



ユーザーガイド

Amazon CloudWatch Logs



Amazon CloudWatch Logs: ユーザーガイド

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

Amazon の商標とトレードドレスは、Amazon 以外の製品またはサービスとの関連において、顧客に混乱を招いたり、Amazon の名誉または信用を毀損するような方法で使用することはできません。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

Amazon CloudWatch Logs とは	1
機能	1
関連 AWS サービス	3
料金	4
概念	4
請求とコスト	6
ログクラス	7
サポートされている機能	7
開始方法	10
前提条件	10
にサインアップする AWS アカウント	10
管理アクセスを持つユーザーを作成する	11
Command Line Interface をセットアップする	12
統合した CloudWatch エージェントの使用	13
前の CloudWatch エージェントの使用	13
CloudWatch Logs エージェントの前提条件	14
クイックスタート: 実行中の EC2 Linux インスタンスにエージェントをインストールする ...	15
クイックスタート: EC2 Linux インスタンスの起動時にエージェントをインストールする	22
クイックスタート: Windows サーバー 2016 インスタンスで CloudWatch Logs を使用する	26
クイックスタート: Windows Server 2012 および Windows Server 2008 インスタンスでの CloudWatch Logs の使用	37
CloudWatch Logs エージェントの状態を報告する	48
CloudWatch Logs エージェントを起動する	48
CloudWatch Logs エージェントを停止する	49
CloudWatch Logs エージェントのリファレンス	49
でのクイックスタート CloudFormation	61
HTTP エンドポイントを介したログ取り込み	63
一般的な動作	63
ベアラートークン認証	64
オプション 1: AWS コンソールを使用したクイックスタート	65
オプション 2: 手動セットアップ	67
CloudWatch Logs API キーを生成して使用するためのアクセス許可を制御する	70
API キーの更新	72

侵害された API キーへの対応	73
API キーのセキュリティのベストプラクティス	74
CloudTrail を使用した API キーの使用状況のログ記録	75
OTLP エンドポイント	76
リクエストの形式	76
リクエスト例	76
レスポンス	77
OTLP 固有の動作	78
HLC エンドポイント	78
入力モード	78
イベントフィールド (必須)	79
時間フィールド (オプション)	79
Content-Type	80
許容される JSON 値タイプ	80
エンドポイントフォーマット	80
リクエストの形式	81
リクエスト例	82
ベストプラクティス	82
制限事項	83
NDJSON エンドポイント	83
リクエストの形式	84
許容される JSON 値タイプ	84
タイムスタンプフィールド	85
無効な行	86
リクエスト例	86
レスポンス	86
ベストプラクティス	87
制限事項	88
構造化 JSON エンドポイント	88
リクエストの形式	88
許容される JSON 値タイプ	89
タイムスタンプフィールド	89
リクエスト例	90
レスポンス	90
ベストプラクティス	91
制限事項	91

HTTP 取り込みエンドポイントの比較	92
エンドポイントの選択	93
AWS SDKs の使用	94
ログ管理	96
データソースの検出と管理	96
CloudWatch Logs データソースとは	97
開始方法	97
システムフィールド	99
データソースへのアクセス	99
ロググループとの関係	100
データソースによって有効になる機能	100
サポートされているデータソース	101
データソース AWS のサービス でサポート	101
データソースでサポートされているサードパーティーソース	106
カスタムソース	114
CloudWatch Logs Insights を使用したログデータの分析	115
サポートされているクエリ言語	118
CloudWatch Logs Insights クエリ言語 (Logs Insights QL)	118
OpenSearch パイプ処理言語 (PPL)	188
OpenSearch 構造化クエリ言語 (SQL)	200
自然言語を使用した CloudWatch Logs Insights クエリの生成と更新	212
クエリの例	213
サービス改善のためのデータ使用をオプトアウトする	215
サポートされるログと検出されるフィールド	216
JSON ログのフィールド	218
フィールドインデックスを作成してクエリパフォーマンスを改善し、スキャン量を削減する ..	220
フィールドインデックスの構文とクォータ	224
アカウントレベルのフィールドインデックスポリシーを作成する	229
ロググループレベルのフィールドインデックスポリシーを作成する	230
クエリ作成時のロググループの選択オプション	231
フィールドインデックスポリシーを削除した場合の影響	232
ファセットを使用してログをグループ化および探索する	232
ファセットベースのクエリを実行するには	233
ファセットベースのクエリを保存するには	234
アカウントレベルのファセットを作成するには	234
APIsを使用したファセット管理	234

CloudWatch Logs Insights での周囲のログの表示	235
周囲のログの仕組み	235
周囲のログを表示する	236
周囲のログ内で検索する	237
パターン分析	238
パターン分析の開始方法	239
パターンコマンドの詳細	241
クエリの保存と再実行	242
パラメータを使用した保存済みクエリの使用	248
クエリをダッシュボードに追加する、またはクエリ結果をエクスポートする	250
実行中のクエリまたはクエリ履歴を表示する	251
でクエリ結果を暗号化する AWS Key Management Service	252
制限	252
ステップ 1: を作成する AWS KMS key	253
ステップ 2: KMS キーでアクセス許可を設定する	253
ステップ 3: KMS キーをクエリ結果に関連付ける	255
ステップ 4: アカウントのクエリ結果からキーの関連付けを解除する	255
CloudWatch Logs Insights クエリ結果から自然言語の概要を生成する	256
仕組み	256
リージョンの可用性とデータ処理	256
開始方法	257
アクセス許可	257
データプライバシー	257
スケジュールされたクエリによるログ分析の自動化	259
スケジュールされたクエリ of の概念を理解する	259
IAM ロールの分離	259
クロスリージョンおよびクロスアカウントの使用	262
スケジュール式とタイムゾーン処理	263
クエリ言語の選択	264
送信先の選択とユースケース	264
クエリ結果の形式と構造	265
スケジュール式リファレンス	267
ベストプラクティス	270
スケジュールされたクエリの開始方法	271
スケジュールされたクエリの作成	272
スケジュールされたクエリの表示と管理	275

スケジュールされたクエリ実行履歴の表示	277
スケジュールされたクエリの更新	279
スケジュールされたクエリの S3 送信先の設定	281
同じアカウントの Amazon S3 バケットに結果を配信する	281
別のアカウントの Amazon S3 バケットに結果を配信する	282
カスターマネージド AWS KMS キーによる結果の暗号化	283
スケジュールされたクエリのトラブルシューティング	285
クエリの実行がアクセス許可エラーで失敗する	285
クエリのタイムアウト	286
送信先処理が失敗する	287
無効なクエリエラー	287
クエリ同時実行エラー	288
ログ異常検出	289
異常とパターンの重要度と優先度	290
異常可視性期間	290
異常の抑制	291
よくある質問	291
CloudWatch Logs Insights での異常検出の使用	292
ロググループの異常検出を有効にする	292
検出された異常を表示する	294
ログ異常ディテクターにアラームを作成する	296
ログ異常ディテクターによって発行されるメトリクス	299
で異常ディテクターとその結果を暗号化する AWS KMS	299
制限	300
CloudWatch Logs Live Tail を使用したトラブルシューティング	305
を使用して Live Tail セッションを開始する AWS CLI	306
print-only	306
interactive	306
コンソールで Live Tail セッションを開始する	308
クロスアカウントクロスリージョンログの一元化	312
データ一元化の概念	312
ログの一元化の設定	314
前提条件	314
送信先ロググループ名をカスタマイズする	314
一元化ルールの作成	316
一元化ルールの変更	319

一元化ルールの表示	319
一元化ルールの削除	319
一元化ルールのモニタリングとトラブルシューティング	320
一元化ルールのヘルスステータス	320
による一元化 API コールのモニタリング AWS CloudTrail	321
モニタリングの推奨事項	321
ロググループとログストリームの操作	324
ロググループの作成	324
ロググループへのログの送信	325
ログデータを表示する	325
フィルターパターンを使用してログデータを検索する	326
コンソールを使用してログエントリを検索する	326
を使用してログエントリを検索する AWS CLI	327
メトリクスからログへのピボット	327
トラブルシューティング	328
ログデータの保持期間の変更	328
ロググループを削除から保護する	329
ロググループを削除から保護する	330
削除保護の有効化	330
ロググループのタグ付け	331
タグの基本	331
タグ付けを使用したコストの追跡	332
タグの制限	332
を使用したロググループのタグ付け AWS CLI	333
CloudWatch Logs API を使用したロググループのタグ付け	333
を使用してログデータを暗号化する AWS KMS	334
制限	335
ステップ 1: AWS KMS キーを作成する	253
ステップ 2: KMS キーでアクセス許可を設定する	336
ステップ 3: 呼び出し元の IAM プリンシパルにアクセス許可を設定する	339
ステップ 4: KMS キーをロググループに関連付ける	304
ステップ 5: ロググループからキーの関連付けを解除する	304
KMS キーと暗号化コンテキスト	342
機密性の高いログデータをマスキングで保護する	345
データ保護ポリシーを理解する	349
データ保護ポリシーの作成または操作に必要な IAM 権限	351

アカウント全体のデータ保護ポリシーを作成する	356
1 つのロググループ用のデータ保護ポリシーを作成する	359
データをマスクせずに表示する	363
監査結果レポート	363
保護できるデータの種類	365
取り込み中のログの変換	408
ログトランスフォーマーの作成と管理	410
アカウントレベルのトランスフォーマーポリシーを作成する	411
アカウントレベルのトランスフォーマーポリシーの編集または削除	412
log-group-level ログトランスフォーマーを最初から作成する	413
既存のトランスフォーマーをコピーして log-group-level トランスフォーマーを作成する ...	414
log-group-level トランスフォーマーを編集する	415
log-group-level トランスフォーマーを削除する	416
設定可能なパーサータイプのプロセッサ	416
parseJSON	417
grok	418
csv	448
parseKeyValue	451
AWS 販売ログ用の組み込みプロセッサ	452
parseWAF	453
parsePostgres	455
parseCloudfront	456
parseRoute53	458
parseVPC	459
parseToOCSF	460
文字列ミューテーションプロセッサ	462
lowerCaseString	462
upperCaseString	463
splitString	464
substituteString	467
trimString	470
JSON ミューテーションプロセッサ	471
addKeys	472
DeleteKeys	473
moveKeys	474
renameKeys	476

copyValue	478
listToMap	480
データ型コンバータプロセッサ	485
typeConverter	485
datetimeConverter	486
変換メトリクスとエラー	489
Amazon OpenSearch Service で分析する	490
ステップ 1: OpenSearch Service との統合を作成する	492
必要なアクセス許可	492
統合を作成する	500
ステップ 2: 提供されるログダッシュボードを作成する	501
提供されるログダッシュボードの表示、編集、削除	502
CloudWatch Logs または OpenSearch Service で提供されるログダッシュボードを表示する	502
追加の IAM ロールまたは IAM ユーザーにダッシュボード表示アクセス権を付与する	503
ダッシュボード設定を編集する	503
提供されたログダッシュボードを削除する	503
OpenSearch Service とのすべての提供されたログダッシュボード統合を削除する	504
ユーザー向けの IAM ポリシー	505
統合に必要なアクセス許可	507
S3 Tables 統合によるログへのアクセス	509
S3 Tables の統合について	509
コアコンポーネント	509
S3 テーブルへのデータフロー	509
S3 テーブル統合を使用するタイミング	510
前提条件	510
IAM アクセス許可	511
KMS キーポリシー (暗号化されたデータ用)	512
開始方法	513
Amazon S3 Tables と AWS 分析サービスの統合 - Amazon S3 コンソールの使用 (リンク)	515
Lake Formation アクセス許可を設定する	515
分析ツールで接続する	516
メトリクスフィルター	517
概念	518
メトリクスフィルターのフィルターパターン構文	519
メトリクスフィルターのメトリクス値を設定する	521

ログイベントからのメトリクスに寸法を発行する	521
ログイベントの値を使用してメトリクスの値を増分する	525
メトリクスフィルターの作成	526
ロググループのメトリクスフィルターの作成	526
例: ログイベントのカウント	527
例: 語句の出現回数をカウントする	529
例: HTTP 404 コードをカウントする	531
例: HTTP 4xx コードをカウントする	533
例: Apache ログからフィールドを抽出してディメンションを割り当てる	534
メトリクスフィルターの一覧表示	536
メトリクスフィルターの削除	537
サブスクリプションフィルター	539
概念	541
ロググループレベルのサブスクリプションフィルター	542
例 1: Amazon Kinesis Data Streams を使用したサブスクリプションフィルター	543
例 2: を使用したサブスクリプションフィルター AWS Lambda	549
例 3: Amazon Data Firehose を使用したサブスクリプションフィルター	553
例 4: Amazon OpenSearch Service を使用したサブスクリプションフィルター	560
アカウントレベルのサブスクリプションフィルター	561
例 1: Amazon Kinesis Data Streams を使用したサブスクリプションフィルター	562
例 2: を使用したサブスクリプションフィルター AWS Lambda	568
例 3: Amazon Data Firehose を使用したサブスクリプションフィルター	573
クロスアカウント、クロスリージョンのサブスクリプション	580
Amazon Kinesis Data Streams を使用したクロスアカウントクロスリージョンログデータ共有	581
Firehose を使用したクロスアカウント、クロスリージョンのログデータ共有	602
Amazon Kinesis Data Streams を使用したクロスアカウントクロスリージョンアカウントレベルのサブスクリプション	616
Firehose を使用したクロスアカウント、クロスリージョン、アカウントレベルのサブスクリプション	635
混乱した代理の防止	647
ログの再帰防止	649
フィルターパターン構文	651
サポートされている正規表現	652
正規表現を使用した語句の一致	655
非構造化ログイベントの語句の一致	655

JSON ログイベントでの語句の一致	659
スペース区切りのログイベントで語句の一致	667
AWS サービスからのログ記録を有効にする	672
追加のアクセス許可が必要なロギング [V1]	674
CloudWatch Logs に送信されたログ	675
Amazon S3 に送信されたログ	678
Firehose に送信されるログ	683
追加のアクセス許可が必要なロギング [V2]	685
CloudWatch Logs に送信されたログ	686
Amazon S3 に送信されたログ	690
Firehose に送信されるログ	694
X-Ray に送信されたトレース	697
サービス固有のアクセス許可	701
コンソール固有のアクセス許可	701
アカウント間の配信の例	703
配信ソースを作成する	704
Amazon S3 バケットへの配信を設定する	704
Firehose ストリームへの配信を設定する	707
サービス間での不分別な代理処理の防止	710
テレメトリ有効化ルールを使用してログ有効化を自動化する	711
ルール評価階層	711
有効化ルールの作成	712
サードパーティーソースからのログ記録を有効にする	713
直接のサードパーティー統合	713
AWS Security Hub CSPM 経由の追加ソース	713
Amazon S3 へのログデータのエクスポート	719
概念	720
コンソールを使用してログデータを Amazon S3 にエクスポートする	721
同一アカウントのエクスポート (コンソール)	722
クロスアカウントエクスポート (コンソール)	729
を使用してログデータを Amazon S3 にエクスポートする AWS CLI	739
同一アカウントのエクスポート (CLI)	740
クロスアカウントエクスポート (CLI)	747
エクスポートタスクを記述する (CLI)	757
エクスポートタスクをキャンセルする (CLI)	758
OpenSearch Service へのデータストリーミング	759

前提条件	759
ロググループを OpenSearch Service にサブスクライブする	760
コードの例	762
基本	763
アクション	764
シナリオ	831
Amazon ECS Service Connect を設定する	832
最初の Lambda 関数を作成する	848
大規模なクエリを実行する	860
スケジュールされたイベントを使用した Lambda 関数の呼び出し	898
を使用してルックアップテーブルを暗号化する AWS KMS	901
CloudWatch Logs がルックアップテーブル AWS KMS に使用方法	902
必要な アクセス許可	902
ステップ 1: AWS KMS キーを作成する	903
ステップ 2: KMS キーでアクセス許可を設定する	904
ステップ 3: KMS キーをルックアップテーブルに関連付ける	905
考慮事項	906
セキュリティ	907
データ保護	908
保管中の暗号化	909
転送中の暗号化	909
ID とアクセス管理	909
認証	910
アクセスコントロール	910
アクセス管理の概要	910
アイデンティティベースのポリシー (IAM ポリシー) の使用	917
コンプライアンス検証	944
耐障害性	945
インフラストラクチャセキュリティ	945
インターフェイス VPC エンドポイント	945
利用可能な状況	946
CloudWatch Logs 用の VPC エンドポイントの作成	946
VPC と CloudWatch Logs との間の接続のテスト	947
CloudWatch Logs の VPC エンドポイントへのアクセスの制御	947
VPC コンテキストキーのサポート	948
を使用した API およびコンソールオペレーションのログ記録 AWS CloudTrail	949

CloudTrail でのクエリ生成情報	954
ログファイルエントリの理解	955
CloudWatch メトリクスの使用状況のモニタリング	957
CloudWatch Logs のメトリック	957
CloudWatch Logs メトリックのディメンション	961
トランスフォーマーのメトリクスとディメンションをログに記録する	962
一元化メトリクスとディメンション	963
CloudWatch Logs サービスの使用状況メトリクス	967
サービスクォータ	969
CloudWatch Logs サービスクォータの管理	982
ドキュメント履歴	984
AWS 用語集	1000
.....	mi

Amazon CloudWatch Logs とは

Amazon CloudWatch Logs を使用して、Amazon Elastic Compute Cloud (Amazon EC2) インスタンス、Route 53、およびその他のソースからログファイルをモニタリング AWS CloudTrail、保存、アクセスできます。

CloudWatch Logs を使用すると、使用するすべてのシステム、アプリケーション、AWS サービスからのログを 1 つの非常にスケーラブルなサービスに一元化できます。これにより、ログを簡単に表示したり、特定のエラーコードやパターンを検索したり、特定のフィールドに基づいてフィルタリングしたり、将来の分析のために安全にアーカイブしたりできます。CloudWatch Logs では、ソースにかかわらずすべてのログをイベントの 1 つの一貫した流れとして時間順に見ることができます。

CloudWatch Logs は、強力なクエリ言語によるログのクエリ、ログ内の機密データの監査とマスキング、フィルターや埋め込みログ形式を使用したログからのメトリクスの生成もサポートしています。

CloudWatch Logs は 2 つのログクラスをサポートしています。CloudWatch Logs 標準ログクラスのロググループは、すべての CloudWatch Logs 機能をサポートしています。CloudWatch Logs 低頻度アクセスログクラスのロググループでは、取り込み料金が低くなり、標準クラスの機能のサブセットがサポートされます。詳細については、「[ログクラス](#)」を参照してください。

機能

- 柔軟性を高める 2 つのログクラス – CloudWatch Logs には 2 つのログクラスが用意されており、アクセス頻度の低いログに対してコスト効率の高いオプションを利用できます。また、リアルタイムモニタリングやその他の機能を必要とするログには、フル機能オプションもあります。詳細については、「[ログクラス](#)」を参照してください。
- ログデータのクエリ – CloudWatch Logs Insights を使用して、ログデータをインタラクティブに検索および分析できます。クエリを実行することで、運用上の問題に効率的かつ効果的に対応できます。CloudWatch Logs Insights には、いくつかのシンプルで強力なコマンドを備えた専用のクエリ言語が含まれています。提供されているサンプルのクエリ、コマンドの説明、クエリの自動補完、およびログフィールドの検出を利用して簡単に使用を開始できます。サンプルクエリは、いくつかのタイプの AWS サービスログに含まれています。開始するには、「[CloudWatch Logs Insights を使用したログデータの分析](#)」を参照してください。
- フィールドインデックスを作成してクエリをより効率的に作成する – ログイベントでフィールドのフィールドインデックスを作成できます。その後、CloudWatch Logs Insights クエリでフィールドインデックスを使用すると、クエリはインデックス付きフィールドを含まないことがわかって

いるログイベントの処理のスキップを試行します。このクエリによってクエリのスキャン量が減少し、結果をより迅速に返すことができます。開始するには、「[フィールドインデックスを作成してクエリパフォーマンスを改善し、スキャン量を削減する](#)」を参照してください。

- Live Tail を使用した検出とデバッグ — Live Tail を使用すれば、新しいログイベントのストリーミングリストを取り込みに表示して、インシデントのトラブルシューティングをすばやく行うことができます。取り込まれたログをほぼリアルタイムで表示、フィルタリング、強調表示できるため、問題をすばやく検出して解決することができます。指定した用語に基づいてログをフィルタリングしたり、特定の用語を含むログを強調表示したりすることで、探しているものをすぐに見つけることができます。詳細については、「[CloudWatch Logs Live Tail を使用したトラブルシューティング](#)」を参照してください。
- Amazon EC2 インスタンスのログをモニタリングする – CloudWatch Logs を使って、ログデータを使用してアプリケーションとシステムをモニタリングできます。例えば、CloudWatch Logs では、アプリケーションログに存在するエラーの数がトラッキングされ、エラー率が指定のしきい値を超えたときに管理者に通知が送信されます。お客様のログが CloudWatch Logs によるモニタリングに使用されるので、コードの変更は不要です。たとえば、アプリケーションログの特定のリテラルターム (例: 「NullPointerException」) をモニタリングしたり、ログデータの特定の場所でリテラルターム (例: Apache アクセスログの「404」ステータスコード) の発生数をカウントしたりできます。検索した語句が見つかったと、CloudWatch Logs は指定された CloudWatch メトリクスにデータをレポートします。ログデータは、転送時や保管時に暗号化されます。開始するには、「[CloudWatch Logs の開始方法](#)」を参照してください。
- AWS CloudTrail ログに記録されたイベントのモニタリング – CloudWatch でアラームを作成し、CloudTrail によってキャプチャされた特定の API アクティビティの通知を受信し、通知を使用してトラブルシューティングを実行できます。開始するには、AWS CloudTrail ユーザーガイドの「[CloudWatch Logs への CloudTrail Events の送信](#)」を参照してください。
- 機密データの監査とマスキング — ログに機密データがある場合は、データ保護ポリシーを使用して保護できます。これらのポリシーにより、機密性の高いデータを監査してマスクできます。データ保護を有効にすると、デフォルトでは、選択したデータ識別子と一致する機密データがマスクされます。詳細については、「[機密性の高いログデータをマスキングで保護する](#)」を参照してください。
- ログの保持期間 – デフォルトでは、ログは無制限に保持され、失効しません。ロググループごとに保持ポリシーを調整し、無制限の保持期間を維持するか、1 日間～10 年間の保持期間を選択することができます。
- 削除保護 – ロググループとそのログストリームが誤って削除されないようにする保護。ロググループで有効になっている場合、削除保護は明示的に無効になるまですべての削除操作をブロックします。デフォルトでは、削除保護は有効になっていません。このオプション機能は、監査データ

を含むロググループやトラブルシューティングと分析のための本番稼働用アプリケーションログなど、重要な運用データとコンプライアンスデータを意図しない削除から保護するのに役立ちます。

- ログデータをアーカイブする – CloudWatch Logs を使用して高い耐久性のストレージにログデータを保存できます。CloudWatch Logs エージェントにより、ローテーションするログデータもローテーションしないログデータも、ホストからログサービスに簡単にすばやく送信できます。その後は、必要なときに生のログデータにアクセスできます。
- Route 53 DNS クエリのログ – CloudWatch Logs を使用して、Route 53 が受け取る DNS クエリに関するログ情報を使用できます。詳細については、Amazon Route 53 デベロッパーガイドの「[DNS クエリのログ](#)」を参照してください。
- アカウントとリージョン間でログを一元化する – CloudWatch Logs Centralization を使用して、マルチアカウント環境とリージョン環境間で取り込まれたログデータを中央のリージョンとアカウントにレプリケートする、クロスアカウントとクロスリージョンの一元化ルールを定義できます。回復性を高めるために中央アカウント内のバックアップリージョンを設定し、中央アカウントで新規作成されたロググループの暗号化設定を定義し、フィルタリング機能が強化された特定のソースアカウントとリージョンからのログイベントについてメトリクスフィルターとサブスクリプションフィルターを作成できます。

関連 AWS サービス

CloudWatch Logs と併せて使用されるサービスは次のとおりです。

- AWS CloudTrail は、AWS Command Line Interface (AWS CLI) AWS マネジメントコンソール、およびその他の サービスによって行われた呼び出しなど、アカウントの CloudWatch Logs API に対して行われた呼び出しをモニタリングできるウェブサービスです。CloudTrail ログ記録がオンになると、CloudTrail は、API 呼び出しをアカウントにキャプチャし、指定する Amazon S3 バケットにログファイルを送信します。リクエストを満たすためにアクションをいくつか実行する必要があったかに応じて、各ログファイルには 1 個以上のレコードが含まれる可能性があります。詳細については AWS CloudTrail、AWS CloudTrail ユーザーガイドの「[とは AWS CloudTrail](#)」を参照してください。CloudWatch が CloudTrail のログファイルに書き込むデータのタイプの例については、「[での CloudWatch Logs API およびコンソールオペレーションのログ記録 AWS CloudTrail](#)」を参照してください。
- AWS Identity and Access Management (IAM) は、ユーザーの AWS リソースへのアクセスを安全に制御するのに役立つウェブサービスです。IAM を使用して、どのユーザーがお客様の AWS リソースを使用できるか (認証)、それらのユーザーがどのリソースをどのような方法で利用できるか (承認) を制御できます。詳細については、IAM ユーザーガイドの「[IAM とは](#)」を参照してください。

- Amazon Kinesis Data Streams は、高速かつ継続的にデータの取り込みと集約を行うためのウェブサービスです。使用されるデータのタイプには、IT インフラストラクチャのログデータ、アプリケーションのログ、ソーシャルメディア、マーケットデータフィード、ウェブのクリックストリームデータなどがあります。データの取り込みと処理の応答時間はリアルタイムであるため、処理は一般的に軽量です。詳細については、Amazon Kinesis Data Streams デベロッパーガイドの「[Amazon Kinesis Data Streams とは](#)」を参照してください。
- AWS Lambda は、新しい情報にすばやく対応するアプリケーションを簡単に構築するためのウェブサービスです。アプリケーションコードを Lambda 関数としてアップロードします。Lambda は可用性の高いコンピューティングインフラストラクチャでお客様のコードを実行し、コンピューティングリソースの管理をすべて担当します。これにはサーバーおよびオペレーティングシステムの管理、キャパシティーのプロビジョニングおよび自動スケーリング、コードおよびセキュリティパッチのデプロイ、モニタリングおよびロギングなどが含まれます。必要な操作は、Lambda がサポートするいずれかの言語でコードを指定するだけです。詳細については、「AWS Lambda デベロッパーガイド」の「[What is AWS Lambda?](#)」を参照してください。

料金

にサインアップすると AWS、無料[AWS 利用](#)枠を使用して CloudWatch Logs を無料で使い始めることができます。

標準料金は、CloudWatch Logs を使用した他のサービスによって格納されたログ (Amazon VPC フローログおよび Lambda ログなど) に適用されます。

料金の詳細については、「[Amazon CloudWatch の料金](#)」を参照してください。

CloudWatch Logs および CloudWatch のコストと使用状況を分析する方法、およびコストを削減するためのベストプラクティスについては、「[CloudWatch の請求とコスト](#)」を参照してください。

Amazon CloudWatch Logs の概念

CloudWatch Logs を理解し使用するために重要な用語と概念を、以下に示します。

ログクラス

CloudWatch Logs には 2 つのクラスのロググループがあります。標準ログクラスは、リアルタイムモニタリングが必要なログや頻繁にアクセスするログのフル機能オプションです。低頻度アクセスログクラスは、アクセス頻度の低いログの低コストオプションです。標準ログクラスの機能のサブセットをサポートします。

ログイベント

ログイベントは、モニタリングされているアプリケーションまたはリソースによって記録されたアクティビティのレコードです。CloudWatch Logs が理解するログイベントレコードには 2 つのプロパティがあります。イベント発生時のタイムスタンプおよび生のイベントメッセージです。イベントメッセージは UTF-8 でエンコードされている必要があります。

ログストリーム

ログストリームは、同じソースを共有する一連のログイベントです。より具体的には、ログストリームは一般的に、モニタリングされているアプリケーションインスタンスやリソースから送信された順序でイベントを表すものです。たとえば、ログストリームは特定のホストの Apache アクセスログと関連付けられる場合があります。ログストリームを必要としなくなった場合、[aws logs delete-log-stream](#) コマンドを使用して、それを削除できます。

ロググループ

ロググループは、保持、監視、アクセス制御について同じ設定を共有するログストリームのグループを定義します。各ログストリームは、1 つのロググループに属している必要があります。たとえば、各ホストから Apache アクセスログの別のログストリームがある場合は、それらのログストリームを `MyWebsite.com/Apache/access_log` という名前の 1 つのロググループにグループ化できます。

1 つのロググループに属することができるログストリームの数に制限はありません。

メトリクスフィルター

メトリクスフィルタを使用して、取り込まれたイベントからメトリクスの監視データを抽出し、CloudWatch メトリクスのデータポイントに変換できます。メトリクスフィルターはロググループに割り当てられ、ロググループに割り当てられたすべてのフィルターはそのログストリームに適用されます。

保持設定

保持設定は、CloudWatch Logs にログイベントを保持する期間を指定するために使用できます。期限切れのログイベントは自動的に削除されます。メトリクスフィルターと同様に、保持設定はロググループに割り当てられ、ロググループに割り当てられた保持期間はそのログストリームに適用されます。

削除保護

削除保護は、ロググループとそのログストリームが誤って削除されないようにする保護手段です。ロググループで有効になっている場合、削除保護は明示的に無効になるまですべての削除操

作をブロックします。デフォルトでは、削除保護は有効になっていません。このオプション機能は、監査データを含むロググループやトラブルシューティングと分析のための本番稼働用アプリケーションログなど、重要な運用データとコンプライアンスデータを意図しない削除から保護するのに役立ちます。

Amazon CloudWatch Logs の請求とコスト

CloudWatch Logs および CloudWatch のコストと使用状況を分析する方法、およびコストを節約するためのベストプラクティスについては、「[CloudWatch の請求とコスト](#)」を参照してください。

料金の詳細については、「[Amazon CloudWatch の料金](#)」を参照してください。

にサインアップすると AWS、無料[AWS 利用](#)枠を使用して CloudWatch Logs を無料で使い始めることができます。

標準料金は、CloudWatch Logs を使用した他のサービスによって格納されたログ (Amazon VPC フローログおよび Lambda ログなど) に適用されます。

ログクラス

CloudWatch Logs には、2 つのクラスのロググループがあります。

- CloudWatch Logs 標準ログクラスは、リアルタイムモニタリングが必要なログや頻繁にアクセスするログに対するフル機能のオプションです。
- CloudWatch Logs 低頻度アクセスログクラスは、ログをコスト効率良く統合するために使用できる新しいログクラスです。このログクラスは、マネージド取り込み、ストレージ、クロスアカウントログ分析、暗号化など、CloudWatch Logs 機能のサブセットを提供し、GB あたりの取り込み料金は低くなります。低頻度アクセスログクラスは、アクセス頻度の低いログのアドホッククエリや事後フォレンジック分析に最適です。

Note

標準ログクラスと低頻度アクセスログクラスの料金は、取り込みコストのみが異なります。ストレージ料金と CloudWatch Logs Insights 料金は、各ログクラスで同じです。

CloudWatch の料金の詳細については、「[Amazon CloudWatch Logs 料金表](#)」をご覧ください。

Important

ロググループの作成後に、ログクラスを変更することはできません。

サポートされている機能

次の表に、各ログクラスの機能を示します。

機能	標準	低頻度 アクセス
フルマネージドログの取り込みとストレージ	はい ✓	はい ✓
クロスアカウント機能	はい ✓	はい ✓
による暗号化 AWS KMS	はい ✓	はい ✓

機能	標準	低頻度 アクセス
CloudWatch Logs Insights クエリコマンド	はい ✓	はい ✓ (ほとんどのコマンド – 「 ログクラスでサポートされている Logs Insights QL コマンド 」を参照)
CloudWatch Logs Insights 検出フィールド	はい ✓	はい ✓
ファセット	はい ✓	いいえ
CloudWatch Pipeline を使用したログの変換	はい ✓	はい ✓
Amazon S3 へのエクスポート	はい ✓	はい ✓
S3 テーブルの統合	はい ✓	はい ✓
スケジュールされたクエリ	はい ✓	はい ✓
OpenSearch PPL または OpenSearch SQL を使用して CloudWatch Logs Insights でクエリを実行	はい ✓	はい ✓
自然言語クエリアシスト	はい ✓	いいえ
CloudWatch Logs 異常検出	はい ✓	いいえ
Live Tail	はい ✓	いいえ
フィールドインデックス作成	はい ✓	いいえ
以前の時間範囲と比較する	はい ✓	いいえ
サブスクリプションフィルター	はい ✓	いいえ

機能	標準	低頻度 アクセス	
GetLogEvents および FilterLogEvents API オペレーション	はい ✓	サポート外。CloudWatch Logs Insights を使用して、低頻度アクセスログクラスのロググループに保存されているログイベントを表示します。	
メトリクスフィルター	はい ✓	いいえ	
Container Insights ログの取り込み	はい ✓	いいえ	
Lambda Insights ログの取り込み	はい ✓	いいえ	
機密性の高いデータをマスキングで保護する	はい ✓	はい ✓	
埋め込みメトリクス形式	はい ✓	いいえ	

Note

これらの 2 つのログクラスに加えて、Delivery ログクラスがあります。Delivery ログクラスは、Amazon S3 または に保存する AWS Lambda ログの配信にのみ使用します Amazon Data Firehose。配信クラスのロググループのログイベントは 2 CloudWatch Logs 日間保持されます。この保持期間は固定されており、変更できません。このログクラスには、CloudWatch Logs Insights クエリなどの豊富な CloudWatch Logs 機能はありません。

CloudWatch Logs の開始方法

Amazon EC2 インスタンスとオンプレミスサーバーから CloudWatch Logs にログを収集するには、統合 CloudWatch エージェントを使用します。ログと高度なメトリクスの両方を 1 つのエージェントで収集できます。Windows Server を実行しているサーバーなど、オペレーティングシステム全体にわたるサポートが提供されています。このエージェントでも優れたパフォーマンスを提供します。

統合 CloudWatch エージェントを使用して CloudWatch メトリクスを収集する場合、ゲスト内の可視性のために、追加のシステムメトリクスを収集できます。また、StatsD または collectd を使用して、カスタムメトリクスを収集することもできます。

詳細については、Amazon CloudWatch ユーザーガイドの「[CloudWatch エージェントのインストール](#)」を参照してください。

Linux を実行しているサーバーからのログの収集のみをサポートする古い CloudWatch Logs エージェントは非推奨となり、サポートされなくなりました。古い CloudWatch Logs エージェントから統合エージェントへの移行については、「[ウィザードを使用して CloudWatch エージェント設定ファイルを作成する](#)」を参照してください。

内容

- [前提条件](#)
- [統合された CloudWatch エージェントを使用して CloudWatch Logs を使用する](#)
- [前の CloudWatch ログエージェントを使用して CloudWatch ログを開始する方法](#)
- [クイックスタート: CloudFormation を使用して CloudWatch Logs の使用を開始する](#)

前提条件

Amazon CloudWatch Logs を使用するには、AWS アカウントが必要です。AWS アカウントでは、サービス (Amazon EC2 など) を使用して、ウェブベースのインターフェイスである CloudWatch コンソールで表示できるログを生成できます。さらに、AWS Command Line Interface () をインストールして設定できますAWS CLI。

にサインアップする AWS アカウント

がない場合は AWS アカウント、次の手順を実行して作成します。

にサインアップするには AWS アカウント

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

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

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

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

管理アクセスを持つユーザーを作成する

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

を保護する AWS アカウントのルートユーザー

1. ルートユーザーを選択し、AWS アカウント E メールアドレスを入力して、アカウント所有者[AWS マネジメントコンソール](#)として にサインインします。次のページでパスワードを入力します。

ルートユーザーを使用してサインインする方法については、「AWS サインイン ユーザーガイド」の「[ルートユーザーとしてサインインする](#)」を参照してください。

2. ルートユーザーの多要素認証 (MFA) を有効にします。

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

管理アクセスを持つユーザーを作成する

1. IAM アイデンティティセンターを有効にします。

手順については、「AWS IAM アイデンティティセンター ユーザーガイド」の「[AWS IAM アイデンティティセンターの有効化](#)」を参照してください。

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

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

管理アクセス権を持つユーザーとしてサインインする

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

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

追加のユーザーにアクセス権を割り当てる

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

手順については、「AWS IAM アイデンティティセンター ユーザーガイド」の「[アクセス許可セットを作成する](#)」を参照してください。

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

手順については、「AWS IAM アイデンティティセンター ユーザーガイド」の「[グループを追加する](#)」を参照してください。

Command Line Interface をセットアップする

を使用して CloudWatch Logs オペレーション AWS CLI を実行できます。

をインストールして設定する方法については AWS CLI、[「ユーザーガイド」の AWS 「コマンドラインインターフェイスのセットアップ」](#)を参照してください。AWS Command Line Interface

統合された CloudWatch エージェントを使用して CloudWatch Logs を使用する

統合された CloudWatch エージェントを使用して CloudWatch Logs を開始する方法については、Amazon CloudWatch ユーザーガイドの「[CloudWatch エージェントを使用した Amazon EC2 Instances インスタンスとオンプレミスサーバーからのメトリクスとログの収集](#)」を参照してください。このセクションに記載されている手順を実行してエージェントのインストールと設定を行い、開始します。エージェントを使用して CloudWatch メトリクスを収集していない場合、メトリクスを参照するセクションはすべて無視できます。

現在古い CloudWatch Logs エージェントを使用していて、新しい統合エージェントの使用に移行する場合は、新しいエージェントパッケージに含まれているウィザードを使用することをお勧めします。このウィザードは、現在の CloudWatch Logs エージェント設定ファイルを読み込み、CloudWatch エージェントを設定して同じログを収集することができます。ウィザードの詳細については、Amazon CloudWatch ユーザーガイドの「[ウィザードを使用して CloudWatch エージェント設定ファイルを作成する](#)」を参照してください。

前の CloudWatch ログエージェントを使用して CloudWatch ログを開始する方法

Important

CloudWatch には、EC2 インスタンスとオンプレミスサーバーからログとメトリクスの両方を収集できる統合 CloudWatch エージェントが含まれています。ログのみの古いエージェントは非推奨となり、サポートされなくなりました。

ログのみの古いエージェントから統合エージェントへの移行については、「[ウィザードを使用して CloudWatch エージェント設定ファイルを作成する](#)」を参照してください。

このセクションの残りの部分では、古い CloudWatch Logs エージェントをまだ使用しているお客様向けに、その使用方法について説明します。

CloudWatch Logs エージェントを使用して、Linux または Windows Server を実行する Amazon EC2 インスタンスのログデータおよび AWS CloudTrailから記録されたイベントを発行できます。代わり

に CloudWatch 統合エージェントを使用してログデータを発行することをお勧めします。新しいエージェントの詳細については、Amazon CloudWatch User Guideの「[CloudWatch エージェントを使用した Amazon EC2 Instances インスタンスとオンプレミスサーバーからのメトリクスとログの収集](#)」を参照してください。

内容

- [CloudWatch Logs エージェントの前提条件](#)
- [クイックスタート: 実行中の EC2 Linux インスタンスに CloudWatch Logs エージェントをインストールして設定する](#)
- [クイックスタート: EC2 Linux インスタンスの起動時に CloudWatch Logs エージェントをインストールして設定する](#)
- [クイックスタート: Windows Server 2016 を実行している Amazon EC2 インスタンスで CloudWatch Logs エージェントを使用してログを CloudWatch Logs に送信できるようにする](#)
- [クイックスタート: Windows Server 2012 および Windows Server 2008 を実行している Amazon EC2 インスタンスで、CloudWatch Logs へのログの送信を有効にする](#)
- [CloudWatch Logs エージェントの状態を報告する](#)
- [CloudWatch Logs エージェントを起動する](#)
- [CloudWatch Logs エージェントを停止する](#)
- [CloudWatch Logs エージェントのリファレンス](#)

CloudWatch Logs エージェントの前提条件

CloudWatch Logs エージェントでは、Python バージョン 2.7、3.0、または 3.3、および次のいずれかのバージョンの Linux が必要です。

- Amazon Linux バージョン 2014.03.02 以降。Amazon Linux 2 はサポートされていません
- Ubuntu Server バージョン 12.04、14.04、または 16.04
- CentOS バージョン 6、6.3、6.4、6.5、または 7.0
- Red Hat Enterprise Linux (RHEL) バージョン 6.5 または 7.0
- Debian 8.0

クイックスタート: 実行中の EC2 Linux インスタンスに CloudWatch Logs エージェントをインストールして設定する

Important

古いログエージェントは廃止する予定です。CloudWatch には、EC2 インスタンスとオンプレミスサーバーからログとメトリクスの両方を収集できる統合エージェントが含まれています。詳細については、「[CloudWatch Logs の開始方法](#)」を参照してください。

古い CloudWatch Logs エージェントから統合エージェントへの移行については、「[ウィザードを使用して CloudWatch エージェント設定ファイルを作成する](#)」を参照してください。

古いログエージェントは、Python の 2.6 から 3.5 までのバージョンしかサポートしていません。さらに、古い CloudWatch Logs エージェントでは、インスタンスメタデータサービスバージョン 2 (IMDSv2) がサポートされていません。サーバーで IMDSv2 を使用している場合は、古い CloudWatch Logs エージェントではなく、新しい統合エージェントを使用する必要があります。

このセクションの残りの部分では、古い CloudWatch Logs エージェントをまだ使用しているお客様向けに、その使用方法について説明します。

Tip

CloudWatch には、EC2 インスタンスとオンプレミスサーバーからログとメトリクスの両方を収集できる新しい統合エージェントが含まれています。古い CloudWatch Logs エージェントをすでに使用しなくなった場合は、新しい統合 CloudWatch エージェントを使用することをお勧めします。詳細については、「[CloudWatch Logs の開始方法](#)」を参照してください。さらに、古いエージェントでは、Instance Metadata Service Version 2 (IMDSv2) がサポートされていません。サーバーで IMDSv2 を使用している場合は、古い CloudWatch Logs エージェントではなく、新しい統合エージェントを使用する必要があります。

このセクションの残りの部分では、古い CloudWatch Logs エージェントの使用について説明します。

実行中の EC2 Linux インスタンスで古い CloudWatch Logs エージェントを設定する

既存の EC2 インスタンスで CloudWatch Logs エージェントインストーラを使用して、CloudWatch Logs エージェントをインストールして設定できます。インストールが完了したら、エージェントの

インストール中に、インスタンスから、作成したログストリームにログが自動的に流れます。エージェントは開始を確認し無効にされるまで実行し続けます。

エージェントの使用に加えて、CloudWatch Logs SDK AWS CLI、または CloudWatch Logs API を使用してログデータを発行することもできます。AWS CLI は、コマンドラインまたはスクリプトを使用してデータを発行するのに最適です。CloudWatch Logs SDK は、アプリケーションから直接、または独自のログ発行アプリケーションを構築してログデータを発行する場合に適しています。

ステップ 1: CloudWatch Logs で使用する IAM ロールまたはユーザーを設定する

CloudWatch Logs エージェントは、IAM ロールとユーザーをサポートします。インスタンスに関連付けられた IAM ロールがすでに存在する場合、その下に IAM ポリシーが含まれていることを確認してください。インスタンスに関連付けられた IAM ロールがまだ存在しない場合は、次のステップで IAM 認証情報を使用するか、IAM ロールインスタンスに割り当てることができます。詳細については、「[インスタンスへの IAM ロールのアタッチ](#)」を参照してください。

IAM ロールまたは CloudWatch Logs ユーザーを設定するには

1. IAM コンソール (<https://console.aws.amazon.com/iam/>) を開きます。
2. ナビゲーションペインで Roles (ロール) を選択してください。
3. ロール名を選択してロールを選択します (名前の横にあるチェックボックスを選択しないでください)。
4. [Attach Policies (ポリシーのアタッチ)] を選択して、[ポリシーの作成] を選択します。

新しいブラウザタブまたはウィンドウが開きます。

5. [JSON] タブを選択して、次の JSON ポリシードキュメントを入力します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams"
      ],
      "Resource": [
```

```
    ""  
  ]  
}  
]  
}
```

6. 完了したら、[ポリシーの確認] を選択します。構文エラーがある場合は、Policy Validator (ポリシー検証) によってレポートされます。
7. [ポリシーの確認] ページで、作成するポリシーの [名前] と [説明] (オプション) を入力します。ポリシーの [Summary] (概要) を参照して、ポリシーによって付与された許可を確認します。次に、[Create policy] (ポリシーの作成) を選択して作業を保存します。
8. ブラウザタブまたはウィンドウを閉じ、ロールの [アクセス権限の追加] を選択します。[更新] を選択し、新しいポリシーを選択してロールに追加します。
9. [Attach Policy] を選択します。

ステップ 2: 既存の Amazon EC2 インスタンスに CloudWatch Logs をインストールして設定する

CloudWatch Logs エージェントのインストール手順は、Amazon EC2 インスタンスが Amazon Linux、Ubuntu、CentOS、Red Hat のどれを実行しているかによって異なります。インスタンスの Linux のバージョンに適切な手順を使用してください。

既存の Amazon Linux インスタンスに CloudWatch Logs をインストールして設定するには

Amazon Linux AMI 2014.09 から、awslogs パッケージを使用した RPM のインストールに CloudWatch Logs エージェントが使用可能です。それ以前のバージョンの Amazon Linux は、`sudo yum update -y` コマンドを使用してインスタンスを更新することで awslogs パッケージにアクセスできます。CloudWatch Logs インストーラを使用する代わりに awslogs パッケージを RPM としてインストールすることで、インスタンスは から定期的なパッケージの更新とパッチを受け取ります。CloudWatch Logs エージェントを手動で再インストール AWS する必要はありません。

Warning

以前に Python スクリプトを使用してエージェントをインストールした場合は、RPM インストール方法を使用して CloudWatch Logs エージェントを更新しないでください。設定に問題が発生して CloudWatch Logs エージェントが CloudWatch にログを送信できなくなる恐れがあります。

1. Amazon Linux インスタンスに接続します。詳細については、「Amazon EC2 ユーザーガイド」の「[Connect to Your Instance](#)」を参照してください。

接続問題の詳細については、「Amazon EC2 ユーザーガイド」「[Amazon EC2 Linux インスタンスへの接続に関する問題のトラブルシューティング](#)」を参照してください。

2. Amazon Linux インスタンスを更新してパッケージリポジトリの最新の変更を取得します。

```
sudo yum update -y
```

3. awslogs パッケージをインストールします。これは Amazon Linux インスタンスで awslogs をインストールする推奨手段です。

