シークレットに関連付けられている KMS キーを検索するには、コンソールにシークレットを表示するか、ListSecrets または DescribeSecret を呼び出します。シークレットが Secrets Manager (aws/secretsmanager) AWS マネージドキー の に関連付けられている場合、これらのオペレーションは KMS キー識別子を返しません。

暗号化されるもの

Secrets Manager ではシークレット値を暗号化しますが、次の値は暗号化しません。

- シークレットの名前と説明
- ローテーション設定
- シークレットに関連付けられた KMS キーの ARN
- アタッチされた AWS タグ

プロセスの暗号化と復号

シークレットのシークレット値を暗号化するには、Secrets Manager は次のプロセスを使用します。

1. Secrets Manager は、シークレットの KMS キーの ID と 256 ビット AES 対称キーのリクエストを使用して、 AWS KMS GenerateDataKey オペレーションを呼び出します。 は、プレーンテキストのデータキーと、KMS キーで暗号化されたそのデータキーのコピー AWS KMS を返します。
2. Secrets Manager は、プレーンテキストのデータキーと Advanced Encryption Standard (AES) アルゴリズムを使用して、外部でシークレット値を暗号化します AWS KMS。次に、使用後可能な限り早く、メモリからプレーンテキストキーが削除されます。

3. Secrets Manager は、暗号化されたデータキーをシークレットのメタデータに保存するので、シークレット値を復号化できます。ただし、Secrets Manager API のいずれも、暗号化されたシークレットまたは暗号化されたデータキーを返しません。

暗号化されたシークレット値を復号するには:

1. Secrets Manager は、 AWS KMS Decrypt オペレーションを呼び出し、暗号化されたデータキーを渡します。
2. AWS KMS はシークレットの KMS キーを使用してデータキーを復号します。プレーンテキストのデータキーを返します。
3. Secrets Manager は、プレーンテキストのデータキーを使用してシークレット値を復号化します。次に、可能な限り早く、メモリからデータキーが削除されます。

KMS キーのアクセス許可

Secrets Manager が暗号化オペレーションで KMS キーを使用する場合、シークレット値をアクセスまたは更新しているユーザーの代わりに動作します。IAM ポリシーまたはキーポリシーでアクセス許可を付与できます。次の Secrets Manager オペレーションにはアクセス AWS KMS 許可が必要です。

- CreateSecret
- GetSecretValue
- PutSecretValue
- UpdateSecret
- ReplicateSecretToRegions

Secrets Manager で発生したリクエストにのみ KMS キーを使用するには、アクセス許可ポリシーで kms:ViaService 条件キーを `secretsmanager.<Region>.amazonaws.com` 値で使用します。

また、暗号化オペレーションに KMS キーを使用する条件として、暗号化コンテキストでキーまたは値を使用することもできます。例えば、IAM またはキーポリシードキュメントで文字列条件演算子を使用したり、制約許可を与えられます。KMS キー付与の伝播には、最大 5 分かかります。詳細については、「CreateGrant」を参照してください。

Secrets Manager による KMS キーの使用方法

Secrets Manager は、KMS キーを使用して次の AWS KMS オペレーションを呼び出します。

GenerateDataKey

Secrets Manager は、次の Secrets Manager オペレーションに応答して、 AWS KMS GenerateDataKey オペレーションを呼び出します。

- CreateSecret — 新しいシークレットにシークレット値が含まれている場合、Secrets Manager は、それを暗号化するための新しいデータキーを要求します。
- PutSecretValue — Secrets Manager は、指定されたシークレット値を暗号化する新しいデータキーを要求します。
- ReplicateSecretToRegions – レプリケートされたシークレットを暗号化するために、Secrets Manager がレプリカリージョン内の KMS キーのデータキーをリクエストします。
- UpdateSecret — シークレット値または KMS キーを変更した場合、Secrets Manager は新しいシークレット値を暗号化するための新しいデータキーを要求します。

rotateSecret 操作は、シークレット値を変更しないため、`GenerateDataKey`を呼び出しません。ただし、`RotateSecret` がシークレット値を変更する Lambda 関数を呼び出す場合、`PutSecretValue` オペレーションの呼び出しにより、`GenerateDataKey` リクエストがトリガーされます。

Decrypt

Secrets Manager は、次の Secrets Manager オペレーションに応答して Decrypt オペレーションを呼び出します。

- GetSecretValue と BatchGetSecretValue — Secrets Manager は、呼び出し元に返す前にシークレット値を復号化します。暗号化されたシークレット値を復号するために、Secrets Manager は AWS KMS Decrypt オペレーションを呼び出してシークレット内の暗号化されたデータキーを復号します。次に、プレーンテキストのデータキーを使って、暗号化されたシークレット値を復号します。バッチコマンドの場合、Secrets Manager は復号化されたキーを再利用できるため、すべての呼び出しが `Decrypt` リクエストにつながるわけではありません。
- PutSecretValue および UpdateSecret — ほとんどの `PutSecretValue` および `UpdateSecret` 要求は `Decrypt` 操作をトリガーしません。ただし、`PutSecretValue` または `UpdateSecret` 要求がシークレットの既存のバージョンでシークレット値を変更しようとすると、Secrets Manager は既存のシークレット値を復号化し、リクエスト内のシークレット値と比較して、それらが同じであることを確認します。このアクションは、Secrets

Manager の操作が冪等であることを保証します。暗号化されたシークレット値を復号するために、Secrets Manager は AWS KMS Decrypt オペレーションを呼び出してシークレット内の暗号化されたデータキーを復号します。次に、プレーンテキストのデータキーを使って、暗号化されたシークレット値を復号します。

- ReplicateSecretToRegions – Secrets Manager は、レプリカリージョンにある新しい KMS キーを使用してシークレット値を再暗号化する前に、まずプライマリリージョンのシークレット値を復号します。

暗号化

Secrets Manager は、次の Secrets Manager オペレーションに応答して Encrypt オペレーションを呼び出します。

- UpdateSecret – KMS キーを変更すると、Secrets Manager は `AWSCURRENT`、`AWSPREVIOUS`、`AWSPENDING` およびシークレットバージョンを保護するデータキーを新しいキーで再暗号化します。

DescribeKey

Secrets Manager は、DescribeKey オペレーションを呼び出して、Secrets Manager コンソールでシークレットを作成または編集するときに KMS キーを一覧表示するかどうかを決定します。

KMS キーへのアクセスの検証

シークレットに関連付けられている KMS キーを確立または変更すると、Secrets Manager は、指定された KMS キーを使用して `GenerateDataKey` および `Decrypt` オペレーションを呼び出します。これらの呼び出しでは、呼び出し元に、これらのオペレーションで KMS キーを使用するアクセス許可があることが確認されます。Secrets Manager は、これらの操作の結果を破棄します。暗号化操作ではそれらを使用しません。

これらのリクエストの `SecretVersionId` キーの暗号化コンテキストの値は `RequestToValidateKeyAccess` であるため、この検証呼び出しを識別できます。

> **Note**
>
> 以前は、Secrets Manager の検証呼び出しに暗号化コンテキストが含まれていませんでした。古い AWS CloudTrail ログでは、暗号化コンテキストのない呼び出しが見つかる場合があります。