```
sudo yum install -y awslogs
```

4. /etc/awslogs/awslogs.conf ファイルを編集して追跡するログを設定します。このファイルの編集方法については、「[CloudWatch Logs エージェントのリファレンス](#)」を参照してください。
5. デフォルトでは、/etc/awslogs/awsccli.conf は us-east-1 リージョンを指します。ログを別の領域にプッシュするには、awsccli.conf ファイルを編集し、その領域を指定します。
6. awslogs サービスを開始します。

```
sudo service awslogs start
```

Amazon Linux 2 を実行している場合は、次のコマンドを使用して awslogs サービスを開始します。

```
sudo systemctl start awslogsd
```

7. (オプション) /var/log/awslogs.log ファイルでサービス開始時に記録されたエラーがあるかどうか確認します。
8. (オプション) システム起動時に毎回 awslogs サービスを起動する場合は、次のコマンドを実行します。

```
sudo chkconfig awslogs on
```

Amazon Linux 2 を実行している場合は、次のコマンドを使用してシステムブートのたびにサービスを開始します。


```
sudo systemctl enable awslogs.service
```

9. エージェントが実行されてしばらくしたら、CloudWatch コンソールに新しく作成されたロググループとログストリームを確認してください。

詳細については、「[CloudWatch Logs に送信されたログデータを表示する](#)」を参照してください。

既存の Ubuntu Server、CentOS、Red Hat のインスタンスに CloudWatch Logs をインストールして設定するには

Ubuntu Server、CentOS、または Red Hat を実行する AMI を使用している場合、次の手順でインスタンスに CloudWatch Logs エージェントを手動インストールします。

1. EC2 インスタンスに接続します。詳細については、「Amazon EC2 ユーザーガイド」の「[Connect to Your Instance](#)」を参照してください。

接続問題の詳細については、「Amazon EC2 ユーザーガイド」「[Amazon EC2 Linux インスタンスへの接続に関する問題のトラブルシューティング](#)」を参照してください。

2. 2つのオプションのいずれかを使用して CloudWatch Logs エージェントインストーラを実行します。インターネットから直接実行するか、ファイルをダウンロードしてスタンドアロンで実行できます。

 Note

CentOS 6.x、Red Hat 6.x、または Ubuntu 12.04 を実行している場合は、インストーラをダウンロードしてスタンドアロンで実行する手順を使用してください。これらのシステムでは、インターネットから直接 CloudWatch Logs エージェントをインストールすることはサポートされていません。

 Note

Ubuntu では、次のコマンドを実行する前に `apt-get update` を実行してください。

インターネットから直接実行するには、次のコマンドを使用してプロンプトに従います。

```
curl https://s3.amazonaws.com/aws-cloudwatch/downloads/latest/awslogs-agent-setup.py -O
```

```
sudo python ./awslogs-agent-setup.py --region us-east-1
```

前のコマンドが機能しない場合は、以下を試してください。

```
sudo python3 ./awslogs-agent-setup.py --region us-east-1
```

スタンドアロンをダウンロードして実行するには、次のコマンドを使用してプロンプトに従います。

```
curl https://s3.amazonaws.com/aws-cloudwatch/downloads/latest/awslogs-agent-setup.py -O
```

```
curl https://s3.amazonaws.com/aws-cloudwatch/downloads/latest/AgentDependencies.tar.gz -O
```

```
tar xvf AgentDependencies.tar.gz -C /tmp/
```

```
sudo python ./awslogs-agent-setup.py --region us-east-1 --dependency-path /tmp/AgentDependencies
```

CloudWatch Logs エージェントをインストールするには、us-east-1、us-west-1、us-west-2、ap-south-1、ap-northeast-2、ap-southeast-1、ap-southeast-2、ap-northeast-1、eu-central-1、eu-west-1、または sa-east-1 の各リージョンを指定します。

Note

awslogs-agent-setup の現行バージョンとバージョン履歴の詳細については、[CHANGELOG.txt](#) を参照してください。

CloudWatch Logs エージェントのインストーラはセットアップ時に特定の情報が必要です。開始する前に、モニタリングするログファイルとそのタイムスタンプ形式を知っておく必要があります。また、次の情報を準備する必要があります。

項目	説明
AWS アクセスキー ID	IAM ロールを使用する場合は Enter キーを押します。それ以外の場合は、AWS アクセスキー ID を入力します。
AWS シークレットアクセスキー	IAM ロールを使用する場合は Enter キーを押します。それ以外の場合は、AWS シークレットアクセスキーを入力します。
デフォルトリージョン名	Enter キーを押します。デフォルトは us-east-2 です。これは、us-east-1、us-west-1、us-west-2、ap-south-1、ap-northeast-2、ap-southeast-1、ap-southeast-2、ap-northeast-1、eu-central-1、eu-west-1、または sa-east-1 に設定できます。
デフォルト出力形式	空白にしたまま Enter キーを押します。
アップロードするログファイルのパス	送信するログデータを含むファイルの場所です。インストーラはパスの候補を表示します。
送信先ロググループ名	ロググループの名前です。インストーラはロググループ名の候補を表示します。
送信先ログストリーム名	デフォルトでは、ホストの名前です。インストーラはホスト名の候補を表示します。
タイムスタンプ形式	指定ログファイル内のタイムスタンプ形式を指定します。独自の形式を指定するには、[custom] を選択します。
初期位置	データをどのようにアップロードできますか データファイルのすべてをアップロードする場合は [start_of_file] に設定します。新しく付け加えられたデータのみをアップロードする場合は [end_of_file] に設定します。

これらの手順が完了すると、インストーラは別のログファイルを設定するかどうか尋ねてきます。各ログファイルについて何回でもプロセスを実行できます。他にモニタリングするログファイルがない場合、別のログをセットアップするようにインストーラに求められた時に [N] を選択します。エージェント設定ファイルの設定の詳細については、「[CloudWatch Logs エージェントのリファレンス](#)」を参照してください。

 Note

複数のログソースから単一のログストリームにデータを送信する設定はサポートされていません。

3. エージェントが実行されてしばらくしたら、CloudWatch コンソールに新しく作成されたロググループとログストリームを確認してください。

詳細については、「[CloudWatch Logs に送信されたログデータを表示する](#)」を参照してください。

クイックスタート: EC2 Linux インスタンスの起動時に CloudWatch Logs エージェントをインストールして設定する

 Tip

このセクションで説明している古い CloudWatch Logs エージェントは非推奨になる予定です。代わりに、ログとメトリクスを両方を収集できる新しい統合 CloudWatch エージェントを使用することを強くお勧めします。さらに、古い CloudWatch Logs エージェントには Python 3.3 以前が必要であり、これらのバージョンはデフォルトでは新しい EC2 インスタンスにインストールされません。統合 CloudWatch エージェントの詳細については、[CloudWatch エージェントのインストール](#)を参照してください。

このセクションの残りの部分では、古い CloudWatch Logs エージェントの使用について説明します。

起動時における EC2 Linux インスタンスへの古い CloudWatch Logs エージェントのインストール

Amazon EC2 ユーザーデータは、インスタンスの起動時にパラメータ情報を渡す Amazon EC2 の機能です。これを使用してインスタンスに CloudWatch Logs エージェントをインストールして設定します。CloudWatch Logs エージェントのインストール情報と設定情報を Amazon EC2 に渡すには、Amazon S3 バケットのようなネットワークの場所に設定ファイルを用意します。

複数のログソースから単一のログストリームにデータを送信する設定はサポートされていません。

前提条件

ロググループとログストリームをすべて記述したエージェントの設定ファイルを作成します。これは、モニタリングするログファイルとそのアップロード先のロググループおよびログストリームが記述されているテキストファイルです。エージェントはこの設定ファイルを使用して記述されたすべてのログファイルのモニタリングおよびアップロードを開始します。エージェント設定ファイルの設定の詳細については、「[CloudWatch Logs エージェントのリファレンス](#)」を参照してください。

以下は、Amazon Linux 2 のサンプルエージェント設定ファイルです。

```
[general]
state_file = /var/lib/awslogs/state/agent-state

[/var/log/messages]
file = /var/log/messages
log_group_name = /var/log/messages
log_stream_name = {instance_id}
datetime_format = %b %d %H:%M:%S
```

以下は、Ubuntu のサンプルエージェント設定ファイルです。

```
[general]
state_file = /var/awslogs/state/agent-state

[/var/log/syslog]
file = /var/log/syslog
log_group_name = /var/log/syslog
log_stream_name = {instance_id}
datetime_format = %b %d %H:%M:%S
```

IAM ロールを設定するには

1. IAM コンソール (<https://console.aws.amazon.com/iam/>) を開きます。
2. ナビゲーションペインで、[Policies]、[Create Policy] の順に選択します。
3. [Create Policy] ページの [Create Your Own Policy] で、[Select] を選択します。カスタムポリシー作成の詳細については、「Amazon EC2 ユーザーガイド」の「[Amazon EC2 のIAM ポリシー](#)」を参照してください。
4. [Review Policy] ページで、[Policy Name] にポリシーの名前を入力します。
5. [Policy Document] に、次のポリシーをコピーして貼り付けます。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams"
      ],
      "Resource": [
        "arn:aws:logs:*:*:*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": [
        "arn:aws:s3:::amzn-s3-demo-bucket/*"
      ]
    }
  ]
}
```

6. [ポリシーを作成] を選択します。

7. ナビゲーションペインで [Roles]、[Create New Role] の順に選択します。
8. [Set Role Name] ページで、ロールの名前を入力し、[Next Step] を選択します。
9. [Select Role Type] ページで、[Amazon EC2] の隣にある [Select] を選択します。
10. [Attach Policy] ページのテーブルのヘッダーで、[Policy Type]、[Customer Managed] の順に選択します。
11. 作成した IAM ポリシーを選択し、[Next Step (次のステップ)] を選択します。
12. [ロールの作成] を選択します。

ユーザーとポリシーの詳細については、IAM ユーザーガイドの「[IAM ユーザーとグループ](#)」および「[IAM ポリシーを管理する](#)」を参照してください。

新しいインスタンスを起動して CloudWatch Logs を有効にするには

1. Amazon EC2 コンソール (<https://console.aws.amazon.com/ec2/>) を開きます。
2. [インスタンスの作成] を選択してください。

詳細については、「Amazon EC2 ユーザーガイド」の「[Launching an Instance](#)」を参照してください。

3. [ステップ 1: Amazon Machine Image (AMI) を選択する] ページで、起動する Linux インスタンスタイプを選択し、[ステップ 2: インスタンスタイプを選択する] ページで [Next: Configure Instance Details] を選択します。

[cloud-init](#) が Amazon Machine Image (AMI) に含まれていることを確認します。Amazon Linux AMIs、および Ubuntu と RHEL の AMIs にはすでに cloud-init が含まれていますが、の CentOS やその他の AMIs には含まれていない AWS Marketplace 場合があります。

4. [ステップ 3: インスタンスの詳細を設定する] ページの [IAM role (IAM ロール)] で、作成した IAM ロールを選択します。
5. [Advanced Details] の [User data] で、以下のスクリプトをボックス内に貼り付けます。その後、スクリプトを更新するには、[-c] オプションの値を、エージェント設定ファイルの位置に変更します。

```
#!/bin/bash
curl https://s3.amazonaws.com/aws-cloudwatch/downloads/latest/awslogs-agent-setup.py -O
chmod +x ./awslogs-agent-setup.py
./awslogs-agent-setup.py -n -r us-east-1 -c s3://amzn-s3-demo-bucket/my-config-file
```

6. そのほかインスタンスへの必要な変更を行い、起動設定を確認して [Launch] を選択します。
7. エージェントが実行されてしばらくしたら、CloudWatch コンソールに新しく作成されたロググループとログストリームを確認してください。

詳細については、「[CloudWatch Logs に送信されたログデータを表示する](#)」を参照してください。

クイックスタート: Windows Server 2016 を実行している Amazon EC2 インスタンスで CloudWatch Logs エージェントを使用してログを CloudWatch Logs に送信できるようにする

Tip

CloudWatch には、EC2 インスタンスとオンプレミスサーバーからログとメトリクスの両方を収集できる新しい統合エージェントが含まれています。新しい統合 CloudWatch エージェントを使用することをお勧めします。詳細については、「[CloudWatch Logs の開始方法](#)」を参照してください。

このセクションの残りの部分では、古い CloudWatch Logs エージェントの使用について説明します。

Windows Server 2016 を実行している Amazon EC2 インスタンスで、古い CloudWatch Logs エージェントを使用して CloudWatch Logs にログを送信できるようにする

Windows Server 2016 を実行しているインスタンスで CloudWatch Logs にログを送信するには、複数の方法を使用できます。このセクションのステップでは、Systems Manager Run Command を使用します。他の可能な方法の詳細については、「[Amazon CloudWatch へのログ、イベント、パフォーマンスカウンターの送信](#)」を参照してください。

ステップ

- [サンプル設定ファイルをダウンロードする](#)
- [CloudWatch の JSON ファイルを設定する](#)
- [Systems Manager 用 IAM ロールを作成する](#)
- [Systems Manager の前提条件を確認する](#)

- [インターネットアクセスを確認する](#)
- [Systems Manager Run Command を使用して CloudWatch Logs を有効化する](#)

サンプル設定ファイルをダウンロードする

サンプルファイル ([AWS.EC2.Windows.CloudWatch.json](#)) をコンピュータにダウンロードします。

CloudWatch の JSON ファイルを設定する

CloudWatch に送信するログを決めるには、設定ファイルで選択して指定します。このファイルを作成し、項目を選択して指定するプロセスは、完了までに 30 分以上かかる場合があります。このタスクを 1 回完了したら、すべてのインスタンスで設定ファイルを再利用できます。

ステップ

- [ステップ 1: CloudWatch Logs を有効化する](#)
- [ステップ 2: CloudWatch 設定を構成する](#)
- [ステップ 3: 送信するデータを設定する](#)
- [ステップ 4: フロー制御を設定する](#)
- [ステップ 5: JSON コンテンツを保存する](#)

ステップ 1: CloudWatch Logs を有効化する

JSON ファイルの先頭で、IsEnableded の「false」を「true」に変更します。

```
"IsEnableded": true,
```

ステップ 2: CloudWatch 設定を構成する

認証情報、リージョン、ロググループ名、およびログストリーム名前空間を指定します。これにより、インスタンスがログデータを CloudWatch Logs に送信できます。同じログデータを複数の異なる宛先に送信する場合は、一意の ID (たとえば「CloudWatchLogs2」および「CloudWatchLogs3」) および各 ID に異なるリージョンを付加してセクションを追加できます。

ログデータを CloudWatch Logs に送信するように設定するには

1. JSON ファイルで、CloudWatchLogs セクションを見つけます。

```
{
  "Id": "CloudWatchLogs",
  "FullName":
  "AWS.EC2.Windows.CloudWatch.CloudWatchLogsOutput,AWS.EC2.Windows.CloudWatch",
  "Parameters": {
    "AccessKey": "",
    "SecretKey": "",
    "Region": "us-east-1",
    "LogGroup": "Default-Log-Group",
    "LogStream": "{instance_id}"
  }
},
```

2. [AccessKey] および [SecretKey] フィールドは空白のままにしておきます。IAM ロールを使用して認証情報を設定します。
3. Region には、ログデータを送信するリージョンを入力します (たとえば、us-east-2)。
4. LogGroup には、ロググループの名前を入力します。この名前は、CloudWatch コンソールの [Log Groups (ロググループ)] 画面に表示されます。
5. LogStream には、送信先のログストリームを入力します。この名前は、CloudWatch コンソールの [Log Groups (ロググループ)] > [Streams (ストリーム)] 画面に表示されます。

デフォルトの {instance_id} を使用する場合、ログストリーム名はこのインスタンスのインスタンス ID です。

存在しないログストリーム名を特定すると、CloudWatch Logs によってログストリームが自動的に作成されます。リテラル文字列、事前定義された変数 {instance_id}、{hostname}、{ip_address}、またはこれらの組み合わせを使用してログストリーム名を定義できます。

ステップ 3: 送信するデータを設定する

イベントログデータ、Event Tracing for Windows (ETW) データ、および他のログデータを CloudWatch Logs に送信できます。

Windows アプリケーションイベントログデータを CloudWatch Logs に送信するには

1. JSON ファイルで、ApplicationEventLog セクションを見つけます。

```
{
```

```
    "Id": "ApplicationEventLog",
    "FullName":
    "AWS.EC2.Windows.CloudWatch.EventLog.EventLogInputComponent,AWS.EC2.Windows.CloudWatch",
    "Parameters": {
        "LogName": "Application",
        "Levels": "1"
    }
},
```

2. Levels には、アップロードするメッセージのタイプを指定します。次のいずれかの値を指定できます。

- 1 - エラーメッセージだけをアップロードします。
- 2 - 警告メッセージだけをアップロードします。
- 4 - 情報メッセージだけをアップロードします。

値を組み合わせて複数のタイプのメッセージを含めることができます。たとえば、値 3 はエラーメッセージ (1) と警告メッセージ (2) をアップロードします。値 7 は、エラーメッセージ (1)、警告メッセージ (2)、情報メッセージ (4) をアップロードします。

セキュリティログデータを CloudWatch Logs に送信するには

1. JSON ファイルで、SecurityEventLog セクションを見つけます。

```
{
    "Id": "SecurityEventLog",
    "FullName":
    "AWS.EC2.Windows.CloudWatch.EventLog.EventLogInputComponent,AWS.EC2.Windows.CloudWatch",
    "Parameters": {
        "LogName": "Security",
        "Levels": "7"
    }
},
```

2. Levels には、7 と入力してすべてのメッセージをアップロードします。

システムイベントログデータを CloudWatch Logs に送信するには

1. JSON ファイルで、SystemEventLog セクションを見つけます。

```
{
  "Id": "SystemEventLog",
  "FullName":
"AWS.EC2.Windows.CloudWatch.EventLog.EventLogInputComponent,AWS.EC2.Windows.CloudWatch",
  "Parameters": {
    "LogName": "System",
    "Levels": "7"
  }
},
```

2. Levels には、アップロードするメッセージのタイプを指定します。次のいずれかの値を指定できます。

- 1 - エラーメッセージだけをアップロードします。
- 2 - 警告メッセージだけをアップロードします。
- 4 - 情報メッセージだけをアップロードします。

値を組み合わせて複数のタイプのメッセージを含めることができます。たとえば、値 3 はエラーメッセージ (1) と警告メッセージ (2) をアップロードします。値 7 は、エラーメッセージ (1)、警告メッセージ (2)、情報メッセージ (4) をアップロードします。

他の種類のイベントログデータを CloudWatch Logs に送信するには

1. JSON ファイルに、新しいセクションを追加します。各セクションには固有の Id が必要です。

```
{
  "Id": "Id-name",
  "FullName":
"AWS.EC2.Windows.CloudWatch.EventLog.EventLogInputComponent,AWS.EC2.Windows.CloudWatch",
  "Parameters": {
    "LogName": "Log-name",
    "Levels": "7"
  }
},
```

2. Id には、アップロードするログの名前を入力します (たとえば、**WindowsBackup**)。
3. LogName には、アップロードするログの名前を入力します。ログの名前を次のように確認できます。

- a. イベントビューワーを開きます。
 - b. ナビゲーションペインで、[Applications and Services Logs] を選択します。
 - c. ログに移動し、[Actions]、[Properties] を選択します。
4. Levels には、アップロードするメッセージのタイプを指定します。次のいずれかの値を指定できます。
- 1 - エラーメッセージだけをアップロードします。
 - 2 - 警告メッセージだけをアップロードします。
 - 4 - 情報メッセージだけをアップロードします。

値を組み合わせることで複数のタイプのメッセージを含めることができます。たとえば、値 3 はエラーメッセージ (1) と警告メッセージ (2) をアップロードします。値 7 は、エラーメッセージ (1)、警告メッセージ (2)、情報メッセージ (4) をアップロードします。

イベントトレース (Windows) データを CloudWatch Logs に送信するには

ETW (Event Tracing for Windows) には、アプリケーションがログを書き込むことができる効率的で、きめ細かいログ記録メカニズムが用意されています。各 ETW は、ログ記録セッションを開始および停止できるセッションマネージャにより制御されます。各セッションには、プロバイダーと 1 つ以上のコンシューマーが存在します。

1. JSON ファイルで、ETW セクションを見つけます。

```
{
  "Id": "ETW",
  "FullName":
  "AWS.EC2.Windows.CloudWatch.EventLog.EventLogInputComponent,AWS.EC2.Windows.CloudWatch",
  "Parameters": {
    "LogName": "Microsoft-Windows-WinINet/Analytic",
    "Levels": "7"
  }
},
```

2. LogName には、アップロードするログの名前を入力します。
3. Levels には、アップロードするメッセージのタイプを指定します。次のいずれかの値を指定できます。

- **1** - エラーメッセージだけをアップロードします。
- **2** - 警告メッセージだけをアップロードします。
- **4** - 情報メッセージだけをアップロードします。

値を組み合わせることで複数のタイプのメッセージを含めることができます。たとえば、値 **3** はエラーメッセージ (**1**) と警告メッセージ (**2**) をアップロードします。値 **7** は、エラーメッセージ (**1**)、警告メッセージ (**2**)、情報メッセージ (**4**) をアップロードします。

カスタムログ (テキストベースのログファイル) を CloudWatch Logs に送信するには

1. JSON ファイルで、CustomLogs セクションを見つけます。

```
{
  "Id": "CustomLogs",
  "FullName":
  "AWS.EC2.Windows.CloudWatch.CustomLog.CustomLogInputComponent,AWS.EC2.Windows.CloudWatch",
  "Parameters": {
    "LogDirectoryPath": "C:\\\\CustomLogs\\",
    "TimestampFormat": "MM/dd/yyyy HH:mm:ss",
    "Encoding": "UTF-8",
    "Filter": "",
    "CultureName": "en-US",
    "TimeZoneKind": "Local",
    "LineCount": "5"
  }
},
```

2. LogDirectoryPath には、ログがインスタンスに格納されるパスを入力します。
3. TimestampFormat には、使用するタイムスタンプ形式を入力します。サポートされる値の詳細については、MSDN の「[カスタムの日付と時刻の書式指定文字列](#)」を参照してください。

Important

ソースログファイルには、各ログ行の先頭にタイムスタンプがあり、タイムスタンプの後にスペースがある必要があります。

4. Encoding には、使用するファイルエンコード (UTF-8 など) を入力します。サポートされる値の一覧については、MSDN の「[Encoding クラス](#)」を参照してください。

Note

表示名ではなく、エンコード名を使用します。

5. (オプション) `Filter` には、ログファイル名のプレフィックスを入力します。すべてのファイルをモニタリングするには、このパラメータを空白のままにします。サポートされる値の詳細については、MSDN の「[FileSystemWatcherFilter プロパティ](#)」を参照してください。
6. (オプション) `CultureName` には、タイムスタンプが記録されているロケールを入力します。`CultureName` が空の場合、Windows インスタンスにより現在使用されているのと同じロケールがデフォルトになります。詳細については、MSDN の「[Product Behavior](#)」トピックの表で、`Language tag` 列を参照してください。

Note

`div`、`div-MV`、`hu`、および `hu-HU` 値は、サポートされていません。

7. (オプション) `TimeZoneKind` には、`Local` または `UTC` を入力します。これを設定すると、ログのタイムスタンプにタイムゾーン情報が含まれていない場合にタイムゾーン情報を提供できます。このパラメータが空になっていて、タイムスタンプにタイムゾーン情報が含まれていない場合、CloudWatch Logs ではデフォルトでローカルタイムゾーンが使用されます。タイムスタンプに既にタイムゾーン情報が含まれている場合、このパラメータは無視されます。
8. (オプション) `LineCount` には、ログファイルを識別するためのヘッダーの行数を入力します。たとえば、IIS のログファイルのヘッダーはほぼ同じです。「5」と入力すると、ログファイルのヘッダーの最初の 3 行が読み取られ、ログファイルを識別できます。IIS ログファイルで、3 番目の行は日時のスタンプですが、タイムスタンプは常にログファイル間で異なるという保証はありません。そのため、ログファイルの一意のフィンガープリントを作成するには、実際のログデータの少なくとも 1 行を含めることをお勧めします。

IIS ログデータを CloudWatch Logs に送信するには

1. JSON ファイルで、`IISLog` セクションを見つけます。

```
{
  "Id": "IISLogs",
  "FullName":
    "AWS.EC2.Windows.CloudWatch.CustomLog.CustomLogInputComponent,AWS.EC2.Windows.CloudWatch",
  "Parameters": {
```

```
    "LogDirectoryPath": "C:\\inetpub\\logs\\LogFiles\\W3SVC1",
    "TimestampFormat": "yyyy-MM-dd HH:mm:ss",
    "Encoding": "UTF-8",
    "Filter": "",
    "CultureName": "en-US",
    "TimeZoneKind": "UTC",
    "LineCount": "5"
  },
},
```

2. LogDirectoryPath には、個々のサイト (C:\inetpub\logs\LogFiles\W3SVCn など) の IIS ログが格納されているフォルダを入力します。

 Note

W3C ログ形式のみサポートされます。IIS、NCSA、カスタム形式はサポートされません。

3. TimestampFormat には、使用するタイムスタンプ形式を入力します。サポートされる値の詳細については、MSDN の「[カスタムの日付と時刻の書式指定文字列](#)」を参照してください。
4. Encoding には、使用するファイルエンコード (UTF-8 など) を入力します。サポートされる値の詳細については、MSDN の「[Encoding クラス](#)」を参照してください。

 Note

表示名ではなく、エンコード名を使用します。

5. (オプション) Filter には、ログファイル名のプレフィックスを入力します。すべてのファイルをモニタリングするには、このパラメータを空白のままにします。サポートされる値の詳細については、MSDN の「[FileSystemWatcherFilter プロパティ](#)」を参照してください。
6. (オプション) CultureName には、タイムスタンプが記録されているロケールを入力します。CultureName が空の場合、Windows インスタンスにより現在使用されているのと同じロケールがデフォルトになります。サポートされる値の詳細については、MSDN の「[Product Behavior](#)」トピックの表で、Language tag 列を参照してください。

 Note

div、div-MV、hu、および hu-HU 値は、サポートされていません。

7. (オプション) `TimeZoneKind` には、`Local` または `UTC` を入力します。これを設定すると、ログのタイムスタンプにタイムゾーン情報が含まれていない場合にタイムゾーン情報を提供できます。このパラメータが空になっていて、タイムスタンプにタイムゾーン情報が含まれていない場合、CloudWatch Logs ではデフォルトでローカルタイムゾーンが使用されます。タイムスタンプに既にタイムゾーン情報が含まれている場合、このパラメータは無視されます。
8. (オプション) `LineCount` には、ログファイルを識別するためのヘッダーの行数を入力します。たとえば、IIS のログファイルのヘッダーはほぼ同じです。「5」と入力すると、ログファイルのヘッダーの最初の 5 行が読み取られ、ログファイルを識別できます。IIS ログファイルで、3 番目の行は日時のスタンプですが、タイムスタンプは常にログファイル間で異なるという保証はありません。そのため、ログファイルの一意のフィンガープリントを作成するには、実際のログデータの少なくとも 1 行を含めることをお勧めします。

ステップ 4: フロー制御を設定する

各データ型は、Flows セクションに対応する送信先を持っている必要があります。たとえば、カスタムログ、ETW ログ、およびシステムログを CloudWatch Logs に送信するには、(`CustomLogs`, `ETW`, `SystemEventLog`), `CloudWatchLogs` を Flows セクションに追加します。

Warning

無効なブロックを追加すると、フローがブロックされます。たとえば、ディスクメトリクス のステップを追加したが、インスタンスにディスクがない場合は、フローのすべてのステップがブロックされます。

同じログファイルを複数の宛先に送信できます。たとえば、アプリケーション ログを `CloudWatchLogs` セクションで定義付けた 2 つの送信先に送信するには、`ApplicationEventLog`, (`CloudWatchLogs`, `CloudWatchLogs2`) を Flows セクションに追加します。

フロー制御を設定するには

1. `AWS.EC2.Windows.CloudWatch.json` ファイルで、「Flows」セクションを見つけます。

```
"Flows": {
  "Flows": [
    "PerformanceCounter,CloudWatch",
```

```
"(PerformanceCounter,PerformanceCounter2), CloudWatch2",  
"(CustomLogs, ETW, SystemEventLog),CloudWatchLogs",  
"CustomLogs, CloudWatchLogs2",  
"ApplicationEventLog,(CloudWatchLogs, CloudWatchLogs2)"  
]  
}
```

2. Flows には、アップロードされる各データ型 (たとえば、ApplicationEventLog) とその送信先 (たとえば、CloudWatchLogs) を追加します。

ステップ 5: JSON コンテンツを保存する

JSON ファイルの編集はこれで完了です。ファイルを保存し、ファイルコンテンツをテキストエディタ内の別のウィンドウに貼り付けます。ファイルコンテンツは、この手順の後のステップで必要になります。

Systems Manager 用 IAM ロールを作成する

インスタンスの認証情報の IAM ロールは、Systems Manager Run Command の実行時に必要になります。このロールにより、Systems Manager はインスタンス上でアクションを実行できます。詳細については、AWS Systems Manager ユーザーガイドの「[Systems Manager セキュリティロールの設定](#)」を参照してください。既存のインスタンスへの IAM ロールのアタッチについては、「Amazon EC2 ユーザーガイド」の「[Attaching an IAM Role to an Instance](#)」を参照してください。

Systems Manager の前提条件を確認する

Systems Manager Run Command を使用して CloudWatch Logs との統合を設定する前に、インスタンスが最低要件を満たしていることを確認してください。詳細については、AWS Systems Manager ユーザーガイドの「[Systems Manager の前提条件](#)」を参照してください。

インターネットアクセスを確認する

CloudWatch にログとイベントのデータを送信するには、Amazon EC2 Windows Server のインスタンスとマネージドインスタンスにアウトバウンドのインターネットアクセスが必要です。インターネットアクセスの詳しい設定方法については、Amazon VPC ユーザーガイドの「[インターネットゲートウェイ](#)」を参照してください。

Systems Manager Run Command を使用して CloudWatch Logs を有効化する

Run Command では、インスタンスの設定をオンデマンドで管理できます。Systems Manager ドキュメントを指定してパラメータを指定し、1 つ以上のインスタンスでコマンドを実行します。イン

スタンスの SSM エージェントは、コマンドを処理し、指定されたとおりにインスタンスを設定します。

Run Command を使用して CloudWatch Logs との統合を設定するには

1. Amazon EC2 コンソール (<https://console.aws.amazon.com/ec2/>) を開きます。
2. SSM コンソール (<https://console.aws.amazon.com/systems-manager/>) を開きます。
3. ナビゲーションペインで、[Run Command] を選択します。
4. [Run a command] を選択します。
5. [Command document] で、[AWS-ConfigureCloudWatch] を選択します。
6. [Target instances (ターゲットインスタンス)] で、CloudWatch Logs と統合するインスタンスを選択します。このリストに表示されていないインスタンスは、Run Command 用に設定されていない場合があります。詳細については、「Amazon EC2 ユーザーガイド」の「[Systems Manager Prerequisites](#)」を参照してください。
7. [Status] で、[Enabled] を選択します。
8. [Properties] で、前のタスクで作成した JSON の内容をコピーして貼り付けます。
9. 残りのオプションフィールドを入力し、[Run] を選択します。

次の手順を使用して、Amazon EC2 コンソールでコマンドの実行結果を表示します。

コンソールでコマンド出力を表示するには

1. コマンドを選択します。
2. [Output] タブを選択します。
3. [View Output] を選択します。コマンド出力ページには、コマンドの実行結果が表示されます。

クイックスタート: Windows Server 2012 および Windows Server 2008 を実行している Amazon EC2 インスタンスで、CloudWatch Logs へのログの送信を有効にする

 Tip

CloudWatch には、EC2 インスタンスとオンプレミスサーバーからログとメトリクスの両方を収集できる新しい統合エージェントが含まれています。新しい統合 CloudWatch エージェ

ントを使用することをお勧めします。詳細については、「[CloudWatch Logs の開始方法](#)」を参照してください。

このセクションの残りの部分では、古い CloudWatch Logs エージェントの使用について説明します。

Windows Server 2012 および Windows Server 2008 を実行している Amazon EC2 インスタンスで、CloudWatch Logs へのログの送信を有効にする

Windows Server 2012 および Windows Server 2008 を実行しているインスタンスで、CloudWatch Logs へのログの送信を有効にするには、以下のステップを実行します。

サンプル設定ファイルをダウンロードする

サンプル JSON ファイル ([AWS.EC2.Windows.CloudWatch.json](#)) をコンピュータにダウンロードします。このファイルは以降のステップで編集します。

CloudWatch の JSON ファイルを設定する

CloudWatch に送信するログを決めるには、JSON 設定ファイルで選択して指定します。このファイルを作成し、項目を選択して指定するプロセスは、完了までに 30 分以上かかる場合があります。このタスクを 1 回完了したら、すべてのインスタンスで設定ファイルを再利用できます。

ステップ

- [ステップ 1: CloudWatch Logs を有効化する](#)
- [ステップ 2: CloudWatch 設定を構成する](#)
- [ステップ 3: 送信するデータを設定する](#)
- [ステップ 4: フロー制御を設定する](#)

ステップ 1: CloudWatch Logs を有効化する

JSON ファイルの先頭で、IsEnabled の「false」を「true」に変更します。

```
"IsEnabled": true,
```

ステップ 2: CloudWatch 設定を構成する

認証情報、リージョン、ロググループ名、およびログストリーム名前空間を指定します。これにより、インスタンスがログデータを CloudWatch Logs に送信できます。同じログデータを複数の異なる

る宛先に送信する場合は、一意の ID (たとえば「CloudWatchLogs2」および「CloudWatchLogs3」) および各 ID に異なるリージョンを付加してセクションを追加できます。

ログデータを CloudWatch Logs に送信するように設定するには

1. JSON ファイルで、CloudWatchLogs セクションを見つけます。

```
{
  "Id": "CloudWatchLogs",
  "FullName":
  "AWS.EC2.Windows.CloudWatch.CloudWatchLogsOutput,AWS.EC2.Windows.CloudWatch",
  "Parameters": {
    "AccessKey": "",
    "SecretKey": "",
    "Region": "us-east-1",
    "LogGroup": "Default-Log-Group",
    "LogStream": "{instance_id}"
  }
},
```

2. [AccessKey] および [SecretKey] フィールドは空白のままにしておきます。IAM ロールを使用して認証情報を設定します。
3. Region には、ログデータを送信するリージョンを入力します (たとえば、us-east-2)。
4. LogGroup には、ロググループの名前を入力します。この名前は、CloudWatch コンソールの [Log Groups (ロググループ)] 画面に表示されます。
5. LogStream には、送信先のログストリームを入力します。この名前は、CloudWatch コンソールの [Log Groups (ロググループ)] > [Streams (ストリーム)] 画面に表示されます。

デフォルトの {instance_id} を使用する場合、ログストリーム名はこのインスタンスのインスタンス ID です。

存在しないログストリーム名を特定すると、CloudWatch Logs によってログストリームが自動的に作成されます。リテラル文字列、事前定義された変数 {instance_id}、{hostname}、{ip_address}、またはこれらの組み合わせを使用してログストリーム名を定義できます。

ステップ 3: 送信するデータを設定する

イベントログデータ、Event Tracing for Windows (ETW) データ、および他のログデータを CloudWatch Logs に送信できます。

Windows アプリケーションイベントログデータを CloudWatch Logs に送信するには

1. JSON ファイルで、ApplicationEventLog セクションを見つけます。

```
{
  "Id": "ApplicationEventLog",
  "FullName":
"AWS.EC2.Windows.CloudWatch.EventLog.EventLogInputComponent,AWS.EC2.Windows.CloudWatch",
  "Parameters": {
    "LogName": "Application",
    "Levels": "1"
  }
},
```

2. Levels には、アップロードするメッセージのタイプを指定します。次のいずれかの値を指定できます。

- 1 - エラーメッセージだけをアップロードします。
- 2 - 警告メッセージだけをアップロードします。
- 4 - 情報メッセージだけをアップロードします。

値を組み合わせて複数のタイプのメッセージを含めることができます。たとえば、値 3 はエラーメッセージ (1) と警告メッセージ (2) をアップロードします。値 7 は、エラーメッセージ (1)、警告メッセージ (2)、情報メッセージ (4) をアップロードします。

セキュリティログデータを CloudWatch Logs に送信するには

1. JSON ファイルで、SecurityEventLog セクションを見つけます。

```
{
  "Id": "SecurityEventLog",
  "FullName":
"AWS.EC2.Windows.CloudWatch.EventLog.EventLogInputComponent,AWS.EC2.Windows.CloudWatch",
  "Parameters": {
    "LogName": "Security",
    "Levels": "7"
  }
},
```

2. Levels には、7 と入力してすべてのメッセージをアップロードします。

システムイベントログデータを CloudWatch Logs に送信するには

1. JSON ファイルで、SystemEventLog セクションを見つけます。

```
{
  "Id": "SystemEventLog",
  "FullName":
  "AWS.EC2.Windows.CloudWatch.EventLog.EventLogInputComponent,AWS.EC2.Windows.CloudWatch",
  "Parameters": {
    "LogName": "System",
    "Levels": "7"
  }
},
```

2. Levels には、アップロードするメッセージのタイプを指定します。次のいずれかの値を指定できます。

- 1 - エラーメッセージだけをアップロードします。
- 2 - 警告メッセージだけをアップロードします。
- 4 - 情報メッセージだけをアップロードします。

値を組み合わせて複数のタイプのメッセージを含めることができます。たとえば、値 **3** はエラーメッセージ (1) と警告メッセージ (2) をアップロードします。値 **7** は、エラーメッセージ (1)、警告メッセージ (2)、情報メッセージ (4) をアップロードします。

他の種類のイベントログデータを CloudWatch Logs に送信するには

1. JSON ファイルに、新しいセクションを追加します。各セクションには固有の Id が必要です。

```
{
  "Id": "Id-name",
  "FullName":
  "AWS.EC2.Windows.CloudWatch.EventLog.EventLogInputComponent,AWS.EC2.Windows.CloudWatch",
  "Parameters": {
    "LogName": "Log-name",
    "Levels": "7"
  }
},
```

2. Id には、アップロードするログの名前を入力します (たとえば、**WindowsBackup**)。

3. LogName には、アップロードするログの名前を入力します。ログの名前を次のように確認できます。
 - a. イベントビューワーを開きます。
 - b. ナビゲーションペインで、[Applications and Services Logs] を選択します。
 - c. ログに移動し、[Actions]、[Properties] を選択します。
4. Levels には、アップロードするメッセージのタイプを指定します。次のいずれかの値を指定できます。
 - 1 - エラーメッセージだけをアップロードします。
 - 2 - 警告メッセージだけをアップロードします。
 - 4 - 情報メッセージだけをアップロードします。

値を組み合わせることで複数のタイプのメッセージを含めることができます。たとえば、値 3 はエラーメッセージ (1) と警告メッセージ (2) をアップロードします。値 7 は、エラーメッセージ (1)、警告メッセージ (2)、情報メッセージ (4) をアップロードします。

イベントトレース (Windows) データを CloudWatch Logs に送信するには

ETW (Event Tracing for Windows) には、アプリケーションがログを書き込むことができる効率的できめ細かいログ記録メカニズムが用意されています。各 ETW は、ログ記録セッションを開始および停止できるセッションマネージャにより制御されます。各セッションには、プロバイダーと 1 つ以上のコンシューマーが存在します。

1. JSON ファイルで、ETW セクションを見つけます。

```
{
  "Id": "ETW",
  "FullName":
  "AWS.EC2.Windows.CloudWatch.EventLog.EventLogInputComponent,AWS.EC2.Windows.CloudWatch",
  "Parameters": {
    "LogName": "Microsoft-Windows-WinINet/Analytic",
    "Levels": "7"
  }
},
```

2. LogName には、アップロードするログの名前を入力します。

3. Levels には、アップロードするメッセージのタイプを指定します。次のいずれかの値を指定できます。
- 1 - エラーメッセージだけをアップロードします。
 - 2 - 警告メッセージだけをアップロードします。
 - 4 - 情報メッセージだけをアップロードします。

値を組み合わせて複数のタイプのメッセージを含めることができます。たとえば、値 3 はエラーメッセージ (1) と警告メッセージ (2) をアップロードします。値 7 は、エラーメッセージ (1)、警告メッセージ (2)、情報メッセージ (4) をアップロードします。

カスタムログ (テキストベースのログファイル) を CloudWatch Logs に送信するには

1. JSON ファイルで、CustomLogs セクションを見つけます。

```
{
  "Id": "CustomLogs",
  "FullName":
"AWS.EC2.Windows.CloudWatch.CustomLog.CustomLogInputComponent,AWS.EC2.Windows.CloudWatch",
  "Parameters": {
    "LogDirectoryPath": "C:\\\\CustomLogs\\\\",
    "TimestampFormat": "MM/dd/yyyy HH:mm:ss",
    "Encoding": "UTF-8",
    "Filter": "",
    "CultureName": "en-US",
    "TimeZoneKind": "Local",
    "LineCount": "5"
  }
},
```

2. LogDirectoryPath には、ログがインスタンスに格納されるパスを入力します。
3. TimestampFormat には、使用するタイムスタンプ形式を入力します。サポートされる値の詳細については、MSDN の「[カスタムの日付と時刻の書式指定文字列](#)」を参照してください。

Important

ソースログファイルには、各ログ行の先頭にタイムスタンプがあり、タイムスタンプの後にスペースがある必要があります。

4. Encoding には、使用するファイルエンコード (UTF-8 など) を入力します。サポートされる値の詳細については、MSDN の「[Encoding クラス](#)」を参照してください。

 Note

表示名ではなく、エンコード名を使用します。

5. (オプション) Filter には、ログファイル名のプレフィックスを入力します。すべてのファイルをモニタリングするには、このパラメータを空白のままにします。サポートされる値の詳細については、MSDN の「[FileSystemWatcherFilter プロパティ](#)」を参照してください。
6. (オプション) CultureName には、タイムスタンプが記録されているロケールを入力します。CultureName が空の場合、Windows インスタンスにより現在使用されているのと同じロケールがデフォルトになります。サポートされる値の詳細については、MSDN の「[Product Behavior](#)」トピックの表で、Language tag 列を参照してください。

 Note

div、div-MV、hu、および hu-HU 値は、サポートされていません。

7. (オプション) TimeZoneKind には、Local または UTC を入力します。これを設定すると、ログのタイムスタンプにタイムゾーン情報が含まれていない場合にタイムゾーン情報を提供できます。このパラメータが空になっていて、タイムスタンプにタイムゾーン情報が含まれていない場合、CloudWatch Logs ではデフォルトでローカルタイムゾーンが使用されます。タイムスタンプに既にタイムゾーン情報が含まれている場合、このパラメータは無視されます。
8. (オプション) LineCount には、ログファイルを識別するためのヘッダーの行数を入力します。たとえば、IIS のログファイルのヘッダーはほぼ同じです。「5」と入力すると、ログファイルのヘッダーの最初の 3 行が読み取られ、ログファイルを識別できます。IIS ログファイルで、3 番目の行は日時のスタンプですが、タイムスタンプは常にログファイル間で異なるという保証はありません。そのため、ログファイルの一意のフィンガープリントを作成するには、実際のログデータの少なくとも 1 行を含めることをお勧めします。

IIS ログデータを CloudWatch Logs に送信するには

1. JSON ファイルで、IISLog セクションを見つけます。

```
{  
  "Id": "IISLogs",
```

```
"FullName":  
"AWS.EC2.Windows.CloudWatch.CustomLog.CustomLogInputComponent,AWS.EC2.Windows.CloudWatch",  
"Parameters": {  
  "LogDirectoryPath": "C:\\inetpub\\logs\\LogFiles\\W3SVC1",  
  "TimestampFormat": "yyyy-MM-dd HH:mm:ss",  
  "Encoding": "UTF-8",  
  "Filter": "",  
  "CultureName": "en-US",  
  "TimeZoneKind": "UTC",  
  "LineCount": "5"  
}  
},
```

2. LogDirectoryPath には、個々のサイト (C:\inetpub\logs\LogFiles\W3SVCn など) の IIS ログが格納されているフォルダを入力します。

 Note

W3C ログ形式のみサポートされます。IIS、NCSA、カスタム形式はサポートされません。

3. TimestampFormat には、使用するタイムスタンプ形式を入力します。サポートされる値の詳細については、MSDN の「[カスタムの日付と時刻の書式指定文字列](#)」を参照してください。
4. Encoding には、使用するファイルエンコード (UTF-8 など) を入力します。サポートされる値の詳細については、MSDN の「[Encoding クラス](#)」を参照してください。

 Note

表示名ではなく、エンコード名を使用します。

5. (オプション) Filter には、ログファイル名のプレフィックスを入力します。すべてのファイルをモニタリングするには、このパラメータを空白のままにします。サポートされる値の詳細については、MSDN の「[FileSystemWatcherFilter プロパティ](#)」を参照してください。
6. (オプション) CultureName には、タイムスタンプが記録されているロケールを入力します。CultureName が空の場合、Windows インスタンスにより現在使用されているのと同じロケールがデフォルトになります。サポートされる値の詳細については、MSDN の「[Product Behavior](#)」トピックの表で、Language tag 列を参照してください。

 Note

div、div-MV、hu、および hu-HU 値は、サポートされていません。

7. (オプション) `TimeZoneKind` には、`Local` または `UTC` を入力します。これを設定すると、ログのタイムスタンプにタイムゾーン情報が含まれていない場合にタイムゾーン情報を提供できます。このパラメータが空になっていて、タイムスタンプにタイムゾーン情報が含まれていない場合、CloudWatch Logs ではデフォルトでローカルタイムゾーンが使用されます。タイムスタンプに既にタイムゾーン情報が含まれている場合、このパラメータは無視されます。
8. (オプション) `LineCount` には、ログファイルを識別するためのヘッダーの行数を入力します。たとえば、IIS のログファイルのヘッダーはほぼ同じです。「5」と入力すると、ログファイルのヘッダーの最初の 5 行が読み取られ、ログファイルを識別できます。IIS ログファイルで、3 番目の行は日時のスタンプですが、タイムスタンプは常にログファイル間で異なるという保証はありません。そのため、ログファイルの一意のフィンガープリントを作成するには、実際のログデータの少なくとも 1 行を含めることをお勧めします。

ステップ 4: フロー制御を設定する

各データ型は、Flows セクションに対応する送信先を持っている必要があります。たとえば、カスタムログ、ETW ログ、およびシステムログを CloudWatch Logs に送信するには、(`CustomLogs`, `ETW`, `SystemEventLog`), `CloudWatchLogs` を Flows セクションに追加します。