AWS マネージドキー (aws/secretsmanager) のキーポリシー

AWS マネージドキー Secrets Manager (aws/secretsmanager) の のキーポリシーは、Secrets Manager がユーザーに代わってリクエストを行う場合にのみ、指定されたオペレーションに KMS キーを使用するアクセス許可をユーザーに付与します。このキーポリシーでは、ユーザーが KMS キーを直接使用することは許可されません。

このキーポリシーは、すべての AWS マネージドキー のポリシーと同様に、サービスによって確立されます。キーポリシーは変更できませんが、いつでも表示できます。詳細については、「Viewing a key policy」を参照してください。

このキーポリシーのポリシーステートメントには次の効果があります

- アカウントのユーザーが暗号化オペレーションに KMS キーを使用できるようにするのは、リクエストがSecrets Manager から送信される場合のみです。kms:ViaService 条件キーで、この制限を適用します。
- AWS アカウントが KMS キープロパティを表示し、許可を取り消すことをユーザーに許可する IAM ポリシーを作成できるようにします。
- Secrets Manager は、KMS キーへのアクセスを取得するときは許可を使用しませんが、このポリシーは、ユーザーに代わって Secrets Manager が KMS キーの許可を作成すること、および、アカウントが KMS キーの使用を Secrets Manager に許可するための許可を取り消すことを、許可します。これらは、 のポリシードキュメントの標準要素です AWS マネージドキー。

以下は、Secrets Manager の例 AWS マネージドキー のキーポリシーです。

JSON

```
{
  "Id": "auto-secretsmanager-2",
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "Allow access through AWS Secrets Manager for all principals in the
 account that are authorized to use AWS Secrets Manager",
      "Effect": "Allow",
      "Principal": {
        "AWS": [
          "*"
        ]
```

```
      },
      "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:CreateGrant",
        "kms:DescribeKey"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "kms:CallerAccount": "111122223333",
          "kms:ViaService": "secretsmanager.us-west-2.amazonaws.com"
        }
      }
    },
    {
      "Sid": "Allow access through AWS Secrets Manager for all principals in the
account that are authorized to use AWS Secrets Manager",
      "Effect": "Allow",
      "Principal": {
        "AWS": [
          "*"
        ]
      },
      "Action": "kms:GenerateDataKey*",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "kms:CallerAccount": "111122223333"
        },
        "StringLike": {
          "kms:ViaService": "secretsmanager.us-west-2.amazonaws.com"
        }
      }
    },
    {
      "Sid": "Allow direct access to key metadata to the account",
      "Effect": "Allow",
      "Principal": {
        "AWS": [
          "arn:aws:iam::111122223333:root"
        ]
      },
```

```
      "Action": [
        "kms:Describe*",
        "kms:Get*",
        "kms:List*",
        "kms:RevokeGrant"
      ],
      "Resource": "*"
    }
  ]
}
```

Secrets Manager の暗号化コンテキスト

暗号化コンテキストは、シークレットデータを含まない一連のキーと値のペアです。データを暗号化するリクエストに暗号化コンテキストを含めると、 は暗号化コンテキストを暗号化されたデータに AWS KMS 暗号化バインドします。データを復号するには、同じ暗号化コンテキストに渡す必要があります。

への GenerateDataKey および Decrypt リクエストでは AWS KMS、Secrets Manager は、次の例に示すように、シークレットとそのバージョンを識別する 2 つの名前と値のペアを持つ暗号化コンテキストを使用します。名前は変わりませんが、組み合わされた暗号化コンテキストの値は、シークレット値ごとに異なります。

```
"encryptionContext": {
    "SecretARN": "arn:aws:secretsmanager:us-east-2:111122223333:secret:test-secret-
a1b2c3",
    "SecretVersionId": "EXAMPLE1-90ab-cdef-fedc-ba987SECRET1"
}
```

暗号化コンテキストを使用して、AWS CloudTrail や Amazon CloudWatch Logs などの監査レコードやログで、ポリシーや許可の認可の条件として、これらの暗号化オペレーションを識別できます。

Secrets Manager の暗号化コンテキストは、2 つの名前と値のペアで構成されます。

- SecretArn – 最初の名前と値のペアがシークレットを識別します。キーは、SecretARN です。値はシークレットの Amazon リソースネーム (ARN) です。

  ```
  "SecretARN": "ARN of an Secrets Manager secret"
  ```

例えば、シークレットの ARN が arn:aws:secretsmanager:us-east-2:111122223333:secret:test-secret-a1b2c3 である場合 、暗号化コンテキストには次のペアが含まれます。

```
"SecretARN": "arn:aws:secretsmanager:us-east-2:111122223333:secret:test-secret-a1b2c3"
```

- SecretVersionID – 2 番目の名前と値のペアは、シークレットのバージョンを識別します。キーは、SecretVersionId です。値は、バージョン ID です。

```
"SecretVersionId": "<version-id>"
```

例えば、シークレットのバージョン ID が EXAMPLE1-90ab-cdef-fedc-ba987SECRET1 である場合、暗号化コンテキストには次のペアが含まれます。

```
"SecretVersionId": "EXAMPLE1-90ab-cdef-fedc-ba987SECRET1"
```

シークレットの KMS キーを確立または変更すると、Secrets Manager は GenerateDataKey および Decrypt リクエストを に送信 AWS KMS して、呼び出し元がこれらのオペレーションに KMS キーを使用するアクセス許可を持っていることを確認します。レスポンスは廃棄され、シークレット値では使用されません。

これらの検証リクエストでは、SecretARN の値がシークレットの実際の ARN となりますが、SecretVersionId 値は、次の暗号化コンテキストの例に示すように RequestToValidateKeyAccess になります。この特殊な値は、ログと監査証跡で検証リクエストを識別するうえで役立ちます。

```
"encryptionContext": {
    "SecretARN": "arn:aws:secretsmanager:us-east-2:111122223333:secret:test-secret-a1b2c3",
    "SecretVersionId": "RequestToValidateKeyAccess"
}
```

> **Note**
>
> 以前は、Secrets Manager の検証要求に暗号化コンテキストが含まれていませんでした。古い AWS CloudTrail ログでは、暗号化コンテキストのない呼び出しが見つかる場合があります。

Secrets Manager の とのやり取りをモニタリングする AWS KMS

AWS CloudTrail および Amazon CloudWatch Logs を使用して、Secrets Manager が AWS KMS ユーザーに代わって に送信するリクエストを追跡できます。シークレットの使用のモニタリングについては、「シークレットを監視する」を参照してください。

GenerateDataKey

シークレットでシークレット値を作成または変更すると、Secrets Manager はシークレットの KMS キー AWS KMS を指定する GenerateDataKey リクエストを に送信します。