 Warning

無効なブロックを追加すると、フローがブロックされます。たとえば、ディスクメトリクスのステップを追加したが、インスタンスにディスクがない場合は、フローのすべてのステップがブロックされます。

同じログファイルを複数の宛先に送信できます。たとえば、アプリケーションログを `CloudWatchLogs` セクションで定義付けた 2 つの送信先に送信するには、`ApplicationEventLog`, (`CloudWatchLogs`, `CloudWatchLogs2`) を Flows セクションに追加します。

フロー制御を設定するには

1. AWS.EC2.Windows.CloudWatch.json ファイルで、「Flows」セクションを見つけます。

```
"Flows": {  
  "Flows": [  
    "PerformanceCounter,CloudWatch",  
    "(PerformanceCounter,PerformanceCounter2), CloudWatch2",  
    "(CustomLogs, ETW, SystemEventLog),CloudWatchLogs",  
    "CustomLogs, CloudWatchLogs2",  
    "ApplicationEventLog,(CloudWatchLogs, CloudWatchLogs2)"  
  ]  
}
```

2. Flows には、アップロードされる各データ型 (たとえば、ApplicationEventLog) とその送信先 (たとえば、CloudWatchLogs) を追加します。

JSON ファイルの編集はこれで完了です。これは、後のステップで使用します。

エージェントを起動する

Windows Server 2012 または Windows Server 2008 を実行している Amazon EC2 インスタンスが、CloudWatch Logs にログを送信できるようにするには、EC2 Config サービス (EC2Config.exe) を使用します。インスタンスには EC2Config 4.0 以降が必要であり、この手順を使用できます。

EC2Config 4.x を使用して CloudWatch を設定するには

1. この手順で前に編集した AWS.EC2.Windows.CloudWatch.json ファイルのエンコーディングを確認します。BOM のない UTF-8 エンコーディングのみがサポートされています。次に、Windows Server 2008 - 2012 R2 インスタンスで、C:\Program Files\Amazon\SSM\Plugins\awsCloudWatch\ フォルダにファイルを保存します。
2. Windows サービスのコントロールパネルを使用するか、次の PowerShell コマンドを送信して、SSM エージェント (AmazonSSMAgent.exe) を開始または再起動します。

```
PS C:\> Restart-Service AmazonSSMAgent
```

SSM エージェントは、再起動後に設定ファイルを検出し、CloudWatch 統合のためにインスタンスを設定します。ローカル設定ファイルのパラメータと設定を変更する場合は、変更を反映するために

SSM エージェントを再起動する必要があります。インスタンスで CloudWatch 統合を無効にする場合は、IsEnabled を false に変更して、設定ファイルで変更を保存します。

CloudWatch Logs エージェントの状態を報告する

EC2 インスタンスでの CloudWatch Logs エージェントのステータスをレポートするには、以下の手順を使用します。

エージェントのステータスをレポートするには

1. EC2 インスタンスに接続します。詳細については、「Amazon EC2 ユーザーガイド」の「[Connect to Your Instance](#)」を参照してください。

接続問題の詳細については、「Amazon EC2 ユーザーガイド」の「[Amazon EC2 Linux インスタンスへの接続に関する問題のトラブルシューティング](#)」を参照してください。

2. コマンドプロンプトで、次のコマンドを入力します。

```
sudo service awslogs status
```

Amazon Linux 2 を実行している場合は、次のコマンドを入力します。

```
sudo service awslogsd status
```

3. /var/log/awslogs.log ファイルで CloudWatch Logs エージェントのエラー、警告、問題を確認してください。

CloudWatch Logs エージェントを起動する

EC2 インスタンスの CloudWatch Logs エージェントがインストール後自動的に開始しなかった場合、またはエージェントを停止した場合、以下の手順を使用してエージェントを開始できます。

エージェントを開始するには

1. EC2 インスタンスに接続します。詳細については、「Amazon EC2 ユーザーガイド」の「[Connect to Your Instance](#)」を参照してください。

接続問題の詳細については、「Amazon EC2 ユーザーガイド」の「[Amazon EC2 Linux インスタンスへの接続に関する問題のトラブルシューティング](#)」を参照してください。

2. コマンドプロンプトで、次のコマンドを入力します。

```
sudo service awslogs start
```

Amazon Linux 2 を実行している場合は、次のコマンドを入力します。

```
sudo service awslogsd start
```

CloudWatch Logs エージェントを停止する

EC2 インスタンスで CloudWatch Logs エージェントを停止するには、以下の手順を使用します。

エージェントを停止するには

1. EC2 インスタンスに接続します。詳細については、「Amazon EC2 ユーザーガイド」の「[Connect to Your Instance](#)」を参照してください。

接続問題の詳細については、「Amazon EC2 ユーザーガイド」「[Amazon EC2 Linux インスタンスへの接続に関する問題のトラブルシューティング](#)」を参照してください。

2. コマンドプロンプトで、次のコマンドを入力します。

```
sudo service awslogs stop
```

Amazon Linux 2 を実行している場合は、次のコマンドを入力します。

```
sudo service awslogsd stop
```

CloudWatch Logs エージェントのリファレンス

Important

このセクションは、廃止された古い CloudWatch Logs エージェントを使用している方のためのリファレンスです。インスタンスメタデータサービスのバージョン 2 (IMDSv2) を使用している場合は、新しい統合 CloudWatch エージェントを使用する必要があります。ただし、IMDSv2 を使用していない場合でも、古い CloudWatch Logs エージェントではなく、新しい統合 CloudWatch エージェントを使用することを強くお勧めします。新しい統合エージェントの詳細については、「[Collecting metrics and logs from Amazon EC2 instance](#)」

[and on-premises servers with the CloudWatch agent](#)」を参照してください。廃止された CloudWatch Logs エージェントから統合エージェントへの移行については、「[Create the CloudWatch agent configuration file with the wizard](#)」を参照してください。

CloudWatch Logs エージェントは、Amazon EC2 インスタンスから CloudWatch Logs にログデータを自動的に送信する方法を提供します。エージェントには以下のコンポーネントが含まれます。

- CloudWatch Logs にログデータをプッシュ AWS CLI する へのプラグイン。
- データを CloudWatch Logs にプッシュするプロセスを開始するスクリプト (デーモン)。
- デーモンが常に実行中であることを確認する cron ジョブ。

エージェント設定ファイル

CloudWatch Logs エージェント設定ファイルには、CloudWatch Logs エージェントに必要な情報が記述されています。エージェント設定ファイルの [general] セクションは、すべてログストリームに適用する一般的な設定を定義します。[logstream] セクションは、リモートなログストリームにローカルファイルを送信するために必要な情報を定義します。複数の [logstream] セクションを持つことができますが、設定ファイル内にそれぞれ [logstream1]、[logstream2] などの一意の名前を持つ必要があります。ログファイルのデータの最初の行とともにある [logstream] 値は、ログファイルの ID を定義します。

```
[general]
state_file = value
logging_config_file = value
use_gzip_http_content_encoding = [true | false]

[logstream1]
log_group_name = value
log_stream_name = value
datetime_format = value
time_zone = [LOCAL|UTC]
file = value
file_fingerprint_lines = integer | integer-integer
multi_line_start_pattern = regex | {datetime_format}
initial_position = [start_of_file | end_of_file]
encoding = [ascii|utf_8|..]
buffer_duration = integer
batch_count = integer
```



```
batch_size = integer

[logstream2]

...
```

state_file

状態ファイルをどこに保存するかを指定します。

logging_config_file

(オプション) エージェントのログ config ファイルの場所を指定します。ここでエージェントのログ config ファイルを指定しない場合は、デフォルトファイル `awslogs.conf` が使用されます。スクリプトでエージェントをインストールした場合、デフォルトのファイルの場所は `/var/awslogs/etc/awslogs.conf` です。rpm でエージェントをインストールした場合は、`/etc/awslogs/awslogs.conf` です。このファイルは、Python の設定ファイル形式 (<https://docs.python.org/2/library/logging.config.html#logging-config-fileformat>) です。以下の名前のロガーはカスタマイズできます。

```
cwlogs.push
cwlogs.push.reader
cwlogs.push.publisher
cwlogs.push.event
cwlogs.push.batch
cwlogs.push.stream
cwlogs.push.watcher
```

以下のサンプルは、デフォルトの値が INFO であるリーダーとパブリッシャーのレベルを WARNING に変更します。

```
[loggers]
keys=root,cwlogs,reader,publisher

[handlers]
keys=consoleHandler

[formatters]
keys=simpleFormatter

[logger_root]
level=INFO
handlers=consoleHandler
```

```
[logger_cwlogs]
level=INFO
handlers=consoleHandler
qualname=cwlogs.push
propagate=0

[logger_reader]
level=WARNING
handlers=consoleHandler
qualname=cwlogs.push.reader
propagate=0

[logger_publisher]
level=WARNING
handlers=consoleHandler
qualname=cwlogs.push.publisher
propagate=0

[handler_consoleHandler]
class=logging.StreamHandler
level=INFO
formatter=simpleFormatter
args=(sys.stderr,)

[formatter_simpleFormatter]
format=%(asctime)s - %(name)s - %(levelname)s - %(process)d - %(threadName)s -
%(message)s
```

use_gzip_http_content_encoding

true (デフォルト) に設定すると、CloudWatch Logs への圧縮されたペイロードの送信に対して、gzip による HTTP コンテンツのエンコードが有効になります。これにより、CPU の使用率が減り、NetworkOut が少なくなり、Put のレイテンシーが短くなります。この機能を無効にするには、CloudWatch Logs エージェント設定ファイルの [general] (全般) セクションに [use_gzip_http_content_encoding = false] を追加してから、エージェントを再起動します。

Note

この設定は awscli-cwlogs バージョン 1.3.3 以降でのみ使用できます。

log_group_name

送信先ロググループを指定します。ロググループが存在しない場合には、自動的に作成されます。ロググループの名前は 1~512 文字で指定します。ここで使えるのは、a~z、A~Z、0~9、"_" (アンダーバー)、"-" (ハイフン)、"/" (スラッシュ) および "." (ピリオド) です。

log_stream_name

送信先ログストリームを指定します。リテラル文字列、定義済み変数 ({instance_id}、{hostname}、{ip_address})、またはこれらの組み合わせを使用して、ログストリーム名を定義できます。ログストリームが存在しない場合には、自動的に作成されます。

datetime_format

ログからタイムスタンプを入手する方法を指定します。タイムスタンプはログイベントを取得し、メトリクスを生成するために使用されます。現在の時刻は、[datetime_format] が提供されていない場合に各ログイベントで使用されます。提供された [datetime_format] の値がそのログメッセージに対して無効の場合は、適切に解析されたタイムスタンプを持つ最後のログイベントのタイムスタンプが使用されます。以前のログイベントが存在しない場合は、現在の時刻が使用されます。

一般的な datetime_format コードは次のとおりです。Python がサポートする datetime_format コード (datetime.strptime()) も使用できます。タイムゾーンオフセット (%z) もサポートされています。ただし、Python 3.2 までは、コロン (:) のない [+ -] HHMM はサポートされていません。詳細については、「[strftime\(\) および strptime\(\) Behavior](#)」を参照してください。

[%y]: ゼロ詰め 10 進数での年 (世紀なし) です。00, 01, ..., 99

%Y: 10 進数での年 (世紀あり) です。1970、1988、2001、2013

%b: ロケールの省略名称での月です。Jan、Feb ... Dec (en_US) ;

%B: ロケールの正式名称での月です。January、February...December (en_US) ;

%m: ゼロ詰め 10 進数での月です。01, 02, ..., 12

%d: ゼロ詰め 10 進数での日です。01, 02, ..., 31

%H: ゼロ詰め 10 進数での時 (24 時間形式の時計) です。00, 01, ..., 23

%I: ゼロ詰め 10 進数での時 (12 時間形式の時計) です。01, 02, ..., 12

%p: ロケールで AM または PM に相当するものです。

%M: ゼロ詰め 10 進数での分です。00, 01, ..., 59

%S: ゼロ詰め 10 進数での秒です。00, 01, ..., 59

%f: 左ゼロ詰め 10 進数でのマイクロ秒です。000000, ..., 999999

%z: +HHMM または -HHMM 形式の UTC オフセットです。+0000、-0400、+1030

形式の例:

Syslog: '%b %d %H:%M:%S', e.g. Jan 23 20:59:29

Log4j: '%d %b %Y %H:%M:%S', e.g. 24 Jan 2014 05:00:00

ISO8601: '%Y-%m-%dT%H:%M:%S%z', e.g. 2014-02-20T05:20:20+0000

time_zone

ログイベントのタイムスタンプのタイムゾーンを指定します。サポートされる 2 つの値は UTC および LOCAL です。デフォルトは LOCAL です。タイムゾーンが [datetime_format] に基づいて推定できない場合に使用されます。

および

CloudWatch Logs にプッシュするログファイルを指定します。ファイルは、特定のファイルまたは複数のファイルを指すことができます (/var/log/system.log* のようにワイルドカードを使用)。ファイルの変更時間に基づいて、最新のファイルのみが CloudWatch Logs にプッシュされます。access_log.2014-06-01-01 と access_log.2014-06-01-02 など同じ形式の一連のファイルを指定するにはワイルドカードの使用をお勧めします。ただし、access_log_80 と access_log_443 のように複数の種類のファイルには使用しないでください。複数の種類のファイルを指定するには、設定ファイルに別のストリームログのエントリを追加して、各種類のログファイルが異なるログストリームに行くようにします。圧縮ファイルはサポートされていません。

file_fingerprint_lines

ファイルを識別するための行範囲を指定します。有効な値は「1」「2-5」のように単一の数字またはハイフンで区切られた 2 つの数字です。デフォルト値は「1」です。最初の行を使用してフィンガープリントを計算します。指定された行がすべて存在しない限り、フィンガープリント行は CloudWatch Logs に送信されません。

multi_line_start_pattern

ログメッセージの開始を識別するパターンを指定します。ログメッセージは、パターンに一致する 1 行と、それに続くパターンに一致しない行で構成されます。有効な値は正規表現または

{datetime_format} です。{datetime_format} を使用する場合は、datetime_format オプションを指定する必要があります。デフォルト値は「^[^\s]」です。よって、空白文字以外の文字で始まる行で前のログメッセージを終了し、新しいログメッセージを開始します。

initial_position

データの読み出しをどこから開始するかを指定します (start_of_file または end_of_file)。デフォルトは start_of_file です。そのログストリームに保持されている状態がない場合にのみ使用されます。

encoding

ファイルを正しく読み込むことができるように、ログファイルのエンコードを指定します。デフォルトは utf_8 です。Python の codecs.decode() がサポートするエンコードを使用できます。

Warning

正しくないエンコードを指定すると、デコードできない文字がそのほかの文字に置き換えられるため、データ損失が生じる可能性があります。

一般的なエンコードを次に示します。

ascii, big5, big5hkscs, cp037, cp424, cp437, cp500, cp720, cp737, cp775, cp850, cp852, cp855, cp856, cp857, cp858, cp860, cp861, cp862, cp863, cp864, cp865, cp866, cp869, cp874, cp875, cp932, cp949, cp950, cp1006, cp1026, cp1140, cp1250, cp1251, cp1252, cp1253, cp1254, cp1255, cp1256, cp1257, cp1258, euc_jp, euc_jis_2004, euc_jisx0213, euc_kr, gb2312, gbk, gb18030, hz, iso2022_jp, iso2022_jp_1, iso2022_jp_2, iso2022_jp_2004, iso2022_jp_3, iso2022_jp_ext, iso2022_kr, latin_1, iso8859_2, iso8859_3, iso8859_4, iso8859_5, iso8859_6, iso8859_7, iso8859_8, iso8859_9, iso8859_10, iso8859_13, iso8859_14, iso8859_15, iso8859_16, johab, koi8_r, koi8_u, mac_cyrillic, mac_greek, mac_iceland, mac_latin2, mac_roman, mac_turkish, ptcp154, shift_jis, shift_jis_2004, shift_jisx0213, utf_32, utf_32_be, utf_32_le, utf_16, utf_16_be, utf_16_le, utf_7, utf_8, utf_8_sig

buffer_duration

ログイベントのバッチ期間を指定します。最小値は 5000ms で、デフォルト値は 5000ms です。

batch_count

バッチのログイベントの最大値を 10000 までの値で指定します。デフォルト値は 10000 です。

batch_size

バッチのログイベントの最大値を 1048576 バイトまでのバイト値で指定します。デフォルト値は 1048576 バイトです。このサイズは、UTF-8 のすべてのイベントメッセージの合計に各ログイベントにつき 26 バイトを加算して計算されます。

HTTP プロキシでの CloudWatch Logs エージェントの使用

CloudWatch Logs エージェントは HTTP プロキシで使用できます。

Note

HTTP プロキシは awslogs-agent-setup.py バージョン 1.3.8 以降でサポートされています。

HTTP プロキシで CloudWatch Logs エージェントを使用するには

1. 次のいずれかを行います。
 - a. CloudWatch Logs エージェントを新たにインストールする場合は、以下のコマンドを実行します。

```
curl https://s3.amazonaws.com/aws-cloudwatch/downloads/latest/awslogs-agent-setup.py -O
```

```
sudo python awslogs-agent-setup.py --region us-east-1 --http-proxy http://your/proxy --https-proxy http://your/proxy --no-proxy 169.254.169.254
```

EC2 インスタンスで Amazon EC2 メタデータサービスへのアクセスを維持するには、`[--no-proxy 169.254.169.254]`(推奨) を使用します。詳細については、「Amazon EC2 ユーザーガイド」の「[Instance Metadata and User Data](#)」を参照してください。

http-proxy および https-proxy の値で、URL 全体を指定します。

- b. CloudWatch Logs エージェントが既にインストールされている場合は、`/var/awslogs/etc/proxy.conf` 編集し、プロキシを追加します。

```
HTTP_PROXY=  
HTTPS_PROXY=  
NO_PROXY=
```

2. エージェントを再起動して、変更を有効にします。

```
sudo service awslogs restart
```

Amazon Linux 2 を使用している場合は、次のコマンドを使用してエージェントを再起動します。

```
sudo service awslogsd restart
```

CloudWatch Logs エージェント設定ファイルのコンパートメント化

awslogs-agent-setup.py バージョン 1.3.8 以降と awscli-cwlogs 1.3.3 以降を使用している場合は、/var/awslogs/etc/config/ に追加の設定ファイルを作成することで、さまざまなコンポーネントのストリーム設定をそれぞれ個別にインポートできます。CloudWatch Logs エージェントが起動すると、これらの追加の設定ファイルにストリーム設定が追加されます。[general] セクションの設定プロパティはメインの設定ファイル (/var/awslogs/etc/awslogs.conf) で定義する必要があり、/var/awslogs/etc/config/ にある追加の設定ファイルで定義しても無視されます。

rpm でエージェントをインストールしたため /var/awslogs/etc/config/ ディレクトリがない場合は、代わりに /etc/awslogs/config/ ディレクトリを使用できます。

エージェントを再起動して、変更を有効にします。

```
sudo service awslogs restart
```

Amazon Linux 2 を使用している場合は、次のコマンドを使用してエージェントを再起動します。

```
sudo service awslogsd restart
```

CloudWatch Logs エージェントに関するよくある質問

どのようなファイルローテーションがサポートされていますか。

次のファイルローテーション機能がサポートされています。

- 既存のファイルを数字サフィックスをつけた名前に変更し、その後、元の名前の空のログファイルを作成し直します。たとえば、`/var/log/syslog.log` が `/var/log/syslog.log.1` という名前に変更されます。`/var/log/syslog.log.1` が前回のローテーションにより既に存在する場合は、`/var/log/syslog.log.2` という名前に変更されます。
- 元のログファイルを、コピーを作成した後切り捨てます。たとえば、`/var/log/syslog.log` を `/var/log/syslog.log.1` にコピーし、`/var/log/syslog.log` を切り捨てます。この場合、データを損失する恐れがあるため、このファイルローテーション機能の使用にはご注意ください。
- 古いファイルと共通のパターンを持つ新しいファイルを作成します。たとえば `/var/log/syslog.log.2014-01-01` をそのまま残し、`/var/log/syslog.log.2014-01-02` を作成します。

ファイルのフィンガープリント (ソース ID) は、ログストリームキーとファイルのコンテンツの 1 行目をハッシュして計算されます。この動作をオーバーライドするには、`[file_fingerprint_lines]` オプションを使用できます。ファイルのローテーションが発生した場合、新しいファイルには新しいコンテンツがあり古いファイルにはコンテンツの追加がないと思われるため、エージェントは古いファイルの読み込みが完了した後は、新しいファイルをプッシュします。

使用しているエージェントのバージョンを確認する方法

セットアップスクリプトから CloudWatch Logs エージェントをインストールした場合、`[/var/awslogs/bin/awslogs-version.sh]` で使用しているエージェントのバージョンを確認することができます。エージェントのバージョンと主要な依存関係がプリントアウトされます。yum から CloudWatch Logs エージェントをインストールした場合には、`["yum info awslogs"]` と `["yum info aws-cli-plugin-cloudwatch-logs"]` で CloudWatch Logs エージェントとプラグインのバージョンを確認することができます。

ログのエントリは、どのようにログイベントに変換されるのですか。

ログイベントには 2 つのプロパティが含まれます。イベント発生時のタイムスタンプおよび生のログメッセージです。デフォルトでは、空白文字以外の文字で始まる行は、前のログメッセージがある場合はこれを終了して新しいログメッセージを開始します。この動作をオーバーライドするには、`[multi_line_start_pattern]` を使用します。パターンに一致する行が新しいログメッセージを開始します。パターンには正規表現または「`{datetime_format}`」を使用できます。例えば、それぞれのログメッセージの 1 行目が「`2014-01-02T13:13:01Z`」のようなタイムスタンプを持っている場合、`multi_line_start_pattern` は「`\d{4}-\d{2}-\d{2}T\d{2}:\d{2}:\d{2}Z`」と設定できます。設定を簡略化するために、`datetime_format` オプションが指定されている場合は「`{datetime_format}`」変数を使用できます。同じ例で、`datetime_format` が「`%Y-%m-%dT%H:%M:%S%z`」に設定されている場合、`multi_line_start_pattern` は「`{datetime_format}`」だけにできます。

現在の時刻は、`[datetime_format]` が提供されていない場合に各ログイベントで使用されます。提供された `[datetime_format]` がそのログメッセージに対して無効の場合は、適切に解析されたタイ

タイムスタンプを持つ最後のログイベントのタイムスタンプが使用されます。以前のログイベントが存在しない場合は、現在の時刻が使用されます。ログイベントが現在の時刻または前のログイベントの時刻にフォールバックした場合は、警告メッセージが記録されます。

タイムスタンプはログイベントを取得し、メトリクスを生成するために使用されます。誤った形式を指定した場合、ログイベントが取得できなくなり誤ったメトリクスが生成されます。

ログイベントはどのようにバッチされていますか。

次の条件のいずれかが満たされる場合、バッチがフルになり発行されます。

1. 最初のログイベントが追加されてから、[buffer_duration] の時間が経過した。
2. 累積されたログイベントの [batch_size] 未満ですが、新しいログイベントを追加すると [batch_size] を超過します。
3. ログイベント数は [batch_count] に達しました。
4. バッチのログイベントは 24 時間以上になりませんが、新しいログイベントを追加すると 24 時間の制約を超過します。

ログエントリ、ログイベント、またはバッチがスキップまたは切り捨てられるのはどのような原因がありますか。

PutLogEvents オペレーションの制約に従って、次の問題によりログイベントまたはバッチがスキップされる場合があります。

 Note

データがスキップされた場合、CloudWatch Logs エージェントはログに警告を書き込みます。

1. ログイベントのサイズが 256 KB を超過した場合、ログイベントは完全にスキップされます。
2. ログイベントのタイムスタンプが 2 時間以上未来の場合、ログイベントはスキップされます。
3. ログイベントのタイムスタンプが 14 日以上過去の場合、ログイベントはスキップされます。
4. ログイベントがロググループの保持期間よりも古い場合、バッチはすべてスキップされます。
5. 単一の PutLogEvents リクエストでログイベントのバッチが 24 時間実行されている場合、PutLogEvents オペレーションは失敗します。

エージェントを停止させた場合、データ損失や重複が発生しますか。

状態ファイルが使用可能であり、最後に実行されたときからファイルのローテーションが発生していなければ、発生しません。CloudWatch Logs エージェントは停止した場所から再開してログデータのプッシュを続行できます。

同一または異なるホストの異なるログデータを同じログストリームに指定できますか。

複数のログソースから単一のログストリームにデータを送信する設定はサポートされていません。

エージェントはどの API に呼び出しを作成しますか (またはどのアクションを IAM ポリシーに含める必要がありますか)?

CloudWatch Logs エージェントに

は、CreateLogGroup、CreateLogStream、DescribeLogStreams、DescribeLogGrooupd、Put
アクションを実行するためのアクセス権限が必要です。最新のエージェントを使用している場合
には、DescribeLogStreams は必要ありません。以下の IAM ポリシーの例を参照してくださ
い。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams",
        "logs:DescribeLogGroups",
        "logs:PutRetentionPolicy"
      ],
      "Resource": [
        "arn:aws:logs:*:*:*"
      ]
    }
  ]
}
```

CloudWatch Logs エージェントに自動的にロググループまたはログストリームを作成させたくありません。エージェントによるロググループとログストリームの再作成を禁止する方法を教えてください。

IAM ポリシーで、エージェントを次のオペレーション (DescribeLogStreams、PutLogEvents) のみに制限できます。

エージェントから CreateLogGroup および CreateLogStream 権限を取り消す前に、エージェントが使用するロググループとログストリームの両方を作成してください。ログエージェントは、CreateLogGroup および CreateLogStream 権限の両方がない限り、作成されたロググループにログストリームを作成できません。

トラブルシューティング時にはどのログを調べますか。

エージェントのインストールログは /var/log/awslogs-agent-setup.log に、エージェントログは /var/log/awslogs.log にあります。

クイックスタート: CloudFormation を使用して CloudWatch Logs の使用を開始する

AWS CloudFormation では、JSON 形式で AWS リソースを記述およびプロビジョニングできます。この方法の利点は、AWS リソースのコレクションを 1 つのユニットとして管理できることと、AWS リソースをリージョン間で簡単にレプリケートできることです。

AWS を使用してプロビジョニングする場合は CloudFormation、使用するリソースを AWS 記述するテンプレートを作成します。次の例は、ロググループと、404 の発生数をカウントし、この数をロググループに送信するメトリクスフィルタを作成するテンプレートスニペットです。

```
"WebServerLogGroup": {
  "Type": "AWS::Logs::LogGroup",
  "Properties": {
    "RetentionInDays": 7
  }
},

"404MetricFilter": {
  "Type": "AWS::Logs::MetricFilter",
  "Properties": {
    "LogGroupName": {
      "Ref": "WebServerLogGroup"
```

```
    },
    "FilterPattern": "[ip, identity, user_id, timestamp, request, status_code =
404, size, ...]",
    "MetricTransformations": [
        {
            "MetricValue": "1",
            "MetricNamespace": "test/404s",
            "MetricName": "test404Count"
        }
    ]
}
}
```

これは基本的な例です。を使用して、より豊富な CloudWatch Logs デプロイを設定できます CloudFormation。テンプレート例の詳細については、AWS CloudFormation ユーザーガイドの「[Amazon CloudWatch Logs テンプレートスニペット](#)」を参照してください。開始方法の詳細については、AWS CloudFormation ユーザーガイドの「[AWS CloudFormationの開始方法](#)」を参照してください。

HTTP エンドポイントを介したログ取り込み

Amazon CloudWatch Logs には、シンプルな HTTP POST リクエストを使用して CloudWatch Logs に直接ログを送信できる HTTP エンドポイントが用意されています。これらのエンドポイントは、SigV4 トークン認証とベアラートークン認証の両方をサポートしています。

⚠ Important

AWS SDK 統合が可能なすべての本稼働ワークロードには、SigV4 認証を使用することをお勧めします。SigV4 は短期認証情報を使用し、最も強力なセキュリティ体制を提供します。ベアラートークン (API キー) 認証は、AWS SDK 統合をサポートしていないサードパーティーのログフォワーダーなど、SigV4 が不可能なシナリオを対象としています。詳細については、IAM ユーザーガイドの[「長期アクセスキーの代替」](#)を参照してください。

CloudWatch Logs は、次の HTTP 取り込みエンドポイントをサポートしています。

Endpoint	パス	Content-Type	形式
OpenTelemetry Logs	/v1/logs	application/json 、または application/x-protobuf	OTLP JSON または Protobuf
HLC Logs	/services/collector/event	application/json	HLC 形式
ND-JSON Logs	/ingest/bulk	application/json 、または application/x-ndjson	改行で区切られた JSON
Structured JSON Logs	/ingest/json	application/json	JSON オブジェクトまたは配列

一般的な動作

すべての HTTP 取り込みエンドポイントは、次の動作を共有します。

認証

すべてのエンドポイントは、SigV4 トークン認証とベアラートークン認証の両方をサポートしています。

- SigV4 (推奨) – 標準 AWS 署名バージョン 4 の署名。アプリケーションまたはインフラストラクチャが AWS SDK をサポートしている場合、またはリクエストに署名できる場合は常に SigV4 を使用します。SigV4 は短期認証情報を使用し、最も安全な認証方法です。
- ベアラートークン – Authorization: Bearer <ACWL token>ヘッダーを使用します。
 - トークンは有効な ACWL ベアラートークンである必要があります。セットアップ手順については、「」を参照してください[the section called “ベアラートークン認証”](#)。
 - logs:PutLogEvents および logs:CallWithBearerToken IAM アクセス許可が必要です。

ロググループとログストリーム

- ヘッダー経由で提供: x-aws-log-groupおよび x-aws-log-stream
- クエリパラメータ?logGroup=<name>&logStream=<name>は、OTLP を除くすべてのエンドポイントでもサポートされています。
- 同じパラメータにクエリパラメータとヘッダーの両方を使用することはできません。
- ロググループとログストリームの両方が必要です。

レスポンス

- 成功: 本文HTTP 200付き {}
- 検証エラー: HTTP 400
- 認証の失敗: HTTP 401

ベアラートークン認証の設定

いずれかの HTTP 取り込みエンドポイントでベアラートークン認証を使用してログを送信する前に、以下を行う必要があります。

- CloudWatch Logs アクセス許可を持つ IAM ユーザーを作成する
- サービス固有の認証情報の生成 (ベアラートークン)
- ロググループとログストリームを作成する

- ロググループでベアラートークン認証を有効にする

Important

これが可能なすべてのワークロードに対して、短期認証情報で SigV4 認証を使用することをお勧めします。SigV4 は、最も強力なセキュリティ体制を提供します。API キー (ベアラートークン) の使用は、短期的な認証情報ベースの認証が不可能なシナリオに制限します。セキュリティ要件が高いアプリケーションに CloudWatch Logs を組み込む準備ができたなら、短期認証情報に切り替える必要があります。詳細については、IAM ユーザーガイドの [「長期アクセスキーの代替」](#) を参照してください。

オプション 1: AWS コンソールを使用したクイックスタート

AWS マネジメントコンソールは、HTTP エンドポイントアクセス用の API キーを生成するための合理化されたワークフローを提供します。

コンソールを使用して HTTP エンドポイントアクセスを設定するには

1. AWS マネジメントコンソールにサインインします。
2. CloudWatch > Settings > Logs に移動します。
3. API キー セクションで、API キーの生成 を選択します。
4. API キーの有効期限について、以下のいずれかを実行します。
 - API キーの有効期限として 1、5、30、90、または 365 日を選択します。
 - カスタム期間を選択して、カスタム API キーの有効期限を指定します。
 - 選択 有効期限なし (推奨されません)。
5. [API キーの生成] を選択します。

コンソールは自動的に以下を実行します。

- 適切なアクセス許可を持つ新しい IAM ユーザーを作成します
 - [CloudWatchLogsAPIKeyAccess](#) 管理ポリシー (logs:PutLogEventsおよび アクセスlogs:CallWithBearerToken許可を含む) をアタッチします
 - サービス固有の認証情報を生成します (API キー)
6. 表示された認証情報をコピーして安全に保存します。

- API キー ID (サービス固有の認証情報 ID)
- API キーシークレット (ベアラートークン)

 Important

API キーシークレットをすぐに保存します。このキーは後で取得することはできません。紛失した場合は、新しい API キーを生成する必要があります。

7. ログを保存するロググループとログストリームを作成します。

```
# Create the log group
aws logs create-log-group \
  --log-group-name /aws/hlc-logs/my-application \
  --region us-east-1

# Create the log stream
aws logs create-log-stream \
  --log-group-name /aws/hlc-logs/my-application \
  --log-stream-name application-stream-001 \
  --region us-east-1
```

8. ロググループでベアラートークン認証を有効にします。

```
aws logs put-bearer-token-authentication \
  --log-group-identifier /aws/hlc-logs/my-application \
  --bearer-token-authentication-enabled \
  --region us-east-1
```

設定の検証:

```
aws logs describe-log-groups \
  --log-group-name-prefix /aws/hlc-logs/my-application \
  --region us-east-1
```

含まれるアクセス許可: 自動的に作成された IAM ユーザーには、次のアクセス許可があります。

- `logs:PutLogEvents` – CloudWatch Logs にログイベントを送信する
- `logs:CallWithBearerToken` – ベアラートークンを使用して認証する

- kms:Describe*、kms:GenerateDataKey*、kms:Decrypt – KMS で暗号化されたロググループにアクセスする (条件はログサービスに制限されます)

オプション 2: 手動セットアップ

IAM 設定をより詳細に制御する場合、またはアクセス許可をカスタマイズする必要がある場合は、HTTP エンドポイントアクセスを手動で設定できます。

ステップ 1: IAM ユーザーを作成する

ログの取り込みに使用される IAM ユーザーを作成します。

1. AWS マネジメントコンソールにサインインし、IAM に移動します。
2. 左のナビゲーションペインで、[ユーザー] を選択します。
3. [ユーザーの作成] を選択します。
4. ユーザー名 (などcloudwatch-logs-hlc-user) を入力します。
5. [次へ] を選択します。
6. 次のいずれかの IAM ポリシーをアタッチします。

オプション A: 管理ポリシーを使用する (推奨)

[CloudWatchLogsAPIKeyAccess](#) 管理ポリシーをアタッチします。

オプション B: カスタムポリシーを作成する

次の IAM ポリシーを作成してアタッチします。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "LogsAPIs",
      "Effect": "Allow",
      "Action": [
        "logs:CallWithBearerToken",
        "logs:PutLogEvents"
      ],
      "Resource": "*"
    }
  ],
}
```

```
{
  "Sid": "KMSAPIs",
  "Effect": "Allow",
  "Action": [
    "kms:Describe*",
    "kms:GenerateDataKey*",
    "kms:Decrypt"
  ],
  "Condition": {
    "StringEquals": {
      "kms:ViaService": [
        "logs.*.amazonaws.com"
      ]
    }
  },
  "Resource": "arn:aws:kms:*:*:key/*"
}
```

7. Next を選択し、Create user を選択します。

Note

KMS で暗号化されたロググループにログを送信する場合は、KMS アクセス許可が必要です。この条件は、CloudWatch Logs サービスを介して使用されるキーのみに KMS アクセスを制限します。

ステップ 2: サービス固有の認証情報を生成する (API キー)

CreateServiceSpecificCredential API を使用して CloudWatch Logs API キーを生成します。

[CreateServiceSpecificCredential](#) [create-service-specific-credential](#) CLI コマンドを使用することもできます。認証情報の有効期間には、1～36600 日の値を指定できます。認証情報の有効期間を指定しないと、API キーは期限切れになりません。

有効期限が 30 日の API キーを生成するには:

```
aws iam create-service-specific-credential \
  --user-name cloudwatch-logs-hlc-user \
  --service-name logs.amazonaws.com \
```

```
--credential-age-days 30
```

レスポンスは [ServiceSpecificCredential](#) オブジェクトです。ServiceCredentialSecret 値は CloudWatch Logs API キー (ベアラートークン) です。

Important

ServiceCredentialSecret 値は後で取得できないため、安全に保管してください。紛失した場合は、新しい API キーを生成する必要があります。

ステップ 3: ロググループとログストリームを作成する

ログを保存するロググループとログストリームを作成します。

```
# Create the log group
aws logs create-log-group \
  --log-group-name /aws/hlc-logs/my-application \
  --region us-east-1

# Create the log stream
aws logs create-log-stream \
  --log-group-name /aws/hlc-logs/my-application \
  --log-stream-name application-stream-001 \
  --region us-east-1
```

ステップ 4: ベアラートークン認証を有効にする

ロググループでベアラートークン認証を有効にします。

```
aws logs put-bearer-token-authentication \
  --log-group-identifier /aws/hlc-logs/my-application \
  --bearer-token-authentication-enabled \
  --region us-east-1
```

設定の検証:

```
aws logs describe-log-groups \
  --log-group-name-prefix /aws/hlc-logs/my-application \
  --region us-east-1
```

CloudWatch Logs API キーを生成して使用するためのアクセス許可を制御する

CloudWatch Logs API キーの生成と使用は、CloudWatch Logs と IAM サービスの両方でアクションと条件キーによって制御されます。

CloudWatch Logs API キーの生成の制御

[iam:CreateServiceSpecificCredential](#) アクションは、サービス固有のキー (CloudWatch Logs API キーなど) の生成を制御します。このアクションの範囲をリソースとして IAM ユーザーに限定し、キーを生成できるユーザーを制限できます。

次の条件キーを使用して、`iam:CreateServiceSpecificCredential` アクションのアクセス許可に条件を課すことができます。

- [iam:ServiceSpecificCredentialAgeDays](#) – 条件でキーの有効期限を日数で指定できます。例えば、この条件キーを使用して、90 日以内に期限切れになる API キーの作成のみ許可できます。
- [iam:ServiceSpecificCredentialServiceName](#) – 条件でサービスの名前を指定できます。たとえば、この条件キーを使用して CloudWatch Logs の API キーの作成のみを許可し、他のサービスの作成は許可しません。

CloudWatch Logs API キーの使用の制御

`logs:CallWithBearerToken` アクションは、CloudWatch Logs API キーの使用を制御します。ID が CloudWatch Logs API キーを使用しないようにするには、キーに関連付けられた IAM ユーザーに `logs:CallWithBearerToken` アクションを拒否するポリシーをアタッチします。

ポリシーの例

ID が CloudWatch Logs API キーを生成して使用しないようにする

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "DenyCWLAPIKeys",
      "Effect": "Deny",
      "Action": [
        "iam:CreateServiceSpecificCredential",
        "logs:CallWithBearerToken"
      ]
    }
  ]
}
```

```
    ],  
    "Resource": "*"    
  }  
]  
}
```

Warning

このポリシーは、AWS サービス固有の認証情報の作成をサポートするすべてのサービスの認証情報の作成を防止します。詳細については、「[IAM ユーザーのサービス固有の認証情報](#)」を参照してください。

ID が CloudWatch Logs API キーを使用しないようにする

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Effect": "Deny",  
      "Action": "logs:CallWithBearerToken",  
      "Resource": "*"    
    }  
  ]  
}
```

CloudWatch Logs キーが 90 日以内に期限切れになる場合にのみ作成を許可する

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "Action": "iam:CreateServiceSpecificCredential",  
      "Resource": "arn:aws:iam::123456789012:user/username",  
      "Condition": {  
        "StringEquals": {  
          "iam:ServiceSpecificCredentialServiceName": "logs.amazonaws.com"  
        },  
        "NumericLessThanEquals": {  
          "iam:ServiceSpecificCredentialAgeDays": "90"  
        }  
      }  
    }  
  ]  
}
```

```
}  
  }  
}  
]  
}
```

API キーの更新

API キーを定期的にローテーションすると、不正アクセスのリスクが軽減されます。組織のセキュリティポリシーに沿ったローテーションスケジュールを確立することをお勧めします。

ローテーションプロセス

ログ配信を中断せずに API キーをローテーションするには、次の手順に従います。

1. IAM ユーザーの新しい (セカンダリ) 認証情報を作成します。

```
aws iam create-service-specific-credential \  
  --user-name cloudwatch-logs-hlc-user \  
  --service-name logs.amazonaws.com \  
  --credential-age-days 90
```

2. (オプション) 安全な取得と自動ローテーション AWS Secrets Manager のために、新しい認証情報を に保存します。
3. 新しい認証情報をベンダーのポータルにインポートするか、アプリケーション設定を更新して新しい API キーを使用します。
4. 元の認証情報を非アクティブに設定します。

```
aws iam update-service-specific-credential \  
  --user-name cloudwatch-logs-hlc-user \  
  --service-specific-credential-id ACCA1234EXAMPLE1234 \  
  --status Inactive
```

5. CloudWatch でロググループのIncomingBytesメトリクスをモニタリングすることで、ログ配信に影響がないことを確認します。詳細については、「[CloudWatch メトリクスでのモニタリング](#)」を参照してください。
6. 新しいキーで正常に配信されたことを確認したら、前の認証情報を削除します。

```
aws iam delete-service-specific-credential \  
  --service-specific-credential-id ACCA1234EXAMPLE1234
```

キーの有効期限のモニタリング

既存の API キーの作成日とステータスを確認するには、[list-service-specific-credentials](#) コマンドを使用します。

```
aws iam list-service-specific-credentials \  
  --user-name cloudwatch-logs-hlc-user \  
  --service-name logs.amazonaws.com
```

レスポンスには、各認証情報 Status の CreateDate と が含まれます。この情報を使用して、有効期限が近づいているか、ローテーションポリシーで許可されているよりも長くアクティブになっているキーを特定します。

侵害された API キーへの対応

API キーが侵害された疑いがある場合は、すぐに次の手順を実行します。

1. キーをすぐに無効にして、それ以上の不正使用を防止します。

```
aws iam update-service-specific-credential \  
  --user-name cloudwatch-logs-hlc-user \  
  --service-specific-credential-id ACCA1234EXAMPLE1234 \  
  --status Inactive
```

2. CloudTrail ログを確認して、不正アクセスの範囲を判断します。API キーの使用状況の監査を有効にする方法については、[the section called “CloudTrail を使用した API キーの使用状況のログ記録”](#)「」を参照してください。
3. 「」で説明されているローテーションプロセスに従って代替キーを作成します[the section called “ローテーションプロセス”](#)。
4. 置換が完了したら、侵害されたキーを削除します。

```
aws iam delete-service-specific-credential \  
  --service-specific-credential-id ACCA1234EXAMPLE1234
```

5. 調査中に IAM ユーザーのすべてのベアラートークンアクセスを直ちにブロックする必要がある場合は、拒否ポリシーをアタッチします。

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

```
"Statement": {
  "Effect": "Deny",
  "Action": "logs:CallWithBearerToken",
  "Resource": "*"
}
```

Note

API を使用してこれらのアクションを実行するには、CloudWatch Logs API キーではなく AWS 認証情報で認証する必要があります。

次の IAM API オペレーションを使用して、侵害されたキーを管理することもできます。

- [ResetServiceSpecificCredential](#) – キーをリセットして、認証情報を削除せずに新しいパスワードを生成します。キーの有効期限が切れていることはできません。

API キーのセキュリティのベストプラクティス

CloudWatch Logs API キーを保護するには、次のベストプラクティスに従います。

- ソースコードに API キーを埋め込まないでください。アプリケーションコードで API キーをハードコードしたり、バージョン管理システムにコミットしたりしないでください。キーが誤ってパブリックリポジトリにコミットされた場合、AWS 自動スキャンはキーにフラグを付ける可能性があるため、すぐにキーをローテーションする必要があります。
- シークレットマネージャーを使用します。API キーを [AWS Secrets Manager](#) または同等のシークレット管理ソリューションに保存します。これにより、一元的なアクセスコントロール、監査ログ記録、自動ローテーションが可能になります。
- すべてのキーの有効期限を設定します。API キーを作成するときは、常に `--credential-age-days` 値を指定します。組織全体でキーの最大有効期間を適用するには、IAM `iam:ServiceSpecificCredentialAgeDays` 条件キーを使用します。例については、[the section called “CloudWatch Logs キーが 90 日以内に期限切れになる場合にのみ作成を許可する”](#)を参照してください。
- 最小特権のアクセス許可を適用します。IAM ユーザーのアクセス許可の範囲を、必要なロググループとアクションのみに設定します。マネージド [CloudWatchLogsAPIKeyAccess](#) ポリシーを開始点として使用し、必要に応じてさらに制限します。

- CloudTrail ログ記録を有効にします。の CloudTrail データイベントを有効にして API キーの使用状況を監査しますAWS::Logs::LogGroupAuthorization。 [「the section called “CloudTrail を使用した API キーの使用状況のログ記録”」](#) を参照してください。
- IAM Access Analyzer でモニタリングします。 [IAM Access Analyzer](#) を使用して、API キー IAM ユーザーに関連付けられた未使用の認証情報と過度に寛容なポリシーを特定します。
- キーを定期的にローテーションします。ローテーションスケジュールを設定し、「」で説明されているプロセスに従います[the section called “API キーの更新”](#)。

CloudTrail を使用した API キーの使用状況のログ記録

を使用して AWS CloudTrail、CloudWatch Logs API キーの使用に関するデータイベントを記録できます。CloudWatch Logs はCallWithBearerToken呼び出しAWS::Logs::LogGroupAuthorizationのデータイベントを出力するため、API キーを使用してログを送信するタイミングと方法を監査できます。

CloudWatch Logs API キーの使用に対して CloudTrail CloudTrail ログ記録を有効にするには:

Note

証跡に指定する S3 バケットには、CloudTrail がログファイルを書き込むことを許可するバケットポリシーが必要です。詳細については、[CloudTrail の Amazon S3 バケットポリシー](#)」を参照してください。

1. 証跡を作成します。

```
aws cloudtrail create-trail \  
  --name cloudwatch-logs-api-key-audit \  
  --s3-bucket-name my-cloudtrail-bucket \  
  --region us-east-1
```

2. CloudWatch Logs ロググループ認可イベントをキャプチャするように高度なイベントセレクタを設定します。

```
aws cloudtrail put-event-selectors \  
  --region us-east-1 \  
  --trail-name cloudwatch-logs-api-key-audit \  
  --advanced-event-selectors '[{  
    "Name": "CloudWatch Logs API key authorization events",
```

```
"FieldSelectors": [
  { "Field": "eventCategory", "Equals": ["Data"] },
  { "Field": "resources.type", "Equals":
["AWS::Logs::LogGroupAuthorization"] }
]
```

3. 証跡のログ記録を開始する:

```
aws cloudtrail start-logging \
  --name cloudwatch-logs-api-key-audit \
  --region us-east-1
```

OTLP エンドポイントを使用したログの送信 (OpenTelemetry Logs)

OpenTelemetry Logs エンドポイント (/v1/logs) は、OpenTelemetry Protocol (OTLP) ログデータを JSON エンコードまたは Protobuf エンコードで受け入れます。設定や使用状況など、OTLP エンドポイントの詳細については、[OpenTelemetry を使用して CloudWatch にメトリクスとトレースを送信する](#)」を参照してください。

ベアラートークン認証を使用している場合は、続行する[the section called “ベアラートークン認証”](#)前に のセットアップステップを完了してください。

リクエストの形式

- 方法: POST
- Content-Type: application/json または application/x-protobuf
- ロググループ: x-aws-log-group ヘッダーのみ (クエリパラメータはサポートされていません)
- ログストリーム: x-aws-log-stream ヘッダー

リクエスト例

```
curl -X POST "https://logs.<region>.amazonaws.com/v1/logs" \
  -H "Authorization: Bearer ACWL<token>" \
  -H "Content-Type: application/json" \
  -H "x-aws-log-group: MyLogGroup" \
```

```
-H "x-aws-log-stream: MyLogStream" \  
-d '{  
  "resourceLogs": [  
    {  
      "resource": {  
        "attributes": [  
          {  
            "key": "service.name",  
            "value": { "stringValue": "my-service" }  
          }  
        ]  
      },  
      "scopeLogs": [  
        {  
          "scope": {  
            "name": "my-library",  
            "version": "1.0.0"  
          },  
          "logRecords": [  
            {  
              "timeUnixNano": "1741900000000000000",  
              "severityNumber": 9,  
              "severityText": "INFO",  
              "body": {  
                "stringValue": "User logged in successfully"  
              },  
              "attributes": [  
                {  
                  "key": "user.id",  
                  "value": { "stringValue": "12345" }  
                }  
              ]  
            }  
          ]  
        }  
      ]  
    }  
  ]  
}'
```

レスポンス

Success (すべてのイベントが受け入れられる):

```
HTTP 200 OK
{}
```

部分的な成功 (一部のイベントが拒否):

```
{
  "partialSuccess": {
    "rejectedLogRecords": 5,
    "errorMessage": "{\"tooOldLogEventCount\": 3, \"tooNewLogEventCount\": 1, \"expiredLogEventCount\": 1}"
  }
}
```

リクエスト Content-Type が `application/x-protobuf` の場合、レスポンスは同じフィールドを持つシリアル化された `ExportLogsServiceResponse` protobuf メッセージとして返されます。

OTLP 固有の動作

以下の動作は OTLP エンドポイントに固有であり、他の HTTP 取り込みエンドポイントには存在しません。

- `Retry-After` ヘッダー – クライアントが再試行するタイミングを示す 503 および 429 レスポンスに含まれます。

HLC エンドポイントを使用したログの送信 (HLC Logs)

HLC Logs エンドポイント (`/services/collector/event`) は、HTTP Log Collector (HLC) 形式に基づいています。

ベアラートークン認証を使用している場合は、続行する [the section called “ベアラートークン認証”](#) 前に のセットアップステップを完了してください。

入力モード

各イベントは、`"event"` 必須フィールドを持つ JSON オブジェクトです。オプションのメタデータフィールド: `"time"`、`"host"`、`"source"`、`"sourcetype"`、`"index"`。

単一イベント:

```
{"event":"Hello world!","time":1486683865.0}
```

イベントの JSON 配列:

```
[  
  {"event":"msg1","time":1486683865.0},  
  {"event":"msg2","time":1486683866.0}  
]
```

連結/バッチイベント (配列ラッパーなし):

```
{"event":"msg1","time":1486683865.0}{ "event":"msg2","time":1486683866.0}
```

イベントフィールド (必須)

"event" フィールドは必須です。その値は任意の JSON タイプにすることができます。

```
{"event":"a string message"}  
{"event":{"message":"structured data","severity":"INFO"}}  
{"event":42}  
{"event":true}
```

"event" フィールドのないオブジェクトはサイレントにスキップされます。

```
{"message":"this is skipped – no event field"}
```

時間フィールド (オプション)

"time" フィールドはエポック秒 (ミリ秒ではなく) で、オプションで 10 進数は 1 秒未満の精度です。

形式	例	次のように解釈されます
浮動小数点数	"time":1486683865.500	1486683865500 ミリ秒
整数	"time":1486683865	1486683865000 ミリ秒
文字列 (浮動小数点)	"time":"1486683865.500"	1486683865500 ミリ秒

形式	例	次のように解釈されます
文字列 (整数)	"time":"1486683865"	1486683865000 ミリ秒
Missing (見つからない)	(時間フィールドなし)	サーバーの現在の時刻
無効	"time":"invalid"	サーバーの現在の時刻

Content-Type

のみが受け入れられapplication/jsonです。

許容される JSON 値タイプ

最上位タイプ	行動
を使用したオブジェクト "event"	承諾
なしのオブジェクト "event"	スキップ済み
オブジェクトの配列	各要素は個別に処理されます
連結オブジェクト	個別に処理される各オブジェクト
プリミティブ (文字列、数値、ブール値、null)	スキップ済み

エンドポイントフォーマット

HLC エンドポイント URL は次の形式に従います。

```
https://logs.<region>.amazonaws.com/services/collector/event?
logGroup=<name>&logStream=<name>[&entityName=<name>&entityEnvironment=<environment>]
```

必須パラメータ:

- <region> – AWS リージョン (例: us-east-1、eu-west-1)
- logGroup – URL エンコードされたロググループ名

- `logStream` – URL エンコードされたログストリーム名

オプションのパラメータ:

オプションで、次のクエリパラメータを含めることで、ログイベントをServiceエンティティに関連付けることができます。HLC エンドポイントを介して送信されるログはカスタムテレメトリであるため、自動的にエンティティに関連付けられません。これらのパラメータを指定することで、CloudWatch Logs は `KeyAttributes.Type` に設定してエンティティを作成しService、ログイベントに関連付けます。これにより、CloudWatch の Explore 関連機能は、これらのログを同じサービスの他のテレメトリ (メトリクス、トレース、ログ) と関連付けることができるため、さまざまなシグナルタイプのアプリケーションのトラブルシューティングとモニタリングが容易になります。エンティティおよび関連するテレメトリの詳細については、[「カスタムテレメトリへの関連情報の追加」](#)を参照してください。

- `entityName` – ログイベントに関連付けるサービスエンティティの名前。この値はエンティティ `KeyAttributes.Name` (`my-application` や など `api.my-service.com`) として保存されます。
- `entityEnvironment` – サービスがホストされている環境、またはサービスが属する環境。この値はエンティティ `KeyAttributes.Environment` (、 、 など `ec2:default:eks:my-cluster/default`) `production` として保存されます。

リクエストの形式

次のヘッダーと本文を含む HTTP POST を使用してログを送信します。

ヘッダー:

- `Authorization: Bearer <your-bearer-token>`
- `Content-Type: application/json`

本文形式:

リクエスト本文は、イベントの配列を含む JSON 形式である必要があります。

```
{
  "event": [
    {
      "time": 1730141374.001,
```

```
    "event": "Application started successfully",
    "host": "web-server-1",
    "source": "application.log",
    "severity": "info"
  },
  {
    "time": 1730141374.457,
    "event": "User login successful",
    "host": "web-server-1",
    "source": "auth.log",
    "user": "john.doe"
  }
]
```

フィールドの説明:

- **time** – UNIX エポックタイムスタンプを秒単位で、オプションで 10 進数を秒未満の精度で指定 (オプション)
- **event** – ログメッセージまたはイベントデータ (必須)
- **host** – ソースホスト名または識別子 (オプション)
- **source** – ログソース識別子 (オプション)

必要に応じて、追加のカスタムフィールドを含めることができます。

リクエスト例

```
curl -X POST "https://logs.<region>.amazonaws.com/services/collector/event?
logGroup=MyLogGroup&logStream=MyStream" \
-H "Authorization: Bearer ACWL<token>" \
-H "Content-Type: application/json" \
-d '{"event":{"message":"User logged
in","user_id":"u-123"},"time":1486683865.0,"host":"web-01","source":"auth-service"}'
```

ベストプラクティス

イベントのバッチ処理

パフォーマンスと効率を向上させるには:

- 可能であれば、1 つのリクエストで複数のイベントをバッチ処理する
- 推奨バッチサイズ: リクエストあたり 10 ~ 100 イベント
- 最大リクエストサイズ: 1 MB

エラー処理

アプリケーションに適切なエラー処理を実装します。一般的な HTTP ステータスコード:

- 200 OK – 正常に取り込まれたログ
- 400 Bad Request – 無効なリクエスト形式またはパラメータ
- 401 Unauthorized – 無効または期限切れのベアラートークン
- 403 Forbidden – アクセス許可が不十分
- 404 Not Found – ロググループまたはストリームが存在しません
- 429 Too Many Requests – レート制限を超えました
- 500 Internal Server Error – サービスエラー (エクスポネンシャルバックオフで再試行)

制限事項

- 最大イベントサイズ: イベントあたり 256 KB
- 最大リクエストサイズ: 1 MB
- リクエストあたりの最大イベント数: 10,000
- ロググループ名は CloudWatch Logs の命名規則に従う必要があります
- ベアラートークン認証を使用する場合は、ロググループでベアラートークン認証を有効にする必要があります。

NDJSON エンドポイントを使用したログの送信 (ND-JSON Logs)

ND-JSON Logs エンドポイント (/ingest/bulk) は、[NDJSON \(Newline Delimited JSON\)](#) 形式のログを受け入れます。各行には、改行文字で区切られた 1 つの JSON 値のみが含まれます。

ベアラートークン認証を使用している場合は、続行する[the section called “ベアラートークン認証”](#)前に のセットアップステップを完了してください。

リクエストの形式

(LF) または \n (CRLF) で区切って、行ごとに 1 つの \r\n JSON 値を送信します。空の行はサイレントに無視されます。

```
{"timestamp":1771007942000,"message":"event one","level":"INFO"}
{"timestamp":1771007943000,"message":"event two","level":"ERROR"}
{"timestamp":1771007944000,"message":"event three","level":"DEBUG"}
```

application/json と の両方 application/x-ndjson が Content-Type として受け入れられます。

許容される JSON 値タイプ

NDJSON 仕様 (RFC 8259) に従って、有効な JSON 値は各行で受け入れられます。

JSON オブジェクト (最も一般的):

```
{"timestamp":1771007942000,"message":"User logged in","service":"auth"}
{"timestamp":1771007943000,"error":"Connection timeout","service":"api"}
```

JSON 配列 (個々のイベントにフラット化):

```
[{"timestamp":1000,"message":"a"}, {"timestamp":2000,"message":"b"}]
```

この 1 行は 2 つのイベントを生成します。各配列要素は個別のログイベントになります。

プリミティブ値:

```
"a plain string log message"
42
true
null
```

各プリミティブは、サーバーの現在のタイムスタンプを持つ独自のイベントになります。

混合型:

```
{"timestamp":1771007942000,"message":"structured event"}
```

```
"unstructured string message"
42
{"timestamp":1771007943000,"error":"something failed"}
```

4 行はすべて有効なイベントとして受け入れられます。

行の内容	行動
JSON オブジェクト	受け入れられ、存在する場合はタイムスタンプが抽出されます
JSON 配列	フラット化 – 各要素は個別のイベントになります
空の配列 []	受け入れられ、0 個のイベントが生成されます
JSON 文字列。	イベントメッセージとして受け入れられました
JSON 数値	イベントメッセージとして受け入れられました
JSON ブール値	イベントメッセージとして受け入れられました
JSON null	イベントメッセージとして受け入れられました
JSON が無効です	スキップ (カウント、処理続行)
空の行	無視 (スキップとしてカウントされません)

タイムスタンプフィールド

"timestamp" フィールドはエポックミリ秒 (秒ではなく) です。

形式	例	次のように解釈されます
数値 (ミリ)	"timestamp":1771007942000	1771007942000 ミリ秒
Missing (見つからない)	(タイムスタンプフィールドなし)	サーバーの現在の時刻
数値以外	"timestamp":"invalid"	サーバーの現在の時刻
オブジェクト以外の行	"hello", 42, true	サーバーの現在の時刻

無効な行

有効な JSON ではない行は、サイレントにスキップされ、カウントされます。処理は次の行に進みます。

```
{ "message": "valid event" }
this is not valid json
{ "message": "another valid event" }
```

結果: 2 つのイベントが取り込まれ、1 つはスキップされました。戻り値 HTTP 200。

すべての行が無効の場合、は HTTP 400とともに を返します "All events were invalid"。

リクエスト例

```
curl -X POST "https://logs.<region>.amazonaws.com/ingest/bulk?
logGroup=MyLogGroup&logStream=MyStream" \
-H "Authorization: Bearer ACWL<token>" \
-H "Content-Type: application/x-ndjson" \
-d '{ "timestamp":1771007942000,"message":"User logged in","level":"INFO"}
{"timestamp":1771007943000,"message":"Query took 42ms","level":"DEBUG"}
{"timestamp":1771007944000,"error":"Connection refused","level":"ERROR"}'
```

レスポンス

Success (すべてのイベントが受け入れられる):

```
HTTP 200 OK
```

```
{}
```

部分的な成功 (一部のイベントが拒否):

```
{
  "partialSuccess": {
    "rejectedLogRecords": 5,
    "errorMessage": "{\"tooOldLogEventCount\": 3, \"tooNewLogEventCount\": 1, \"expiredLogEventCount\": 1}"
  }
}
```

`rejectedLogRecords` フィールドは、拒否されたイベントの総数です。`errorMessage` フィールドには、拒否理由別の JSON エンコードされた内訳が含まれます。

- `tooOldLogEventCount` – タイムスタンプが保持期間より古いイベント
- `tooNewLogEventCount` – タイムスタンプが遠すぎるイベント
- `expiredLogEventCount` – 処理中に期限切れになったイベント

ベストプラクティス

イベントのバッチ処理

パフォーマンスと効率を向上させるには:

- 可能であれば、1 つのリクエストで複数のイベントをバッチ処理する
- 推奨バッチサイズ: リクエストあたり 10 ~ 100 イベント
- 最大リクエストサイズ: 1 MB

エラー処理

アプリケーションに適切なエラー処理を実装します。一般的な HTTP ステータスコード:

- 200 OK – 正常に取り込まれたログ
- 400 Bad Request – 無効なリクエスト形式またはパラメータ
- 401 Unauthorized – 無効または期限切れのベアータークン
- 403 Forbidden – アクセス許可が不十分

- 404 Not Found – ロググループまたはストリームが存在しません
- 429 Too Many Requests – レート制限を超えました
- 500 Internal Server Error – サービスエラー (エクスポネンシャルバックオフで再試行)

制限事項

- 最大イベントサイズ: イベントあたり 256 KB
- 最大リクエストサイズ: 1 MB
- リクエストあたりの最大イベント数: 10,000
- ロググループ名は CloudWatch Logs の命名規則に従う必要があります
- ベアラートークン認証を使用する場合は、ロググループでベアラートークン認証を有効にする必要があります。

構造化 JSON エンドポイントを使用したログの送信 (構造化 JSON ログ)

構造化 JSON ログエンドポイント (/ingest/json) は、単一の JSON オブジェクトまたはオブジェクトの JSON 配列の標準 JSON を受け入れます。このエンドポイントは、各イベントが JSON オブジェクトである構造化ログデータ用に設計されています。

ベアラートークン認証を使用している場合は、続行する[the section called “ベアラートークン認証”](#)前に のセットアップステップを完了してください。

リクエストの形式

Content-Type としてのみ application/json 受け入れられます。

単一の JSON オブジェクト:

```
{"timestamp":1771007942000,"message":"single event","level":"INFO"}
```

オブジェクトの JSON 配列:

```
[{"timestamp":1771007942000,"message":"event one","level":"INFO"}, {"timestamp":1771007943000,"message":"event two","level":"ERROR"}]
```

]

許容される JSON 値タイプ

このエンドポイントは厳密です。JSON オブジェクトのみがイベントとして受け入れられます。

Input	行動
単一の JSON オブジェクト	1 つのイベントとして受け入れられました
オブジェクトの JSON 配列	各オブジェクトが個別のイベントになる
空の配列 []	受け入れられ、0 個のイベントが生成されます
配列内のオブジェクト以外 (文字列、数値など)	スキップ済み
最上位プリミティブ ("hello"、42)	スキップ済み
連結オブジェクト {...}{...}	解析された最初のオブジェクトのみ

例 – 混合型を持つ配列:

```
[
  {"timestamp":1771007942000,"message":"valid object"},
  "just a string",
  42,
  {"timestamp":1771007943000,"message":"another valid object"}
]
```

結果: 取り込まれた 2 つのイベント (オブジェクト)、スキップされた 2 つのイベント (文字列と数値)。

タイムスタンプフィールド

"timestamp" フィールドは、NDJSON エンドポイントと同じエポックミリ秒単位です。

形式	例	次のように解釈されます
数値 (ミリ秒)	"timestamp":1771007942000	1771007942000 ミリ秒
Missing (見つからない)	(タイムスタンプフィールドなし)	サーバーの現在の時刻
数値以外	"timestamp":"invalid"	サーバーの現在の時刻

リクエスト例

```
curl -X POST "https://logs.<region>.amazonaws.com/ingest/json?
logGroup=MyLogGroup&logStream=MyStream" \
-H "Authorization: Bearer ACWL<token>" \
-H "Content-Type: application/json" \
-d '[{"timestamp":1771007942000,"message":"User logged in","user_id":"u-123"},
{"timestamp":1771007943000,"message":"Order placed","order_id":"o-456"}]'
```

レスポンス

Success (すべてのイベントが受け入れられる):

```
HTTP 200 OK
{}
```

部分的な成功 (一部のイベントが拒否):

```
{
  "partialSuccess": {
    "rejectedLogRecords": 5,
    "errorMessage": "{\"tooOldLogEventCount\": 3, \"tooNewLogEventCount\": 1,
    \"expiredLogEventCount\": 1}"
  }
}
```

rejectedLogRecords フィールドは、拒否されたイベントの総数です。errorMessage フィールドには、拒否理由別の JSON エンコードされた内訳が含まれます。

- tooOldLogEventCount – タイムスタンプが保持期間より古いイベント

- tooNewLogEventCount – タイムスタンプが遠すぎるイベント
- expiredLogEventCount – 処理中に期限切れになったイベント

ベストプラクティス

イベントのバッチ処理

パフォーマンスと効率を向上させるには:

- 可能であれば、1 つのリクエストで複数のイベントをバッチ処理する
- 推奨バッチサイズ: リクエストあたり 10 ~ 100 イベント
- 最大リクエストサイズ: 1 MB

エラー処理

アプリケーションに適切なエラー処理を実装します。一般的な HTTP ステータスコード:

- 200 OK – 正常に取り込まれたログ
- 400 Bad Request – 無効なリクエスト形式またはパラメータ
- 401 Unauthorized – 無効または期限切れのベアラートークン
- 403 Forbidden – アクセス許可が不十分
- 404 Not Found – ロググループまたはストリームが存在しません
- 429 Too Many Requests – レート制限を超えました
- 500 Internal Server Error – サービスエラー (エクスポネンシャルバックオフで再試行)

制限事項

- 最大イベントサイズ: イベントあたり 256 KB
- 最大リクエストサイズ: 1 MB
- リクエストあたりの最大イベント数: 10,000
- ロググループ名は CloudWatch Logs の命名規則に従う必要があります
- ベアラートークン認証を使用する場合は、ロググループでベアラートークン認証を有効にする必要があります。

HTTP 取り込みエンドポイントの比較

機能	HLC ログ	ND-JSON ログ	構造化 JSON ログ	OpenTelemetry ログ
パス	/services/collector/event	/ingest/bulk	/ingest/json	/v1/logs
Content-Type	application/json	application/json 、または application/x-ndjson	application/json	application/json 、または application/x-protobuf
タイムスタンプフィールド	"time" (秒)	"timestamp" (ミリ秒)	"timestamp" (ミリ秒)	"timeUnixNano" (ナノ秒)
必須フィールド	"event"	なし	なし	OTLP 構造 ("resourceLogs")
部分的な成功レスポンス	いいえ	はい	はい	はい
クエリパラメータのサポート	はい	はい	はい	いいえ (ヘッダーのみ)
エンティティメタデータ	はい	はい	あり	なし
プリミティブを受け入れます	いいえ	あり	なし	いいえ
ラインベースの解析	いいえ	あり	なし	いいえ

機能	HLC ログ	ND-JSON ログ	構造化 JSON ログ	OpenTelemetry ログ
Protobuf のサポート	いいえ	なし	なし	はい
Retry-After ヘッダー	いいえ	なし	なし	はい

エンドポイントの選択

- HLC 形式を使用しますか? HLC ログを使用します。既存の HLC ペイロードは最小限の変更で機能します。
- ストリーミングline-by-lineログ ND-JSON ログを使用します。1 行に 1 つのイベントを出力するログパイプラインに最適です。最も柔軟性が高い — 任意の JSON 値タイプを受け入れます。
- 構造化された JSON ペイロードを送信しますか? 構造化 JSON ログを使用します。正しい形式の JSON オブジェクトまたは配列を生成するアプリケーションに最適です。
- OpenTelemetry を既に使用していますか? OpenTelemetry Logs を使用します。OTLP JSON または Protobuf 形式を受け入れ、再試行セマンティクスを使用した部分的な成功レスポンスをサポートします。

AWS SDK での CloudWatch Logs の使用

AWS Software Development Kit (SDKs)は、多くの一般的なプログラミング言語で使用できます。各 SDK には、デベロッパーが好みの言語でアプリケーションを簡単に構築できるようになる API、コード例、およびドキュメントが提供されています。

SDK ドキュメント	コード例
AWS SDK for C++	AWS SDK for C++ コード例
AWS CLI	AWS CLI コード例
AWS SDK for Go	AWS SDK for Go コード例
AWS SDK for Java	AWS SDK for Java コード例
AWS SDK for JavaScript	AWS SDK for JavaScript コード例
AWS SDK for Kotlin	AWS SDK for Kotlin コード例
AWS SDK for .NET	AWS SDK for .NET コード例
AWS SDK for PHP	AWS SDK for PHP コード例
AWS Tools for PowerShell	AWS Tools for PowerShell コード例
AWS SDK for Python (Boto3)	AWS SDK for Python (Boto3) コード例
AWS SDK for Ruby	AWS SDK for Ruby コード例
AWS SDK for Rust	AWS SDK for Rust コード例
AWS SDK for SAP ABAP	AWS SDK for SAP ABAP コード例
AWS SDK for Swift	AWS SDK for Swift コード例

CloudWatch Logs の固有の例については、「[SDK を使用した CloudWatch Logs のコード例 AWS SDKs](#)」を参照してください。

可用性の例

必要なものが見つからなかった場合。このページの下側にある [Provide feedback] リンクから、コードの例をリクエストしてください。

ログ管理

CloudWatch Logs は、ログデータのより効果的な整理、変換、分析に役立つ高度なログ管理機能を提供します。これらの機能には、[クロスアカウントおよびクロスリージョンのデータ一元化](#)、[自動データ検出とスキーマ管理](#)、[取り込み中のログ変換](#)、[インタラクティブなログ探索のためのファセットを使用した拡張分析](#)が含まれます。

トピック

- [データソースの検出と管理](#)
- [データソースによって有効になる機能](#)
- [サポートされているデータソース](#)

データソースの検出と管理

CloudWatch Logs は、データソースとタイプ別にログデータを自動的に検出して分類するため、大規模なログの理解と管理が容易になります。この機能は、Amazon VPC フローログ、CloudTrail、Route AWS 53 などのベンダーソースとサードパーティーのセキュリティツールのスキーマ検出を提供します。

ログ管理コンソールには、ロググループだけでなく、データソースとタイプ別に整理されたログの概要が表示されます。この組織は、以下に役立ちます。

- AWS サービス、サードパーティーソース (Okta や CrowdStrike など)、カスタムソース別に分類されたログを表示する
- ログデータのスキーマと構造を自動的に理解する
- 検出されたスキーマフィールドに基づいてフィールドインデックスポリシーを作成する
- 異なるデータソース間でログをより効率的に管理する
- さまざまなデータソースによるログのクエリ

[サポートされている AWS サービスの CloudWatch Logs ログ記録を有効にすると](#)、CloudWatch Logs はログに適切なスキーマを自動的に適用します。この自動スキーマアプリケーションは、整合性を維持し、ログ構造を即座に把握するのに役立ちます。

CloudWatch Logs データソースとは

CloudWatch Logs データソースは、ログを生成するソースに基づいてログデータを整理および分類する新しい方法を提供する機能です。CloudWatch Logs は従来、ロググループを使用してログを整理していましたが、データソースは、元のサービスとログタイプ別にログをグループ化する追加の組織レイヤーを提供します。

データソースの仕組み

データソースは、サービスベースのログ整理と、AWS インフラストラクチャ全体の簡素化された検出を提供します。特定のサービスからログを簡単に検索し、個々のロググループ名や構造を知ることなく、ログタイプでフィルタリングできます。

サードパーティーソースの場合、およびオプションでアプリケーションログソースの場合、データソースは CloudWatch パイプラインと連携してログ进行分类します。ログを取り込んで変換するようにパイプラインを設定するときは、データソース名とタイプを指定します。その後、CloudWatch Logs はパイプラインが処理するすべてのログを自動的に分類します。詳細については、「[Amazon CloudWatch ユーザーガイド](#)」の「[CloudWatch パイプライン](#)」を参照してください。Amazon CloudWatch

データソースは、2 つのキー識別子を使用してログ进行分类します。

- データソース名: ログを生成する AWS サービス、サードパーティーのソース、またはアプリケーション (Route 53、Amazon VPC、CloudTrail、Okta SSO、CrowdStrike Falcon など)。
- データソースタイプ: そのサービスによって生成された特定のタイプのログ。

スキーマは、存在するフィールドや情報の整理方法など、ログデータの構造を定義します。1 つのデータソースで、スキーマと目的が異なる複数のタイプのログを生成できます。たとえば、AWS CloudTrail データソースには、管理イベント (リソースの作成や削除などのコントロールプレーンオペレーションを追跡する) とデータイベント (S3 オブジェクトアクセスなどのデータプレーンオペレーションを追跡する) の 2 つのタイプがあります。各タイプには、さまざまな種類の情報をキャプチャするため、異なるスキーマがあります。

開始方法

CloudWatch Logs は、ログをオリジンに基づいてデータソースに分類します。メソッドは、使用しているログのタイプによって異なります。

AWS のサービス ログ

[サポートされている AWS のサービス](#) からのログは、設定を必要とせずにデータソースごとに自動的にグループ化されます。CloudWatch Logs はこれらのログを認識し、元のサービスに基づいて適切なデータソース名とタイプを適用します。

サードパーティーのログ

サードパーティーのログには、データソースの分類にパイプラインが必要です。Microsoft Office 365、Okta、CrowdStrike、Palo Alto Networks などのサポートされているサードパーティーソースからログを取り込むようにパイプラインを設定する場合は、パイプライン設定で[データソース名とタイプ](#)を指定します。CloudWatch Logs は、パイプラインがそれらの識別子を使用して処理するすべてのログを自動的に分類します。

Pipelines は、必要に応じてサードパーティーのログを Open Cybersecurity Schema Framework (OCSF) 形式に変換して、標準化されたセキュリティイベント分析を行うことができます。OCSF 変換を有効にすると、データソース名とタイプは OCSF スキーママッピングに基づいて自動的に決定されます。OCSF 変換を使用しない場合は、パイプライン設定でデータソース名とタイプを指定します。

アプリケーションログ

カスタムアプリケーションログの場合、次のいずれかの方法を使用してデータソース別に分類できます。

- ロググループのタグ - キー `cw:datasource:name` と `cw:datasource:type` を使用してロググループにタグを追加し、ロググループに取り込まれたすべてのログのデータソース名とタイプをそれぞれ指定します。タグ値は最大 64 文字で、小文字、数字、アンダースコアのみを含めることができます。文字または数字で始める必要があり、二重アンダースコア (__) を含めることはできません。
- パイプライン設定 - アプリケーションログを取り込むときに、ログ処理パイプラインを介してデータソース情報を設定します。

Note

AWS サービスログとの競合を避けるため、データソース名を「aws」または「amazon」で始めることはできません。

システムフィールド

CloudWatch Logs は、データソース別に分類されたログに 3 つのシステムフィールドを自動的に追加します。これらのフィールドはデフォルトのファセットとして機能します。

- @data_source_name - データソースの名前、または未確定の場合は「不明」が含まれます。
- @data_source_type - データソースのタイプ、または未決定の場合は「不明」が含まれます。
- @data_format - ログデータの形式を示します。

データソース名またはタイプを特定できない場合、これらのフィールドは「不明」に設定されます。「不明」値を持つデータソースは、引き続き コンソールの「ログ管理」の下にあるファセットとデータソーステーブルに表示され、分類されていないログとそのロググループの送信元を特定できません。

@data_format フィールドには次のいずれかの値が含まれます:

- Default - 変更せずに取り込まれたログ。
- Custom - パイプラインプロセッサを介して処理されたログ、またはデータソース名/タイプタグを持つロググループに取り込まれたログ。
- OCSF-<version> - パイプラインで OCSF (Open Cybersecurity Schema Framework) プロセッサで処理されたログ。
- AWS-OTEL-LOG-V<version> - CloudWatch OTLP エンドポイントを介して取り込まれた OpenTelemetry ログ。
- AWS-OTEL-TRACE-V<version> - CloudWatch OTLP エンドポイントを介して取り込まれた OpenTelemetry トレース。

これらのシステムフィールドを使用すると、ソースと形式に基づいてログをフィルタリングしてクエリできるため、さまざまなオリジンからのログの操作やパイプラインの処理が容易になります。

データソースへのアクセス

コンソール

CloudWatch Logs コンソールで、ログ管理タブを使用してデータソースにアクセスします。CloudWatch Logs は、データソースとタイプ別にログデータを自動的に統合し、新しく取り込まれたデータを継続的に検出します。データソースリストから、パイプラインの作成、フィールドインデックスとファセットの定義を行うことができます。

AWS CLI

次のコマンドを使用して、アカウント内の個別のデータソースとログのタイプを一覧表示します。

```
aws logs list-aggregate-log-group-summaries --group-by DATA_SOURCE_NAME_AND_TYPE
```

ロググループとの関係

データソースは、ロググループを置き換えるのではなく補完します。ログは以前と同様にロググループに引き続き保存されますが、データソース情報も自動的にタグ付けされます。このデュアル組織を使用すると、次のことが可能になります。

- きめ細かなアクセスコントロールと保持ポリシーにロググループを使用する
- サービスベースのログ検出と分析にデータソースを使用する
- 必要に応じて、いずれかの組織的な方法を使用してログをクエリする

データソースを使用すると、AWS インフラストラクチャ全体でログデータをサービス中心に表示できるため、大規模なログの操作が容易になります。

データソースによって有効になる機能

データソースは、フィールド検出と一貫したデータ構造を通じて、高度なログ処理と分析機能を可能にします。

- **ファセット:** ファセットは、クエリを記述せずにインタラクティブなフィルタリングと分析を提供するインデックス付きログフィールドです。CloudWatch Logs はデータソース名とタイプのファセットを自動的に作成し、検出されたフィールドにファセットポリシーを作成してトラブルシューティングを高速化できます。ファセットは CloudWatch Logs Insights で値の分布とカウントを表示するため、point-and-click探索を通じてパターンを簡単に識別できます。
- **パイプライン:** 特定のデータソース名とタイプのすべてのログに適用される変換パイプラインを作成します。これにより、同じソースからのログに対して一貫した処理ルールを定義できます。
- **フィールド検出:** CloudWatch Logs は、パイプラインプロセッサに基づいて、各データソース名とタイプの組み合わせのフィールドとそのデータ型を自動的に検出します。AWS マネージドログの場合、フィールド構造は事前定義されています。アプリケーションログについては、一貫したログ形式を維持し、明確に定義されたフィールド構造を必要とする Amazon S3 テーブルなどの分析ツールとの互換性を最大化することをお勧めします。

GetLogFields API を使用して、任意のデータソースのフィールドとそのタイプの完全なリストを表示できます。

```
aws logs get-log-fields --data-source-name <name> --data-source-type <type>
```

このフィールドの検出と整合性により、ログデータを処理する際に外部ツールが予測可能なフィールド構造を操作できるため、高度な分析と統合が可能になります。

サポートされているデータソース

CloudWatch Logs は、AWS サービス、サードパーティーのセキュリティおよび IT ツール、カスタムアプリケーションソースなど、幅広いデータソースをサポートしています。このページでは、サポートされているすべてのデータソースの包括的なカタログを提供します。データソースの検出と管理方法については、「」を参照してください[the section called “データソースの検出と管理”](#)。

トピック

- [データソース AWS のサービス でサポート](#)
- [データソースでサポートされているサードパーティーソース](#)
- [カスタムソース](#)

データソース AWS のサービス でサポート

次の表に、CloudWatch Logs によって自動的にデータソースとして分類 AWS のサービス される を示します。

データソース名 (@data_source_name フィールド)	データソースタイプ (@data_source_type フィールド)
amazon_api_gateway	access
amazon_bedrock_agentcore	browser_usage
amazon_bedrock_agentcore	code_interpreter_application
amazon_bedrock_agentcore	code_interpreter_usage
amazon_bedrock_agentcore	gateway_application

データソース名 (@data_source_name フィールド)	データソースタイプ (@data_source_type フィールド)
amazon_bedrock_agentcore	identity_workload_application
amazon_bedrock_agentcore	memory_application
amazon_bedrock_agentcore	online_evaluation_config
amazon_bedrock_agentcore	runtime_application
amazon_bedrock_agentcore	runtime_usage
amazon_bedrock_agents	application
amazon_bedrock_agents	event
amazon_bedrock_knowledge_bases	application
amazon_cloudfront	access
amazon_cloudfront	connection
amazon_cloudwatch	rum_app_monitor
amazon_cognito	user_pool
amazon_ec2	verified_access
amazon_eks	api_server
amazon_eks	audit
amazon_eks	authenticator
amazon_eks	controller_manager
amazon_eks	scheduler
amazon_elasticache	cluster
amazon_eventbridge	eventbus_error

データソース名 (@data_source_name フィールド)	データソースタイプ (@data_source_type フィールド)
amazon_eventbridge	eventbus_info
amazon_eventbridge	pipes_execution
amazon_interactive_video_service	chat
amazon_managed_prometheus	scraper
amazon_managed_prometheus	workspace
amazon_msk	broker
amazon_msk	connect
amazon_opensearch_service	pipeline
amazon_q_business	events
amazon_q_business	sync_job
amazon_q_connect	events
amazon_route53	global_resolver_query
amazon_route53	hosted_zones
amazon_route53	profiles_resolver_query
amazon_route53	resolver_query
amazon_sagemaker	workteam_activity
amazon_ses	ingress_endpoints
amazon_ses	rule_sets
amazon_ses	traffic_policy
amazon_vpc	flow

データソース名 (@data_source_name フィールド)	データソースタイプ (@data_source_type フィールド)
amazon_vpc	route_server_peer
amazon_vpc_lattice	access
amazon_vpc_lattice	resource_access
amazon_workmail	access_control
amazon_workmail	authentication
amazon_workmail	personal_access
amazon_workmail	workmail_access
amazon_workmail	workmail_availability
aws_b2b_data_interchange	execution
aws_backup	data_access
aws_backup	hypervisor
aws_clean_rooms	analysis
aws_client_vpn	connection
aws_client_vpn	event
aws_cloudtrail	data
aws_cloudtrail	management
aws_elemental_mediapackage	egress_access
aws_elemental_mediapackage	ingress_access
aws_elemental_mediatailor	ad_decision
aws_elemental_mediatailor	manifest

データソース名 (@data_source_name フィールド)	データソースタイプ (@data_source_type フィールド)
aws_elemental_mediataylor	transcode
aws_entity_resolution	id_mapping_workflow
aws_entity_resolution	matching_workflow
aws_iot_fleetwise	error
aws_mainframe_modernization	batch_job
aws_mainframe_modernization	config
aws_mainframe_modernization	console
aws_mainframe_modernization	dataset_import
aws_network_firewall	alert
aws_network_firewall	flow
aws_network_firewall	tls
aws_nlb	access
aws_pcs	job_completion
aws_pcs	scheduler
aws_security_hub	compliance_finding
aws_security_hub	data_security_finding
aws_security_hub	detection_finding
aws_security_hub	vulnerability_finding
aws_security_hub_cspm	asff_finding
aws_shield	protection_flow

データソース名 (@data_source_name フィールド)	データソースタイプ (@data_source_type フィールド)
aws_step_functions	express
aws_step_functions	standard
aws_transfer_family	server
aws_waf	access

データソースでサポートされているサードパーティーソース

次の表に、パイプラインを介して取り込むときに CloudWatch Logs によってデータソースとして自動的に分類されるサードパーティーソースを示します。

データソース名 (@data_source_name フィールド)	データソースタイプ (@data_source_type フィールド)
cisco_umbrella	data_security_finding
cisco_umbrella	dns_activity
cisco_umbrella	entity_management
cisco_umbrella	network_activity
crowdstrike_falcon	detection_finding
crowdstrike_falcon	process_activity
github_auditlogs	account_change
github_auditlogs	api_activity
github_auditlogs	entity_management
microsoft_entraid	account_change
microsoft_entraid	authentication

データソース名 (@data_source_name フィールド)	データソースタイプ (@data_source_type フィールド)
microsoft_entraid	entity_management
microsoft_entraid	user_access_management
microsoft_office365	account_change
microsoft_office365	application_lifecycle
microsoft_office365	authentication
microsoft_office365	compliance_finding
microsoft_office365	detection_finding
microsoft_office365	email_activity
microsoft_office365	file_hosting_activity
microsoft_office365	group_management
microsoft_office365	incident_finding
microsoft_office365	user_access_management
microsoft_office365	vulnerability_finding
microsoft_office365	web_resources_activity
microsoft_windows	account_change
microsoft_windows	authentication
microsoft_windows	entity_management
microsoft_windows	event_log_activity
microsoft_windows	file_system_activity
microsoft_windows	group_management

データソース名 (@data_source_name フィールド)	データソースタイプ (@data_source_type フィールド)
microsoft_windows	kernel_activity
okta_auth0	api_activity
okta_auth0	authentication
okta_sso	api_activity
okta_sso	authentication
okta_sso	detection_finding
okta_sso	entity_management
paloaltonetworks_nextgenerationfirewall	authentication
paloaltonetworks_nextgenerationfirewall	detection_finding
paloaltonetworks_nextgenerationfirewall	network_activity
paloaltonetworks_nextgenerationfirewall	process_activity
sentinelone_endpointsecurity	dns_activity
sentinelone_endpointsecurity	file_system_activity
sentinelone_endpointsecurity	http_activity
sentinelone_endpointsecurity	process_activity
servicenow_cmdb	api_activity
servicenow_cmdb	datastore_activity

データソース名 (@data_source_name フィールド)	データソースタイプ (@data_source_type フィールド)
servicenow_cmdb	entity_management
wiz_cnapp	api_activity
wiz_cnapp	authentication
wiz_cnapp	compliance_finding
wiz_cnapp	detection_finding
wiz_cnapp	vulnerability_finding
zscaler_internetaccess	authentication
zscaler_internetaccess	dns_activity
zscaler_internetaccess	http_activity
zscaler_internetaccess	network_activity

AWS Security Hub CSPM を介した追加のサードパーティーソース

追加のサードパーティーのセキュリティ検出結果は、AWS Security Hub CSPM 統合を通じて入手できます。次のパートナーは、検出結果を Security Hub CSPM に送信し、CloudWatch Logs のデータソースとして利用できます。これらの統合の包括的な詳細については、[「Security Hub ユーザーガイド」](#)の「[Security Hub CSPM とのサードパーティー製品統合](#)」を参照してください。AWS

Partner	統合
3CORESec – NTA	Security Hub CSPM 経由で結果を送信する
アラートロジック – SIEMless Threat Management	Security Hub CSPM 経由で結果を送信する

Partner	統合
Aqua Security – クラウドネイティブセキュリティプラットフォーム	Security Hub CSPM 経由で結果を送信する
Aqua Security – Kube-bench	Security Hub CSPM 経由で結果を送信する
Armor – Armor Anywhere	Security Hub CSPM 経由で結果を送信する
AttackIQ	Security Hub CSPM 経由で結果を送信する
Barracuda Networks – Cloud Security Guardian	Security Hub CSPM 経由で結果を送信する
BigID – BigID Enterprise	Security Hub CSPM 経由で結果を送信する
ブルー六角形	Security Hub CSPM 経由で結果を送信する
チェックポイント – CloudGuard IaaS	Security Hub CSPM 経由で結果を送信する
チェックポイント – CloudGuard 姿勢管理	Security Hub CSPM 経由で結果を送信する
Claroty – xDome	Security Hub CSPM 経由で結果を送信する
クラウドストレージセキュリティ – Amazon S3 のウイルス対策	Security Hub CSPM 経由で結果を送信する
コントラストセキュリティ – コントラスト評価	Security Hub CSPM 経由で結果を送信する
CrowdStrike – CrowdStrike Falcon	Security Hub CSPM 経由で結果を送信する

Partner	統合
CyberArk – 特権脅威分析	Security Hub CSPM 経由で結果を送信する
データ定理	Security Hub CSPM 経由で結果を送信する
Drata	Security Hub CSPM 経由で結果を送信する
Forcepoint – CASB	Security Hub CSPM 経由で結果を送信する
Forcepoint – クラウドセキュリティゲートウェイ	Security Hub CSPM 経由で結果を送信する
Forcepoint – DLP	Security Hub CSPM 経由で結果を送信する
Forcepoint – NGFW	Security Hub CSPM 経由で結果を送信する
ファーク	Security Hub CSPM 経由で結果を送信する
Guardicore – Centra	Security Hub CSPM 経由で結果を送信する
HackerOne — 脆弱性インテリジェンス	Security Hub CSPM 経由で結果を送信する
JFrog – Xray	Security Hub CSPM 経由で結果を送信する
Juniper Networks – vSRX 次世代ファイアウォール	Security Hub CSPM 経由で結果を送信する
k9 セキュリティ – Access Analyzer	Security Hub CSPM 経由で結果を送信する

Partner	統合
レースワーク	Security Hub CSPM 経由で結果を送信する
McAfee – MVISION CNAPP	Security Hub CSPM 経由で結果を送信する
Netscout – サイバー調査担当者	Security Hub CSPM 経由で結果を送信する
Orca – クラウドセキュリティプラットフォーム	Security Hub CSPM 経由で結果を送信する
Palo Alto Networks – Prisma クラウドコンピューティング	Security Hub CSPM 経由で結果を送信する
Palo Alto Networks – Prisma Cloud Enterprise	Security Hub CSPM 経由で結果を送信する
Plerion – クラウドセキュリティプラットフォーム	Security Hub CSPM 経由で結果を送信する
プロウラー	Security Hub CSPM 経由で結果を送信する
Qualys – 脆弱性管理	Security Hub CSPM 経由で結果を送信する
Rapid7 – InsightVM	Security Hub CSPM 経由で結果を送信する
SentinelOne	Security Hub CSPM 経由で結果を送信する
Snyk	Security Hub CSPM 経由で結果を送信する
Sonrai Security – Sonrai Dig	Security Hub CSPM 経由で結果を送信する

Partner	統合
Sophos – サーバー保護	Security Hub CSPM 経由で結果を送信する
StackRox – Kubernetes セキュリティ	Security Hub CSPM 経由で結果を送信する
サモロジック – マシンデータ分析	Security Hub CSPM 経由で結果を送信する
Symantec – クラウドワークロード保護	Security Hub CSPM 経由で結果を送信する
Tenable.io	Security Hub CSPM 経由で結果を送信する
Trend Micro – Cloud One	Security Hub CSPM 経由で結果を送信する
Vectra – Cognito Detect	Security Hub CSPM 経由で結果を送信する
Wiz	Security Hub CSPM 経由で結果を送信する
Caveonix – Caveonix クラウド	Security Hub CSPM を介して検出結果を送受信する
Cloud Custodian	Security Hub CSPM を介して検出結果を送受信する
DisruptOps	Security Hub CSPM を介して検出結果を送受信する
キオン	Security Hub CSPM を介して検出結果を送受信する
Turbot	Security Hub CSPM を介して検出結果を送受信する

Note

このリストには、書き込み時に結果を送信する Security Hub パートナー統合が反映されています。AWS Security Hub は定期的に新しいパートナー統合を追加するため、利用可能なパートナーのup-to-dateリストについては、[「Security Hub ユーザーガイド」の「Security Hub CSPM とのサードパーティー製品統合」](#)を参照してください。AWS

カスタムソース

カスタムアプリケーションログでは、次のいずれかの方法を使用して、独自のデータソースの分類を定義できます。

- ロググループタグ – キー `cw:datasource:name` と `cw:datasource:type` を使用してロググループにタグを追加し、ロググループに取り込まれたすべてのログのデータソース名とタイプを指定します。
- パイプライン設定 – アプリケーションログを取り込むときに、ログ処理パイプラインを介してデータソース情報を設定します。

Note

AWS サービスログとの競合を避けるため、カスタムデータソース名は「aws」または「amazon」で始めることはできません。

CloudWatch Logs Insights を使用したログデータの分析

CloudWatch Logs Insights を使用すると、Amazon CloudWatch Logs のログデータをインタラクティブに検索し分析することができます。クエリを実行して、運用上の問題により効率的かつ効果的に対応できます。ロググループを使用したクエリに加えて、ファセット、データソース、データ型を使用してクエリを実行できます。問題が発生した場合は、CloudWatch Logs Insights を使用して、潜在的な原因を特定し、デプロイされた修正を検証することができます。ダッシュボードに追加されたクエリを含め、アカウントあたりの同時 CloudWatch Logs Insights QL は 100 に制限されています。さらに、OpenSearch Service PPL または OpenSearch Service SQL のいずれかに対して 15 OpenSearch の同時クエリを実行できます。

CloudWatch Logs Insights は、クエリに使用できる 3 つのクエリ言語をサポートしています。

- いくつかのシンプルで強力なコマンドを備えた目的別の Logs Insights クエリ言語 (Logs Insights QL)。
- OpenSearch Service パイプ処理言語 (PPL)。OpenSearch PPL を使用すると、パイプ (|) で区切られた一連のコマンドを使用してログを分析できます。

OpenSearch PPL を使用すると、パイプ処理されたコマンドを使用してデータを取得、クエリ、分析できるため、複雑なクエリの理解と構成が容易になります。構文を使用すると、コマンドを連鎖してデータの変換と処理を行えます。PPL を使用すると、データをフィルターして集計し、豊富な一連の算術、文字列、日付、条件付きなどの関数を用いて分析できます。

- OpenSearch Service 構造化クエリ言語 (SQL)。OpenSearch SQL クエリを使用すると、宣言的な方法でログを分析できます。SELECT、FROM、WHERE、GROUP BY、HAVING などのコマンドや、SQL で使用できるその他のさまざまなコマンドや関数を使用できます。ロググループ間で JOINS を実行し、サブクエリを使用してログ間でデータを関連付け、一連の豊富な JSON、算術、文字列、条件付きなどの SQL 関数を使用してログに対する強力な分析を実行できます。

SQL コマンドまたは PPL コマンドを使用する場合は、フィールドをバッククォートの特殊文字 (非アルファベットおよび非数値) で囲んでクエリを正常に実行してください。例えば、@message、Operation.Export、Test::Field をバッククォートで囲みます。フィールド名がアルファベットのみの場合はバッククォートで囲む必要はありません。

CloudWatch Logs Insights には、任意のクエリ言語で利用できる以下の機能があります。

- Amazon Route 53、Amazon VPC などのサービスからの AWS ログの[ログフィールド](#)、および[ログイベント](#)を JSON として出力するアプリケーションまたはカスタムログの自動検出。AWS Lambda AWS CloudTrail
- [フィールドインデックス](#)を作成してコストを削減し、特に多数のロググループまたはログイベントのクエリで結果を高速化。ログイベントで一般的なフィールドのフィールドインデックスを作成したら、クエリで使用できます。このクエリは、インデックス付きフィールドが含まれていないことがわかっているログイベントの処理をスキップし、データ処理量を抑制します。

 Note

filterIndex コマンドは Logs Insights QL でのみ使用できます。

- ログイベントの[パターンの検出と分析](#)。パターンは、ログフィールド間で繰り返される共有テキスト構造です。クエリの結果を表示するときは、[パターン]タブを選択して、結果のサンプルに基づいて CloudWatch Logs が検出したパターンを表示できます。
- [クエリの保存](#)、クエリ履歴の表示、保存されたクエリの再実行、[パラメータを使用した保存されたクエリの使用](#)。
- [ダッシュボードへのクエリの追加](#)。
- [によるクエリ結果の暗号化 AWS Key Management Service](#)。
- [自然言語を使用したクエリ生成](#)により、自然言語を使用して CloudWatch Logs Insights クエリを作成できます。探しているデータについて質問あるいは説明すると、AI はそのプロンプトに基づいてクエリを生成し、そのクエリの仕組みを詳しく説明します。
- [ファセットを使用して、ログをグループ化、フィルタリング、インタラクティブに探索します](#)。
- [ログを囲む](#)と、クエリ結果の特定のログレコードの前後にログ行を表示して、重要なログイベントに関するコンテキストを即座に取得できます。選択したレコードの前後に 5、10、20、50、または 100 ログ行を表示するように範囲を設定し、周囲のログ内の特定のキーワードを検索できます。

以下の CloudWatch Logs Insights 機能は、Logs Insights QL を使用する場合にのみサポートされます。

- ロググループのログイベントと以前の期間のログイベントを比較する[比較クエリ](#)。

⚠ Important

CloudWatch Logs Insights は、ロググループの作成時刻より前のタイムスタンプを持つログイベントにはアクセスすることができません。

CloudWatch のクロスアカウントオブザーバビリティでモニタリングアカウントとして設定されたアカウントにサインインしている場合、このモニタリングアカウントにリンクされているソースアカウントのロググループで CloudWatch Logs Insights クエリを実行できます。異なるアカウントにある複数のロググループをクエリするクエリを実行できます。詳細については、「[CloudWatch のクロスアカウントオブザーバビリティ](#)」を参照してください。

また、Logs Insights QL を使用してクエリを作成する場合は、自然言語を使用して CloudWatch Logs Insights クエリを作成することもできます。そのためには、どのようなデータを探しているのかを質問したり、具体的に説明したりしてみてください。AI 支援機能が働いて、プロンプトに基づいてクエリが生成され、クエリの仕組みを説明した文が 1 行ずつ表示されます。詳細については、「[Use natural language to generate and update CloudWatch Logs Insights queries](#)」を参照してください。

サポートされているクエリ言語のいずれかを使用したクエリは、完了していない場合、60 分後にタイムアウトします。クエリ結果は 7 日間利用できます。

CloudWatch Logs Insights クエリでは、クエリ言語に関係なく、クエリされたデータの量に基づいて料金が発生します。詳細については、「[Amazon CloudWatch 料金表](#)」をご覧ください。

CloudWatch Logs Insights を使用して、2018 年 11 月 5 日以降に CloudWatch Logs に送信されたログデータを検索できます。

⚠ Important

ネットワークセキュリティチームがウェブソケットの使用を許可しない場合は、現在 CloudWatch コンソールの CloudWatch Logs Insights 部分にアクセスすることはできません。API を使用して CloudWatch Logs Insights のクエリ機能を使用できます。詳細については、Amazon CloudWatch Logs API リファレンスの「[StartQuery](#)」を参照してください。

内容

- [サポートされているクエリ言語](#)
- [自然言語を使用した CloudWatch Logs Insights クエリの生成と更新](#)

- [サポートされるログと検出されるフィールド](#)
- [フィールドインデックスを作成してクエリパフォーマンスを改善し、スキャン量を削減する](#)
- [ファセットを使用してログをグループ化および探索する](#)
- [CloudWatch Logs Insights での周囲のログの表示](#)
- [パターン分析](#)
- [CloudWatch Logs Insights クエリの保存と再実行](#)
- [クエリをダッシュボードに追加する、またはクエリ結果をエクスポートする](#)
- [実行中のクエリまたはクエリ履歴を表示する](#)
- [でクエリ結果を暗号化する AWS Key Management Service](#)
- [CloudWatch Logs Insights クエリ結果から自然言語の概要を生成する](#)

サポートされているクエリ言語

以下のセクションでは、各クエリ言語でサポートされるコマンドがリストアップされています。また、構文形式の説明とサンプルクエリも提供されています。

トピック

- [CloudWatch Logs Insights クエリ言語 \(Logs Insights QL\)](#)
- [OpenSearch パイプ処理言語 \(PPL\)](#)
- [OpenSearch 構造化クエリ言語 \(SQL\)](#)

CloudWatch Logs Insights クエリ言語 (Logs Insights QL)

このセクションには、Logs Insights QL のコマンドと関数の完全なドキュメントが含まれています。また、この言語のサンプルクエリも含まれています。

使用できる他のクエリ言語について詳しくは、「[OpenSearch Service PPL](#)」、「[OpenSearch Service SQL](#)」、「[CloudWatch Metrics Insights](#)」を参照してください。

トピック

- [CloudWatch Logs Insights 言語のクエリ構文](#)
- [Logs Insights QL の使用を開始する: クエリチュートリアル](#)
- [サンプルクエリ](#)

- [\(diff\) を以前の時間範囲と比較する](#)
- [グラフでログデータを視覚化する](#)

CloudWatch Logs Insights 言語のクエリ構文

このセクションでは、Logs Insights QL の詳細について説明します。クエリ構文は、一般的な関数、算術演算と比較演算、正規表現など、さまざまな関数とオペレーションをサポートしています。

Important

大規模なクエリを実行して過剰な料金が発生するのを防ぐため、次のベストプラクティス留意してください。

- 各クエリに必要なロググループのみを選択します。
- クエリには、常にできるだけ狭い時間範囲を指定します。
- コンソールを使用してクエリを実行する場合は、CloudWatch Logs Insights コンソールページを閉じる前にすべてのクエリをキャンセルします。それ以外の場合、クエリは完了するまで実行され続けます。
- CloudWatch Logs Insights ウィジェットをダッシュボードに追加するときは、更新ごとに新しいクエリが開始されるため、ダッシュボードが高頻度で更新されないようにしてください。

複数のコマンドを含むクエリを作成するときは、コマンドをパイプ文字 (|) で区切ります。

コメントを含むクエリを作成するときは、コメントをハッシュ文字 (#) で区切ります。

Note

CloudWatch Logs Insights は、さまざまなログタイプのフィールドを自動で検出し、@ 文字で始まるフィールドを生成します。これらのフィールドの詳細については、「Amazon CloudWatch ユーザーガイド」の「[サポートされるログと検出されるフィールド](#)」を参照してください。

次の表で、各コマンドについて簡単に説明します。この表の後に、各コマンドについての詳細な説明と例を示します。

Note

すべての Logs Insights QL クエリコマンドは、標準ログクラスのロググループでサポートされています。低頻度アクセスログクラスのロググループは `pattern`、`diff`、`unmask` を除くすべての Logs Insights QL クエリコマンドをサポートします。

<u>anomaly</u>	機械学習を使用して、ログデータの異常なパターンを識別します。
<u>display</u>	クエリ結果に特定のフィールドを表示します。
<u>fields</u>	クエリ結果に特定のフィールドを表示し、クエリで使用するフィールド値を変更したり新しいフィールドを作成したりするときに使用できる関数と演算をサポートします。
<u>filter</u>	クエリをフィルタリングし、1 つ以上の条件に一致するログイベントのみを返します。
<u>filterIndex</u>	フィールドインデックスに記載されているフィールドでインデックス作成され、そのフィールドインデックスの値も含まれるロググループのみをスキャンするようにクエリに強制します。これにより、このフィールドインデックスのクエリで指定された値を含むこれらのロググループのログイベントのみをスキャンしようとすることで、スキャンされるボリュームが減少します。 このコマンドは、低頻度アクセスログクラスのロググループではサポートされていません。
<u>pattern</u>	自動的にログデータをパターンにクラスター化します。パターンは、ログフィールド間で繰り返される共有テキスト構造です。CloudWatch Logs Insights には、ログイベントで検出されたパターンを分析する方法が用意されています。詳細については、「 パターン分析 」を参照してください。
<u>diff</u>	リクエストされた期間に検出されたログイベントと、以前の期間と同じ長さのログイベントを比較して、傾向を検索し、特定のログイベントが新しいかどうかを確認することができます。

<u>parse</u>	ログフィールドからデータを抽出し、クエリで処理できる抽出フィールドを作成します。 parse は、ワイルドカードを使用する glob モードと正規表現の両方をサポートします。
<u>sort</u>	返されたログイベントを昇順 (asc) または降順 (desc) で表示します。
<u>SOURCE</u>	SOURCE をクエリに含めると、クエリに含めるロググループ名のプレフィックス、アカウント識別子、ロググループクラスに基づいて大量のロググループを指定するのに役立ちます。このコマンドは、CloudWatch コンソールではなく、AWS CLI またはプログラムでクエリを作成する場合にのみサポートされます。
<u>stats</u>	ログフィールドの値を使って集計した統計を算出します。
<u>limit</u>	クエリで返すログイベントの最大数を指定します。 sort で「上位 20 件」または「最新の 20 件」の結果を返すソートと一緒に使用すると便利です。
<u>dedup</u>	指定したフィールドの特定の値に基づいて、重複した結果を削除します。
<u>unmask</u>	データ保護ポリシーにより一部のコンテンツがマスクされているログイベントの、すべてのコンテンツを表示します。ロググループのデータ保護の詳細については、「 機密性の高いログデータをマスキングで保護する 」を参照してください。
<u>unnest</u>	入力として取得したリストをフラット化し、リスト内の各要素に対して 1 つのレコードを持つ複数のレコードを生成します。
<u>lookup</u>	フィールド値を一致させることで、ルックアップテーブルのデータを含むログイベントを強化します。ルックアップテーブルを使用して、ユーザーの詳細、アプリケーション名、製品情報などの参照データをクエリ結果に追加します。
<u>join</u>	ソースロググループからのログイベントを、一致するフィールドに基づいて別のロググループまたはクエリ結果からのイベントと組み合わせます。join コマンドを使用して、一致するリクエスト識別子やトランザクション IDs。

subqueries	サブクエリは、別のクエリへの入力として使用できるネストされた Logs Insights クエリです。サブクエリを使用すると、後続のコマンドで消費される中間結果セットを取得できます。
その他の演算と関数	CloudWatch Logs Insightsは、比較、計算、日時、数値、文字列、IP アドレス、一般的な関数と演算も多数サポートしています。

以下のセクションでは、CloudWatch Logs Insights のクエリコマンドについてさらに詳しく説明します。

トピック

- [ログクラスでサポートされている Logs Insights QL コマンド](#)
- [異常](#)
- [display](#)
- [fields](#)
- [フィルター](#)
- [filterIndex](#)
- [SOURCE](#)
- [pattern](#)
- [diff](#)
- [parse](#)
- [sort](#)
- [stats](#)
- [制限](#)
- [重複排除](#)
- [マスクを外す](#)
- [ネスト解除](#)
- [lookup](#)
- [結合](#)
- [subqueries](#)
- [ブール、比較、数値、日時、その他の関数](#)
- [特殊文字を含むフィールド](#)

• [クエリでのエイリアスとコメントの使用](#)

ログクラスでサポートされている Logs Insights QL コマンド

すべての Logs Insights QL クエリコマンドは、標準ログクラスのロググループでサポートされています。低頻度アクセスログクラスのロググループは `pattern`、`diff`、`filterIndex`、`unmask` を除くすべてのクエリコマンドをサポートします。

異常

`anomaly` を使用すると、機械学習を用いてログデータ内の異常なパターンと潜在的な問題を自動的に特定します。

`anomaly` コマンドは既存の `pattern` 機能を拡張し、高度な分析を活用してログデータの潜在的な異常を特定します。`anomaly` を使用すると、ログの異常なパターンや動作を自動的に表示することで運用上の問題の特定と解決にかかる時間を短縮できます。

`anomaly` コマンドは [pattern](#) コマンドと連携し、まずログパターンを識別してからそれらのパターン内の異常を検出します。さらに、`anomaly` を [filter](#) または [sort](#) コマンドと組み合わせ、データの特定のサブセットの異常検出に焦点を当てることもできます。

異常コマンド入力

`anomaly` コマンドは通常、[pattern](#) コマンドの後に使用され、ログデータで識別されるパターンを分析します。このコマンドは追加のパラメータを必要とせず、クエリ内の前述のコマンドからの出力を分析します。

特定される異常のタイプ

`anomaly` コマンドは、次の 5 つの異なるタイプの異常を識別します。

- パターン頻度の異常: アプリケーションが通常よりも多くのエラーメッセージの生成を開始する場合など、特定のログパターンの異常な頻度。
- 新しいパターンの異常: ログに表示される新しいタイプのエラーやメッセージを示す、これまでに見られなかったログパターン。
- トークンバリエーションの異常: ログメッセージのコンテンツの不測の変更。予想されるログ形式の異常なバリエーションを示している可能性があります。
- 数値トークンの異常: パフォーマンスの潜在的な問題やメトリクスの不測の変動を検出するのに役立つ、ログ内の数値の異常な変化。

- HTTP エラーコードの異常: HTTP エラーレスポンスに関連するパターン。特にウェブアプリケーションと API をモニタリングする場合に便利です。

異常コマンド出力

`anomaly` コマンドは、入力データからのすべてのフィールドを保持し、ログデータの異常なパターンを識別するのに役立つ異常検出結果を追加します。

例

次のコマンドは、ログデータのパターンを識別し、それらのパターン内の異常を検出します。

```
fields @timestamp, @message
| pattern @message
| anomaly
```

`anomaly` コマンドはフィルタリングと併用し、特定のログタイプに焦点を当てることができます。

```
fields @timestamp, @message
| filter @type = "REPORT"
| pattern @message
| anomaly
```

`anomaly` コマンドをソートと組み合わせて結果を整理できます。

```
fields @timestamp, @message
| filter @type = "ERROR"
| pattern @message
| anomaly
| sort @timestamp desc
```

display

`display` を使用して、クエリ結果の特定のフィールドを表示します。

`display` コマンドは、指定したフィールドのみを表示します。クエリに複数の `display` コマンドが含まれている場合、クエリ結果には、最後の `display` コマンドで指定したフィールドのみが表示されます。

例: 1 つのフィールドを表示する

コードスニペットは、解析コマンドを使用して @message からデータを抽出し、抽出フィールド loggingType および loggingMessage を作成するクエリの例を示します。クエリは、loggingType の値が ERROR であるすべてのログイベントを返します。display は、クエリ結果に loggingMessage の値のみを表示します。

```
fields @message
| parse @message "[]" * as loggingType, loggingMessage
| filter loggingType = "ERROR"
| display loggingMessage
```

Tip

クエリで 1 回だけ display を使用します。クエリで display を 2 回以上使用すると、クエリの結果には、使用されている display コマンドの直近の実行で指定されたフィールドが表示されます。

fields

fields を使用して、クエリ結果の特定のフィールドを表示します。

クエリに複数の fields コマンドが含まれ、display コマンドが含まれていない場合は、結果に、fields コマンドで指定されたすべてのフィールドが表示されます。

例: 特定のフィールドを表示する

以下は、20 個のログイベントを返し、それらを降順で表示するクエリの例です。@timestamp と @message の値がクエリ結果に表示されます。

```
fields @timestamp, @message
| sort @timestamp desc
| limit 20
```

フィールド値を変更したり、クエリで利用できる新しいフィールドを作成したりするため、fields がサポートしている異なる関数や演算を使用するときは、display ではなく fields を使用します。

fields コマンドを as キーワードと共に使用すると、ログイベント内の関数とフィールドを使用して抽出フィールドを作成できます。例えば、fields ispresent as isRes はクエリの残りの部分で利用できる isRes という名前の抽出フィールドを作成します。

フィルター

`filter` を使用して、1 つ以上の条件に一致するログイベントを取得します。

例: 1 つの条件を使用してログイベントをフィルタリングする

コードスニペットは、`range` の値が 3000 より大きいすべてのログイベントを返すクエリの例を示します。このクエリは、結果を 20 個のログイベントに制限し、ログイベントを `@timestamp` 別に降順で並べ替えます。

```
fields @timestamp, @message
| filter (range>3000)
| sort @timestamp desc
| limit 20
```

例: 複数の条件を使用してログイベントをフィルタリングする

キーワード `and` および `or` を使用して、複数の条件を組み合わせることができます。

コードスニペットは、`range` の値が 3000 より大きく、`accountId` の値が 123456789012 に等しいログイベントを返すクエリの例を示します。このクエリは、結果を 20 個のログイベントに制限し、ログイベントを `@timestamp` 別に降順で並べ替えます。

```
fields @timestamp, @message
| filter (range>3000 and accountId=123456789012)
| sort @timestamp desc
| limit 20
```

インデックス付きフィールドとフィルターコマンド

ロググループのフィールドインデックスを作成している場合は、それらのフィールドインデックスを活用して `filter` クエリの効率性を高め、スキャンされるボリュームを減らすことができます。例えば、`requestId` のフィールドインデックスを作成したとします。次に、`filter requestId = value` または `filter requestId IN [value, value, ...]` は、インデックス付きフィールドを含まないことがわかっているログイベントの処理をスキップしようとしています。そのインデックス付きフィールドが含まれていることがわかっているログイベントのみをスキャンしようとする、スキャン量を減らすことができ、クエリが高速になります。

フィールドインデックスおよびその作成方法の詳細については「[フィールドインデックスを作成してクエリパフォーマンスを改善し、スキャン量を削減する](#)」を参照してください。

 Important

フィールドインデックスの改善による恩恵があるのは `filter fieldName =...` と `filter fieldName IN...` を持つクエリのみです。`filter fieldName like` を持つクエリの場合、インデックスは使用されず、選択したロググループのすべてのログイベントを常にスキャンします。

例: インデックスを使用して、特定のリクエスト ID に関連するログイベントを検索する

この例では、`requestId` でフィールドインデックスが作成されていることを前提としています。このフィールドインデックスを使用するロググループの場合、クエリはフィールドインデックスを活用して最小量のログイベントをスキャンし、値が `123456` の `requestId` を持つイベントを見つけようとしています。

```
fields @timestamp, @message
| filter requestId = "123456"
| limit 20
```

フィルターコマンドの一致と正規表現

フィルターコマンドは、正規表現の使用をサポートします。以下の比較演算子 (`=`、`!`、`=`、`<`、`<=`、`>`、`>=`) とブール演算子 (`and`、`or`、および `not`) を使用できます。

キーワード `in` を使用して集合要素関係をテストし、配列内の要素をチェックできます。配列の要素をチェックするには、`in` の後に対象の配列を配置します。ブール演算子 `not` および `in` を使用できます。`in` を使用するクエリを作成して、フィールドに文字列の一致があるログイベントを返すことができます。フィールドは完全な文字列でなければなりません。例えば、次のコードスニペットは、フィールド `logGroup` が完全な文字列 `example_group` であるログイベントを返すために `in` を使用するクエリを示しています。

```
fields @timestamp, @message
| filter logGroup in ["example_group"]
```

キーワードフレーズ `like` および `not like` を使用して、部分文字列を一致させることができます。正規表現の演算子 `=~` を使用して部分文字列を一致させることができます。`like` および `not like` で部分文字列を一致させるには、単一引用符または二重引用符で一致させたい部分文字列を囲みます。正規表現パターンは、`like` および `not like` と共に使用できます。部分文字列を正

規表現の演算子と一致させるには、一致させたい部分文字列をスラッシュで囲みます。次の例には、`filter` コマンドを使用して部分文字列を照合する方法を示すコードスニペットが含まれます。

例: 部分文字列の一致

以下の例では、`f1` に単語 `Exception` が含まれているログイベントを返します。これら 3 つの例すべてで、大文字と小文字が区別されます。

最初の例では、部分文字列を `like` と一致させます。

```
fields f1, f2, f3
| filter f1 like "Exception"
```

2 番目の例では、部分文字列を `like` および正規表現パターンと一致させます。

```
fields f1, f2, f3
| filter f1 like /Exception/
```

3 番目の例では、部分文字列を正規表現と一致させます。

```
fields f1, f2, f3
| filter f1 =~ /Exception/
```

例: 部分文字列をワイルドカードと一致させる

ピリオド記号 (`.`) を正規表現のワイルドカードとして使用して、部分文字列に一致させることができます。次の例では、クエリは `f1` の値が文字列 `ServiceLog` で始まる一致を返します。

```
fields f1, f2, f3
| filter f1 like /ServiceLog./
```

ピリオド記号 (`.*`) の後にアスタリスク記号を置いて、できるだけ多くの一致を返す貪欲な量指定子を作成することができます。例えば、次のクエリは `f1` の値が文字列 `ServiceLog` で始まるだけでなく、文字列 `ServiceLog` も含む一致を返します。

```
fields f1, f2, f3
| filter f1 like /ServiceLog.*/
```

考えられる一致は、次のようにフォーマットされている可能性があります:

- ServiceLogSampleApiLogGroup
- SampleApiLogGroupServiceLog

例: 一致から部分文字列を除外する

次の例は、f1 に Exception という単語が含まれないログイベントを返すクエリを示しています。この例では大文字と小文字が区別されます。

```
fields f1, f2, f3
| filter f1 not like "Exception"
```

例: 大文字と小文字を区別しないパターンで部分文字列を一致させる

大文字と小文字を区別しない部分文字列を、like および正規表現と一致させることができます。次のパラメータ (?i) を、一致させる部分文字列の前に配置します。次の例は、f1 に単語 Exception または exception が含まれるログイベントを返すクエリを示しています。

```
fields f1, f2, f3
| filter f1 like /(?i)Exception/
```

filterIndex

filterIndex を使用してインデックス付きデータのみを返すには、クエリで指定したフィールドでインデックス付けされたロググループのみをスキャンするようクエリに強制します。このフィールドでインデックス付けされるこれらのロググループの場合、インデックス付けされたフィールドのクエリで指定されたフィールドを含む任意のログイベントを持たないロググループをスキップすることで、クエリがさらに最適化されます。さらに、このフィールドインデックスのクエリで指定された値に一致するこれらのロググループのログイベントのみをスキャンしようとすることで、スキャンされるボリュームが減少します。フィールドインデックスおよびその作成方法の詳細については「[フィールドインデックスを作成してクエリパフォーマンスを改善し、スキャン量を削減する](#)」を参照してください。

インデックス付きフィールドで filterIndex を使用すると、フィールドインデックスを持つロググループとログイベントに実際の検索スペースを制限することで、ペタバイトのログデータを含むロググループを効率的にクエリできます。

例えば、アカウントの一部のロググループで IPAddress のフィールドインデックスを作成したとします。その後、次のクエリを作成してアカウント内のすべてのロググループをクエリし、IPAddress フィールドの値 198.51.100.0 を含むログイベントを検索できます。

```
fields @timestamp, @message
| filterIndex IPAddress = "198.51.100.0"
| limit 20
```

filterIndex コマンドにより、このクエリは IPAddress においてインデックス付けされていないすべてのロググループのスキップを試みます。さらに、インデックス付けされたロググループ内で、クエリは IPAddress フィールドを持つもののそのフィールドの値として 198.51.100.0 が観測されないログイベントをスキップします。

IN 演算子を使用して、インデックス付きフィールドの複数の値のいずれかに結果を展開します。次の例では、IPAddress フィールドで値 198.51.100.0 または 198.51.100.1 を含むログイベントを検索します。

```
fields @timestamp, @message
| filterIndex IPAddress in ["198.51.100.0", "198.51.100.1"]
| limit 20
```

CloudWatch Logs は、標準ログクラスのすべてのロググループにデフォルトのフィールドインデックスを提供します。デフォルトのフィールドインデックスは、次のフィールドで自動的に使用できます。

- @logStream
- @aws.region
- @aws.account
- @source.log
- @data_source_name
- @data_source_type
- @data_format
- traceId
- severityText
- attributes.session.id

CloudWatch Logs は、特定のデータソース名とタイプの組み合わせのデフォルトのフィールドインデックスも提供します。デフォルトのフィールドインデックスは、次のデータソース名とタイプの組み合わせで自動的に使用できます。

データソース名とタイプ	デフォルトのフィールドインデックス
amazon_vpc.flow	action logStatus region flowDirection type
amazon_route53_resolver_query	query_type transport rcode
aws_waf.access	action httpRequest.country
aws_cloudtrail.data aws_cloudtrail.management	eventSource eventName awsRegion userAgent errorCode eventType managementEvent readOnly eventCategory requestId

デフォルトのフィールドインデックスは、ポリシー内で定義した任意のカスタムフィールドインデックスに追加されます。デフォルトのフィールドインデックスは、[フィールドインデックスクォータ](#)にはカウントされません。

filterIndex と filter の比較

filterIndex と filter の違いを説明するために、次のクエリ例を検討してください。IPAddress のフィールドインデックスを 4 つのロググループ向けに作成したものの、5 番目のロググループには作成していないとします。filterIndex を使用する次のクエリは、フィールドのインデックスが付けられていないロググループのスキップをスキップします。インデックス付きロググループごとに、インデックス付きフィールドを持つログイベントのみのスキップを試行します。また、フィールドインデックスの作成後からの結果のみを返します。

```
fields @timestamp, @message
| filterIndex IPAddress = "198.51.100.0"
| limit 20
```

対照的に、同じ 5 つのロググループのクエリに filterIndex の代わりに filter を使用する場合、クエリはインデックス付きロググループの値を含むログイベントだけではなく、インデックス化されていない 5 番目のロググループもスキップし、その 5 番目のロググループ内のすべてのログイベントをスキップします。

```
fields @timestamp, @message
| filter IPAddress = "198.51.100.0"
| limit 20
```

SOURCE

クエリSOURCEに を含めることは、AWS CLI または API を使用してクエリを作成するときに、クエリに含めるロググループやデータソースを指定するのに役立ちます。SOURCE コマンドは、CloudWatch コンソールではなく、AWS CLI および API でのみサポートされます。CloudWatch コンソールを使用してクエリを開始する場合は、コンソールインターフェイスを使用してロググループを指定します。

クエリロググループ

SOURCE を使用してクエリするロググループを指定するには、次のキーワードを使用できます。

- namePrefix は、指定した文字列で始まる名前を持つロググループに対してクエリを実行します。これを省略すると、すべてのロググループがクエリされます。

リストには最大 5 つのプレフィックスを含めることができます。

- `accountIdentifier` は、指定された AWS アカウントのロググループに対してクエリを実行します。これは、モニタリングアカウントでクエリを実行する場合にのみ機能します。これを省略した場合、デフォルトでは、リンクされたすべてのソースアカウントと現在のモニタリングアカウントをクエリします。クロスアカウントオブザーバビリティについては「[CloudWatch cross-account observability](#)」を参照してください。

リストには最大 20 個のアカウント識別子を含めることができます。

- `logGroupClass` は、標準アクセスまたは低頻度アクセスのいずれかの指定されたログクラスにあるロググループに対してクエリを実行します。これを省略した場合、デフォルトの標準ログクラスが使用されます。ログクラスの詳細については、「[ログクラス](#)」を参照してください。

この方法でクエリを実行するために多数のロググループを指定できるため、作成したフィールドインデックスを利用するクエリでのみ `SOURCE` を使用することをお勧めします。インデックスフィールドの詳細については、「[フィールドインデックスを作成してクエリパフォーマンスを改善し、スキャン量を削減する](#)」を参照してください。

次の例では、アカウント内のすべてのロググループを選択します。これがモニタリングアカウントの場合、モニタリング全体のロググループとすべてのソースアカウントが選択されます。ロググループの総数が 10,000 を超える場合は、別のロググループ選択方法を使用してロググループの数を減らすように求めるエラーが表示されます。

```
SOURCE logGroups()
```

次の例では、111122223333 ソースアカウント内のロググループを選択します。CloudWatch のクロスアカウントオブザーバビリティのモニタリングアカウントでクエリを開始する場合は、すべてのソースアカウントだけではなくそのモニタリングアカウントのロググループもデフォルトで選択されます。

```
SOURCE logGroups(accountIdentifiers:['111122223333'])
```

次の例では、名前プレフィックスに基づいてロググループを選択します。

```
SOURCE logGroups(namePrefix: ['namePrefix1', 'namePrefix2'])
```

次の例では、低頻度アクセスログクラスのすべてのロググループを選択します。`class` 識別子を含めない場合、クエリはデフォルトである標準ログクラスのロググループにのみ適用されます。

```
SOURCE logGroups(class: ['INFREQUENT_ACCESS'])
```

次の例では、特定の名前プレフィックスで始まり、標準ログクラスにある 111122223333 アカウントのロググループを選択します。Standard がデフォルトのログクラス値であるため、このクラスはコマンドで説明されません。

```
SOURCE logGroups(accountIdentifiers:['111122223333'], namePrefix: ['namePrefix1',  
'namePrefix2'])
```

最後の例は、SOURCE コマンドで start-query AWS CLI コマンドを使用する方法を示しています。

```
aws logs start-query  
--region us-east-1  
--start-time 1729728200  
--end-time 1729728215  
--query-string "SOURCE logGroups(namePrefix: ['Query']) | fields @message | limit 5"
```

データソースのクエリ

SOURCE を使用してクエリするデータソースを指定するには、dataSource キーワードを使用できます。リストには最大 10 個のデータソースを含めることができます。

次の例では、amazon_vpc.flow データソースを選択します。

```
SOURCE dataSource(['amazon_vpc.flow'])
```

次の例では、amazon_vpc.flow データソースを選択し、ロググループ名のプレフィックスに基づいてロググループを制限します。

```
SOURCE dataSource(['amazon_vpc.flow']) logGroups(namePrefix: ['namePrefix1'])
```

pattern

pattern を使用してログデータを自動的にパターンにクラスター化します。

パターンは、ログフィールド間で繰り返される共有テキスト構造です。pattern を使用して新たな傾向を発見することや既知のエラーをモニタリングすることに加えて、頻繁に発生するログラインや

コストの高いログラインを特定することができます。CloudWatch Logs Insights は、ログイベント内のパターンを検索してさらに分析するために使用できるコンソールエクスペリエンスも提供します。詳細については、「[パターン分析](#)」を参照してください。

pattern コマンドは一般的なパターンを自動的に識別するため、ログを検索して分析するための出発点として使用できます。また、pattern を [filter](#)、[parse](#)、または [sort](#) コマンドと組み合わせて、より微調整されたクエリでパターンを識別することもできます。

パターンコマンド入力

pattern コマンドでは、@message フィールド、[parse](#) コマンドを使用して作成された抽出フィールド、または 1 つ以上の [String 関数](#) を使用して操作された文字列のいずれかの入力が预期されます。

CloudWatch Logs が動的トークンの表すデータのタイプを推測できない場合は、<トークン - ##> として表示され、## は他の動的トークンと比較して、このトークンがパターン内のどこに表示されるかを示します。

動的トークンの一般的な例には、エラーコード、IP アドレス、タイムスタンプ、リクエスト ID があります。

パターンコマンド出力

pattern コマンドは以下の出力を生成します。

- @pattern: ログイベントフィールド間で繰り返される共有テキスト構造。リクエスト ID やタイムスタンプなど、パターン内の異なるフィールドはトークンによって表現されます。CloudWatch Logs が動的トークンの表すデータのタイプを判断できる場合、トークンは <string-number> として表示されます。### は、トークンが表すデータのタイプの説明です。## は、他の動的トークンと比較して、このトークンがパターン内のどこに表示されるかを示します。

CloudWatch Logs は、名前を含むログイベントのコンテンツの分析に基づいて、その名前の文字列部分を割り当てます。

CloudWatch Logs が動的トークンの表すデータのタイプを推測できない場合は、<トークン - ##> として表示され、## は他の動的トークンと比較して、このトークンがパターン内のどこに表示されるかを示します。

例えば、[INFO] Request time: <Time-1> ms はログメッセージ [INFO] Request time: 327 ms の出力候補です。

- @ratio: 選択した期間のログイベントと、識別されたパターンに一致する特定のロググループのログイベントの割合。例えば、選択したロググループと期間のログイベントの半分がパターンと一致する場合、@ratio は 0.50 を返します。
- @sampleCount: 選択した期間のログイベントと、識別されたパターンに一致する特定のロググループのログイベントの数。
- @severityLabel: ログの重要度またはレベル。ログに含まれる情報の種類を示します。Error、Warning、Info、Debug などが該当します。

例

次のコマンドは、選択した時間範囲内の指定されたロググループ内の構造が似ているログを識別し、パターンと数でグループ化します。

```
pattern @message
```

pattern コマンドは [filter](#) コマンドと組み合わせて使用できます

```
filter @message like /ERROR/  
| pattern @message
```

pattern コマンドは、[parse](#) および [sort](#) コマンドと共に使用できます。

```
filter @message like /ERROR/  
| parse @message 'Failed to do: *' as cause  
| pattern cause  
| sort @sampleCount asc
```

diff

リクエストされた期間に検出されたログイベントと、以前の期間と同じ長さのログイベントを比較します。これにより、傾向を検索し、特定のログイベントが新しいかどうかを確認できます。

diff コマンドに修飾子を追加して、比較する期間を指定します。

- diff は、現在選択されている時間範囲のログイベントを直前の時間範囲のログイベントと比較します。
- diff previousDay は、現在選択されている時間範囲のログイベントを、前日の同じ時刻のログイベントと比較します。

- `diff previousWeek` は、現在選択されている時間範囲のログイベントを、前週の同じ時刻のログイベントと比較します。
- `diff previousMonth` は、現在選択されている時間範囲のログイベントを、前月と同じ時刻のログイベントと比較します。

詳細については、「[\(diff\) を以前の時間範囲と比較する](#)」を参照してください。

parse

`parse` を使用してログフィールドからデータを抽出し、クエリで処理できる抽出フィールドを作成します。ログイベントが指定されたパターンと一致しない場合、結果には表示されますが、抽出されたフィールドは表示されません。は、ワイルドカードを使用した `glob` モードと正規表現の両方 **`parse`** をサポートしています。正規表現の構文の詳細については、「[サポートされている正規表現 \(regex\) 構文](#)」を参照してください。

ネストされた JSON フィールドは正規表現で解析できます。

例: ネストされた JSON フィールドの解析

コードスニペットは、取り込み中にフラット化された JSON ログイベントを解析する方法を示します。

```
{'fieldsA': 'logs', 'fieldsB': [{'fA': 'a1'}, {'fA': 'a2'}]}
```

コードスニペットは、`fieldsA` および `fieldsB` の値を抽出し、抽出フィールド `fld` および `array` を作成する正規表現を含むクエリを示します。

```
parse @message "'fieldsA': '*', 'fieldsB': ['*']" as fld, array
```

名前付きキャプチャグループ

正規表現で **`parse`** を使用すると、名前付きキャプチャグループを使用してパターンをフィールドに取り込むことができます。構文は `parse @message (?<Name>pattern)` です。

次の例では、VPC フローログのキャプチャグループを使用して、ENI を `NetworkInterface` という名前のフィールドに抽出します。

```
parse @message /(?(?<NetworkInterface>eni-.*?)/ | display NetworkInterface, @message
```

Note

JSON ログイベントは取り込み中にフラット化されます。現在、ネストされた JSON フィールドを glob 表現で解析することはサポートされていません。解析できるのは、200 個以下のログイベントフィールドを含む JSON ログイベントのみです。ネストされた JSON フィールドを解析するときは、クエリ内の正規表現を JSON ログイベントの形式と一致するようにフォーマットする必要があります。

解析コマンドの例

glob 式を使用して、ログフィールド **@message** から、抽出フィールド **@user**、**@method**、**@latency** を抽出し、**@method** および **@user** との一意の組み合わせごとに平均レイテンシーを返します。

```
parse @message "user=*, method:*, latency := *" as @user,  
    @method, @latency | stats avg(@latency) by @method,  
    @user
```

正規表現を使用して、ログフィールド **@message** から、フィールド **@user2**、**@method2**、**@latency2** を抽出し、**@method2** および **@user2** との一意の組み合わせごとに平均レイテンシーを返します。

```
parse @message /user=(?<user2>.*?), method:(?<method2>.*?),  
    latency := (?<latency2>.*?)/ | stats avg(latency2) by @method2,  
    @user2
```

フィールド **loggingTime**、**loggingType**、**loggingMessage** を抽出し、**ERROR** または **INFO** 文字列を含むログイベントをフィルタリングし、**ERROR** 文字列を含むイベントの **loggingMessage** および **loggingType** フィールドのみを表示します。

```
FIELDS @message  
| PARSE @message "*" [*] "*" as loggingTime, loggingType, loggingMessage  
| FILTER loggingType IN ["ERROR", "INFO"]  
| DISPLAY loggingMessage, loggingType = "ERROR" as isError
```


例えば、Amazon VPC フローログの次のクエリでは、ホスト間のパケット転送の上位 15 件を検索します。

```
stats sum(packets) as packetsTransferred by srcAddr, dstAddr
  | sort packetsTransferred desc
  | limit 15
```

stats

stats を使用して、ログデータを棒グラフ、折れ線グラフ、積み上げ面グラフなどで視覚化します。これにより、ログデータのパターンをより効率的に特定できます。CloudWatch Logs Insights は、stats 関数と 1 つ以上の集計関数を使用するクエリの視覚化を生成します。

例えば、Route 53 ロググループの次のクエリは、Route 53 レコードの 1 時間あたりのディストリビューションをクエリタイプ別に視覚化して返します。

```
stats count(*) by queryType, bin(1h)
```

このようなクエリはすべて、棒グラフを生成できます。クエリで bin() 関数を使用して、時間の経過とともにデータを 1 つのフィールドでグループ化する場合、折れ線グラフや積み上げ面グラフも表示できます。

bin 関数では、次の時間単位と略語がサポートされています。複数の文字を含むすべての単位と略語では、s の複数形への追加がサポートされています。したがって、hr および hrs の両方とも時間を指定して機能します。

- millisecond ms msec
- second s sec
- minute m min
- hour h hr
- day d
- week w
- month mo mon
- quarter q qtr
- year y yr

トピック

- [時系列データを視覚化](#)
- [フィールド別にグループ化されたログデータを視覚化](#)
- [1つのクエリで複数の stats コマンドを使用する](#)
- [統計と併用する関数](#)

時系列データを視覚化

時系列の視覚化は、次の特性を持つクエリで機能します。

- 1つ以上の集計関数が含まれているクエリ。詳細については、「[Aggregation Functions in the Stats Command](#)」を参照してください。
- `bin()` 関数を使用して1つのフィールドでデータをグループ化するクエリ。

これらのクエリは、折れ線グラフ、積み上げ面グラフ、棒グラフ、円グラフを生成できます。

例

完全なチュートリアルについては、「[the section called “チュートリアル: 時系列の視覚化を生成するクエリを実行する”](#)」を参照してください。

時系列の視覚化で機能するクエリの他の例を以下に示します。

次のクエリでは、`myfield1` フィールドの平均値の視覚化を生成します。データポイントは5分間隔で作成されます。各データポイントは、それまでの5分間隔のログに基づく `myfield1` 値の平均の集約です。

```
stats avg(myfield1) by bin(5m)
```

次のクエリでは、異なるフィールドに基づく3つの値の視覚化を生成します。データポイントは5分間隔で作成されます。視覚化が生成されるのは、クエリに集計関数が含まれており、グループ化フィールドとして `bin()` が使用されているためです。

```
stats avg(myfield1), min(myfield2), max(myfield3) by bin(5m)
```

折れ線グラフと積み上げ面グラフの制限

ログエントリ情報を集計するが、`bin()` 関数を使用しないクエリでは、棒グラフを生成できます。ただし、これらのクエリは折れ線グラフや積み上げ面グラフを生成できません。これらのタイプの

クエリの詳細については、「[the section called “フィールド別にグループ化されたログデータを視覚化”](#)」を参照してください。

フィールド別にグループ化されたログデータを視覚化

stats 関数と 1 つ以上の集計関数を使用するクエリの棒グラフを作成できます。詳細については、「[Aggregation Functions in the Stats Command](#)」を参照してください。

視覚化を表示するには、クエリを実行します。次に、[Visualization (視覚化)] タブを選択し、[Line (線)] の横にある矢印を選択して、[Bar (棒)] を選択します。棒グラフでは、視覚化は最大 100 本の棒に制限されています。

例

完全なチュートリアルについては、「[the section called “チュートリアル: ログフィールド別にグループ化された視覚化を生成するクエリを実行する”](#)」を参照してください。次の段落では、フィールド別の視覚化のクエリに関する他の例を示します。

次の VPC フローログクエリは、各宛先アドレスについて、セッションごとに転送された平均バイト数を検出します。

```
stats avg(bytes) by dstAddr
```

また、結果の値ごとに複数の棒を含むグラフを生成することもできます。たとえば、次の VPC フローログクエリは、各宛先アドレスについて、セッションごとに転送された平均および最大バイト数を検出します。

```
stats avg(bytes), max(bytes) by dstAddr
```

次のクエリは、各クエリタイプの Amazon Route 53 クエリログの数を検出します。

```
stats count(*) by queryType
```

1 つのクエリで複数の stats コマンドを使用する

1 つのクエリで最大 2 つの stats コマンドを使用できます。これにより、最初の集計の出力に対して追加の集計を実行できます。

例: 2 つの **stats** コマンドによるクエリ

例えば、次のクエリは、最初に 5 分間のビンの合計トラフィック量を検索し、次に、その 5 分間のビンの中で最大、最低、および平均のトラフィック量を計算します。

```
FIELDS strlen(@message) AS message_length
| STATS sum(message_length)/1024/1024 as logs_mb BY bin(5m)
| STATS max(logs_mb) AS peak_ingest_mb,
      min(logs_mb) AS min_ingest_mb,
      avg(logs_mb) AS avg_ingest_mb
```

例: 複数の stats コマンドを **filter**、**fields**、**bin** などの他の関数と組み合わせます。

1 つのクエリで、2 つの stats コマンドを、filter や fields などの他のコマンドと組み合わせることができます。例えば、次のクエリは、セッション内の異なる IP アドレス数を調べ、クライアントプラットフォームごとにセッション数を調べて、それらの IP アドレスをフィルタリングして、最後にクライアントプラットフォームごとのセッションリクエストの平均を求めます。

```
STATS count_distinct(client_ip) AS session_ips,
      count(*) AS requests BY session_id, client_platform
| FILTER session_ips > 1
| STATS count(*) AS multiple_ip_sessions,
      sum(requests) / count(*) AS avg_session_requests BY client_platform
```

クエリでは、bin と dateceil の関数を複数の stats コマンドと共に使用できます。例えば、次のクエリは、最初にメッセージを 5 分のブロックに結合し、次に 5 分間のブロックを 10 分のブロックに集約して、各 10 分ブロック内の最大、最低、および平均のトラフィック量を計算します。

```
FIELDS strlen(@message) AS message_length
| STATS sum(message_length) / 1024 / 1024 AS logs_mb BY BIN(5m) as @t
| STATS max(logs_mb) AS peak_ingest_mb,
      min(logs_mb) AS min_ingest_mb,
      avg(logs_mb) AS avg_ingest_mb BY dateceil(@t, 10m)
```

注意事項と制限事項

1 つのクエリにつき、最大 2 つの stats コマンドを持つことができます。このクォータは変更できません。

sort または limit コマンドを使用する場合は、2 番目の stats コマンドの後に指定する必要があります。2 番目の stats コマンドより前に置くと、クエリは無効になります。

クエリに 2 つの stats コマンドがある場合、1 つ目の stats 集計が完了するまで、クエリの結果の一部は表示されなくなります。

1 つのクエリにある 2 番目の stats コマンドでは、1 番目の stats コマンドで定義されているフィールドのみを参照できます。例えば、最初の stats 集計以降 @message フィールドが使用できなくなるため、次のクエリは無効です。

```
FIELDS @message
| STATS SUM(Fault) by Operation
# You can only reference `SUM(Fault)` or Operation at this point
| STATS MAX(strlen(@message)) AS MaxMessageSize # Invalid reference to @message
```

最初の stats コマンドの後に参照するフィールドは、すべて最初の stats コマンドで定義する必要があります。

```
STATS sum(x) as sum_x by y, z
| STATS max(sum_x) as max_x by z
# You can only reference `max(sum_x)`, max_x or z at this point
```

Important

この bin 関数は常に @timestamp フィールドを暗黙的に使用します。つまり、2 番目の stats コマンドでは、1 番目の stats コマンドを使用して timestamp フィールドを伝達しないと bin を使用できないということです。例えば、以下のクエリは有効ではありません。

```
FIELDS strlen(@message) AS message_length
| STATS sum(message_length) AS ingested_bytes BY @logStream
| STATS avg(ingested_bytes) BY bin(5m) # Invalid reference to @timestamp field
```

代わりに、最初の stats コマンドで @timestamp フィールドを定義し、次の例のように 2 番目の stats コマンドで dateceil と共にそれを使用できます。