GenerateDataKey 演算を記録するイベントは、次のようなサンプルイベントになります。リクエストは secretsmanager.amazonaws.com によって起動されます。このパラメータには、シークレットの KMS キーの Amazon リソースネーム (ARN)、256 ビットキーを要求するキー識別子、およびシークレットとバージョンを識別する暗号化コンテキストが含まれます。


```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AROAIGDTESTANDEXAMPLE:user01",
        "arn": "arn:aws:sts::111122223333:assumed-role/Admin/user01",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2018-05-31T23:23:41Z"
            }
        },
        "invokedBy": "secretsmanager.amazonaws.com"
    },
    "eventTime": "2018-05-31T23:23:41Z",
    "eventSource": "kms.amazonaws.com",
```

```
    "eventName": "GenerateDataKey",
    "awsRegion": "us-east-2",
    "sourceIPAddress": "secretsmanager.amazonaws.com",
    "userAgent": "secretsmanager.amazonaws.com",
    "requestParameters": {
        "keyId": "arn:aws:kms:us-
east-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
        "keySpec": "AES_256",
        "encryptionContext": {
            "SecretARN": "arn:aws:secretsmanager:us-east-2:111122223333:secret:test-
secret-a1b2c3",
            "SecretVersionId": "EXAMPLE1-90ab-cdef-fedc-ba987SECRET1"
        }
    },
    "responseElements": null,
    "requestID": "a7d4dd6f-6529-11e8-9881-67744a270888",
    "eventID": "af7476b6-62d7-42c2-bc02-5ce86c21ed36",
    "readOnly": true,
    "resources": [
        {
            "ARN": "arn:aws:kms:us-
east-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
            "accountId": "111122223333",
            "type": "AWS::KMS::Key"
        }
    ],
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}
```


Decrypt

シークレットのシークレット値を取得または変更すると、Secrets Manager は Decrypt リクエストを に送信 AWS KMS して、暗号化されたデータキーを復号します。バッチコマンドの場合、Secrets Manager は復号化されたキーを再利用できるため、すべての呼び出しが Decrypt リクエストにつながるわけではありません。

Decrypt 演算を記録するイベントは、次のようなサンプルイベントになります。ユーザーは、テーブルにアクセスする AWS アカウントのプリンシパルです。パラメータには、暗号化されたテーブルキー (暗号化テキスト BLOB として) と、テーブルと AWS アカウントを識別する暗号化コンテキストが含まれます。 は、暗号化テキストから KMS キーの ID AWS KMS を取得します。

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AROAIGDTESTANDEXAMPLE:user01",
        "arn": "arn:aws:sts::111122223333:assumed-role/Admin/user01",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2018-05-31T23:36:09Z"
            }
        },
        "invokedBy": "secretsmanager.amazonaws.com"
    },
    "eventTime": "2018-05-31T23:36:09Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "Decrypt",
    "awsRegion": "us-east-2",
    "sourceIPAddress": "secretsmanager.amazonaws.com",
    "userAgent": "secretsmanager.amazonaws.com",
    "requestParameters": {
        "encryptionContext": {
            "SecretARN": "arn:aws:secretsmanager:us-east-2:111122223333:secret:test-
secret-a1b2c3",
            "SecretVersionId": "EXAMPLE1-90ab-cdef-fedc-ba987SECRET1"
        }
    },
    "responseElements": null,
    "requestID": "658c6a08-652b-11e8-a6d4-ffee2046048a",
    "eventID": "f333ec5c-7fc1-46b1-b985-cbda13719611",
    "readOnly": true,
    "resources": [
        {
            "ARN": "arn:aws:kms:us-
east-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
            "accountId": "111122223333",
            "type": "AWS::KMS::Key"
        }
    ],
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
```

```
}
```

暗号化

シークレットに関連付けられた KMS キーを変更すると、Secrets Manager は Encrypt リクエストを に送信 AWS KMS してAWSCURRENT、、AWSPREVIOUS、および シークAWSPENDINGレットバージョンを新しいキーで再暗号化します。シークレットを別のリージョンにレプリケートすると、Secrets Manager は Encrypt リクエストも AWS KMS に送信します。

Encrypt 演算を記録するイベントは、次のようなサンプルイベントになります。ユーザーは、テーブルにアクセスする AWS アカウントのプリンシパルです。

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AROAIGDTESTANDEXAMPLE:user01",
        "arn": "arn:aws:sts::111122223333:assumed-role/Admin/user01",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "attributes": {
                "creationDate": "2023-06-09T18:11:34Z",
                "mfaAuthenticated": "false"
            }
        },
        "invokedBy": "secretsmanager.amazonaws.com"
    },
    "eventTime": "2023-06-09T18:11:34Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "Encrypt",
    "awsRegion": "us-east-2",
    "sourceIPAddress": "secretsmanager.amazonaws.com",
    "userAgent": "secretsmanager.amazonaws.com",
    "requestParameters": {
        "keyId": "arn:aws:kms:us-east-2:111122223333:key/EXAMPLE1-f1c8-4dce-8777-
aa071ddefdcc",
        "encryptionAlgorithm": "SYMMETRIC_DEFAULT",
        "encryptionContext": {
            "SecretARN": "arn:aws:secretsmanager:us-
east-2:111122223333:secret:ChangeKeyTest-5yKnKS",
            "SecretVersionId": "EXAMPLE1-5c55-4d7c-9277-1b79a5e8bc50"
        }
```

```
    },
    "responseElements": null,
    "requestID": "129bd54c-1975-4c00-9b03-f79f90e61d60",
    "eventID": "f7d9ff39-15ab-47d8-b94c-56586de4ab68",
    "readOnly": true,
    "resources": [
        {
            "accountId": "AWS Internal",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-west-2:111122223333:key/EXAMPLE1-f1c8-4dce-8777-
aa071ddefdcc"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```


のインフラストラクチャセキュリティ AWS Secrets Manager

マネージドサービスである AWS Secrets Manager は、 AWS グローバルネットワークセキュリティによって保護されています。 AWS セキュリティサービスと がインフラストラクチャ AWS を保護する方法については、AWS 「 クラウドセキュリティ」を参照してください。インフラストラクチャセキュリティのベストプラクティスを使用して AWS 環境を設計するには、「Security Pillar AWS Well-Architected Framework」の「Infrastructure Protection」を参照してください。

ネットワークを介した Secrets Manager へのアクセスには、TSL を使用した API が発行するAWS が使用されます。これらの Secrets Manager API はネットワークの任意の場所から呼び出すことができます。ただし、Secrets Manager はリソースベースのアクセスポリシーをサポートしており、それらのポリシーには、ソース IP アドレスに基づく制限を含めることができます。また、Secrets Manager ソースのポリシーを使用して、特定の仮想プライベートクラウド (VPC) エンドポイント、または特定の VPC からのシークレットへのアクセスを制御することもできます。これにより、実質的にネットワーク内の特定の VPC からのみ、特定のシークレットへの AWS ネットワークアクセスが分離されます。詳細については、「the section called “VPC エンドポイント (AWS PrivateLink)”」を参照してください。

AWS Secrets Manager VPC エンドポイントの使用

パブリックインターネットからアクセスできないプライベートネットワーク上で、できるだけ多くのインフラストラクチャを実行することをお勧めします。VPC と Secrets Manager とのプライベート接続は、インターフェイス VPC エンドポイントを作成すると、確立できます。インターフェイスエンドポイントは、インターネットゲートウェイAWS PrivateLink、NAT デバイス、VPN 接続、または Direct Connect 接続なしで Secrets Manager APIs にプライベートにアクセスできるテクノロジーである を利用しています。VPC にあるインスタンスは、Secrets Manager API と通信するときに、パブリック IP アドレスを必要としません。VPC と Secrets Manager 間のトラフィックは、 AWS ネットワークを離れません。詳細については、「Amazon VPC ユーザーガイド」の「インターフェイス VPC エンドポイント (AWS PrivateLink)」を参照してください。

シークレットマネージャーが Lambda ローテーション関数を使用して、例えばデータベース認証情報を含むシークレットをローテーションすると、Lambda 関数は、データベースと Secrets Manager の両方にリクエストを送信します。コンソールを使用して自動ローテーションをオンにすると、Secrets Manager はデータベースと同じ VPC に Lambda 関数を作成します。Lambda ローテーション関数から Secrets Manager へのリクエストが Amazon ネットワークから出ないように、同じ VPC に Secrets Manager エンドポイントを作成することをお勧めします。

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 (`secretsmanager.us-east-1.amazonaws.com` など) を使って Secrets Manager への API リクエストを実行できます。詳細については、「Amazon VPC ユーザーガイド」の「インターフェイスエンドポイントを介したサービスへのアクセス」を参照してください。

アクセス許可ポリシーに条件を含めることで、Secrets Manager へのリクエストが VPC アクセスから来るようにすることができます。詳細については、「the section called “例: アクセス許可と VPC”」を参照してください。

AWS CloudTrail ログを使用して、VPC エンドポイントを介したシークレットの使用を監査できます。

Secrets Manager の VPC エンドポイントを作成するには

1. 「Amazon VPC ユーザーガイド」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。次のいずれかのサービス名を使用します。

   - `com.amazonaws.region.secretsmanager`
   - `com.amazonaws.region.secretsmanager-fips`

2. エンドポイントへのアクセスを制御するには、「エンドポイントポリシーを使用した VPC エンドポイントへのアクセス制御」を参照してください。
3. IPv6 とデュアルスタックのアドレス指定を使用するには、「IPv4 および IPv6 アクセス」を参照してください。

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を介した Secrets Manager へのフルアクセスが許可されています。VPC から Secrets Manager に許可されるアクセスを制御するには、カスタム・エンドポイント・ポリシーをインターフェースの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: Secrets Manager 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。この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すると、指定されたシークレットでリストされた Secrets Manager アクションへのアクセスが許可されます。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "Allow all users to use GetSecretValue and DescribeSecret on the
 specified secret.",
      "Effect": "Allow",
      "Principal": "*",
```

```
      "Action": [
        "secretsmanager:GetSecretValue",
        "secretsmanager:DescribeSecret"
      ],
      "Resource": "arn:aws:secretsmanager:us-
east-1:111122223333:secret:secretName-AbCdEf"
    }
  ]
}
```