```
FIELDS strlen(@message) AS message_length
| STATS sum(message_length) AS ingested_bytes, max(@timestamp) as @t BY
@logStream
| STATS avg(ingested_bytes) BY dateceil(@t, 5m)
```

統計と併用する関数

CloudWatch Logs Insights は、統計集計関数と統計非集計関数の両方をサポートしています。

statsaggregation 関数は、stats コマンドで使します。また、他の関数の引数としても使します。

関数	結果タイプ	説明
<code>avg(fieldName: NumericLogField)</code>	数値	指定したフィールドの値の平均。
<code>count()</code> <code>count(fieldName: LogField)</code>	数値	ロギイベントをカウントします。count() (または count(*)) は、クエリによって返されたすべてのイベントをカウントし、count(fieldName) は指定されたフィールド名を含むすべてのレコードをカウントします。
<code>count_distinct(fieldName: LogField)</code>	数値	フィールドの一意な値の数を返します。このフィールドの濃度が非常に高い場合 (一意な値が多数含まれている場合)、count_distinct から返される値は単なる概算値です。
<code>max(fieldName: LogField)</code>	LogFieldValue	クエリを実行したログにおける、このログフィールドの値の最大数。
<code>min(fieldName: LogField)</code>	LogFieldValue	クエリを実行したログにおける、このログフィールドの値の最小数。
<code>pct(fieldName: LogFieldValue, percent: number)</code>	LogFieldValue	パーセンタイルは、データセットにおける値の相対的な位置を示します。たとえば、pct(@duration, 95) が @duration 値を返した場合、@duration の値の 95% がこの値より低く、5% がこの値より高くなります。
<code>stddev(fieldName: NumericLogField)</code>	数値	指定されたフィールドの値の標準偏差。

関数	結果タイプ	説明
<code>sum(fieldName: NumericLogField)</code>	数値	指定したフィールドの値の合計。

統計非集計関数

非集約関数は、`stats` コマンドで使います。また、他の関数の引数としても使います。

関数	結果タイプ	説明
<code>earliest(fieldName: LogField)</code>	LogField	クエリを実行したうち最も早いタイムスタンプがあるログイベントから <code>fieldName</code> の値を返します。
<code>latest(fieldName: LogField)</code>	LogField	クエリを実行したうち最も遅いタイムスタンプがあるログイベントから <code>fieldName</code> の値を返します。
<code>sortsFirst(fieldName: LogField)</code>	LogField	クエリを実行したログをソートすると最初に来る <code>fieldName</code> の値を返します。
<code>sortsLast(fieldName: LogField)</code>	LogField	クエリを実行したログをソートすると最後に来る <code>fieldName</code> の値を返します。

制限

`limit` を使用して、クエリで返すログイベントの数を指定します。`limit` を省略すると、クエリは結果に最大 10,000 個のログイベントを返します。

例えば、以下の例は、最新の 25 のログイベントのみを返しています。

```
fields @timestamp, @message | sort @timestamp desc | limit 25
```

重複排除

指定したフィールドの特定の値に基づいて、重複した結果を削除するときは、`dedup` を使います。`dedup` は 1 つ以上のフィールドで使用できます。`dedup` を使ってフィールドを 1 つ指定する

と、そのフィールドの一意の値ごとに 1 つのログイベントのみが返されます。複数のフィールドを指定すると、そのフィールドの一意の値の組み合わせごとに 1 つのログイベントが返されます。

重複はソート順に基づいて破棄され、ソート順の最初の結果だけが保持されます。dedup コマンドを実行する前に、結果をソートすることが推奨されます。dedup を実行する前に結果がソートされていない場合は、@timestamp を使用しているデフォルトの降順のソート順が使用されます。

NULL 値は、評価において重複とは見なされません。指定したフィールドのいずれかに NULL 値が含まれるログイベントは保持されます。NULL 値のフィールドを削除するには、isPresent(field) 関数を使用して **filter** を実行します。

dedup コマンドの後のクエリで利用できるクエリコマンドは、limit だけです。

クエリdedupで を使用すると、コンソールに「Showing X of Y records」などのメッセージが表示されます。ここで、X は重複排除された結果の数、Y は重複排除前に一致したレコードの合計数です。これは、重複したレコードが削除されたことを示すものであり、データが欠落していることを意味するものではありません。

例: **server** という名前のフィールドの、一意の値ごとに、最新のログイベントのみを表示します。

次の例では、server の一意の値ごとに、最新のイベントの timestamp、server、severity、message フィールドのみを表示します。

```
fields @timestamp, server, severity, message
| sort @timestamp desc
| dedup server
```

CloudWatch Logs Insights クエリのその他の例については、「[一般的なクエリ](#)」を参照してください。

マスクを外す

データ保護ポリシーにより一部のコンテンツがマスクされているログイベントのすべてのコンテンツを表示するには unmask を使用します。このコマンドを使用するには、logs:Unmask アクセス許可が必要です。

ロググループのデータ保護の詳細については、「[機密性の高いログデータをマスキングで保護する](#)」を参照してください。

ネスト解除

`unnest` を使用し、入力として取得されたリストをフラット化し、リスト内の各要素に対して 1 つのレコードを持つ複数のレコードを生成します。フィールドに含まれる項目の数に基づいて、このコマンドは現在のレコードを破棄し、新しいレコードを生成します。各レコードには、項目を表す `unnested_field` が含まれます。他のすべてのフィールドは元のレコードから取得されます。

`unnest` の入力は `LIST` で、`jsonParse` 関数から取得されます。詳細については、「[構造型](#)」を参照してください。MAP、String、numbers などの他のタイプは、`unnest` に 1 つの項目を含むリストとして扱われます。

コマンド構造

以下の例はこのコマンドの形式を説明しています。

```
unnest field into unnested_field
```

クエリの例

次の例では、JSON オブジェクト文字列を解析し、フィールドイベントのリストを展開します。

```
fields jsonParse(@message) as json_message
| unnest json_message.events into event
| display event.name
```

このクエリ例のログイベントは、次のように JSON 文字列になります。

```
{
  "events": [
    {
      "name": "exception"
    },
    {
      "name": "user action"
    }
  ]
}
```

この場合、サンプルクエリはクエリ結果に 2 つのレコードを生成します。1 つは `exception` として `event.name` を使用し、もう 1 つはユーザーアクションとして `event.name` を使用します。

クエリの例

次の例では、リストをフラット化し、項目を除外します。

```
fields jsonParse(@message) as js
| unnest js.accounts into account
| filter account.type = "internal"
```

クエリの例

次の例では集計用のリストをフラット化します。

```
fields jsonParse(trimmedData) as accounts
| unnest accounts into account
| stats sum(account.droppedSpans) as n by account.accountId
| sort n desc
| limit 10
```

lookup

を使用してlookup、ルックアップテーブルからの参照データを使用してクエリ結果を強化します。ルックアップテーブルには、Amazon CloudWatch Logs にアップロードする CSV データが含まれています。クエリが実行されると、lookup コマンドはログイベントのフィールドをルックアップテーブルのフィールドと照合し、指定された出力フィールドを結果に追加します。

ユーザー IDsユーザーの詳細にマッピングする、製品コードを製品情報にマッピングする、エラーコードをエラーの説明にマッピングするなどのデータエンリッチメントシナリオには、ルックアップテーブルを使用します。

ルックアップテーブルの作成と管理

クエリで lookup コマンドを使用する前に、ルックアップテーブルを作成する必要があります。ルックアップテーブルは、CloudWatch コンソールから、または Amazon CloudWatch Logs API を使用して作成および管理できます。

ルックアップテーブルを作成するには (コンソール)

1. CloudWatch コンソールの <https://console.aws.amazon.com/cloudwatch/> を開いてください。
2. ナビゲーションペインで、設定を選択し、ログタブを選択します。
3. ルックアップテーブルまでスクロールし、管理を選択します。
4. ルックアップテーブルの作成を選択します。

5. ルックアップテーブルの名前を入力します。名前には、英数字、ハイフン、アンダースコアのみを含めることができます。
6. (オプション) 説明を入力します。
7. CSV ファイルをアップロードします。ファイルには、列名を含むヘッダー行を含める必要があり、UTF-8 エンコーディングを使用し、10 MB を超えることはできません。
8. (オプション) テーブルデータを暗号化する AWS KMS キーを指定します。
9. [作成] を選択します。

ルックアップテーブルを作成したら、CloudWatch Logs Insights クエリエディタで表示できます。ルックアップテーブルタブを選択して、使用可能なテーブルとそのフィールドを参照します。

ルックアップテーブルを更新するには、テーブルを選択し、アクション、更新を選択します。新しい CSV ファイルをアップロードして、既存のすべてのコンテンツを置き換えます。ルックアップテーブルを削除するには、アクション、削除を選択します。

Note

アカウントごとに、ごとに最大 100 個のルックアップテーブルを作成できます AWS リージョン。CSV ファイルは最大 10 MB です。Amazon CloudWatch Logs API を使用してルックアップテーブルを管理することもできます。詳細については、Amazon CloudWatch Logs API リファレンスの「[CreateLookupTable](#)」を参照してください。

Note

ルックアップテーブルが KMS キーで暗号化されている場合、そのルックアップテーブルを参照するクエリで StartQuery API を使用するには、呼び出し元がキー (ルックアップテーブルの暗号化に使用される KMS キー) に対する kms:Decrypt アクセス許可を持っている必要があります。詳細については、「[を使用して CloudWatch Logs のルックアップテーブルを暗号化する AWS Key Management Service](#)」を参照してください。

検索のクエリ構文

コマンド構造

このコマンドの形式を次に示します。

```
lookup table lookup-field as log-field [,...] output-mode output-field [,...]
```

コマンドは、次の引数を使用します。

- *table* – 使用するルックアップテーブルの名前。
- *lookup-field* – 照合するルックアップテーブルのフィールド。
- *log-field* – 一致するログイベントのフィールド。一致は完全で大文字と小文字が区別されます。
- *lookup-field* as *log-field* [,...] – 複数の一致フィールドペアをカンマで区切って指定できます。複数のペアを指定する場合、結果を生成するには、ルックアップテーブルの行がすべてのフィールドと一致する必要があります (AND ロジック)。
- *output-mode* – 出力フィールドを結果に追加する方法を指定します。以下のいずれかを使用します。
 - OUTPUT – 出力フィールドを結果に追加します。ログイベントに同じ名前のフィールドがすでに存在する場合、ルックアップテーブル値で上書きされます。一致が見つからない場合、フィールドは null に設定されます。
 - OUTPUTNEW – フィールドがログイベントにまだ存在しない場合にのみ、出力フィールドを結果に追加します。フィールドにすでに値がある場合、元の値は保持されます。一致が見つからない場合、フィールドは変更されません。
- *output-field* – 結果に追加するルックアップテーブルの 1 つ以上のフィールド。

例: ユーザーの詳細を使用してログイベントを強化する

id フィールドを含むイベントを含むロググループと、列 id、`name`、`email` および `user_data` を含む という名前のルックアップテーブルがあるとします department。次のクエリは、各ログイベントをルックアップテーブルのユーザー名、E メール、部門で強化します。

```
fields action, status, name, email, department
| lookup user_data id OUTPUT name, email, department
```

例: 集計でルックアップを使用する

集計関数でルックアップ出力フィールドを使用できます。次のクエリは、ユーザーの詳細でログイベントを強化し、E メールアドレス別にグループ化されたイベントをカウントします。

```
fields user_id, action, username, email, department
```

```
| lookup user_data user_id OUTPUT username, email, department
| stats count(*) by email
```

例: フィルターでルックアップを使用する

ルックアップによって返されたフィールドに基づいて結果をフィルタリングできます。次のクエリは、ログイベントを強化し、特定の部門のイベントのみを表示するようにフィルタリングします。

```
fields user_id, action
| lookup user_data user_id OUTPUT username, email, department
| filter department = "Engineering"
```

例: OUTPUTNEW を使用して上書きせずに強化する

ログイベントに既に hostname フィールドが含まれているが、場合によっては空の場合は、OUTPUTNEW を使用して、既存の値を上書きせずに欠損値を入力します。

```
fields srcAddr, hostname
| lookup known_hosts ip_address as srcAddr OUTPUTNEW hostname, region
```

例: 複数の一致フィールドでルックアップを使用する

複数のフィールドで一致させることができます。次のクエリは、srcAddr と の両方をルックアップテーブルdstPortと照合して、既知のネットワークサービスを識別します。

```
fields @timestamp, srcAddr, dstAddr, dstPort
| lookup network_services ip_address as srcAddr, port as dstPort OUTPUT service_name,
  owner
| filter ispresent(service_name)
```

結合

ソースロググループからのログイベントを、一致するフィールドに基づいて別のロググループまたはクエリ結果からのイベントと組み合わせます。

join コマンドを使用して、一致するリクエスト識別子やトランザクション IDs。

構文

```
join type=<join_type> left=<left_alias> right=<right_alias>
  where <left_alias>.<field>=<right_alias>.<field>
```

```
(SOURCE <right_log_group>)
```

パラメータ

- `<right_log_group>` – 結合するセカンダリデータソース。
- `<left_alias>` および `<right_alias>` – フィールドを左 (プライマリ) データソースと右 (セカンダリ) データソースから区別するエイリアス。
- `where <field>` – 結合キーとして使用されるフィールドを指定します。フィールドは両方のデータソースに存在する必要があります。
- `type=<join_type>` (オプション) – 結合タイプを指定します。次の値を指定できます。
 - `inner` (デフォルト) – 一致するレコードのみを返します
 - `left` – プライマリデータソースからのすべてのレコードと、セカンダリデータソースからの一致するレコードを返します。

例

Example例 1: API Gateway リクエストを Lambda 実行ログと関連付ける

この例では、API Gateway アクセスログを Lambda 関数ログに結合して、受信リクエストとバックエンド処理を関連付ける方法を示します。これはend-to-endのリクエストフローのトラブルシューティングや、特定の API リクエストに対応する Lambda 呼び出しの特定に役立ちます。

```
filter status >= 500
| join type=inner left=api right=lambda
  where api.requestId=lambda.requestId
  (SOURCE '/aws/lambda/my-function')
| fields api.requestId, api.status, api.latency, lambda.duration, lambda.memoryUsed
| sort api.latency desc
```

このクエリ:

1. API Gateway アクセスログをクエリし、サーバーエラーをフィルタリングする (ステータス $\geq$ 500)
2. 両方のログソースに表示される `requestId` フィールドを使用して Lambda 関数ログと結合する
3. エイリアス (`api` および `lambda`) を使用して、各ソースからフィールドを区別します
4. Lambda の実行時間とメモリ使用量とともに API レイテンシーを示す複合情報を返します。

5. API レイテンシーで結果をソートして、最も遅いリクエストを特定します

Example例 2: マイクロサービス間の分散トランザクションを追跡する

マイクロサービスアーキテクチャで問題をデバッグする場合、多くの場合、複数のサービスにわたるトランザクションを追跡する必要があります。この例では、共通のトランザクション ID を使用して 2 つの異なるサービスのログを結合する方法を示します。

```
filter eventType = "ORDER_CREATED"
| join type=left left=order right=payment
  where order.transactionId=payment.transactionId
  (SOURCE '/aws/lambda/payment-service')
| filter payment.eventType = "PAYMENT_PROCESSED" or !ispresent(payment.eventType)
| fields order.transactionId, order.orderId, order.customerId,
  payment.paymentStatus, payment.amount
| filter payment.paymentStatus != "SUCCESS" or !ispresent(payment.paymentStatus)
```

このクエリ:

1. 注文サービスからの注文作成イベントから開始します
2. 支払いレコードleft joinが一致しない場合でも、を使用してすべての注文を含めます
3. 共有transactionIdフィールドを使用して支払い処理イベントに参加する
4. 最終結果をフィルタリングして、支払いに失敗した注文または支払いレコードが欠落している注文のみを表示します

ここで左結合は重要です。これは、作成されたが、システム障害を示す可能性のある対応する支払いイベントがなかった注文が表示されるためです。

行動

- プライマリデータソース (左側) が最初に処理されます。
- セカンダリデータソースは、指定された結合キーを使用して評価および照合されます。
- 一致は、結合フィールドの等価比較を使用して実行されます。
- 左結合の場合、プライマリデータソースからの一致しないレコードは、セカンダリフィールドの null 値で保持されます。

注意事項と制限事項

- 等価 (=) 条件のみがサポートされています。
- クエリごとにサポートされる結合コマンドは 1 つだけです。
- 結合キーは両方のデータソースに存在し、互換性のあるタイプである必要があります。
- 結合を使用するクエリは、より多くのデータをスキャンしてコストが高くなる可能性があります。
- クエリのパフォーマンスを確保するために、セカンダリデータソースの一意のキー値の数は 50,000 に制限されています。
- 結合の右側にあるサブクエリはサポートされていません。

関連コマンド

- [フィールド](#)
- [フィルター](#)
- [解析](#)
- [stats](#)
- [ソート](#)
- [制限](#)

subqueries

サブクエリは、別のクエリへの入力として使用できるネストされた Logs Insights クエリです。サブクエリを使用すると、後続のコマンドで消費される中間結果セットを取得できます。

構文

フィルターのサブクエリ

```
filter <field> in (  
    <subquery>  
)
```

パラメータ

- <subquery> – 結果セットを返す有効な Logs Insights クエリ。サブクエリは、外部クエリによって参照されるフィールドを生成する必要があります。

例

Example例 1: ダウンストリームサービスでエラーが発生したリクエストを検索する

この例では、サブクエリを使用して、ダウンストリームサービスでエラーが発生したメインサービスのリクエストを識別する方法を示します。これは、分散システムのカスケード障害のトラブルシューティングに役立ちます。

```
filter requestId in (
  SOURCE '/aws/lambda/database-service'
  | filter errorType = "DatabaseConnectionTimeout"
  | fields requestId
)
| fields @timestamp, requestId, endpoint, userId, responseTime
| sort @timestamp desc
```

このクエリ:

1. サブクエリは、接続タイムアウトが発生したデータベースサービスからすべてのrequestId値を検索します。
2. 外部クエリは、メインサービスのログをフィルタリングして、エラーが発生しやすいリクエストIDs
3. 結果には、影響を受けたエンドポイントやユーザーなど、ダウンストリームで失敗したリクエストの完全なコンテキストが表示されます。

このパターンは、ダウンストリーム障害のアップストリームへの影響を理解するのに役立ちます。

Example例 2: ターゲット調査のために頻繁に失敗するリクエストを特定する

この例では、集計でサブクエリを使用して、繰り返し失敗するリクエストを検索する方法を示します。多くの場合、一時的なエラーではなく体系的な問題を示します。

```
filter requestId in (
  SOURCE '/aws/lambda/payment-processor'
  | filter status = "FAILED"
  | stats count(*) as failureCount by requestId
  | filter failureCount > 3
  | fields requestId
)
| fields @timestamp, requestId, customerId, amount, failureReason
```

```
| sort @timestamp asc
```

このクエリ:

1. サブクエリは、失敗した支払い試行を集計し、3 回以上失敗したリクエスト IDs を示します。
2. 外部クエリは、問題のあるリクエスト IDs
3. 再試行の進行状況を示すために結果が時系列的にソートされます

これにより、より詳細な調査が必要な一時的な障害 (単一発生) と永続的な問題 (複数の障害) を区別できます。

行動

- サブクエリは、外部クエリとは独立して実行されます。
- 結果は、外部クエリによって消費される前にマテリアライズされます。
- 外部クエリで利用できるのは、サブクエリで明示的に選択されたフィールドのみです。

注意事項と制限事項

- サブクエリは、外部クエリによって参照されるフィールドを返す必要があります。
- ネストされたサブクエリはサポートされていません。
- サブクエリにより、クエリの実行時間とコストが増加する可能性があります。
- 相関サブクエリはサポートされていません。
- 内部クエリの実行は 30 秒に制限されています。

関連コマンド

- [フィールド](#)
- [フィルター](#)
- [解析](#)
- [stats](#)
- [ソート](#)
- [制限](#)

ブール、比較、数値、日時、その他の関数

CloudWatch Logs Insights は、以下のセクションで説明するように、クエリ内の他の多くの演算や関数をサポートしています。

トピック

- [算術演算子](#)
- [ブール演算子](#)
- [比較演算子](#)
- [数値演算子](#)
- [構造型](#)
- [日時関数](#)
- [一般関数](#)
- [JSON 関数](#)
- [IP アドレス文字列関数](#)
- [文字列関数](#)

算術演算子

算術演算子は、数値データ型を引数として受け入れ、数値結果を返します。算術演算子は、`filter` コマンドと `fields` コマンドで使います。また、他の関数の引数としても使います。

運用	説明
$a + b$	加算
$a - b$	減算
$a * b$	乗算
a / b	除算
$a ^ b$	指数 (2 ^ 3 は 8 を返します)
$a \% b$	残余または剰余 (10 % 3 は 1 を返します)

ブール演算子

ブール演算子 **and**、**or**、および **not** を使用します。

Note

ブール演算子は、TRUE または FALSE の値を返す関数でのみ使用します。

比較演算子

比較演算子は、すべてのデータ型を引数として受け入れ、ブール値の結果を返します。比較オペレーションは、`filter` コマンドで使用します。また、他の関数の引数としても使用します。

演算子	説明
=	Equal
!=	等しくない
<	Less than
>	Greater than
<=	以下
>=	以上

数値演算子

数値オペレーションは、数値データ型を引数として受け入れ、数値結果を返します。数値オペレーションは、`filter` コマンドと `fields` コマンドで使用します。また、他の関数の引数としても使用します。

運用	結果タイプ	説明
<code>abs(a: number)</code>	数値	絶対値
<code>ceil(a: number)</code>	数値	上限 (a の値より大きい最小整数) に切り上げられます。

運用	結果タイプ	説明
<code>floor(a: number)</code>	数値	下限 (a の値より小さい最大整数) に切り下げられます。
<code>greatest(a: number, ...numbers: number[])</code>	数値	最大値を返します
<code>least(a: number, ...numbers: number[])</code>	数値	最小値を返します
<code>log(a: number)</code>	数値	自然対数
<code>sqrt(a: number)</code>	数値	平方根

構造型

マップまたはリストは、クエリの属性にアクセスして使用できる、CloudWatch Logs Insights の構造型です。

例: マップまたはリストを取得するには

`jsonParse` を使用して、JSON 文字列であるフィールドを解析し、マップまたはリストにします。

```
fields jsonParse(@message) as json_message
```

例: 属性にアクセスするには

ドットアクセス演算子 (`map.attribute`) を使用して、マップ内の項目にアクセスします。マップ内の属性に特殊文字が含まれている場合は、バッククォートを使用して属性名を囲みます (`map.attributes.`special.char``)。

```
fields jsonParse(@message) as json_message
| stats count() by json_message.status_code
```

ブラケットアクセス演算子 (`list[index]`) を使用して、リスト内の特定の位置にある項目を取得します。

```
fields jsonParse(@message) as json_message
| filter json_message.users[1].action = "PutData"
```

キー名に特殊文字が含まれている場合は、特殊文字をバッククォート (`) でラップします。

```
fields jsonParse(@message) as json_message
| filter json_message.`user.id` = "123"
```

例: 空の結果

マップとリストは、文字列、数値、日時関数では null として扱われます。

```
fields jsonParse(@message) as json_message
| display toupper(json_message)
```

マップとリストを他のフィールドと比較すると false になります。

Note

dedup、pattern、sort、stats におけるマップとリストの使用はサポートされていません。

日時関数

日時関数

日時関数は、fields コマンドと filter コマンドで使用します。また、他の関数の引数としても使用します。これらの関数では、集計関数を使用してクエリの時間バケットを作成します。数値と次のいずれかで構成される期間を使用します。

- ms はミリ秒
- s は秒
- m は分
- h は時間

たとえば、10m は 10 分、1h は 1 時間です。

Note

日時関数に最適な時間単位を使用します。CloudWatch Logs は、選択した時間単位に従ってリクエストを制限します。例えば、s を使用するすべてのリクエストの最大値として 60 を上限とします。したがって、bin(300s) を指定すると、CloudWatch Logs はこれを 60 秒として実際に実装します。60 は 1 分間の秒数であるため、CloudWatch Logs は s で 60 を超える数値を使用しません。5 分間のバケットを作成するには、代わりに bin(5m) を使用します。

ms の上限は 1000、s と m の上限は 60、h の上限は 24 です。

次の表は、クエリコマンドで使用できるさまざまな日付時刻関数のリストを示したものです。このリストには、各関数の結果タイプと説明が記載されています。

Tip

クエリコマンドを作成するときに、時間間隔セレクタを使用してクエリの対象とする期間を選択できます。例えば、5～30 分間隔、1 時間、3 時間、12 時間間隔、またはカスタム時間枠の期間を設定できます。また、特定の日付の間で期間を指定することもできます。

関数	結果タイプ	説明
bin(period: Period)	タイムスタンプ	@timestamp の値を特定の期間に切り上げ、次に切り詰めます。例えば、bin(5m) は @timestamp の値を最も近い 5 分に四捨五入します。 これを使用して、複数のログエントリをクエリにまとめることができます。次の例では、1 時間あたりの例外の数を返します。 <pre>filter @message like /Exception/ stats count(*) as exceptionCount by bin(1h) sort exceptionCount desc</pre>

関数	結果タイプ	説明
		bin 関数では、次の時間単位と略語がサポートされています。複数の文字を含むすべての単位と略語では、s の複数形への追加がサポートされています。したがって、hr および hrs の両方とも時間を指定して機能します。 • millisecond ms msec • second s sec • minute m min • hour h hr • day d • week w • month mo mon • quarter q qtr • year y yr
<code>datefloor(timestamp: Timestamp, period: Period)</code>	タイムスタンプ	タイムスタンプを特定の期間に切り詰めます。たとえば、 <code>datefloor(@timestamp, 1h)</code> は <code>@timestamp</code> のすべての値を 1 時間の下限に切り詰めます。
<code>dateceil(timestamp: Timestamp, period: Period)</code>	タイムスタンプ	タイムスタンプを特定の期間に切り上げ、次に切り詰めます。たとえば、 <code>dateceil(@timestamp, 1h)</code> は <code>@timestamp</code> のすべての値を 1 時間の上限に切り詰めます。
<code>fromMillis(fieldName: number)</code>	タイムスタンプ	入力フィールドを Unix エポックからのミリ秒数として解釈し、タイムスタンプに変換します。

関数	結果タイプ	説明
<code>toMillis(fieldName: Timestamp)</code>	数値	指定されたフィールドで見つかったタイムスタンプを、Unix エポックからのミリ秒を表す数値に変換します。例えば、<code>toMillis(@timestamp)</code> はタイムスタンプを <code>2022-01-14T13:18:031.000-08:00</code> から <code>1642195111000</code> に変換します。
<code>now()</code>	数値	クエリ処理が開始された時刻をエポック秒単位で返します。この関数は引数を取りません。 これを使用して、現在の時刻に従ってクエリ結果をフィルタリングできます。 例えば、次のクエリは、過去 2 時間の 4xx エラーをすべて返します。 <pre>parse @message "Status Code: *;" as statusCode\n filter statusCode >= 400 and statusCode <= 499 \n filter toMillis(@timestamp) >= (now() * 1000 - 7200000)</pre> 次の例では、<code>error</code> または <code>failure</code> という単語のいずれかを含む過去 5 時間のすべてのログエントリを返します。 <pre>fields @timestamp, @message filter @message like /(?(i)(error failure)/ filter toMillis(@timestamp) >= (now() * 1000 - 18000000)</pre>

Note

現在、CloudWatch Logs Insights では、可読性のあるタイムスタンプが記録されているログのフィルタリングをサポートしていません。

一般関数

一般関数

一般関数は、`fields` コマンドと `filter` コマンドで使います。また、他の関数の引数としても使います。

関数	結果タイプ	説明
<code>ispresent(fieldName: LogField)</code>	ブール値	フィールドが存在する場合は <code>true</code> を返します
<code>coalesce(fieldName: LogField, ...fieldNames: LogField[])</code>	LogField	リストから最初の <code>null</code> でない値を返します

JSON 関数

JSON 関数

JSON 関数は、`fields` コマンドと `filter` コマンドで使います。また、他の関数の引数としても使います。

関数	結果タイプ	説明
<code>jsonParse(fieldName: string)</code>	マップ リスト 空	入力が JSON オブジェクトまたは JSON 配列の文字列表現である場合、マップまたはリストを返します。入力がその表現のいずれかでない場合、空の値を返します。

関数	結果タイプ	説明
<code>jsonStringify(fieldName: Map List)</code>	String	マップまたはリストデータから JSON 文字列を返します。

IP アドレス文字列関数

IP アドレス文字列関数

IP アドレス文字列関数は、`filter` コマンドと `fields` コマンドで使用します。また、他の関数の引数としても使用します。

関数	結果タイプ	説明
<code>isValidIp(fieldName: string)</code>	ブール型	フィールドが有効な IPv4 または IPv6 アドレスである場合、 <code>true</code> を返します。
<code>isValidIPv4(fieldName: string)</code>	ブール型	フィールドが有効な IPv4 アドレスである場合、 <code>true</code> を返します。
<code>isValidIPv6(fieldName: string)</code>	ブール型	フィールドが有効な IPv6 アドレスである場合、 <code>true</code> を返します。
<code>isIpInSubnet(fieldName: string, subnet: string)</code>	ブール型	指定された v4 または v6 サブネット内でフィールドが有効な IPv4 または IPv6 アドレスである場合、 <code>true</code> を返します。サブネットを指定するときは、 <code>192.0.2.0/24</code> または <code>2001:db8::/32</code> などの CIDR 表記を使用します。 <code>192.0.2.0</code> または <code>2001:db8::</code> は CIDR ブロックの開始アドレスです。
<code>isIPv4InSubnet(fieldName: string, subnet: string)</code>	boolean	指定された v4 サブネット内でフィールドが有効な IPv4 アドレスである場合、 <code>true</code> を返します。サブネットを指定するときは、 <code>192.0.2.0/24</code> などの CIDR 表記を使用します。 <code>192.0.2.0</code> は CIDR ブロックの開始アドレスです。

関数	結果タイプ	説明
<code>isIpv6InSubnet(fieldName: string, subnet: string)</code>	boolean	指定された v6 サブネット内でフィールドが有効な IPv6 アドレスである場合、true を返します。サブネットを指定するときは、2001:db8::/32 などの CIDR 表記を使用します。2001:db8:: は CIDR ブロックの開始アドレスです。

文字列関数

文字列関数

文字列関数は、`fields` コマンドと `filter` コマンドで使います。また、他の関数の引数としても使います。

関数	結果タイプ	説明
<code>isempty(fieldName: string)</code>	Number	フィールドが欠落しているか、空の文字列である場合、1 を返します。
<code>isblank(fieldName: string)</code>	Number	フィールドが欠落しているか、空の文字列であるか、空白が含まれている場合、1 を返します。
<code>concat(str: string, ...strings: string[])</code>	文字列	複数の文字列を連結します。
<code>ltrim(str: string)</code> <code>ltrim(str: string, trimChars: string)</code>	文字列	関数に 2 番目の文字列引数がない場合、文字列の左側からホワイトスペースを削除します。関数に 2 番目の文字列引数がある場合、ホワイトスペースは削除されません。その場合、 <code>str</code> の左から <code>trimChars</code> 個の文

関数	結果タイプ	説明
		字が削除されます。たとえば、 <code>ltrim("xyZxyfooxyZ","xyZ")</code> は "fooxyZ" を返します。
<code>rtrim(str: string)</code> <code>rtrim(str: string, trimChars: string)</code>	文字列	関数に 2 番目の文字列引数がない場合、文字列の右側からホワイトスペースを削除します。関数に 2 番目の文字列引数がある場合、ホワイトスペースは削除されません。その場合、 <code>str</code> の右から <code>trimChars</code> 個の文字が削除されます。たとえば、 <code>rtrim("xyZfooxyxyZ","xyZ")</code> は "xyZfoo" を返します。
<code>trim(str: string)</code> <code>trim(str: string, trimChars: string)</code>	文字列	関数に 2 番目の文字列引数がない場合、文字列の両方の端からホワイトスペースを削除します。関数に 2 番目の文字列引数がある場合、ホワイトスペースは削除されません。その場合、 <code>str</code> の両方から <code>trimChars</code> 個の文字が削除されます。たとえば、 <code>trim("xyZxyfooxyxyZ","xyZ")</code> は "foo" を返します。
<code>strlen(str: string)</code>	数値	文字列の長さを Unicode コードポイントで返します。
<code>toupper(str: string)</code>	文字列	文字列を大文字に変換します。

関数	結果タイプ	説明
<code>tolower(str: string)</code>	文字列	文字列を小文字に変換します。
<code>substr(str: string, startIndex: number)</code> <code>substr(str: string, startIndex: number, length: number)</code>	文字列	数値引数で指定されたインデックスから文字列の末尾までの部分文字列を返します。関数に 2 番目の数値引数がある場合、この引数には取得される部分文字列の長さが含まれます。たとえば、 <code>substr("xyzfooxyz", 3, 3)</code> は "foo" を返します。
<code>replace(fieldName: string, searchValue: string, replaceValue: string)</code>	文字列	<code>searchValue</code> の <code>fieldName: string</code> のすべてのインスタンスを <code>replaceValue</code> に置き換えます。 例えば、関数 <code>replace(logGroup, "smoke_test", "Smoke")</code> はフィールド <code>logGroup</code> に文字列値 <code>smoke_test</code> を含むログイベントを検索し、その値を文字列 <code>Smoke</code> に置き換えます。
<code>strcontains(str: string, searchValue: string)</code>	数値	<code>str</code> に <code>searchValue</code> が含まれている場合は 1 を返し、それ以外の場合は 0 を返します。

特殊文字を含むフィールド

フィールドに @ 記号またはピリオド (.) 以外の英数字以外の文字が含まれている場合は、フィールドをバッククォート文字 (`) で囲む必要があります。例えば、ログフィールド `foo-bar` では英数字以外の文字であるハイフン (``foo-bar``) が含まれているため、バッククォート (`-`) で囲む必要があります。

クエリでのエイリアスとコメントの使用

エイリアスを含むクエリを作成します。ログフィールドの名前を変更するために、またはフィールドに値を抽出する場合にエイリアスを使用します。キーワード `as` を使用して、ログフィールドまたは結果にエイリアスを指定します。クエリ内で複数のエイリアスを使用できます。次のコマンド内でエイリアスを使用できます。

- `fields`
- `parse`
- `sort`
- `stats`

次の例では、エイリアスを含むクエリを作成する方法を示します。

例

クエリの `fields` コマンドはエイリアスを含みます。

```
fields @timestamp, @message, accountId as ID
| sort @timestamp desc
| limit 20
```

クエリは、フィールド `@timestamp`、`@message`、および `accountId` の値を返します。結果は降順でソートされ、20 に制限されます。ID の値は、エイリアス `accountId` の下に一覧表示されます。

例

クエリの `sort` および `stats` コマンドはエイリアスを含みます。

```
stats count(*) by duration as time
| sort time desc
```


クエリは、ロググループでフィールド `duration` が発生した回数をカウントし、結果を降順で並べ替えます。`duration` の値は、エイリアス `time` の下に一覧表示されます。

コメントの使用

CloudWatch Logs Insights は、クエリ内でのコメントをサポートしています。ハッシュ文字 (#) を使用してコメントを開始します。コメントを使用して、クエリまたはドキュメントクエリの行を無視できます。

例: クエリ

次のクエリを実行すると、2 行目は無視されます。

```
fields @timestamp, @message, accountId
# | filter accountId not like "7983124201998"
| sort @timestamp desc
| limit 20
```

Logs Insights QL の使用を開始する: クエリチュートリアル

以下のセクションでは、Logs Insights QL の使用を開始する際に役立つサンプルのクエリのチュートリアルをご覧ください。

トピック

- [チュートリアル: サンプルクエリを実行および変更する](#)
- [チュートリアル: 集計関数を使用してクエリを実行する](#)
- [チュートリアル: ログフィールド別にグループ化された視覚化を生成するクエリを実行する](#)
- [チュートリアル: 時系列の視覚化を生成するクエリを実行する](#)

チュートリアル: サンプルクエリを実行および変更する

以下のチュートリアル通りに、CloudWatch Logs Insights の使用を開始できます。Logs Insights QL のサンプルクエリを実行し、次にこのクエリを変更して再実行する方法を示します。

クエリを実行するには、CloudWatch Logs に保存済みのログが必要です。CloudWatch Logs をすでに使用していて、ロググループとログストリームが設定済みである場合は、開始する準備が整っています。また、AWS CloudTrail、Amazon Route 53、または Amazon VPC などのサービスを使用し、それらのサービスから CloudWatch Logs に移動するようにログを設定している場合は、既

にログが存在する可能性があります。CloudWatch Logs にログを送信する方法の詳細については、「[CloudWatch Logs の開始方法](#)」を参照してください。

CloudWatch Logs Insights のクエリは、ログイベントから一連のフィールドを返すか、ログイベントに対して実行された数学的な集約やその他のオペレーションの結果を返します。このチュートリアルでは、ログイベントのリストを返すクエリを示します。

サンプルクエリを実行する

CloudWatch Logs Insights サンプルクエリを実行するには

1. CloudWatch コンソールの <https://console.aws.amazon.com/cloudwatch/> を開いてください。
2. ナビゲーションペインで、[Logs] (ログ)、[Logs Insights] (ログのインサイト) の順に選択します。

[Logs Insights] (ログのインサイト) ページでは、Logs Insights QL のクエリエディタにデフォルトクエリが表示されます。デフォルトでは、最新の 20 件のログイベントが返されます。

3. [Select log group] (ロググループの選択) ドロップダウンから、クエリを実行するロググループを 1 つ以上選択します。

これが CloudWatch のクロスアカウントオブザーバビリティのモニタリングアカウントの場合は、モニタリングアカウントだけでなくソースアカウントのロググループも選択できます。1 つのクエリで複数のアカウントのログを一度にクエリできます。

ロググループは、ロググループ名、アカウント ID、またはアカウントラベルでフィルタリングできます。

標準ログクラスのロググループを選択すると、CloudWatch Logs Insights はグループ内のデータフィールドを自動的に検出します。検出されたフィールドを表示するには、ページの右上あたりにある [Fields] (フィールド) メニューを選択します。

Note

検出されたフィールドは、標準ログクラスのロググループでのみサポートされます。ログクラスの詳細については、「[ログクラス](#)」を参照してください。

4. (オプション) 時間間隔セレクタを使用して、クエリを実行する期間を選択します。
5 ~ 30 分間隔、1 時間、3 時間、12 時間間隔、またはカスタム時間枠を選択できます。
5. [Run] (実行) を選択して結果を表示します。

このチュートリアルでは、最近追加されたログイベントが 20 件表示されます。

また、CloudWatch Logs は、このロググループのログイベントを時間の経過に従って棒グラフで表示します。この棒グラフは、表に示されるイベントだけでなく、クエリと時間範囲に一致するロググループ内のイベントの分布も示します。

6. 返されたログイベントのすべてのフィールドを表示するには、番号付きイベントの左にある三角形のドロップダウンアイコンを選択します。

サンプルクエリを変更する

このチュートリアルでは、サンプルクエリを変更して、最新のログイベントを 50 件表示します。

前のチュートリアルをまだ実行していない場合は、今すぐ実行してください。このチュートリアルは、前のチュートリアルが終了した箇所から開始します。

Note

CloudWatch Logs Insights に用意されている一部のサンプルクエリでは、`limit` の代わりに `head` または `tail` コマンドを使用します。これらのコマンドは非推奨であり、`limit` に置き換えられています。ユーザーが記述するすべてのクエリで、`limit` または `head` の代わりに `tail` を使用します。

CloudWatch Logs Insights のサンプルクエリを変更するには

1. クエリエディタで、20 を 50 に変更し、[実行] を選択します。

新しいクエリの結果が表示されます。デフォルトの時間範囲でロググループに十分なデータがあるとして、これで 50 件のログイベントが一覧表示されます。

2. (オプション) 作成したクエリは保存できます。このクエリを保存するには、[保存] を選択します。詳細については、「[CloudWatch Logs Insights クエリの保存と再実行](#)」を参照してください。

サンプルクエリにフィルターコマンドを追加する

このチュートリアルでは、クエリエディタを使用してクエリに対してより強力な変更を行う方法を示します。このチュートリアルでは、取得したログイベントのフィールドに基づいて、前のクエリの結果をフィルタリングします。

前のチュートリアルをまだ実行していない場合は、今すぐ実行してください。このチュートリアルは、前のチュートリアルが終了した箇所から開始します。

前のクエリにフィルターコマンドを追加するには

1. フィルタリングするフィールドを決定します。過去 15 分間に選択したロググループに含まれるログイベントで CloudWatch Logs により検出された最も一般的なフィールドと、各フィールドが出現するログイベントの割合を確認するには、ページの右側にある [Fields (フィールド)] を選択します。

特定のログイベントに含まれているフィールドを表示するには、その行の左にあるアイコンを選択します。

ログ内のイベントに応じて、ログイベントに [awsRegion] フィールドが表示される場合があります。このチュートリアルの残りの部分では、フィルターフィールドとして [awsRegion] を使用しますが、このフィールドが使用できない場合は、別のフィールドを使用できます。

2. クエリエディタボックスで [50] の後にカーソルを置き、Enter キーを押します。
3. 新しい行で、最初に | (パイプ文字) とスペースを入力します。CloudWatch Logs Insights クエリのコマンドは、パイプ文字で区切る必要があります。
4. **filter awsRegion="us-east-1"** と入力します。
5. [Run (実行)] を選択します。

クエリが再度実行されます。今回は、新しいフィルターに一致する 50 件の最新の結果が表示されます。

別のフィールドにフィルターを適用してエラーが発生した場合は、必要に応じてフィールド名をエスケープします。フィールド名に英数字以外の文字が含まれている場合は、フィールド名の前後にバッククォート文字 (``) を挿入します (例: ``error-code`="102"`)。

英数字以外の文字を含むフィールド名にはバッククォート文字を使用する必要がありますが、値には必要ありません。値は常に引用符 (") で囲まれます。

Logs Insights QL には、強力なクエリ機能として、複数の正規表現のコマンドとサポート、算術演算、統計演算などが含まれています。詳細については、「[CloudWatch Logs Insights 言語のクエリ構文](#)」を参照してください。

チュートリアル: 集計関数を使用してクエリを実行する

集約関数は、stats コマンドで使用できます。また、他の関数の引数としても使用できます。このチュートリアルでは、指定したフィールドを含むログイベントの数をカウントするクエリコマンドを実行します。このクエリコマンドは、指定したフィールドの値でグループ化された合計数を返します。集計関数の詳細については、「Amazon CloudWatch Logs ユーザーガイド」の「[サポートされているオペレーションと関数](#)」を参照してください。

集計関数を使用したクエリの実行方法

1. CloudWatch コンソールの <https://console.aws.amazon.com/cloudwatch/> を開いてください。
2. ナビゲーションペインで、[Logs] (ログ)、[Logs Insights] (ログのインサイト) の順に選択します。
3. Logs Insights QL タブが選択されていることを確認します。
4. [Select log group] (ロググループの選択) ドロップダウンから、クエリを実行するロググループを 1 つ以上選択します。

これが CloudWatch のクロスアカウントオブザーバビリティのモニタリングアカウントの場合は、モニタリングアカウントだけでなくソースアカウントのロググループも選択できます。1 つのクエリで複数のアカウントのログを一度にクエリできます。

ロググループは、ロググループ名、アカウント ID、またはアカウントラベルでフィルタリングできます。

ロググループを選択すると、標準クラスロググループの場合 CloudWatch Logs Insights はロググループ内のデータフィールドを自動的に検出します。検出されたフィールドを表示するには、ページの右上あたりにある [Fields] (フィールド) メニューを選択します。

5. クエリエディタでデフォルトのクエリを削除し、次のコマンドを入力します。

```
stats count(*) by fieldName
```

6. *fieldName* を [Fields] (フィールド) メニューから検出されたフィールドに置換します。

[Fields] (フィールド) メニューはページの右上にあり、CloudWatch Logs Insights がロググループ内で検出したすべての検出フィールドがそこに表示されます。

7. [Run] (実行) を選択してクエリの結果を表示します。

クエリの結果には、クエリコマンドに一致するロググループ内のレコード数と、指定したフィールドの値でグループ化された合計数が表示されます。

チュートリアル: ログフィールド別にグループ化された視覚化を生成するクエリを実行する

`stats` 関数を使用するクエリを実行して、返された値をログエントリ内の 1 つ以上のフィールドの値別にグループ化すると、結果を棒グラフ、円グラフ、折れ線グラフ、積み上げ面グラフとして表示できます。これにより、ログの傾向をより効率的に視覚化できます。

視覚化用のクエリを実行するには

1. CloudWatch コンソールの <https://console.aws.amazon.com/cloudwatch/> を開いてください。
2. ナビゲーションペインで、[Logs] (ログ)、[Logs Insights] (ログのインサイト) の順に選択します。
3. [Select log group] (ロググループの選択) ドロップダウンから、クエリを実行するロググループを 1 つ以上選択します。

これが CloudWatch のクロスアカウントオブザーバビリティのモニタリングアカウントの場合は、モニタリングアカウントだけでなくソースアカウントのロググループも選択できます。1 つのクエリで複数のアカウントのログを一度にクエリできます。

ロググループは、ロググループ名、アカウント ID、またはアカウントラベルでフィルタリングできます。

4. クエリエディタで、現在の表示内容を削除し、以下の `stats` 関数を入力して、[クエリの実行] を選択します。

```
stats count(*) by @logStream
| limit 100
```

結果には、各ログストリームのロググループ内のログイベント数が表示されます。結果は 100 行に制限されます。

5. [Visualization (視覚化)] タブを選択します。
6. [線] の横にある矢印を選択し、[バー] を選択します。

棒グラフが表示され、ロググループ内のログストリームごとに棒が表示されます。

チュートリアル: 時系列の視覚化を生成するクエリを実行する

`bin()` 関数を使用するクエリを実行して、返された値を期間別にグループ化すると、結果を折れ線グラフ、積み上げ面グラフ、円グラフ、棒グラフとして表示できます。これにより、時間の経過に伴うログイベントの傾向をより効率的に視覚化できます。

視覚化用のクエリを実行するには

1. CloudWatch コンソールの <https://console.aws.amazon.com/cloudwatch/> を開いてください。
2. ナビゲーションペインで、[Logs] (ログ)、[Logs Insights] (ログのインサイト) の順に選択します。
3. Logs Insights QL タブが選択されていることを確認します。
4. [Select log group] (ロググループの選択) ドロップダウンから、クエリを実行するロググループを 1 つ以上選択します。

これが CloudWatch のクロスアカウントオブザーバビリティのモニタリングアカウントの場合は、モニタリングアカウントだけでなくソースアカウントのロググループも選択できます。1 つのクエリで複数のアカウントのログを一度にクエリできます。

ロググループは、ロググループ名、アカウント ID、またはアカウントラベルでフィルタリングできます。

5. クエリエディタで、現在の表示内容を削除し、以下の stats 関数を入力して、[クエリの実行] を選択します。