共有サブネット

自分と共有されているサブネットで VPC エンドポイントを作成、説明、変更、または削除することはできません。ただし、VPC エンドポイントを使用することはできます。VPC 共有の詳細については、「Amazon Virtual Private Cloud ユーザーガイド」の「VPC を他のアカウントと共有する」を参照してください。

IAM ポリシーによる API アクセスコントロール

IAM ポリシーを使用して IP アドレス AWS のサービス に基づいて へのアクセスを制御する場合は、IPv6 アドレス範囲を含めるようにポリシーを更新する必要がある場合があります。このガイドでは、IPv4 と IPv6 の違いと、両方のプロトコルをサポートするように IAM ポリシーを更新する方法について説明します。これらの変更を実装することで、IPv6 をサポートしながら AWS リソースへの安全なアクセスを維持できます。

IPv6 とは

IPv6 は、最終的に IPv4 を IPv6 に置き換えることを意図した次世代の IP 規格です。以前のバージョンの IPv4 は、32 ビットのアドレス指定方式を使用して 43 億台のデバイスをサポートしていました。IPv6 は代わりに 128 ビットのアドレス指定を使用して、約 340 兆 x1 兆倍 x1 兆倍 (つまり 2 の 128 乗) のデバイスをサポートします。

詳細については、VPC IPv6 のウェブページを参照してください。

IPv6 アドレスの例を次に示します。

```
2001:cdba:0000:0000:0000:0000:3257:9652 # This is a full, unabbreviated IPv6 address.
2001:cdba:0:0:0:0:3257:9652             # The same address with leading zeros in each
 group omitted
```

```
2001:cdba::3257:965                    # A compressed version of the same address.
```

IAM デュアルスタック (IPv4 および IPv6) ポリシー

IAM ポリシーを使用して Secrets Manager API へのアクセスを制御し、設定された範囲外の IP アドレスからの Secrets Manager API へのアクセスをブロックできます。

Secrets Manager API の secretsmanager.{region}.amazonaws.com デュアルスタックエンドポイントは、IPv6 と IPv4 の両方をサポートしています。

IPv4 と IPv6 の両方をサポートする必要がある場合は、IPv6 アドレスを処理するように、IP アドレスフィルタリングポリシーを更新します。それ以外の場合は、IPv6 経由で Secrets Manager に接続できなくなる場合があります。

この変更を行うべき状況

この変更は、aws:sourceIp を含むポリシーでデュアルアドレス指定を使用する場合に影響します。デュアルアドレス指定とは、ネットワークが IPv4 と IPv6 の両方をサポートすることを意味します。

デュアルアドレス指定を使用する場合は、現在 IPv4 形式のアドレスを使用している IAM ポリシーを、IPv6 形式のアドレスを含むように更新する必要があります。

この変更を行うべきではない状況

この変更は、IPv4 ネットワークのみを使用する場合には影響しません。

IAM ポリシーに IPv6 を追加する

IAM ポリシーは、aws:SourceIp 条件キーを使用して、特定の IP アドレスからのアクセスを制御します。ネットワークでデュアルアドレス指定 (IPv4 および IPv6) を使用している場合は、IAM ポリシーを更新して IPv6 アドレス範囲を含めます。

ポリシーの Condition 要素で、IP アドレス条件に IpAddress および NotIpAddress 演算子を使用します。さまざまな有効な IPv6 アドレス形式を処理できないため、文字列演算子を使用しないでください。

以下の例では、aws:SourceIp を使用します。VPC の場合は、代わりに aws:VpcSourceIp を使用します。

以下は、IAM ユーザーガイドのソース IP リファレンスポリシー AWS に基づいて へのアクセスを拒否します。2 つの IPv4 アドレス範囲、192.0.2.0/24 および 203.0.113.0/24 を一覧表示する Condition 要素の NotIpAddress は、API へのアクセスが拒否されます。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Effect": "Deny",
        "Action": "*",
        "Resource": "*",
        "Condition": {
            "NotIpAddress": {
                "aws:SourceIp": [
                    "192.0.2.0/24",
                    "203.0.113.0/24"
                ]
            },
            "Bool": {
                "aws:ViaAWSService": "false"
            }
        }
    }
}
```

このポリシーを更新するには、Condition 要素を変更して、IPv6 アドレス範囲 の 2001:DB8:1234:5678::/64 と 2001:cdba:3257:8593::/64 を含むように更新します。

> **Note**
>
> 既存の IPv4 アドレスを削除しないでください。下位互換性のために必要です。

```
"Condition": {
                "NotIpAddress": {
                    "aws:SourceIp": [
                        "192.0.2.0/24", <<DO NOT REMOVE existing IPv4 address>>
                        "203.0.113.0/24", <<DO NOT REMOVE existing IPv4 address>>
                        "2001:DB8:1234:5678::/64", <<New IPv6 IP address>>
```

```
                    "2001:cdba:3257:8593::/64" <<New IPv6 IP address>>
                ]
            },
            "Bool": {
                "aws:ViaAWSService": "false"
            }
        }
```

VPC のこのポリシーを更新するには、aws:SourceIp の代わりに aws:VpcSourceIp を使用します。

```
"Condition": {
            "NotIpAddress": {
                "aws:VpcSourceIp": [
                    "10.0.2.0/24", <<DO NOT REMOVE existing IPv4 address>>
                    "10.0.113.0/24", <<DO NOT REMOVE existing IPv4 address>>
                    "fc00:DB8:1234:5678::/64", <<New IPv6 IP address>>
                    "fc00:cdba:3257:8593::/64" <<New IPv6 IP address>>
                ]
            },
            "Bool": {
                "aws:ViaAWSService": "false"
            }
        }
```

クライアントが IPv6 をサポートしていることを検証する

secretsmanager.{region}.amazonaws.com エンドポイントを使用する場合は、そのエンドポイントに接続できることを確認します。以下の手順では、検証を実行する方法について説明します。

この例では、Linux と curl バージョン 8.6.0 を使用し、amazonaws.com エンドポイントにある IPv6 対応エンドポイントを持つ AWS Secrets Manager サービスエンドポイントを使用します。

> **Note**
>
> secretsmanager.{region}.amazonaws.com は、一般的なデュアルスタックの命名規則とは異なります。Secrets Manager エンドポイントの完全なリストについては、「AWS Secrets Manager エンドポイント」を参照してください。
> を、サービスがあるリージョンと同じリージョン AWS リージョン に変更します。この例では、米国東部 (バージニア北部) – us-east-1 エンドポイントを使用します。

1. 次の dig コマンドを使用して、エンドポイントが IPv6 アドレスで解決されることを確認します。

```
$ dig +short AAAA secretsmanager.us-east-1.amazonaws.com

> 2600:1f18:e2f:4e05:1a8a:948e:7c08:c1c3
```

2. 次の curl コマンドを使用して、クライアント ネットワークが IPv6 接続を確立できるかどうかを確認します。レスポンスコードが 404 の場合は接続が成功したことを意味し、0 の場合は接続が失敗したことを意味します。

```
$ curl --ipv6 -o /dev/null --silent -w "\nremote ip: %{remote_ip}\nresponse code:
 %{response_code}\n" https://secretsmanager.us-east-1.amazonaws.com

> remote ip: 2600:1f18:e2f:4e05:1a8a:948e:7c08:c1c3
> response code: 404
```

リモート IP アドレスが特定され、さらにレスポンスコードが `0` でない場合は、IPv6 を使用してエンドポイントへのネットワーク接続が正常に確立されています。リモート IP は IPv6 アドレスである必要があります。これは、オペレーティングシステムがクライアントに有効なプロトコルを選択する必要があるからです。

リモート IP が空白の場合、またはレスポンスコードが `0` の場合は、クライアントネットワークまたはエンドポイントへのネットワークパスは IPv4 専用です。これを確認するには、次の curl コマンドを使用します。

```
$ curl -o /dev/null --silent -w "\nremote ip: %{remote_ip}\nresponse code:
 %{response_code}\n" https://secretsmanager.us-east-1.amazonaws.com

> remote ip: 3.123.154.250
> response code: 404
```

の耐障害性 AWS Secrets Manager

AWS は、 AWS リージョン およびアベイラビリティーゾーンを中心にグローバルインフラストラクチャを構築します。 は、低レイテンシー、高スループット、高度に冗長なネットワークで接続する複数の物理的に分離および分離されたアベイラビリティーゾーン AWS リージョン を提供します。アベイラビリティーゾーンでは、ゾーン間で中断することなく自動的にフェールオーバーするアプリ

ケーションとデータベースを設計および運用することができます。アベイラビリティーゾーンは、従来の単一または複数のデータセンターインフラストラクチャよりも優れた可用性、耐障害性、および拡張性をもたらします。

障害耐性とディザスタリカバリの詳細については、「信頼性の柱 – AWS 優れたアーキテクチャフレームワーク」を参照してください。

AWS リージョン およびアベイラビリティーゾーンの詳細については、AWS 「 グローバルインフラストラクチャ」を参照してください。

ポスト量子 TLS

Secrets Manager は、Transport Layer Security (TLS) ネットワーク暗号化プロトコル用のハイブリッドポスト量子キー交換オプションをサポートします。この TLS オプションは、Secrets Manager API エンドポイントに接続するときに使用できます。この機能はポスト量子アルゴリズムが標準化される前に提供されているため、これらのキー交換プロトコルの Secrets Manager コールへの影響のテストを開始できます。これらのオプションのハイブリッドポスト量子キー交換機能は、現在使用している TLS 暗号化と同等以上に安全であり、セキュリティ上のさらなる利点をもたらす可能性があります。ただし、現在使用されている従来のキー交換プロトコルと比較して、レイテンシーとスループットに影響します。AWSワークロード認証情報プロバイダーは、デフォルトで最高優先度のキー交換としてポスト量子 ML-KEM キー交換を使用します。

現在暗号化されているデータを潜在的な将来の攻撃から保護するために、AWSは量子耐性アルゴリズムまたはポスト量子アルゴリズムの開発に暗号コミュニティに参加しています。Secrets Manager エンドポイントにハイブリッドポスト量子キー交換暗号スイートを実装しました。これらのハイブリッド暗号スイートは、従来の要素とポスト量子要素を組み合わせたもので、これにより TLS 接続が少なくとも従来の暗号スイートと同じくらい強力になります。ただし、ハイブリッド暗号スイートのパフォーマンス特性と帯域幅要件は従来のキー交換メカニズムのものとは異なるため、API コールでテストすることをお勧めします。

Secrets Manager は、中国リージョンを除くすべてのリージョンで PQTLS をサポートしています。

ハイブリッドポスト量子 TLS の設定する

1. AWS共通ランタイムクライアントを Maven の依存関係に追加します。利用可能な最新バージョンを使用することをお勧めします。たとえば、以下のステートメントはバージョン 2.20.0 を追加します。

```
<dependency>
```

```
  <groupId>software.amazon.awssdk</groupId>
  <artifactId>aws-crt-client</artifactId>
  <version>2.20.0</version>
</dependency>
```

2. AWS SDK for Java 2.x をプロジェクトに追加し、初期化します。HTTP クライアントでハイブリッドポスト量子暗号スイートを有効にします。

```
SdkAsyncHttpClient awsCrtHttpClient = AwsCrtAsyncHttpClient.builder()
            .postQuantumTlsEnabled(true)
            .build();
```

3. Secrets Manager 非同期クライアントを作成します。