```
stats count(*) by bin(30s)
```

結果として、CloudWatch Logs が 30 秒間隔で受信したロググループ内のログイベントの数が表示されます。

6. [Visualization (視覚化)] タブを選択します。

結果が折れ線グラフとして表示されます。棒グラフ、円グラフ、積み上げ面グラフに切り替えるには、グラフの右上で [Line (線)] を選択します。

サンプルクエリ

このセクションでは、[CloudWatch コンソール](#)で実行できる一般的で便利なクエリコマンドの一覧を紹介します。クエリコマンドの実行方法については、「Amazon CloudWatch Logs ユーザーガイド」の「[チュートリアル: サンプルクエリを実行および変更する](#)」を参照してください。

クエリ構文の詳細については、「[CloudWatch Logs Insights 言語のクエリ構文](#)」を参照してください。

トピック

- [一般的なクエリ](#)
- [Lambda ログのクエリ](#)
- [Amazon VPC フローログのクエリ](#)
- [Route 53 ログのクエリ](#)
- [CloudTrail ログのクエリ](#)
- [のクエリ Amazon API Gateway](#)
- [NAT ゲートウェイに対するクエリ](#)
- [Apache サーバーのログに対するクエリ](#)
- [Amazon EventBridge のクエリ](#)
- [解析コマンドの例](#)

一般的なクエリ

最近追加された 25 件のログイベントを検索します。

```
fields @timestamp, @message | sort @timestamp desc | limit 25
```

1 時間あたりの例外数のリストを表示します。

```
filter @message like /Exception/  
  | stats count(*) as exceptionCount by bin(1h)  
  | sort exceptionCount desc
```

例外ではないログイベントのリストを取得します。

```
fields @message | filter @message not like /Exception/
```

server フィールドの一意の値ごとに最新のログイベントを表示します。

```
fields @timestamp, server, severity, message  
  | sort @timestamp asc  
  | dedup server
```

各 **severity** タイプの、**server** フィールドの一意の値ごとに最新のログイベントを表示します。


```
fields @timestamp, server, severity, message
| sort @timestamp desc
| dedup server, severity
```

Lambda ログのクエリ

過剰にプロビジョニングされたメモリの量を確認します。

```
filter @type = "REPORT"
| stats max(@memorySize / 1000 / 1000) as provisionedMemoryMB,
    min(@maxMemoryUsed / 1000 / 1000) as smallestMemoryRequestMB,
    avg(@maxMemoryUsed / 1000 / 1000) as avgMemoryUsedMB,
    max(@maxMemoryUsed / 1000 / 1000) as maxMemoryUsedMB,
    provisionedMemoryMB - maxMemoryUsedMB as overProvisionedMB
```

レイテンシーレポートを作成します。

```
filter @type = "REPORT" |
    stats avg(@duration), max(@duration), min(@duration) by bin(5m)
```

遅い関数呼び出しを検索し、再試行やクライアント側コードが原因で発生する可能性のある重複リクエストを削除します。このクエリでは、**@duration** はミリ秒単位です。

```
fields @timestamp, @requestId, @message, @logStream
| filter @type = "REPORT" and @duration > 1000
| sort @timestamp desc
| dedup @requestId
| limit 20
```

Amazon VPC フローログのクエリ

ホスト間での上位 15 件のパケット転送を検索します:

```
stats sum(packets) as packetsTransferred by srcAddr, dstAddr
| sort packetsTransferred desc
| limit 15
```

特定のサブネットにおけるホストの上位 15 バイトの転送を検索します。

```
filter isIpv4InSubnet(srcAddr, "192.0.2.0/24")
  | stats sum(bytes) as bytesTransferred by dstAddr
  | sort bytesTransferred desc
  | limit 15
```

データ転送プロトコルとして UDP を使用する IP アドレスを検索します。

```
filter protocol=17 | stats count(*) by srcAddr
```

キャプチャウィンドウでフローレコードがスキップされた IP アドレスを検索します。

```
filter logStatus="SKIPDATA"
  | stats count(*) by bin(1h) as t
  | sort t
```

接続のたびに 1 つのレコードを検索し、ネットワークの接続問題の解決を促します。

```
fields @timestamp, srcAddr, dstAddr, srcPort, dstPort, protocol, bytes
| filter logStream = 'vpc-flow-logs' and interfaceId = 'eni-0123456789abcdef0'
| sort @timestamp desc
| dedup srcAddr, dstAddr, srcPort, dstPort, protocol
| limit 20
```

Route 53 ログのクエリ

クエリタイプ別に 1 時間あたりのレコードのディストリビューションを検索します。

```
stats count(*) by queryType, bin(1h)
```

リクエスト数が最大である 10 件の DNS リゾルバーを検索します。

```
stats count(*) as numRequests by resolverIp
  | sort numRequests desc
  | limit 10
```

サーバーが DNS リクエストを完了できなかったレコード数をドメイン別およびサブドメイン別に検索します。

```
filter responseCode="SERVFAIL" | stats count(*) by queryName
```

CloudTrail ログのクエリ

各サービス、イベントタイプ、AWS リージョンのログエントリの数を検索します。

```
stats count(*) by eventSource, eventName, awsRegion
```

特定の AWS リージョンで開始または停止された Amazon EC2 ホストを検索します。

```
filter (eventName="StartInstances" or eventName="StopInstances") and awsRegion="us-east-2"
```

新しく作成された IAM ユーザーの AWS リージョン、ユーザー名、ARNs を検索します。

```
filter eventName="CreateUser"  
  | fields awsRegion, requestParameters.userName, responseElements.user.arn
```

API **UpdateTrail** の呼び出し中に例外が発生したレコードの数を検索します。

```
filter eventName="UpdateTrail" and ispresent(errorCode)  
  | stats count(*) by errorCode, errorMessage
```

TLS 1.0 または 1.1 が使用されたログエントリを検索します。

```
filter tlsDetails.tlsVersion in [ "TLSv1", "TLSv1.1" ]  
  | stats count(*) as numOutdatedTlsCalls by userIdentity.accountId, recipientAccountId,  
    eventSource, eventName, awsRegion, tlsDetails.tlsVersion, tlsDetails.cipherSuite,  
    userAgent  
  | sort eventSource, eventName, awsRegion, tlsDetails.tlsVersion
```

TLS バージョン 1.0 または 1.1 を使用したサービスごとの呼び出し数を検索します。

```
filter tlsDetails.tlsVersion in [ "TLSv1", "TLSv1.1" ]
```

```
| stats count(*) as numOutdatedTlsCalls by eventSource
| sort numOutdatedTlsCalls desc
```

のクエリ Amazon API Gateway

最新の 4XX エラーを 10 件検索します。

```
fields @timestamp, status, ip, path, httpMethod
| filter status>=400 and status<=499
| sort @timestamp desc
| limit 10
```

Amazon API Gateway アクセスロググループ内で最も長く実行されている 10 個の Amazon API Gateway リクエストを特定する

```
fields @timestamp, status, ip, path, httpMethod, responseLatency
| sort responseLatency desc
| limit 10
```

Amazon API Gateway アクセスロググループで最もよく使用されている API パスのリストを返します。

```
stats count(*) as requestCount by path
| sort requestCount desc
| limit 10
```

Amazon API Gateway アクセスロググループの統合レイテンシーレポートを作成します。

```
filter status=200
| stats avg(integrationLatency), max(integrationLatency),
min(integrationLatency) by bin(1m)
```

NAT ゲートウェイに対するクエリ

AWS 請求額が通常よりも高い場合は、CloudWatch Logs Insights を使用して上位の寄与要因を見つけることができます。次のクエリコマンドの詳細については、[「VPC の NAT ゲートウェイを介したトラフィックの上位寄与要因を見つけるにはどうすればよいですか？」](#)を参照してください。AWS プレミアムサポートページを参照してください。

Note

次のクエリコマンドの「x.x.x.x」の部分をお使いの NAT ゲートウェイのプライベート IP に置き換え、「y.y」を VPC CIDR アドレス範囲の第 1 および 第 2 オクテットの値に置き換えます。

NAT ゲートウェイ経由で最も多くのトラフィックを送信しているインスタンスを検索します。

```
filter (dstAddr like 'x.x.x.x' and srcAddr like 'y.y.')
| stats sum(bytes) as bytesTransferred by srcAddr, dstAddr
| sort bytesTransferred desc
| limit 10
```

NAT ゲートウェイ内のインスタンスとの間で送受信されているトラフィックを特定します。

```
filter (dstAddr like 'x.x.x.x' and srcAddr like 'y.y.') or (srcAddr like 'xxx.xx.xx.xx'
and dstAddr like 'y.y.')
| stats sum(bytes) as bytesTransferred by srcAddr, dstAddr
| sort bytesTransferred desc
| limit 10
```

VPC 内のインスタンスがアップロードとダウンロードの通信で最も頻繁に使用している、インターネット上の送信先を特定します。

アップロードの場合

```
filter (srcAddr like 'x.x.x.x' and dstAddr not like 'y.y.')
| stats sum(bytes) as bytesTransferred by srcAddr, dstAddr
| sort bytesTransferred desc
| limit 10
```

ダウンロードの場合

```
filter (dstAddr like 'x.x.x.x' and srcAddr not like 'y.y.')
| stats sum(bytes) as bytesTransferred by srcAddr, dstAddr
| sort bytesTransferred desc
| limit 10
```

Apache サーバーのログに対するクエリ

CloudWatch Logs Insights では Apache サーバーのログにクエリできます。以下のクエリの詳細については、AWS Cloud Operations & Migrations ブログの「[Simplifying Apache server logs with CloudWatch Logs Insights](#)」を参照してください。

アクセスログを確認してアプリケーションの /admin パスでトラフィックをチェックできるように、最も関連性の高いフィールドを検索します。

```
fields @timestamp, remoteIP, request, status, filename | sort @timestamp desc
| filter filename="/var/www/html/admin"
| limit 20
```

メインページにアクセスした際のステータスコードが「200」(成功) になっている箇所を探し、一意の GET リクエストの数を見つけます。

```
fields @timestamp, remoteIP, method, status
| filter status="200" and referrer= http://34.250.27.141/ and method= "GET"
| stats count_distinct(remoteIP) as UniqueVisits
| limit 10
```

Apache サービスが再起動した回数を確認します。

```
fields @timestamp, function, process, message
| filter message like "resuming normal operations"
| sort @timestamp desc
| limit 20
```

Amazon EventBridge のクエリ

イベント詳細タイプ別にグループ化された EventBridge イベントの数を取得します。

```
fields @timestamp, @message
| stats count(*) as numberOfEvents by `detail-type`
| sort numberOfEvents desc
```

解析コマンドの例

glob 式を使用して、ログフィールド **@message** から、抽出フィールド **@user**、**@method**、**@latency** を抽出し、**@method** および **@user** との一意の組み合わせごとに平均レイテンシーを返します。

```
parse @message "user=*, method:*, latency := *" as @user,  
    @method, @latency | stats avg(@latency) by @method,  
    @user
```

正規表現を使用して、ログフィールド **@message** から、フィールド **@user2**、**@method2**、**@latency2** を抽出し、**@method2** および **@user2** との一意の組み合わせごとに平均レイテンシーを返します。

```
parse @message /user=(?<user2>.*?), method:(?<method2>.*?),  
    latency := (?<latency2>.*?)/ | stats avg(latency2) by @method2,  
    @user2
```

フィールド **loggingTime**、**loggingType**、**loggingMessage** を抽出し、**ERROR** または **INFO** 文字列を含むログイベントをフィルタリングし、**ERROR** 文字列を含むイベントの **loggingMessage** および **loggingType** フィールドのみを表示します。

```
FIELDS @message  
| PARSE @message "*" [*] "*" as loggingTime, loggingType, loggingMessage  
| FILTER loggingType IN ["ERROR", "INFO"]  
| DISPLAY loggingMessage, loggingType = "ERROR" as isError
```

(diff) を以前の時間範囲と比較する

CloudWatch Logs Insights と Logs Insights QL を併用し、ログイベントの変化を経時的に比較できます。最近の時間範囲に取り込まれたログイベントと、直前の期間のログを比較できます。または、同様の過去の期間と比較することもできます。これにより、ログ内のエラーが最近発生したものか、または既に発生していたかを調べることができ、他の傾向を見つけやすくなります。

比較クエリは、未加工のログイベントではなく、結果のパターンのみを返します。返されるパターンを基に、ログイベントの傾向と経時的な変化をすばやく確認できるようになります。比較クエリを実行してパターン結果を取得したら、関心のあるパターンの raw ログイベントのサンプルを確認できます。ログパターンの詳細については、「[パターン分析](#)」を参照してください。

比較クエリを実行すると、選択した元のクエリ期間と比較期間の 2 つの異なる期間に対してクエリが分析されます。比較期間は、常に元のクエリ期間と同じ長さです。比較のデフォルトの時間間隔は次のとおりです。

- 前の期間 — クエリ期間の直前の期間と比較します。
- 前日 — クエリ期間の 1 日前の期間と比較します。

- 前週 — クエリ期間の 1 週間前の期間と比較します。
- 前月 — クエリ期間の 1 か月前の期間と比較します。

 Note

比較を使用したクエリでは、組み合わせた時間範囲で単一の CloudWatch Logs Insights クエリを実行する場合と同様の料金が発生します。詳細については、「[Amazon CloudWatch 料金表](#)」をご覧ください。

比較クエリを実行するには

1. CloudWatch コンソールの <https://console.aws.amazon.com/cloudwatch/> を開いてください。
2. ナビゲーションペインで、[ログ]、[Logs Insights] を選択します。

デフォルトのクエリがクエリボックスに表示されます。

3. Logs Insights QL タブが選択されていることを確認します。
4. デフォルトのクエリを保持するか、別のクエリを入力します。
5. [ロググループを選択] ドロップダウンから、クエリを実行するロググループを 1 つ以上選択します。
6. (オプション) 時間間隔セレクタを使用して、クエリを実行する期間を選択します。デフォルトのクエリは、過去 1 時間のログデータです。
7. 時間範囲セレクタで、[比較] を選択します。次に、元のログと比較する前の期間を選択し、[適用] を選択します。
8. [Run query] (クエリの実行) を選択します。

クエリが比較期間からデータを取得できるようにするため、diff コマンドがクエリに追加されます。

9. [パターン] タブを選択して結果を表示します。

テーブルに表示される情報は、次のとおりです。

- 各 [パターン] では、パターンの可変部分が動的トークン記号 `<string-number>` に置き換えられます。`###` は、トークンが表すデータのタイプの説明です。`##` は、他の動的トークンと比較して、このトークンがパターン内のどこに表示されるかを示します。詳細については、「[パターン分析](#)」を参照してください。

- [イベント数] は、元のより最新の期間にそのパターンを持つログイベントの数です。
 - [イベント数の差異] は、現在の期間と比較期間の一致するログイベントの数の差です。正の差は、現在の期間にこのようなイベントが多いことを意味します。
 - [差異の説明] は、現在の期間と比較期間の間のパターンの変化を簡単に要約します。
 - [重要度タイプ] は、FATAL、ERROR、WARN などのログイベントで検出された単語に基づいて、このパターンを持つログイベントの推定重要度です。
10. リスト内のパターンの 1 つをさらに検査するには、[検査] 列の アイコンを選択してパターンの 1 つを選択します。

[パターン検査] ペインが表示され、以下が表示されます。

- [パターン]。パターン内のトークンを選択して、そのトークンの値を分析します。
- クエリされた時間範囲におけるパターンの出現回数を示すヒストグラム。これにより、パターンの発生の急増など、興味深い傾向を特定できます。
- [ログサンプル] タブには、選択したパターンに一致するログイベントがいくつか表示されます。
- [トークンの値] タブには、選択している場合は、選択した動的トークンの値が表示されます。

 Note

トークンごとに最大 10 個のトークン値がキャプチャされます。トークン数は正確ではない可能性があります。CloudWatch Logs は確率カウンターを使用して、絶対値ではなくトークン数を生成します。

- [関連パターン] タブには、検査するパターンとほぼ同じ時間に頻繁に発生した他のパターンが表示されます。例えば、ERROR メッセージのパターンに通常、追加の詳細を含む INFO とマークされた別のログイベントが伴っていた場合、そのパターンがここに表示されます。

グラフでログデータを視覚化する

棒グラフ、折れ線グラフ、積層型面グラフなどの視覚化を使用して、ログデータのパターンをより効率的に識別できます。CloudWatch Logs Insights は、stats 関数と 1 つ以上の集計関数を使用するクエリの視覚化を生成します。詳細については、「[stats](#)」を参照してください。

OpenSearch パイプ処理言語 (PPL)

このセクションでは、OpenSearch PPL を使用した CloudWatch Logs のクエリの基本的な概要を示します。PPL を使用すると、パイプ集計コマンドを使用してデータを取得、クエリ、分析できるため、複雑なクエリの理解と作成が容易になります。この構文は Unix パイプに基づいており、コマンドを連鎖してデータの変換と処理を行えます。PPL を使用すると、データをフィルターして集計し、豊富な一連の算術、文字列、日付、条件付きなどの関数を用いて分析できます。

PPL クエリSOURCEに を含めることは、AWS CLI または API を使用してクエリを作成するときに、ロググループのフィールドインデックスと、クエリに含めるデータソースを指定するのに役立ちます。SOURCE コマンドは、CloudWatch コンソールではなく、AWS CLI および API でのみサポートされます。CloudWatch コンソールを使用してクエリを開始するときは、コンソールインターフェイスを使用してロググループとデータソースの名前とタイプを指定します。

aws:fieldIndex を使用してインデックス付きデータのみを返すには、クエリで指定したフィールドでインデックス付けされたロググループのみをスキャンするようクエリに強制します。関連するロググループは、filterIndex コマンドで指定されたフィールドに基づいて自動的に選択されます。これにより、クエリで指定された フィールドを含むログイベントを持たないロググループをスキップし、このフィールドインデックスのクエリで指定された値と一致するロググループのみをスキャンすることで、スキャンされたボリュームが減少します。aws:fieldIndex を使用して、ソースコマンドのフィールド名と値とともにフィールド名を指定し、指定されたフィールドと値を含むインデックス付きデータのみをクエリします。詳細については、[フィールドインデックスを作成してクエリパフォーマンスを改善し、スキャン量を削減する](#)を参照してください。

OpenSearch PPL は、標準ログクラスのロググループのクエリに使用できます。

Note

CloudWatch Logs でサポートされているすべての OpenSearch PPL クエリコマンドと構文や制限の詳細については、「OpenSearch Service デベロッパーガイド」の「[サポートされている PPL コマンド](#)」を参照してください。

使用できる他のクエリ言語については、「[CloudWatch Logs Insights](#)」、「[OpenSearch Service SQL](#)」、「[CloudWatchMetrics Insights](#)」を参照してください。

コマンドまたは関数	クエリの例	説明
fields	<code>fields field1, field2</code>	射影が必要な一連のフィールドを表示します。
join	<code>LEFT JOIN left=l, right=r on l.id = r.id `join_right_lg` fields l.field_1, r.field_2</code>	2つのデータセットを結合します。
where	<code>where field1="success" where field2 != "i-023fe0a90929d8822" fields field3, field4, field5,field6 head 1000</code>	指定した条件に基づいてデータをフィルタリングします。
aws:field Index	<code>source = [`aws:fieldIndex`="region", `region` = "us-west-2"] where status = 200 head 10</code>	クエリで指定したフィールドでインデックスが作成されたロググループのみをスキャンするようにクエリに強制することで、インデックス付きデータのみを返します。
stats	<code>stats count(), count(field1), min(field 1), max(field1), avg(field1) by field2 head 1000</code>	集計と計算を実行します。
parse	<code>parse field1 ".*/(?<field2>[^/]+\$)" where field2 = "requestId" fields field1, field2 head 1000</code>	文字列から正規表現 (regex) パターンを抽出し、抽出されたパターンを表示します。抽出

コマンドまたは関数	クエリの例	説明
		されたパターンはその後、新しいフィールドの作成やデータのフィルタリングに使用できます。
sort	<pre>stats count(), count(field1), min(field1) as field1Alias, max(`field1`), avg(`field1`) by field2 sort -field1Alias head 1000</pre>	表示された結果をフィールド名で並べ替えます。sort-FieldName を使用して降順で並べ替えます。
eval	<pre>eval field2 = field1 * 2 fields field1, field2 head 20</pre>	フィールドの値を変更または処理し、別のフィールドに保存します。これは、列を数学的に変更したり、列に文字列関数を適用したり、列に日付関数を適用したりするのに有効です。
rename	<pre>rename field2 as field1 fields field1;</pre>	検索結果の 1 つ以上のフィールドの名前を変更します。

コマンドまたは関数	クエリの例	説明
head	<code>fields `@message` head 20</code>	表示されるクエリ結果を最初の N 行に制限します。
top	<code>top 2 field1 by field2</code>	フィールド内で最も頻度の高い値を抽出します。
dedup	<code>dedup field1 fields field1, field2, field3</code>	指定したフィールドに基づいて重複するエントリを削除します。
rare	<code>rare field1 by field2</code>	フィールドリスト内のすべてのフィールドの最低頻度の値を検索します。
subquery	<code>where field_1 IN [search source=`subquery_lg` fields field_2] fields id, field_1</code>	PPL ステートメント内でネストされた複雑なクエリを実行します。
trendline	<code>trendline sma(2, field1) as field1Alias</code>	フィールドの移動平均を計算します。

コマンドまたは関数	クエリの例	説明
eventStats	<code>eventstats sum(field1) by field2</code>	計算された要約統計量を使用してイベントデータを補強します。イベント内の指定されたフィールドを分析し、さまざまな統計量を計算して、その結果を新しいフィールドとして各イベントに追加します。
expand	<code>eval tags_array_string = json_extract(`@message`, '\$.tags') eval tags_array = json_array(json_extract(tags_string, '\$[0]'), json_extract(tags_string, '\$[1]')) expand tags_array as color_tags</code>	複数の値を含むフィールドを別々の行に分割し、指定されたフィールドの値ごとに新しい行を作成します。
fillnull	<code>fields `@timestamp`, error_code, status_code fillnull using status_code = "UNKNOWN", error_code = "UNKNOWN"</code>	null フィールドに指定した値を入力します。1 つ以上のフィールドで使用できます。

コマンドまたは関数	クエリの例	説明
flatten	<pre>eval metadata_struct = json_object('size', json_extract(metadata_string, '\$.size'), 'color', json_extract(metadata_string, '\$.color')) flatten metadata_struct as (meta_size, meta_color)</pre>	フィールドをフラット化します。フィールドは <code>struct<?, ?></code> または <code>array<struct<?, ?>></code> のタイプである必要があります。
cidrmatch	<pre>where cidrmatch(ip, '2003:db8::/32') fields ip</pre>	指定された IP アドレスが指定された CIDR 範囲内にあるかどうかを確認します。
fieldsummary	<pre>where field1 != 200 fieldsummary includefields= field1 nulls=true</pre>	各フィールドの基本統計 (カウント、個別カウント、最小、最大、平均、stddev、平均) を計算します。
grok	<pre>grok email '.*@%{HOSTNAME:host}' fields email, host</pre>	grok パターンでテキストフィールドを解析し、検索結果に結果を追加します。

コマンドまたは関数	クエリの例	説明
文字列関数	<pre>eval field1Len = LENGTH(field1) fields field1Len</pre>	PPL クエリ内の文字列およびテキストデータを操作および変換できる PPL の組み込み関数。例えば、大文字と小文字の変換、文字列の組み合わせ、パーツの抽出、テキストのクリーニングなどです。

コマンドまたは関数	クエリの例	説明
日付および時刻関数	<pre>eval newDate = ADDDATE(DATE('2020-08-26'), 1) fields newDate</pre>	PPL クエリで日付とタイムスタンプデータを処理および変換するための組み込み関数。例えば、date_add、date_format、datediff、date-sub、timestampadd、timestampdiff、current_timezone、utc_timestamp、current_date などで
条件関数	<pre>eval field2 = isnull(field1) fields field2, field1, field3</pre>	特定のフィールド条件をチェックし、式を条件付きで評価する組み込み関数。例えば、field1 が null の場合、field 2 を返します。

コマンドまたは関数	クエリの例	説明
数学関数	<pre>eval field2 = ACOS(field1) fields field1</pre>	PPL クエリで数学的計算と変換を実行するための組み込み関数。例えば、abs (絶対値)、round (四捨五入数値)、sqrt (平方根)、pow (出力計算)、ceil (最も近い整数に切り上げ) などです。
暗号化関数	<pre>eval crypto = MD5(field) head 1000</pre>	指定されたフィールドのハッシュを計算するには

コマンドまたは関数	クエリの例	説明
JSON 関数	<pre>eval valid_json = json('[1,2,3,{"f1":1,"f2":[5,6]},4]') fields valid_json</pre>	配列、抽出、検証など、JSON を処理するための組み込み関数。例えば、 <code>json_object</code> 、 <code>json_array</code> 、 <code>to_json_string</code> 、 <code>json_array_length</code> 、 <code>json_extract</code> 、 <code>json_keys</code> 、 <code>json_valid</code> などです。

クエリスコープ

クエリに SOURCE を含めることは、AWS CLI または API を使用してクエリを作成するときに、クエリに含めるロググループを指定するのに役立ちます。SOURCE コマンドは、CloudWatch コンソールではなく、AWS CLI および API でのみサポートされています。CloudWatch コンソールを使用してクエリを開始するときは、コンソールインターフェイスを使用してロググループとデータソースの名前とタイプを指定します。

PPL の `source` コマンドは、それらを指定する複数の方法をサポートするようになりました。

1. ロググループ
2. フィールドインデックス - 新規
3. データソースとタイプ - 新規

ロググループ

ロググループのソース選択は、検索する必要がある正確なロググループ (複数可) がわかっている場合に使用できます。

```
source = [lg:`aws/lambda/my-function`] | where status = 200 | head 10
```

フィールドインデックス

フィールドインデックスベースのソース選択は、インデックスが作成されたフィールドをフィルターがターゲットにする場合、インデックスが付けられたデータのみ結果を制限することで、クエリされるデータの量を減らします。関連するロググループは、`filterIndex` コマンドで指定されたフィールドに基づいて自動的に選択されます。フィールドインデックスとその作成方法の詳細については、「[Create field indexes to improve query performance and reduce scan volume](#)」を参照してください。

`aws:fieldIndex` を使用してインデックス付きデータのみを返すには、クエリで指定したフィールドでインデックス付けされたロググループのみをスキャンするようクエリに強制します。このフィールドでインデックス付けされるこれらのロググループの場合、インデックス付けされたフィールドのクエリで指定されたフィールドを含む任意のログイベントを持たないロググループをスキップすることで、クエリがさらに最適化されます。さらに、このフィールドインデックスのクエリで指定された値に一致するこれらのロググループのログイベントのみをスキャンしようとすることで、スキャンされるボリュームが減少します。フィールドインデックスとその作成方法の詳細については、「[Create field indexes to improve query performance and reduce scan volume](#)」を参照してください。

PPL では、`aws:fieldIndex` を使用して、インデックスとして扱うキーと値のペアを指定します。構文は次のとおりです。

```
source = [`aws:fieldIndex`="region", `region` = "us-west-2"] | where status = 200 | head 10
```

次のとおりです。

- ``aws:fieldIndex`="region"` はリージョンをフィールドインデックスとして識別します。
 - 注: `=` の代わりに、顧客は `IN` を使用して複数のインデックスを指定できます (以下の例)
- ``region`="us-west-2"` は、適用するフィルター条件を識別します。
 - 注: `=` の代わりに、顧客は `IN` を使用して複数の値を指定できます (以下の例を参照)

お客様は次のように複数の fieldIndexes を指定できます。

```
source = [`aws:fieldIndex` IN ("status", "region"), `status` = 200, `region` IN ("us-west-2", "us-east-1")] | head 10
```

データソースとタイプ

データソースとタイプベースのソース選択は、クエリが必要な正確なデータソースがわかっている場合に使用できます。このクエリは、指定されたデータソースとタイプを含む 1 つ以上のロググループで実行されます。

```
source = [ds:`data_source.type` ] | where status = 200 | head 10
```

データソースクエリでサポートされる PPL

PPL でデータソースをクエリするためのユースケースをサポートするには、動的ソースセレクト句を使用できます。この構文を使用して、検索コマンドでデータソースを指定することで、データソースをクエリできます。最大 10 個のデータソースを指定できます。

[Syntax] (構文)

```
source=[ds:`DataSource1.Type1`, ds:`DataSource2.Type2`, ...ds:`DataSourceN.TypeN`]
```

クエリの例

```
search source=[ds:`DataSource1.Type1`, ds:`DataSource2.Type2`] | fields field1, field2
```

組み合わせられた例

お客様はすべてのソース選択演算子を任意の順序で指定でき、結果は適用されるすべての条件の共通部分になります。

たとえば、/aws/lambda/my-function-1 には、さまざまなインデックスを含む複数のデータソースとタイプが含まれている場合があります。次のクエリが実行されると、返される結果にはソースイベントとタイプ DataSource1.Type1 のイベントのみが含まれ、'status' = 200 の基準に一致します。

```
search source=[
  ds:`DataSource1.Type1`,
  lg:`/aws/lambda/my-function-1`,
```

```
`aws:fieldIndex` IN ("status"), `status` = 200  
]
```

制限事項

OpenSearch PPL を使用して CloudWatch Logs Insights でクエリを実行する場合、次の制限が適用されます。

- データソースクエリでは、結合コマンドまたはサブクエリコマンドを使用できません。

OpenSearch 構造化クエリ言語 (SQL)

このセクションでは、OpenSearch SQL を使用した CloudWatch Logs のクエリの基本的な概要を示します。リレーショナルデータベースの使用に慣れている場合は、使い慣れたオプションを提供します。OpenSearch SQL は SQL 機能のサブセットを提供するため、アドホッククエリやデータ分析タスクの実行に適しています。OpenSearch SQL では、SELECT、FROM、WHERE、GROUP BY、HAVING、その他のさまざまな SQL コマンドや関数などのコマンドを使用できます。ロググループ間で JOINS を実行し、サブクエリを使用してロググループ間でデータを関連付け、一連の豊富な JSON、算術、文字列、条件付きなどの SQL 関数を使用してログとセキュリティデータに対する強力な分析を実行できます。

filterIndex を使用してインデックス付きデータのみを返すには、クエリで指定したフィールドでインデックス付けされたロググループのみをスキャンするようクエリに強制します。クエリで指定されたフィールドを含むログイベントを持たないロググループをスキップし、このフィールドインデックスのクエリで指定された値と一致するロググループのみをスキャンすることで、スキャンされたボリュームを減らします。filterIndex を使用してフィールド名を指定し、フィールド名と値を指定して、指定されたフィールドと値を含むインデックス付きデータのみをクエリします。

OpenSearch SQL は、標準ログクラスのロググループのクエリに使用できます。SQL は、データソース名とデータソースタイプを使用したクエリもサポートしています。

Note

次の表は、CloudWatch Logs でサポートされている SQL コマンドと関数の一覧です。構文を含むすべての OpenSearch SQL コマンドの詳細については、「[OpenSearch Service デベロッパーガイド](#)」の「[サポートされている SQL コマンド](#)」を参照してください。使用できる他のクエリ言語については、「[CloudWatch Logs Insights](#)」、「[OpenSearch Service PPL](#)」、「[CloudWatchMetrics Insights](#)」を参照してください。

サポートされている SQL コマンド

Note

クエリの例の列では、クエリするデータソースに応じて、*<logGroup>* を適宜置き換えます。

コマンドまたは関数	クエリの例	説明
SELECT	SELECT `@message`, Operation FROM `LogGroupA`	射影された値を表示します。
FROM	SELECT `@message`, Operation FROM `LogGroupA`	データを取得するソーステーブル (複数可) またはビュー (複数可) を指定する組み込み句。さまざまなタイプの結合とサブクエリをサポートします。
WHERE	SELECT * FROM `LogGroupA` WHERE Operation = 'x'	指定されたフィールド基準に基づいてログイベントをフィルタリングします。
filterIndex	SELECT * FROM `filterIndex('region' = 'us-east-1')` WHERE status = 200 LIMIT 10;	クエリで指定したフィールドでインデックスが作成されたロググループのみを

コマンドまたは関数	クエリの例	説明
		スキャンするようにクエリに強制することで、インデックス付きデータのみを返します。
グループ化の条件	<pre>SELECT `@logStream`, COUNT(*) as log_count FROM `LogGroupA` GROUP BY `@logStream`</pre>	グループはカテゴリに基づいてイベントをログに記録し、統計に基づいて平均を見つけます。
HAVING	<pre>SELECT `@logStream`, COUNT(*) as log_count FROM `LogGroupA` GROUP BY `@logStream` HAVING log_count > 100</pre>	グループ化条件に基づいて結果をフィルタリングします。
ORDER BY	<pre>SELECT * FROM `LogGroupA` ORDER BY `@timestamp` DESC</pre>	ORDER BY 句のフィールドに基づいて結果が並べ替えられます。昇順または降順のいずれかで並べ替えることができます。

コマンドまたは関数	クエリの例	説明
JOIN	<pre>SELECT A.`@message`, B.`@timestamp` FROM `LogGroupA` as A INNER JOIN `LogGroupB` as B ON A.`requestId` = B.`requestId`</pre>	共通フィールドに基づいて2つのテーブルの結果を結合します。Inner JOIN または Left Outer Join を指定する必要があります。
LIMIT	<pre>Select * from `LogGroupA` limit 10</pre>	表示されるクエリ結果を最初のN行に制限します。
文字列関数	<pre>SELECT upper(Operation) , lower(Opera- tion), Operation FROM `LogGroupA`</pre>	SQL クエリ内の文字列およびテキストデータを操作および変換できる SQL の組み込み関数。例えば、大文字と小文字の変換、文字列の組み合わせ、パーツの抽出、テキストのクリーニングなどです。

コマンドまたは関数	クエリの例	説明
日付関数	<pre>SELECT current_date() as today, date_add(current_date(), 30) as thirty_days_later, last_day(current_date()) as month_end FROM `LogGroupA`</pre>	SQL クエリで日付とタイムスタンプデータを処理および変換するための組み込み関数。例えば、date_add、date_format、datediff、current_date などです。
条件関数	<pre>SELECT Operation, IF(Error > 0, 'High', 'Low') as error_category FROM `LogGroupA`;</pre>	指定された条件に基づいてアクションを実行するか、式を条件付きで評価する組み込み関数。例えば、CASE と IF が挙げられます。
集計関数	<pre>SELECT AVG(bytes) as bytesWritten FROM `LogGroupA`</pre>	複数の行で計算を実行して 1 つの要約値を生成する組み込み関数。例えば、SUM、COUNT、AVG、MAX、MIN などです。

コマンドまたは関数	クエリの例	説明
JSON 関数	<pre>SELECT get_json_object(json_column, '\$.name') as name FROM `LogGroupA`</pre>	SQL クエリ (from_json、to_json、get_json_object、json_tuple など) 内の JSON 形式のデータを解析、抽出、変更、クエリするための組み込み関数。データセット内の JSON 構造を操作できます。
配列関数	<pre>SELECT scores, size(scores) as length, array_contains(scores, 90) as has_90 FROM `LogGroupA`;</pre>	SQL クエリで配列タイプの列を操作するための組み込み関数。配列データ (サイズ、展開、array_contains など) に対するアクセス、変更、分析などの操作を可能にします。

コマンドまたは関数	クエリの例	説明
Window 関数	<pre>SELECT field1, field2, RANK() OVER (ORDER BY field2 DESC) as field2Rank FROM `LogGroupA`;</pre>	現在の行 (ウィンドウ) に関連する指定された行セットで計算を実行する組み込み関数。ランキング、合計実行、移動平均などの操作を有効にします。例えば、ROW_NUMBER、RANK、LAG、LEAD などです。
変換関数	<pre>SELECT CAST('123' AS INT) as converted_number, CAST(123 AS STRING) as converted_string FROM `LogGroupA`</pre>	SQL クエリ内でデータをあるタイプから別のタイプに変換するための組み込み関数。データ型変換と形式変換を可能にします。例えば、CAST、TO_DATE、TO_TIMESTAMP、BINARY などです。

コマンドまたは関数	クエリの例	説明
述語関数	<pre>SELECT scores, size(scores) as length, array_contains(scores, 90) as has_90 FROM `LogGroupA`;</pre>	条件を評価し、指定された条件またはパターンに基づいてブール値 (true/false) を返す組み込み関数。例えば、IN、LIKE、BETWEEN、IS NULL、EXISTS などです。
複数のロググループを選択する	<pre>SELECT lg1.field1, lg1.field2 from `logGroups( logGroupIdentifier: ['LogGroup1', 'LogGroup2'])` as lg1 where lg1.field3= "Success"</pre>	SELECT ステートメントで複数のロググループを指定できます。
複数のデータソースを選択する	<pre>SELECT ds1.field1, ds1.field2 from `dataSource(['DataSource1', 'DataSource2'])` as ds1 where ds1.field3= "Success"</pre>	SELECT ステートメントで複数のデータソースを指定できます

multi-log-group クエリでサポートされている SQL

SQL で複数のロググループをクエリするためのユースケースをサポートするには、logGroups コマンドを使用できます。この構文を使用し、FROM コマンドでそれらを指定することで複数のロググループをクエリできます。

構文:

```
`logGroups(
  logGroupIdentifier: ['LogGroup1', 'LogGroup2', ... 'LogGroupn']
```

```
)
```

この構文では、`logGroupIdentifier` パラメータに最大 50 個のロググループを指定できます。モニタリングアカウントのロググループを参照するには、`LogGroup` の名前の代わりに ARN を使用します。

クエリの例:

```
SELECT LG1.Column1, LG1.Column2 from `logGroups(  
    logGroupIdentifier: ['LogGroup1', 'LogGroup2']  
)` as LG1 WHERE LG1.Column1 = 'ABC'
```

FROM ステートメントの後に複数のロググループを含む次の構文は、CloudWatch Logs のクエリではサポートされていません。

```
SELECT Column1, Column2 FROM 'LogGroup1', 'LogGroup2', ... 'LogGroupn'  
WHERE Column1 = 'ABC'
```

データソースクエリでサポートされている SQL

SQL でデータソースをクエリするためのユースケースをサポートするには、`dataSource` コマンドを使用できます。この構文を使用して、`FROM` コマンドでデータソースを指定することで、データソースをクエリできます。最大 10 個のデータソースを指定できます。

[Syntax] (構文)

```
`dataSource(  
    ['DataSource1', 'DataSource2', ... 'DataSourcen']  
)`
```

クエリの例

```
SELECT DS1.Column1, DS1.Column2 from `dataSource(  
    ['DataSource1', 'DataSource2']  
)` as DS1 WHERE DS1.Column1 = 'ABC'
```

クエリスコープ

AWS CLI および API では、ロググループ、データソースとタイプ、フィールドインデックスを使用して、クエリするログを指定できます。

ロググループ

ロググループのソース選択は、検索する必要がある正確なロググループ (複数可) がわかっている場合に使用できます。

```
SELECT * FROM `logGroups(logGroupIdentifier: ['/aws/lambda/my-function'])`;
```

データソースとタイプ

お客様は、データソース名とデータソースタイプを使用してログをクエリできます。

データソースとタイプベースのソース選択は、クエリが必要な正確なデータソースがわかっている場合に使用できます。このクエリは、指定されたデータソースとタイプを含む 1 つ以上のロググループで実行されます。

SQL でデータソースをクエリするためのユースケースをサポートするには、dataSource コマンドを使用できます。この構文を使用して、FROM コマンドでデータソースを指定することで、データソースをクエリできます。最大 10 個のデータソースを指定できます。

構文:

```
`dataSource(  
    ['DataSource1.Type1', 'DataSource2.Type2', ... 'DataSourceN.TypeN']  
)`
```

クエリの例:

```
SELECT DS1.Column1, DS1.Column2 from `dataSource(  
    ['DataSource1.Type1', 'DataSource2.Type2']  
)` as DS1 WHERE DS1.Column1 = 'ABC'
```

データソースによるクエリの詳細については、「」を参照してください [ファセットを使用してログをグループ化および探索する](#)。

組み合わせられた例

お客様はバックティック内のすべてのソース選択演算子を任意の順序で指定でき、結果は適用されるすべての条件の共通部分に基づきます。

たとえば、/aws/lambda/my-function-1 には、さまざまなインデックスを含む複数のデータソースとタイプが含まれている場合があります。次のクエリが実行されると、返される結果にはソースイベントとタイプ DataSource1.Type1 のイベントのみが含まれ、'status' = 200 の基準に一致します。

```
SELECT * FROM `
  logGroups(logGroupIdentifier: ['/aws/lambda/my-function'])
  filterIndex('status' = 200)
  dataSource(['DataSource1.Type1'])
`;
```

フィールドインデックス

フィールドインデックススペースのソース選択は、フィルターがインデックス付きフィールドをターゲットにするときに、関連するロググループを自動的に識別し、スキャンボリュームとクエリランタイムを削減します。

`filterIndex` を使用してインデックス付きデータのみを返すには、クエリで指定したフィールドでインデックス付けされたロググループのみをスキャンするようクエリに強制します。このフィールドでインデックス付けされるこれらのロググループの場合、インデックス付けされたフィールドのクエリで指定されたフィールドを含む任意のログイベントを持たないロググループをスキップすることで、クエリがさらに最適化されます。さらに、このフィールドインデックスのクエリで指定された値に一致するこれらのロググループのログイベントのみをスキャンしようとすることで、スキャンされるボリュームが減少します。フィールドインデックスとその作成方法の詳細については、[「フィールドインデックスを作成してクエリのパフォーマンスを向上させ、スキャンボリュームを減らす」](#)を参照してください。

SQL では、`filterIndex` を使用して、インデックスとして扱うキーと値のペアを指定します。構文は次のとおりです。

```
SELECT * FROM `filterIndex('region' = 'us-east-1')`;
```

次のとおりです。

1. `filterIndex(...)` は、キー値をフィールドインデックスとして扱います。各キーと値のペアはカンマで区切られます (以下の例)
2. 「region」 = 「us-east-1」は、適用する実際の条件を指定します。
 - a. 注: = の代わりに、顧客は IN を使用して複数の値を指定できます (以下の例を参照)

複数の `filterIndex` を使用すると、「AND」を使用して条件が結合されます。この例では、us-east-1 または us-west-2 の status=200 と region に一致するログがクエリされます。

```
SELECT * FROM `filterIndex('status' = 200, 'region' IN ['us-east-1', 'us-west-2'])`;
```


制限事項

OpenSearch SQL を使用して CloudWatch Logs Insights でクエリを実行する場合、次の制限が適用されます。

- SELECT ステートメントに含めることができる JOIN は 1 つだけです。
- データソースクエリで JOIN またはサブクエリを使用することはできません。
- ネストされたサブクエリは 1 つのレベルのみサポートされます。
- セミコロン (;) で区切られた複数のステートメントのクエリはサポートされていません。
- フィールド名が同一であるが、大文字や小文字のみが異なる (field1 および FIELD1 など) クエリはサポートされていません。

例えば、次のクエリはサポートされません。

```
Select AWSAccountId, AwsAccountId from LogGroup
```

ただし、このフィールド名 (@logStream) は両方のロググループで同一であるため、次のクエリがサポートされています。

```
Select a.`@logStream`, b.`@logStream` from Table A INNER Join Table B on a.id = b.id
```

- 関数と式はフィールド名に対して操作し、FROM 句で指定されたロググループを持つ SELECT ステートメントの一部である必要があります。

例として、次のクエリはサポートされていません。

```
SELECT cos(10) FROM LogGroup
```

このクエリはサポートされています。

```
SELECT cos(field1) FROM LogGroup
```

- SQL コマンドまたは PPL コマンドを使用する場合は、特定のフィールドをバッククオートで囲んでクエリを正常に実行します。特殊文字 (アルファベットや数字でないもの) を含むフィールドにはバッククオートが必要です。例えば、@message、Operation.Export、Test::Field をバッククオートで囲みます。フィールド名がアルファベットのみの場合はバッククオートで囲む必要はありません。

シンプルなフィールドを含むクエリの例:

```
SELECT SessionToken, Operation, StartTime FROM `LogGroup-A`  
LIMIT 1000;
```

バッククオートが付け足された類似のクエリ:

```
SELECT `@SessionToken`, `@Operation`, `@StartTime` FROM `LogGroup-A` LIMIT 1000;
```

自然言語を使用した CloudWatch Logs Insights クエリの生成と更新

CloudWatch Logs は自然言語クエリ機能をサポートしており、[CloudWatch Logs Insights](#)、[OpenSearch Service PPL](#)、[OpenSearch Service SQL](#)、[CloudWatch Metrics Insights](#) のクエリを生成し、更新する際に便利です。

この機能を使用すると、どのような CloudWatch Logs データを探しているのかを平易な英語で質問したり、具体的に説明したりすることができます。自然言語機能が働いて、入力したプロンプトに基づいてクエリが生成され、クエリの仕組みを説明した文が 1 行ずつ表示されます。また、さらにデータを詳しく調査できるように、生成されたクエリを更新することもできます。

環境に応じて、「転送されたバイト数で上位 100 のソース IP アドレスは何ですか?」や「Lambda 関数の最も遅いリクエストを 10 個見つけてください。」などのプロンプトを入力できます。

Note

自然言語クエリ機能はリージョン別のサービスです。一部のリージョンでは、この機能は、クエリ プロンプトを処理するために、米国のリージョンに対してクロスリージョン呼び出しを行います。詳細については、「[Amazon CloudWatch が自然言語クエリ結果の要約とクエリ生成のリージョン別サポートを拡張する](#)」を参照してください。

この機能を使用して CloudWatch Logs Insights クエリを生成するには、CloudWatch Logs Insights クエリエディタを開き、クエリするロググループを選択し、[Generate query] を選択します。

⚠ Important

自然言語クエリ機能を使用するに

は、[CloudWatchLogsFullAccess](#)、[CloudWatchLogsReadOnlyAccess](#)、[AdministratorAccess](#)、[ReadOn](#)

の IAM ポリシーでサインインするか、cloudwatch:GenerateQuery のアクセス権限が必要です。

クエリの例

このセクションの例では、自然言語機能を使用して、クエリを生成し更新する方法について説明します。

i Note

CloudWatch Logs Insights クエリエディタと構文の詳細については、「[CloudWatch Logs Insights query syntax](#)」を参照してください。

例: 自然言語クエリを生成する

自然言語を使用してクエリを生成するには、プロンプトを入力し、[新しいクエリを生成] を選択します。これらの例は、基本的な検索を実行するクエリを示しています。

Prompt

以下は、最も遅い 10 個の Lambda 関数の呼び出しを検索する機能を指示するプロンプトの例です。