```
SecretsManagerAsyncClient SecretsManagerAsync = SecretsManagerAsyncClient.builder()
            .httpClient(awsCrtHttpClient)
            .build();
```

これで Secrets Manager API オペレーションを呼び出すと、コールはハイブリッドポスト量子 TLS を使用して Secrets Manager エンドポイントに送信されます。

ハイブリッドポスト量子 TLS の使用の詳細については、次を参照してください。

- AWS SDK for Java 2.xデベロッパーガイドとAWS SDK for Java 2.xリリースされたブログ記事。
- 「Introducing s2n-tls, a New Open Source TLS Implementation」(新しいオープンソース TLS 実装の のご紹介) および「Using s2n-tls」(の使用)。
- 米国国立標準技術研究所 (NIST) のポスト量子暗号。
- Hybrid Post-Quantum Key Encapsulation Methods (PQ KEM) for Transport Layer Security 1.2 (TLS) (Transport Layer Security 1.2 (TLS) 用のハイブリッド Post-Quantum Key Encapsulation Methods (PQ KEM))。

Secrets Manager のポスト量子 TLS は、中国AWS リージョンを除くすべての で使用できます。

トラブルシューティング AWS Secrets Manager

こちらの情報は、Secrets Manager を操作するときに発生する可能性がある問題の、診断や修復に役立ちます。

ローテーションに関連する問題については、「the section called “ローテーションのトラブルシューティング”」を参照してください。

トピック

- 「アクセスが拒否されました」のメッセージ
- 一時的なセキュリティ認証情報に「アクセスが拒否されました」と表示される
- 変更がすぐに表示されない。
- 「ピアによる接続のリセット」エラー
- シークレットの作成時に「非対称 KMS キーを持つデータキーを生成できません」
- AWS CLI または AWS SDK オペレーションが部分的な ARN からシークレットを見つけられない
- このシークレットは AWS サービスによって管理されるため、そのサービスを使用して更新する必要があります。
- Transform: AWS::SecretsManager-2024-09-16 の使用時に Python モジュールのインポートが失敗する
- マネージド外部シークレットがローテーションしていない

「アクセスが拒否されました」のメッセージ

Secrets Manager に対して GetSecretValue や CreateSecret などの API コールを行う場合、その呼び出しを行う IAM アクセス許可が必要です。また、シークレットが配置されている AWS アカウントと AWS リージョンでこれらの API コールを実行していることも確認する必要があります。コンソールを使用する場合、コンソールはユーザーに代わって同じ API コールを行うため、IAM アクセス許可も必要です。管理者は、IAM ユーザーまたは自分がメンバーであるグループに IAM ポリシーをアタッチすることで、権限を付与できます。ポリシーが時間帯または IP アドレス制限などの条件を含む権限を付与する記述をしている場合は、リクエストを送信する際にそれらの条件を満たす必要もあります。IAM ユーザー、グループ、またはロールのポリシーの表示や修正の詳細については、「IAM ユーザーガイド」の「Working with Policies」を参照してください。Secrets Manager に必要な許可の詳細ついては、「the section called “認証とアクセスコントロール”」を参照してください。

手動で API リクエストに署名する (AWS SDK を使用しない) 場合は、正しくリクエストに署名していることを確認します。

一時的なセキュリティ認証情報に「アクセスが拒否されました」と表示される

リクエストの作成に使用している IAM ユーザーまたはロールに適切なアクセス許可があることを確認します。一時的なセキュリティ認証情報のアクセス許可は IAM ユーザーまたはロールから取得されます。つまり、アクセス許可は IAM ユーザーまたはロールに付与されたものに限定されます。一時的なセキュリティ認証情報のアクセス権限がどのように決定されるかについては、「IAM ユーザーガイド」の「Controlling Permissions for Temporary Security Credentials」を参照してください。

リクエストが正しく署名されており、そのリクエストの形式が整っていることを確認します。詳細については、選択した SDK のツールキットドキュメント、または IAM ユーザーガイドの「一時的なセキュリティ認証情報を使用して AWS リソースへのアクセスをリクエストする」を参照してください。

一時的な認証情報が失効していないことを確認します。詳細については、「IAM ユーザーガイド」の「Requesting Temporary Security Credentials」を参照してください。

Secrets Manager に必要な許可の詳細ついては、「the section called “認証とアクセスコントロール”」を参照してください。

変更がすぐに表示されない。

Secrets Manager では、結果整合性と呼ばれる分散コンピューティングモデルが使用されています。Secrets Manager (または他の AWS サービス) で行った変更は、可能なすべてのエンドポイントから表示されるまでに時間がかかります。この遅延は、サーバー間、レプリケーションゾーン間、世界中のリージョン間でのデータ送信にかかる時間から発生している場合もあります。Secrets Manager ではパフォーマンス向上のためにキャッシュも使用しているため、これが原因で遅延が発生することがあります。変更は、以前にキャッシュされたデータがタイムアウトになるまで反映されない場合があります。

発生する可能性のあるこれらの遅延を考慮して、グローバルなアプリケーションを設計します。また、ある場所で行われた変更が他の場所で直ちに表示されない場合でも、期待どおりに機能するように確かめて下さい。

他のサービス AWS が結果整合性の影響を受ける方法の詳細については、以下を参照してください。

- 「Amazon Redshift Database デベロッパーガイド」の「Managing data consistency」
- 「Amazon Simple Storage Service ユーザーガイド」の「Amazon S3 Data Consistency Model」
- 「Ensuring Consistency When Using Amazon S3 and Amazon EMR for ETL Workflows」(AWS ビッグデータブログ)
- 「Amazon EC2 API リファレンス」の「Amazon EC2 Eventual Consistency」

「ピアによる接続のリセット」エラー

Secrets Manager を呼び出すと、ピアエラーによって接続がリセットされることがあります。これは、クライアントが使用している、または使用しようとしている TCP 接続をサーバーが閉じたときに発生します。キャパシティを管理し、可用性を維持するために、Secrets Manager を含む AWS サービスはいつでも接続を閉じることができます。

これは、アプリケーションが再利用するアイドル状態のキープアライブ接続に最も頻繁に影響しますが、新しく確立された接続にも影響する可能性があります。リセットは、接続ライフサイクル管理の通常の部分です。シークレットまたはアカウントに問題があることを示すものではありません。

AWS SDKs は接続プールを管理し、デフォルトで新しい接続でこれらのエラーを再試行するため、通常はアクションを実行する必要はありません。これらのエラーが発生した場合は、次の操作を行います。

- 最新バージョンの AWS SDK または を使用していることを確認します AWS CLI。デフォルトの再試行動作をオフにしていないことを確認してください。詳細については、「AWS SDK とツールのリファレンスガイド」の「再試行動作」を参照してください。
- AWS SDK を使用する代わりに HTTP 接続を直接管理する場合は、新しい接続でリクエストを再試行します。などの読み取りオペレーション`GetSecretValue`はべき等で、再試行しても安全です。
- 毎回の使用時に Secrets Manager を呼び出す代わりに、クライアントでシークレット値をキャッシュします。キャッシュにより、アプリケーションが開く接続の数が減少し、一時的なネットワークエラーに対する耐障害性が向上します。詳細については、「キャッシュを使用してシークレットを取得する」を参照してください。

これらのエラーが長期間持続する場合、原因は通常の接続サイクル以外である可能性があります。原因を特定するには、 AWS リージョンの Secrets Manager の AWS Health Dashboard イベントを確

認します。また、VPC エンドポイント、プロキシ、ファイアウォール、DNS 設定など、サービスエンドポイントへのネットワークパスも確認します。問題が解決しない場合は、 AWS サポートにお問い合わせください。

シークレットの作成時に「非対称 KMS キーを持つデータキーを生成できません」

Secrets Manager は、シークレットに関連付けられている対称 KMS キーを使用して、各シークレット値のデータキーを生成します。非対称 KMS キーを使用することはできません。非対称 KMS キーではなく対称暗号化キーが使用されていることを確認します。手順については、「非対称 KMS キーの識別」を参照してください。

AWS CLI または AWS SDK オペレーションが部分的な ARN からシークレットを見つけられない

多くの場合、Secrets Manager は完全な ARN ではなく ARN の一部からシークレットを見つけることができます。しかし、シークレットの名前がハイフンで終わり、その後に 6 つの文字が続く場合、Secrets Manager は ARN の一部からシークレットを見つけられないことがあります。代わりに、完全な ARN またはシークレットの名前を使用することを推奨します。

詳細

Secrets Manager は、シークレット ARN が確実に一意であるようにするのに役立つよう、シークレット名の末尾に 6 つのランダムな文字を含めます。元のシークレットが削除され、同じ名前で新しいシークレットが作成された場合、これらの文字により 2 つのシークレット ARN は異なったものとなります。ARN が異なるため、古いシークレットにアクセスできるユーザーであっても、新しいシークレットへのアクセスを自動的に取得するわけではありません。

Secrets Manager は、リージョン、アカウント、シークレット名、ハイフン、さらに 6 文字を含むシークレットの ARN を次のように構築します。

```
arn:aws:secretsmanager:us-east-2:111122223333:secret:SecretName-abcdef
```

シークレット名がハイフンと 6 文字で終わる場合、ARN の一部のみを使用すると、完全な ARN を指定しているかのようにSecrets Manager に表示されます。たとえば、ARN で `MySecret-abcdef` という名前のシークレットがあるとします。

arn:aws:secretsmanager:us-east-2:111122223333:secret:MySecret-abcdef-nutBrk

シークレット ARN の一部のみを使用する次のオペレーションを呼び出すと、Secrets Manager がシークレットを見つけられないことがあります。

```
$ aws secretsmanager describe-secret --secret-id arn:aws:secretsmanager:us-
east-2:111122223333:secret:MySecret-abcdef
```

このシークレットは AWS サービスによって管理されるため、そのサービスを使用して更新する必要があります。

シークレットを変更しようとしているときにこのメッセージが表示された場合、シークレットはメッセージに記載されている管理サービスを使用してのみ更新できます。詳細については、「他のサービスによって管理されるシークレット」を参照してください。

シークレットを管理しているユーザーを特定するには、シークレット名を確認します。他のサービスによって管理されるシークレットには、そのサービスの ID がプレフィックスとして付けられます。または、 で describe-secret を AWS CLI呼び出し、フィールド を確認しますOwningService。

Transform: AWS::SecretsManager-2024-09-16 の使用時に Python モジュールのインポートが失敗する

Transform: AWS::SecretsManager-2024-09-16 を使用していて、ローテーション Lambda 関数の実行時に Python モジュールのインポートに失敗した場合、問題は互換性のない Runtime 値が原因である可能性があります。この変換バージョンでは、 AWS CloudFormation はランタイムバージョン、コード、共有オブジェクトファイルを自動的に管理します。これらをユーザーが管理する必要はありません。

マネージド外部シークレットがローテーションしていない

サードパーティーアプリケーションによって管理されるシークレット が想定どおりに回転しない場合、Secrets Manager は のRotationFailedイベントに詳細を記録します AWS CloudTrail。このイベントには、ローテーションが失敗した理由に関する特定の情報が含まれます。シークレットの最新のRotationFailedイベントを確認して、原因を特定します。CloudTrail での Secrets Manager イベントの表示の詳細については、「」を参照してくださいthe section called “で口グ記録する AWS CloudTrail ”。

マネージド外部シークレットのローテーション失敗の一般的な原因は次のとおりです。

- 認証情報が無効です。シークレットに保存されている認証情報、または使用する場合は管理者シークレットに保存されている認証情報は有効でないか、パートナーアプリケーションでシークレットを更新するアクセス許可がありません。シークレット値に、統合パートナーが必要とする形式の正しいフィールドが含まれていること、およびパートナーアプリケーションで認証情報がまだアクティブであることを確認します。各パートナーの必須フィールドについては、「」を参照してください統合パートナー。
- 外部リソースへのブロックされたネットワークアクセス。パートナーアプリケーションは、Secrets Manager が管理する外部シークレットローテーショントラフィックをブロックします。外部リソースが IP アドレスによるインバウンドアクセスを制限する場合は、ローテーション呼び出しの送信元の IP 範囲を許可リストに登録する必要があります。 AWSマネージドプレフィックスリスト を追加します。ここでcom.amazonaws.*region*.secretsmanager-managed-external-secrets、*region* はシークレットが存在する AWS リージョンです。マネージドプレフィックスリストを使用すると、基になる IP 範囲が変更されても進入ルールが自動的に最新の状態になります。詳細については、「Amazon VPC ユーザーガイド」の「AWSマネージドプレフィックスリスト」を参照してください。

  プログラムでプレフィックスリストを参照する場合は、GetManagedPrefixListEntries API を使用して、許可リストに登録する必要がある IPsを一覧表示できます。このプレフィックスリストを使用して進入ルールを定期的に更新して、最新の状態に保つ必要があります。

マネージド外部シークレットのローテーションの詳細については、「」を参照してくださいSecrets Manager が管理する外部シークレットのローテーション。マネージドプレフィックスリストを含むセキュリティとアクセス許可の詳細については、「」を参照してくださいセキュリティおよびアクセス許可。

AWS Secrets Manager クォータ

Secrets Manager の読み取り API は TPS クォータが高く、あまり呼び出されないコントロールプレーン API は TPS クォータが低くなります。10 分に 1 回以上の持続頻度で `PutSecretValue` または `UpdateSecret` を呼び出すことは避けることが推奨されます。`PutSecretValue` または `UpdateSecret` を呼び出してシークレット値を更新すると、Secrets Manager はシークレットの新しいバージョンを作成します。Secrets Manager は、ラベルのないバージョンが 100 を超えると削除しますが、24 時間以内に作成されたバージョンは削除しません。10 分に 1 回以上の頻度でシークレット値を更新すると、Secrets Manager が削除した数よりも多くバージョンが作成され、シークレットバージョンのクォータに達します。

お使いのアカウントで複数のリージョンを運用できます。各クォータは各リージョンに固有です。

あるアプリケーションが別のアカウントが所有するシークレット AWS アカウント を使用する場合、クロスアカウントリクエストと呼ばれます。クロスアカウントリクエストでは、Secrets Manager は、シークレットを所有するアカウントではなく、リクエストを行うアイデンティティのアカウントをスロットリングします。例えば、アカウント A のアイデンティティがアカウント B のシークレット を使用する場合、このシークレットの使用は、アカウント A のクォータにのみ適用されます。

Secrets Manager のクォータ

名前	デフォルト	引き上げ可能	説明
DeleteResourcePolicy、GetResourcePolicy、PutResourcePolicy および ValidateResourcePolicy API リクエストの合計レート	サポートされている各リージョン: 50/秒	いいえ	DeleteResourcePolicy、GetResourcePolicy、PutResourcePolicy および ValidateResourcePolicy API リクエストの合計の 1 秒あたり

名前	デフォルト	引き上げ可能	説明
			の最大トランザクション数。
PutSecretValue、RemoveRegionsFromReplication、ReplicateSecretToRegion、StopReplicationToReplica、UpdateSecret および UpdateSecretVersionStage API リクエストの合計レート	サポートされている各リージョン: 50/秒	いいえ	PutSecretValue、RemoveRegionsFromReplication、ReplicateSecretToRegion、StopReplicationToReplica、UpdateSecret および UpdateSecretVersionStage API リクエストの合計の 1 秒あたりの最大トランザクション数。
RestoreSecret API リクエストの合計レート	サポートされている各リージョン: 50/秒	いいえ	RestoreSecret API リクエストの 1 秒あたりの最大トランザクション数。
RotateSecret および CancelRotateSecret API リクエストの合計レート	サポートされている各リージョン: 50/秒	いいえ	RotateSecret および CancelRotateSecret API リクエストの合計の 1 秒あたりの最大トランザクション数。
TagResource および UntagResource API リクエストの合計レート	サポートされている各リージョン: 50/秒	いいえ	TagResource および UntagResource API リクエストの合計の 1 秒あたりの最大トランザクション数。

名前	デフォルト	引き上げ可能	説明
BatchGetSecretValue API リクエストのレート	サポートされている各リージョン: 100/秒	いいえ	BatchGetSecretValue API リクエストの 1 秒あたりの最大トランザクション数。
CreateSecret API リクエストのレート	サポートされている各リージョン: 50/秒	いいえ	CreateSecret API リクエストの 1 秒あたりの最大トランザクション数。
DeleteSecret API リクエストのレート	サポートされている各リージョン: 50/秒	いいえ	DeleteSecret API リクエストの 1 秒あたりの最大トランザクション数。
DescribeSecret API リクエストのレート	サポートされている各リージョン: 40,000/秒	いいえ	DescribeSecret API リクエストの 1 秒あたりの最大トランザクション数。
GetRandomPassword API リクエストのレート	サポートされている各リージョン: 50/秒	いいえ	GetRandomPassword API リクエストの 1 秒あたりの最大トランザクション数。
GetSecretValue API リクエストのレート	サポートされている各リージョン: 10,000/秒	いいえ	GetSecretValue API リクエストの 1 秒あたりの最大トランザクション数。
ListSecretVersionIds API リクエストのリスト	サポートされている各リージョン : 50/秒	いいえ	ListSecretVersionIds API リクエストの 1 秒あたりの最大トランザクション数

名前	デフォルト	引き上げ可能	説明
ListSecrets API リクエストのレート	サポートされている各リージョン: 100/秒	いいえ	ListSecrets API リクエストの 1 秒あたりの最大トランザクション数
リソースベースのポリシーの長さ	サポートされている各リージョン: 20,480	いいえ	シークレットにアタッチされているリソースベースのアクセス権限ポリシーの最大文字数。
シークレット値のサイズ	サポートされている各リージョン: 65,536 バイト	いいえ	暗号化されたシークレット値の最大サイズ。シークレット値が文字列の場合、これはシークレット値で許可される文字数です。
Secrets	サポートされている各リージョン: 500,000	いいえ	この AWS アカウントの各 AWS リージョンのシークレットの最大数。
シークレットのすべてのバージョンにアタッチされたステージングラベル	サポートされている各リージョン: 20	いいえ	シークレットのすべてのバージョンにアタッチされたステージングラベルの最大数。
シークレットあたりのバージョン	サポートされている各リージョン: 100	いいえ	1 つのシークレットのバージョンの最大数。

アプリケーションへの再試行を追加する

予期しない問題により AWS 、Secrets Manager への呼び出しが失敗することがあります。あるいは、Secrets Manager によるレート制限が原因で、呼び出しが失敗する場合もあります。API リクエストクォータを超えると、Secrets Manager はリクエストをスロットルします。それ以外の場合は有効なリクエストを拒否し、throttling というエラーを出力します。どちらの種類の失敗に関しても、短い待機時間後に、呼び出しを再試行することをお勧めします。これは、バックオフと再試行の戦略と呼ばれます。

以下のようなエラーが発生した場合は、アプリケーションコードに再試行の処理を追加します。

一時的なエラーおよび例外

- `RequestTimeout`
- `RequestTimeoutException`
- `PriorRequestNotComplete`
- `ConnectionError`
- `HTTPClientError`

サービス側のスロットリングと制限のエラーおよび例外

- `Throttling`
- `ThrottlingException`
- `ThrottledException`
- `RequestThrottledException`
- `TooManyRequestsException`
- `ProvisionedThroughputExceededException`
- `TransactionInProgressException`
- `RequestLimitExceeded`
- `BandwidthLimitExceeded`
- `LimitExceededException`
- `RequestThrottled`
- `SlowDown`

再試行、エクスポネンシャルバックオフ、ジッターに関する詳細およびコード例については、次のリソースを参照してください。

- エクスポネンシャルバックオフとジッター
- ジッターを伴うタイムアウト、再試行、およびバックオフ
- エラーの再試行とエクスポネンシャルバックオフ。 AWS

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 Secrets Manager。このドキュメントの更新に関する通知を受け取るには、RSS フィードにサブスクライブできます。

変更	説明	日付
AWS 管理ポリシーの更新	`AWSSecretsManagerClientReadOnlyAccess` 管理ポリシーに `BatchGetSecretValue` および アクセス`ListSecrets` 許可が含まれるようになりました。詳細については、「Secrets Manager updates to AWS managed policies」を参照してください。	2026 年 6 月 2 日
新しい AWS 管理ポリシー	Secrets Manager は、クライアントアプリケーションのシークレットへの読み取り専用アクセス`AWSSecretsManagerClientReadOnlyAccess` を提供する新しい マネージドポリシーをリリースしました。詳細については、「Secrets Manager updates to AWS managed policies」を参照してください。	2025 年 11 月 5 日
コスト割り当てタグのサポートの追加	Secrets Manager でコスト配分タグがサポートされるようになり、お客様は部門、チーム、またはアプリケーションごとにコストを分類して追	2025 年 5 月 27 日

	跡できるようになりました。詳細については、「 でのコスト配分タグ AWS Secrets Managerの使用」を参照してください。	
IPv6 とデュアルスタックのサポートを追加	Secrets Manager がデュアルスタックのエンドポイントをサポートするようになりました。詳細については、「IPv4 および IPv6 アクセス」を参照してください。	2024 年 12 月 20 日
Secrets Manager の AWS マネージドポリシーへの変更	`SecretsManagerReadWrite` マネージドポリシーには、`redshift-serverless` アクセス許可が含まれています。詳細については、「 の AWS 管理ポリシー」を参照してください。AWS Secrets Manager	2024 年 3 月 12 日

以前の更新

次の表に、2024 年 2 月以前の AWS Secrets Manager ユーザーガイドの各リリースにおける重要な変更点を示します。

変更	説明	日付
一般提供	これは、Secrets Manager の最初の一般リリースです。	2018 年 4 月 4 日

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