```
Find the 10 slowest requests
```

クエリ

以下は、自然言語機能がプロンプトに基づいて生成した CloudWatch Logs Insights クエリ言語を使用したクエリです。プロンプトがクエリの前にあるコメントにどのように表示されるかに注意してください。クエリの後に、クエリの仕組みを説明した文があります。

```
# Find the 10 slowest requests  
fields @timestamp, @message, @duration
```

```
| sort @duration desc
| limit 10
# This query retrieves the timestamp, message and duration fields from the logs and
  sorts them in descending order by duration to find the 10 slowest requests.
```

Note

プロンプトとクエリの仕組みを説明した文が表示されないようにするには、エディタの歯車アイコンを使用します。

Prompt

OpenSearch SQL クエリを生成するには、OpenSearch SQL タブを選択し、クエリジェネレータープロンプトボックスを開いて自然言語プロンプトを入力します。OpenSearch SQL クエリを生成するために自然言語機能が使用するプロンプトの例を以下に示します。

```
Give me the number of errors and exceptions per hour
```

クエリ

以下は、1 時間あたりで集計されたエラーと例外の数を見つけるために使用できる、そのプロンプトによって生成された SQL クエリです。

```
SELECT DATE_FORMAT(`@timestamp`, 'yyyy-MM-dd HH') AS hour,
        COUNT(*) AS error_count
FROM `aws/lambda/CloudWatchOdysseyQueryGen`
WHERE `@message` LIKE '%error%'
      OR `@message` LIKE '%exception%'
GROUP BY DATE_FORMAT(`@timestamp`, 'yyyy-MM-dd HH')
ORDER BY hour
```

Prompt

OpenSearch PPL クエリを生成するには、OpenSearch PPL タブを選択し、クエリジェネレータープロンプトボックスを開いて自然言語プロンプトを入力します。OpenSearch PPL クエリを生成するために自然言語機能が使用するプロンプトの例を以下に示します。

```
Give me all unique exception messages
```

クエリ

以下は、ログ内の一意的例外メッセージを検索するために使用できる、そのプロンプトによって生成された PPL クエリです。

```
dedup @message  
| fields @message
```

例: 自然言語クエリを更新する

生成済みのプロンプトを編集し、[クエリを更新] を選択することで、クエリを更新できます。

プロンプトの更新

次の例は、先ほどのプロンプトを更新したものを示しています。最も遅い 10 個の Lambda 関数の呼び出しを検索するプロンプトの代わりに、このプロンプトは最も遅い 20 個の Lambda 関数の呼び出しを検索し、追加のログイベントのために別の列を含めています。

```
Show top 20 slowest requests instead and display requestId as a column
```

クエリの更新

以下は、CloudWatch Logs Insights クエリ言語を使用して更新されたクエリの例です。更新したプロンプトが更新したクエリの前にあるコメントにどのように表示されるかに注意してください。クエリの後に、元のクエリがどのように更新されたかを説明した文があります。

```
# Show top 20 slowest requests instead and display requestId as a column  
fields @timestamp, @message, @requestId, @duration  
| sort @duration desc  
| limit 20  
# This query modifies the original query by replacing the @message field with the  
  @requestId field and changing the limit from 10 to 20 to return the top 20 log events  
  by duration instead of the top 10.
```

サービス改善のためのデータ使用をオプトアウトする

AI モデルをトレーニングし、該当するクエリを生成するために提供した自然言語プロンプトデータは、サービスを提供して維持するためにのみ使用されます。このデータは、CloudWatch Logs

Insights の品質を高めるために使用される可能性があります。お客様の信頼、プライバシー、コンテンツのセキュリティが当社の最優先事項です。詳細については、「[AWS サービス規約](#)」と「[AWS responsible AI policy](#)」を参照してください。

自然言語クエリの開発や品質改善に自分のコンテンツが使用されないようにオプトアウトするには、AI サービスオプトアウトポリシーを作成します。クエリ生成機能などのすべての CloudWatch Logs AI 機能データ収集をオプトアウトするには、CloudWatch Logs のオプトアウトポリシーを作成する必要があります。詳細については、「AWS Organizations ユーザーガイド」の「[AI サービスオプトアウトポリシー](#)」を参照してください。

サポートされるログと検出されるフィールド

CloudWatch Logs Insights は、さまざまなタイプのログをサポートします。Amazon CloudWatch Logs の標準クラスロググループに送信されるすべてのログについて、CloudWatch Logs Insights は 5 つのシステムフィールドを自動的に生成します。

- @message は、生の未解析のログイベントを示します。これは、[InputLogevent](#) の message フィールドに相当します。
- @timestamp には、ログイベントの timestamp フィールドに含まれるイベントタイムスタンプが含まれます。これは、[InputLogevent](#) の timestamp フィールドに相当します。
- @ingestionTime は、ログイベントが CloudWatch Logs によって受信された時間を示します。
- @logStream は、ログイベントの追加先のログストリームの名前を示します。ログストリームは、生成時と同じプロセスでログをグループ化します。
- @log は、**account-id:log-group-name** の形式のロググループ識別子です。これは、複数のロググループにクエリを実行する場合に、特定のイベントが属しているロググループを識別するのに役立ちます。
- @entity には、[Explore 関連のテレメトリ](#)機能のエンティティに関連するフラット化された JSON が含まれています。

例えば、この JSON はエンティティを表すことができます。

```
{
  "Entity": {
    "KeyAttributes": {
      "Type": "Service",
      "Name": "PetClinic"
    },
```

```
"Attributes": {  
  "PlatformType": "AWS::EC2",  
  "EC2.InstanceId": "i-1234567890123"  
}
```

このエンティティの場合、抽出されたシステムフィールドは次のとおりです。

```
@entity.KeyAttributes.Type = Service  
@entity.KeyAttributes.Name = PetClinic  
@entity.Attributes.PlatformType = AWS::EC2  
@entity.Attributes.EC2.InstanceId = i-1234567890123
```

Note

フィールド検出は、標準ログクラスのロググループでのみサポートされます。ログクラスの詳細については、「[ログクラス](#)」を参照してください。

CloudWatch Logs Insights では、生成されるフィールドの先頭に @ 記号を挿入します。

CloudWatch Logs は、多くのログタイプで、ログ内のログフィールドも自動的に検出します。これらの自動検出フィールドを以下の表に示します。

CloudWatch Logs Insights が自動的に検出しないフィールドを持つ他のログタイプについては、parse コマンドを使用して抽出フィールドを抽出および作成してクエリで使用できます。詳細については、「[CloudWatch Logs Insights 言語のクエリ構文](#)」を参照してください。

検出されたログフィールドの名前が @ 文字で始まる場合は、この名前の先頭に @ が追加されて CloudWatch Logs Insights に表示されます。たとえば、ログフィールド名が @example.com である場合、このフィールド名は @@example.com と表示されます。

Note

@message、@timestamp、@log を除き、検出されたフィールドのフィールドインデックスを作成できます。フィールドインデックスの詳細については、「[フィールドインデックスを作成してクエリパフォーマンスを改善し、スキャン量を削減する](#)」を参照してください。

ログタイプ	検出されるログフィールド
Amazon VPC フローログ	@timestamp , @logStream , @message, accountId , endTime, interfaceId , logStatus , startTime , version, action, bytes, dstAddr, dstPort, packets, protocol, srcAddr, srcPort
Route 53 ログ	@timestamp , @logStream , @message, edgeLocation , ednsClientSubnet , hostZoneId , protocol, queryName , queryTimestamp , queryType , resolverIp , responseCode , version
Lambda ログ	@timestamp , @logStream , @message, @requestId , @duration, @billedDuration , @type, @maxMemoryUsed , @memorySize Lambda ログ行に X-Ray トレース ID が含まれている場合は、@xrayTraceId および @xraySegmentId フィールドも含まれます。 CloudWatch Logs Insights は、Lambda ログのログフィールドを自動的に検出します。ただし、各ログイベントに埋め込まれている最初の JSON フラグメントのみが検出されます。Lambda ログイベントに複数の JSON フラグメントが含まれている場合は、parse コマンドを使用してログフィールドを解析して抽出できます。詳細については、「JSON ログのフィールド」を参照してください。
CloudTrail ログ JSON 形式のログ	詳細については、「 JSON ログのフィールド 」を参照してください。
その他のログタイプ	@timestamp , @ingestionTime , @logStream , @message, @log.

JSON ログのフィールド

CloudWatch Logs Insights では、ドット表記を使用して JSON フィールドを表します。このセクションでは、ドット表記を使用して JSON フィールドにアクセスする方法を、JSON イベントとコードスニペットによる例で説明します。

例: JSON イベント


```
{
  "eventVersion": "1.0",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "EX_PRINCIPAL_ID",
    "arn": "arn: aws: iam: : 123456789012: user/Alice",
    "accessKeyId": "EXAMPLE_KEY_ID",
    "accountId": "123456789012",
    "userName": "Alice"
  },
  "eventTime": "2014-03-06T21: 22: 54Z",
  "eventSource": "ec2.amazonaws.com",
  "eventName": "StartInstances",
  "awsRegion": "us-east-2",
  "sourceIPAddress": "192.0.2.255",
  "userAgent": "ec2-api-tools1.6.12.2",
  "requestParameters": {
    "instancesSet": {
      "items": [
        {
          "instanceId": "i-abcde123"
        }
      ]
    }
  },
  "responseElements": {
    "instancesSet": {
      "items": [
        {
          "instanceId": "i-abcde123",
          "currentState": {
            "code": 0,
            "name": "pending"
          },
          "previousState": {
            "code": 80,
            "name": "stopped"
          }
        }
      ]
    }
  }
}
```

サンプルの JSON イベントには、`userIdentity` という名前のオブジェクトが含まれています。`userIdentity` には `type` という名前のフィールドが含まれます。ドット表記を使用して `type` の値を表すには、`userIdentity.type` を使用します。

サンプル JSON イベントには、ネストされたフィールド名と値のリストにフラット化された配列が含まれています。`requestParameters.instancesSet` の最初の項目である `instanceId` の値を表すには、`requestParameters.instancesSet.items.0.instanceId` を使用します。`instanceId` フィールドの前にある番号 `0` は、`items` フィールドの値の場所を指します。次の例には、JSON ログイベントでネストされた JSON フィールドにアクセスする方法を示すコードスニペットが含まれています。

例: クエリ

```
fields @timestamp, @message
| filter requestParameters.instancesSet.items.0.instanceId="i-abcde123"
| sort @timestamp desc
```

このコードスニペットは、ネストされた JSON フィールド `instanceId` の値にアクセスする `filter` コマンドと共にドット表記を使用するクエリを示します。このクエリは、`instanceId` の値が `"i-abcde123"` に等しいメッセージをフィルタリングし、指定した値を含むログイベントをすべて返します。

Note

CloudWatch Logs Insights は、1 つの JSON ログから最大 200 個のログイベントフィールドを抽出できます。抽出されない追加のフィールドについては、`parse` コマンドを使用して、メッセージフィールドの未処理の未解析ログイベントからこれらのフィールドを抽出できます。`parse` コマンドの詳細については、「Amazon CloudWatch ユーザーガイド」の「[クエリ構文](#)」を参照してください。

フィールドインデックスを作成してクエリパフォーマンスを改善し、スキャン量を削減する

ログイベントでフィールドのフィールドインデックスを作成して、同等性ベースの検索を効率的に行うことができます。その後、CloudWatch Logs Insights クエリでフィールドインデックスを使用すると、クエリはインデックス付きフィールドを含まないことがわかっているログイベントの処理のス

キップを試行します。これによってフィールドインデックスを使用するクエリのスキャン量が減少し、結果をより迅速に返すことができます。これにより、数千のロググループ全体で合計ログのペタバイトをすばやく検索し、関連するログをより迅速に見つけることができます。インデックス付けに適したフィールドは、多くの場合、クエリが必要なフィールドです。値のカーディナリティが高いフィールドは、フィールドインデックスの候補としても適しています。これらのフィールドインデックスを使用するクエリは、一致するログイベントをターゲット値に制限するため、より迅速に完了するからです。

例えば、requestId のフィールドインデックスを作成したとします。次に、requestId = **value** または を含むそのロググループの CloudWatch Logs Insights クエリ `requestId IN [value, value, ...]` は、そのインデックス付きフィールドとクエリされた値を含むことがわかっており、CloudWatch Logs がそのフィールドの値を過去に検出したログイベントのみを処理しようとしています。

フィールドインデックスを活用して、多数のロググループの効率的なクエリを作成することもできます。filter コマンドの代わりにクエリで filterIndex コマンドを使用すると、フィールドインデックスを持つログイベントで選択したロググループに対してクエリが実行されます。これらのクエリは、最大 5 つのロググループ名のプレフィックスを指定することで、最大 10,000 個のロググループをスキャンできます。これが CloudWatch のクロスアカウントオブザーバビリティのモニタリングアカウントの場合は、すべてのソースアカウントを選択するか個々のソースアカウントを指定してロググループを選択できます。

インデックス付きのフィールドでは、大文字と小文字が区別されます。例えば、RequestId のフィールドインデックスは、requestId を含むログイベントと一致しません。

フィールドインデックスは、JSON ログとサービスログの構造化ログ形式でのみサポートされます。

CloudWatch Logs は、標準ログクラスのすべてのロググループにデフォルトのフィールドインデックスを提供します。デフォルトのフィールドインデックスは、次のフィールドで自動的に使用できます。

- @logStream
- @aws.region
- @aws.account
- @source.log
- @data_source_name
- @data_source_type

- @data_format
- traceId
- severityText
- attributes.session.id

CloudWatch Logs は、特定のデータソース名とタイプの組み合わせのデフォルトのフィールドインデックスも提供します。デフォルトのフィールドインデックスは、次のデータソース名とタイプの組み合わせで自動的に使用できます。

データソース名とタイプ	デフォルトのフィールドインデックス
amazon_vpc.flow	action logStatus region flowDirection type
amazon_route53_resolver_query	query_type transport rcode
aws_waf.access	action httpRequest.country
aws_cloudtrail.data aws_cloudtrail.management	eventSource eventName awsRegion userAgent errorCode

データソース名とタイプ	デフォルトのフィールドインデックス
	eventType
	managementEvent
	readOnly
	eventCategory
	requestId

デフォルトのフィールドインデックスは、ポリシー内で定義した任意のカスタムフィールドインデックスに追加されます。デフォルトのフィールドインデックスは、[フィールドインデックスクォータ](#)にはカウントされません。

CloudWatch Logs は、インデックスポリシーの作成後に取り込まれたログイベントのみをインデックス化します。ポリシーの作成前に取り込まれたログイベントはインデックス化されません。フィールドインデックスを作成すると、一致する各ログイベントは、ログイベントの取り込み時間から 30 日間インデックスが付いたままになります。

Note

モニタリングアカウントにフィールドインデックスポリシーを作成する場合、そのポリシーはリンクされたソースアカウントのロググループには使用されません。フィールドインデックスポリシーは、それが作成されたアカウントにのみ適用されます。

このセクションの残りのトピックでは、フィールドインデックスを作成する方法について説明します。クエリでフィールドインデックスを参照する方法については「[filterIndex](#)」および「[フィルター](#)」を参照してください。

トピック

- [フィールドインデックスの構文とクォータ](#)
- [アカウントレベルのフィールドインデックスポリシーを作成する](#)
- [ロググループレベルのフィールドインデックスポリシーを作成する](#)
- [クエリ作成時のロググループの選択オプション](#)
- [フィールドインデックスポリシーを削除した場合の影響](#)

フィールドインデックスの構文とクォータ

フィールドインデックスを作成するには、フィールドインデックスポリシーを作成します。アカウント全体に適用されるアカウントレベルのインデックスポリシーを作成できます。また、単一のロググループのみに適用されるポリシーを作成することもできます。アカウント全体のインデックスポリシーでは、アカウント内のすべてのロググループに適用されるインデックスポリシーを設定できます。ロググループ名のプレフィックスによって選択された、アカウントのロググループのサブセットに適用されるアカウントレベルのインデックスポリシーを作成することもできます。同じアカウントに複数のアカウントレベルのポリシーがある場合、これらのポリシーのロググループ名のプレフィックスを重複させることはできません。同様に、特定のデータソース名とタイプの組み合わせに適用されるアカウントレベルのインデックスポリシーを作成できます。データソース名とタイプの組み合わせごとに作成できるアカウントポリシーは 1 つだけです。

ロググループレベルのフィールドインデックスポリシーは、アカウントレベルのフィールドインデックスポリシーを上書きします。これは、ロググループ全体に適用されます (選択基準のないアカウントレベルのポリシーや、ロググループ名のプレフィックスベースの選択基準を持つアカウントレベルのポリシーなど)。ログイベントレベルで一致するアカウントレベルのポリシー (特定のデータソース名とタイプの組み合わせなど) は、ロググループ全体に一致するポリシーに加えて適用されます。ロググループレベルのインデックスポリシーを作成する場合、そのロググループは、ロググループレベルで一致するアカウントレベルのポリシーを使用しません。

ログイベントとフィールドインデックスの名前の一致では、大文字と小文字が区別されます。例えば、RequestId のフィールドインデックスは、requestId を含むログイベントと一致しません。

最大 40 のアカウントレベルのインデックスポリシーを持つことができ、そのうち 20 はロググループ名のプレフィックス選択基準を使用し、20 はデータソースベースの選択基準を使用できます。ロググループ名のプレフィックスに対してフィルタリングされた複数のアカウントレベルのインデックスポリシーがある場合、同じまたは重複するロググループ名のプレフィックスを 2 つ使用することはできません。例えば、my-log で始まるロググループに対してフィルタリングされたポリシーが 1 つある場合、別のフィールドインデックスポリシーを my-logpprod または my-logging に対してフィルタリングすることはできません。同様に、データソース名とタイプの組み合わせにフィルタリングされた複数のアカウントレベルのインデックスポリシーがある場合、2 つのポリシーで同じデータソース名とタイプを使用することはできません。たとえば、1 つのポリシーをデータソース名 amazon_vpc とデータソースタイプにフィルタリングした場合 flow、この組み合わせで別のポリシーを作成することはできません。

名前プレフィックスがなく、すべてのロググループに適用されるアカウントレベルのインデックスポリシーがある場合、ロググループ名プレフィックスフィルターを持つ他のアカウントレベルのイン

デックスポリシーを作成することはできません。データソース名とタイプフィルターを使用するアカウントレベルのインデックスポリシーを作成できます。

各インデックスポリシーには、次のクォータと制限があります。

- 最大 20 個のフィールドをポリシーに含めることができます。
- 各フィールド名には最大 100 文字を含めることができます。
- @ で始まるロググループにカスタムフィールドのインデックスを作成するには、フィールド名の先頭に @ を追加してフィールドを指定する必要があります。例えば、ログイベントに @userId という名前のフィールドが含まれている場合は、@@userId を指定してこのフィールドのインデックスを作成する必要があります。

データソース名とタイプベースの選択基準を持つアカウントレベルのインデックスポリシーでは、追加の制限が適用されます。すべてのフィールドはプリミティブデータ型でなければならない、ネストされたプリミティブは 構造体でのみサポートされます。

生成されたフィールドと予約されたフィールド

CloudWatch Logs Insights は、各ログイベントにシステムフィールドを自動的に生成します。これらの生成されたフィールドには @ のプレフィックスが付きます。生成されたフィールドの詳細については「[サポートされるログと検出されるフィールド](#)」を参照してください。

これらの生成されたフィールドのうち、フィールドインデックスとして使用するために以下がサポートされています。

- @logStream
- @ingestionTime
- @requestId
- @type
- @initDuration
- @duration
- @billedDuration
- @memorySize
- @maxMemoryUsed
- @xrayTraceId

- @xraySegmentId

これらの生成されたフィールドをインデックス付けするには、@で始まるカスタムフィールドに対して行う必要があるため、それらを指定するときに@を追加する必要はありません。例えば、@logStreamのフィールドインデックスを作成するには、@logStreamをフィールドインデックスとして指定します。

CloudWatch Logs は、標準ログクラスのすべてのロググループにデフォルトのフィールドインデックスを提供します。デフォルトのフィールドインデックスは、次のフィールドで自動的に使用できます。

- @logStream
- @aws.region
- @aws.account
- @source.log
- @data_source_name
- @data_source_type
- @data_format
- traceId
- severityText
- attributes.session.id

CloudWatch Logs は、特定のデータソース名とタイプの組み合わせのデフォルトのフィールドインデックスも提供します。デフォルトのフィールドインデックスは、次のデータソース名とタイプの組み合わせで自動的に使用できます。

データソース名とタイプ	デフォルトのフィールドインデックス
amazon_vpc.flow	action
	logStatus
	region
	flowDirection

データソース名とタイプ	デフォルトのフィールドインデックス
	type
amazon_route53_resolver_query	query_type transport rcode
aws_waf.access	action httpRequest.country
aws_cloudtrail.data aws_cloudtrail.management	eventSource eventName awsRegion userAgent errorCode eventType managementEvent readOnly eventCategory requestId

デフォルトのフィールドインデックスは、ポリシー内で定義した任意のカスタムフィールドインデックスに追加されます。デフォルトのフィールドインデックスは、[フィールドインデックスクォータ](#)にはカウントされません。

JSON ログの子フィールドと配列フィールド

JSON ログのネストされた子フィールドまたは配列フィールドであるフィールドにインデックスを作成できます。

例えば、このログ内の `userIdentity` フィールド内に `accessKeyId` 子フィールドのインデックスを作成できます。

```
{
  "eventVersion": "1.0",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "EXAMPLE_PRINCIPAL_ID",
    "arn": "arn: aws: iam: : 123456789012: user/Alice",
    "accessKeyId": "11112222",
    "accountId": "123456789012",
    "userName": "Alice"
  },
  "eventTime": "2014-03-06T21: 22: 54Z",
  "eventSource": "ec2.amazonaws.com",
  "eventName": "StartInstances",
  "awsRegion": "us-east-2",
  "sourceIPAddress": "192.0.2.255",
  "userAgent": "ec2-api-tools1.6.12.2",
  "requestParameters": {
    "instancesSet": {
      "items": [{
        "instanceId": "i-abcde123",
        "currentState": {
          "code": 0,
          "name": "pending"
        },
        "previousState": {
          "code": 80,
          "name": "stopped"
        }
      }]
    }
  }
}
```

このフィールドを作成するには、フィールドインデックスを作成する時とクエリでそれを指定する時の両方で、ドット表記 (`userIdentity.accessKeyId`) を使用して参照します。クエリは次のようになります。

```
fields @timestamp, @message
| filterIndex userIdentity.accessKeyId = "11112222"
```

前の例のイベントの場合、instanceId フィールドは requestParameters.instancesSet.items 内の配列にあります。フィールドインデックスの作成時とクエリ時の両方でこのフィールドを表すには、requestParameters.instancesSet.items.0.instanceId として参照します。0 は配列内のそのフィールドの場所を参照します。

すると、このフィールドのクエリは次のようになります。

```
fields @timestamp, @message
| filterIndex requestParameters.instancesSet.items.0.instanceId="i-abcde123"
```

アカウントレベルのフィールドインデックスポリシーを作成する

このセクションのステップを使用して、アカウント内のすべてのロググループ、または同じ文字列で始まるロググループ名を持つ複数のロググループに適用されるフィールドインデックスポリシーを作成します。

アカウントレベルのフィールドインデックスポリシーを作成するには

1. CloudWatch コンソールの <https://console.aws.amazon.com/cloudwatch/> を開いてください。
2. ナビゲーションペインで、[設定] を選択し、[ログ] を選択します。
3. [アカウントレベルのインデックスポリシー] セクションで、[管理] を選択します。
4. [インデックスポリシーを作成] を選択します。
5. ポリシー名に、新しいポリシーの名前を入力します。
6. ポリシーの範囲を選択するには、次のいずれかを実行します。
 - [すべての標準ロググループ] を選択して、インデックスポリシーをアカウントのすべての標準クラスロググループに適用します。
 - プレフィックス一致でロググループを選択し、同じ文字列で始まる名前を持つロググループのサブセットにポリシーを適用します。次に、これらのロググループのプレフィックスを [プレフィックス名を入力] に入力します。

プレフィックスを入力したら、[プレフィックスが一致したロググループをプレビュー] を選択して、プレフィックスが期待したロググループと一致することを確認できます。

データソース別のログデータを選択して、ポリシーを特定のデータソース名とタイプの組み合わせに適用します。その後、ドロップダウンメニューからデータソースとデータ型を選択できます。

データソース名とタイプを選択したら、フィールドの取得を選択して、フィールドインデックスとファセットの設定セクションに、使用可能なフィールド、含まれるロググループ、デフォルトフィールドインデックスとカスタムフィールドインデックスなどの関連情報を入力できます。

7. [カスタムインデックスフィールド設定] で [フィールドパスを追加] を選択し、インデックスを作成する最初のフィールドを入力します。

次に、フィールド名の値として使用する文字列を入力するか、ドロップダウンメニューからフィールドを選択します。これは、ログイベントに表示されるものと完全に一致する必要があります。例えば、ログイベントに `requestId` が含まれている場合は、ここに `requestId` と入力する必要があります。`RequestId`、`requestID`、`request Id` は一致しません。

@ 文字で始まるカスタムログフィールドにインデックスを作成する場合は、インデックス文字列を入力するときに追加の @ 文字を含める必要があります。例えば、カスタムログフィールド `@emailname` がある場合は、[フィールドパスの追加] ボックスに `@emailname` と入力します。

CloudWatch Logs が自動的に生成する `@ingestionTime` および `@logStream` フィールドのインデックスを作成することもできます。その場合は、指定するときに @ を追加する必要はありません。

8. (オプション) フィールドパスを指定するだけでなく、ファセットとして設定を選択して、ファセットとしてフィールドを作成できます。
9. 前のステップを繰り返して、最大 20 個のフィールドインデックスを追加します。
10. 終了したら、[Create] (作成) を選択します。

ロググループレベルのフィールドインデックスポリシーを作成する

このセクションのステップを使用して、単一のロググループに適用されるフィールドインデックスポリシーを作成します。

ロググループレベルのフィールドインデックスポリシーを作成するには

1. CloudWatch コンソールの <https://console.aws.amazon.com/cloudwatch/> を開いてください。
2. 左側のナビゲーションペインで、[Logs] (ログ)、[Log groups] (ロググループ) の順に選択します。
3. ロググループの名前を選択します。

4. [フィールドインデックス] タブを選択します。
5. [このロググループのフィールドインデックスを管理] を選択します。
6. [ロググループレベルのフィールドインデックスを管理] で [フィールドパスの追加] を選択し、インデックスを作成する最初のフィールドを入力します。

次に、フィールド名の値として使用する文字列を入力します。これは、ログイベントに表示されるものと完全に一致する必要があります。例えば、ログイベントに `requestId` が含まれている場合は、ここに `requestId` と入力する必要があります。RequestId、requestID、request Id は一致しません。

@ 文字で始まるカスタムログフィールドにインデックスを作成する場合は、インデックス文字列を入力するときに追加の @ 文字を含める必要があります。例えば、カスタムログフィールド `@emailname` がある場合は、[フィールドパスの追加] ボックスに `@emailname` と入力します。

CloudWatch Logs が自動的に生成する `@ingestionTime` および `@logStream` フィールドのインデックスを作成することもできます。その場合は、指定するときに @ を追加する必要はありません。

7. (オプション) フィールドパスを指定するだけでなく、ファセットとして設定を選択して、ファセットとしてフィールドを作成できます。
8. 前のステップを繰り返して、最大 20 個のフィールドインデックスを追加します。
9. 完了したら、[Save] を選択します。

クエリ作成時のロググループの選択オプション

このセクションでは、クエリに含めるロググループを選択するさまざまな方法について説明します。

コンソールでクエリのロググループを選択するには

1. CloudWatch コンソールの <https://console.aws.amazon.com/cloudwatch/> を開いてください。
2. ナビゲーションペインで、[ログ]、[Logs Insights] を選択します。
3. このクエリに使用するクエリ言語を選択します。Logs Insights QL、OpenSearch PPL、OpenSearch SQL のいずれかを選択できます。
4. クエリのロググループを選択する方法は 3 つあります。
 - [ロググループ名] ボックスを使用します。これはデフォルトの選択方法です。この方法では、最大 50 個のロググループ名を入力できます。これが CloudWatch のクロスアカウント

オブザーバビリティのモニタリングアカウントの場合は、モニタリングアカウントだけでなくソースアカウントのロググループも選択できます。1つのクエリで複数のアカウントのログを一度にクエリできます。

- [ロググループの条件] セクションを使用します。このセクションでは、ロググループ名のプレフィックスに基づいてロググループを選択できます。1つのクエリに最大5つのプレフィックスを含めることができます。これらのプレフィックスが名前に含まれているロググループが選択されます。または、[すべてのロググループ] オプションで、アカウントからすべてのロググループを選択します。
- これが CloudWatch クロスアカウントオブザーバビリティのモニタリングアカウントである場合は、アカウントドロップダウンメニューで[すべてのアカウント] を選択して、リンクされたすべてのアカウントからロググループを選択できます。または、このクエリに含めるアカウントを個別に選択できます。

選択肢が 10,000 を超えるロググループと一致する場合、選択肢を絞り込むよう求めるエラーが表示されます。

5. クエリのデフォルトのログクラスは 標準 です。[ログクラス] を使用して低頻度アクセスに変更できます。

の使用 AWS CLI

コマンドラインからクエリを開始するときにこれらのタイプの選択を行うには、クエリで `source` コマンドを使用できます。詳細な説明と例については、[SOURCE](#) を参照してください。

フィールドインデックスポリシーを削除した場合の影響

しばらく有効になっているフィールドインデックスポリシーを削除すると、次のようになります。

- ポリシーが削除されてから最大 30 日間、クエリはインデックスが作成されたログイベントから恩恵を受けることができます。
- ロググループレベルのインデックスポリシーを削除し、そのロググループに適用されるアカウントレベルのポリシーがすでに設定されている場合、アカウントレベルのポリシーは最終的にそのロググループに適用されます。

ファセットを使用してログをグループ化および探索する

ファセットは、クエリを実行せずにデータをインタラクティブにフィルタリングおよびグループ化できるため、ログの分析に役立ちます。ファセットは、ロググループ間のフィルタリング、集約、分

析を可能にするログ内のフィールド (ServiceName や など StatusCode) です。CloudWatch Logs Insights コンソールでファセットフィールドのリストと、選択した時間範囲に基づく各ファセット値のログイベントの数を表示できます。さまざまなファセットと値を選択すると、ファセット値とカウントがリアルタイムで更新されるため、ログをインタラクティブに探索できます。

各ファセットには、選択した時間範囲とクエリスコープに基づいてログ内のフィールドから自動的に抽出され、30 日間保持される使用可能な値とカウントが表示されます。表示されるファセット数は概算です。データソース名やデータソースタイプなどのデフォルトのファセットを使用してログを調べたり、ログ内の任意のフィールドにカスタムファセットを作成したりできます。データソース名はログを生成する AWS のサービスまたはアプリケーション (Route 53、Amazon VPC、CloudTrail など) であり、データソースタイプはそのサービスによって生成される特定のタイプのログです。デフォルトのファセットは CloudWatch によって作成され `@aws.region`、`@data_source_name`、`@data_source_type`、および `@data_format` が含まれます。詳細については、「[ログ管理](#)」を参照してください。ファセットは、アカウントに取り込まれたログでのみ使用できます。クロスアカウントオブザーバビリティを設定している場合、モニタリングアカウントはソースアカウントのログに基づいてファセットを表示できません。

追加のファセットを作成するには、トラブルシューティングに関連するログのフィールドを選択し、インデックスポリシーを使用してファセットを設定します。カスタムファセットの場合、低カードinalityフィールド (Status や ApplicationName など、1 日あたり 100 個未満の一意の値を持つフィールド) にファセットを作成することをお勧めします。1 日あたり 100 個を超える一意の値を持つファセットは高濃度に分類され、これらのファセットの値は表示されません。1 つ以上のファセットを選択し、をクリックしてログ全体でクエリを実行します。

CloudWatch Logs Insights でファセットの使用を開始するには:

1. CloudWatch コンソールの <https://console.aws.amazon.com/cloudwatch/> を開いてください。
2. ナビゲーションペインで、[ログ]、[Logs Insights] を選択します。
3. (オプション) 時間範囲セレクタを使用して、分析する期間を選択します。選択した時間範囲について、使用可能なファセットと値がパネルに表示されます。
4. ファセットを選択してデータを探索し、ファセット間の値分布をリアルタイムで更新します。

100 個を超える一意の値を持つファセットは表示されません。特定の値をクエリするには、代わりにクエリでフィルターを使用します。

ファセットベースのクエリを実行するには

1. ファセット間で 1 つ以上の値を選択します。

2. イベント数は、選択したファセットと値に基づいて更新されます。
3. ファセット値を選択すると、クエリスコープが更新されて選択が反映されます。
4. ファセット値を選択したら、実行をクリックしてクエリを実行します。
5. ファセットごとにサポートされる一意の値の最大数は 100 です。たとえば、ファセットの値が 100 を超える場合、すべてのカウントは「-」と表示され、値が不明であることを示します。

ファセットベースのクエリを保存するには

1. 1 つ以上のファセット値を使用してクエリを作成します。
2. 残りのステップは、Logs Insights クエリの保存と同じです。[CloudWatch Logs Insights クエリの保存](#)を参照してください。
3. 保存済みクエリは、保存済みクエリセクションで使用できます。保存されたクエリを取得すると、クエリに使用されるファセットと値が自動的に含まれるため、ログの分析が容易になります。

アカウントレベルのファセットを作成するには

1. ファセットを作成するには、まずフィールドをインデックスとして作成し、ファセットとして設定する必要があります。ナビゲーションペインで、設定、ログ、アカウントレベルのインデックスポリシーを選択します。または、ファセットパネルでファセットの管理を選択することもできます。
2. 新しいインデックスポリシーの作成を選択します。インデックスポリシーの作成の詳細については、「」を参照してください[アカウントレベルのフィールドインデックスポリシーを作成する](#)。
3. ファセットを作成するには、インデックスポリシーの作成ページで、選択したフィールドのファセットとして設定を確認します。

APIsを使用したファセット管理

ファセット管理は、フィールドインデックスポリシーを使用して実行できます。詳細については、[field index APIs](#)を参照してください。

フィールドインデックス APIs

いいえ。	名前	説明
1	PutIndexPolicy	特定のロググループのフィールドインデックスポリシーを作成または更新します
2	PutAccountPolicy	アカウント内のすべてのロググループまたはロググループのサブセットに適用されるアカウントレベルのデータ保護ポリシー、サブスクリプションフィルターポリシー、フィールドインデックスポリシー、トランスフォーマーポリシー、またはメトリクス抽出ポリシーを作成します。
3	DeleteIndexPolicy	1 つのロググループに適用されたロググループレベルのフィールドインデックスポリシーを削除します。
4	DeleteAccountPolicy	CloudWatch Logs アカウントポリシーを削除します。

CloudWatch Logs Insights での周囲のログの表示

CloudWatch Logs Insights クエリによって返された特定のログレコードの前後に発生したログイベントを表示できます。ログを囲むと、エラー、警告、またはその他の重要なイベントに関するコンテキストを理解するのに役立ちます。

周囲のログの仕組み

CloudWatch Logs Insights クエリを実行すると、結果の各ログレコードにサラウンドログオプションが含まれます。このオプションを選択すると、CloudWatch Logs Insights は、選択したレコードの直前と直後に発生した同じログストリームからの追加のログイベントを表示します。これらのイベントは、元のクエリフィルターと一致しなくても表示されます。

表示する周囲のログ行の数を設定できます。選択したレコードの上下の 5、10、20、50、または 100 ログ行から選択します。

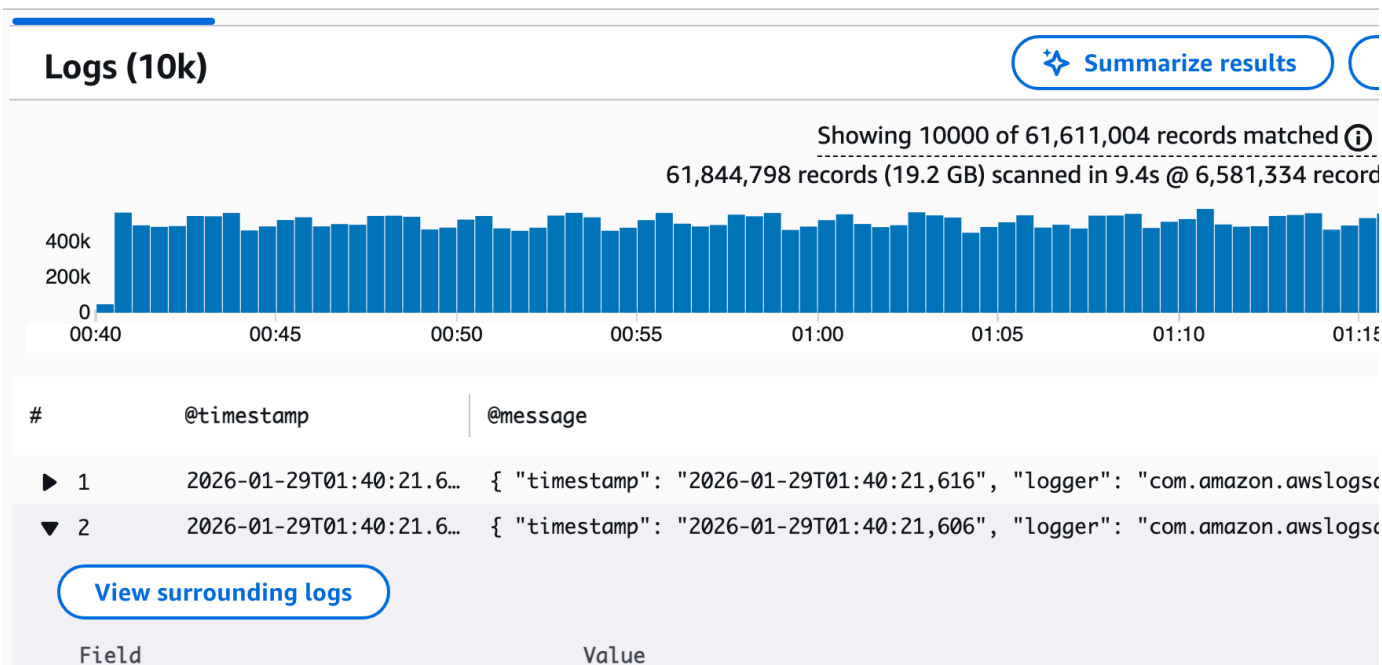
ログを囲むことは、以下のシナリオで役立ちます。

- 失敗の原因となったイベントを表示してエラーをデバッグする
- 分散システムでの警告またはタイムアウトの調査
- アプリケーション全体のイベントのシーケンスを理解する
- 大量のログ環境での問題の分析

周囲のログを表示する

周囲のログを表示するには

1. CloudWatch コンソールの <https://console.aws.amazon.com/cloudwatch/> を開いてください。
2. ナビゲーションペインで、[ログ]、[Logs Insights] を選択します。
3. クエリを実行します。
4. クエリ結果で、調査するログレコードを見つけます。
5. サラウンドログを選択します。



6. イベント数ドロップダウンで、選択したレコードの上下に表示するログイベントの数を選択します。5、10、20、50、または 100 を選択できます。

Surrounding Logs [ApplicationLogs-ip-10-0-42-156.ec2.internalApplicationLogs](#) ✕

Q Search log messages... Log event count +/- 5 ▲

Timestamp	Message
▶ 2026-01-29T01:40:21.246Z	{ "timestamp": "2026-01-29T01:40:21,246", "logger": "com.amazon.awslogsanalysisdataplane.util..." }
▶ 2026-01-29T01:40:21.246Z	{ "timestamp": "2026-01-29T01:40:21,246", "logger": "com.amazon.awslogsanalysisdataplane.util..." }
▶ 2026-01-29T01:40:21.577Z	{ "timestamp": "2026-01-29T01:40:21,577", "logger": "com.amazon.awslogsanalysisdataplane.util..." }
▶ 2026-01-29T01:40:21.587Z	{ "timestamp": "2026-01-29T01:40:21,587", "logger": "com.amazon.awslogsanalysisdataplane.util..." }
▶ 2026-01-29T01:40:21.597Z	{ "timestamp": "2026-01-29T01:40:21,597", "logger": "com.amazon.awslogsanalysisdataplane.util..." }
▶ » 2026-01-29T01:40:21.607Z	{ "timestamp": "2026-01-29T01:40:21,606", "logger": "com.amazon.awslogsanalysisdataplane.util..." }
▶ 2026-01-29T01:40:21.616Z	{ "timestamp": "2026-01-29T01:40:21,616", "logger": "com.amazon.awslogsanalysisdataplane.util..." }
▶ 2026-01-29T01:40:21.625Z	{ "timestamp": "2026-01-29T01:40:21,625", "logger": "com.amazon.awslogsanalysisdataplane.util..." }
▶ 2026-01-29T01:40:21.635Z	{ "timestamp": "2026-01-29T01:40:21,635", "logger": "com.amazon.awslogsanalysisdataplane.util..." }
▶ 2026-01-29T01:40:21.644Z	{ "timestamp": "2026-01-29T01:40:21,644", "logger": "com.amazon.awslogsanalysisdataplane.util..." }
▶ 2026-01-29T01:40:21.653Z	{ "timestamp": "2026-01-29T01:40:21,653", "logger": "com.amazon.awslogsanalysisdataplane.util..." }

There are newer logs to load. [Load more](#)



周囲のログ内で検索する

周囲のログパネルを開くと、特定のキーワードを検索して関連するコンテキストを見つけることができます。

周囲のログ内で検索するには

1. 周囲のログパネルで、検索ボックスに検索語を入力します。
2. 強調表示された一致するログ行を確認します。
3. ナビゲーションコントロールを使用して、一致間を移動します。

Surrounding Logs [ApplicationLogs-ip-10-0-42-156.ec2.internalApplicationLogs](#)

Log event count +/- 5

Timestamp	Message
▶ 2026-01-29T01:40:21.246Z	{ "timestamp": "2026-01-29T01:40:21,246", "logger": "com.amazon.awslogsanalysisdataplane.patt...
▼ 2026-01-29T01:40:21.246Z	{ "timestamp": "2026-01-29T01:40:21,246", "logger": "com.amazon.awslogsanalysisdataplane.patt...

```
{
  "timestamp": "2026-01-29T01:40:21,246",
  "logger": "com.amazon.awslogsanalysisdataplane.patternlibrary.PatternLibraryStoreImpl",
  "level": "INFO",
  "threadID": "33750",
  "threadName": "sdk-async-response-4-18426",
  "message": "Calling putPattern with anomalyDetectorId '80a6b9d7-daa0-44ac-bef7-3ef8662e4347'"
}
```

パターン分析

CloudWatch Logs Insights は、機械学習アルゴリズムを使用して、ログをクエリするときにパターンを検索します。パターンは、ログフィールド間で繰り返される共有テキスト構造です。クエリの結果を表示するときは、[パターン]タブを選択して、結果のサンプルに基づいて CloudWatch Logs が検出したパターンを表示できます。または、クエリに `pattern` コマンドを追加して、一致するログイベントのセット全体のパターンを分析することもできます。

多数のログイベントをいくつかのパターンに圧縮することができるため、パターンは大きなログセットを分析する際に役立ちます。

次の 3 つのログイベントの例を検討してください。

```
2023-01-01 19:00:01 [INFO] Calling DynamoDB to store for resource id 12342342k124-12345
2023-01-01 19:00:02 [INFO] Calling DynamoDB to store for resource id 324892398123-12345
2023-01-01 19:00:03 [INFO] Calling DynamoDB to store for resource id 3ff231242342-12345
```

前のサンプルでは、3 つのログイベントはすべて 1 つのパターンに従っています。

```
<Time-1> [INFO] Calling DynamoDB to store for resource id <ID-2>
```

パターン内のフィールドはトークンと呼ばれます。リクエスト ID やタイムスタンプなど、パターン内の異なるフィールドは動的トークンと呼ばれます。個々の動的トークンは `<string-number>` で表されます。`###` は、トークンが表すデータのタイプの説明です。`##` は、他の動的トークンと比較して、このトークンがパターン内のどこに表示されるかを示します。

動的トークンの一般的な例には、エラーコード、タイムスタンプ、リクエスト ID があります。トークン値は、動的トークンの特定の値を表します。例えば、動的トークンが HTTP エラーコードを表す場合、トークン値は 501 になります。

パターン検出は、CloudWatch Logs 異常ディテクターおよび比較機能でも使用されます。詳細については、「[ログ異常検出](#)」および「[\(diff\) を以前の時間範囲と比較する](#)」を参照してください。

パターン分析の開始方法

パターン検出は、CloudWatch Logs Insights クエリで自動的に実行されます。pattern コマンドを含まないクエリは、結果にログイベントとパターンの両方を取得します。

pattern コマンドをクエリに含めると、一致するログイベントのセット全体に対してパターン分析が実行されます。これにより、より正確なパターン結果が得られますが、pattern コマンドの使用時に未加工のログイベントは返されません。クエリに pattern が含まれていない場合、パターン結果は、返された最初の 1000 個のログイベント、またはクエリで使用した制限値のいずれかに基づいて返されます。pattern をクエリに含めると、[パターン] タブに表示される結果は、クエリに一致するすべてのログイベントから取得されます。

CloudWatch Logs Insights でパターン分析を開始するには

1. CloudWatch コンソールの <https://console.aws.amazon.com/cloudwatch/> を開いてください。
2. ナビゲーションペインで、[ログ]、[Logs Insights] を選択します。

[Logs Insights] (ログのインサイト) ページでは、クエリエディタにデフォルトクエリが表示されます。デフォルトでは、最新の 20 件のログイベントが返されます。

3. クエリボックスの `| limit 20` 行を削除して、クエリを次のようにします。

```
fields @timestamp, @message, @logStream, @log
| sort @timestamp desc
```

4. [ロググループを選択] ドロップダウンから、クエリを実行するロググループを 1 つ以上選択します。
5. (オプション) 時間間隔セレクタを使用して、クエリを実行する期間を選択します。

5 分および 30 分間隔、1 時間、3 時間、12 時間間隔、またはカスタム時間枠を選択できます。

6. [クエリの実行] を選択してクエリを開始します。

クエリの実行が完了すると、[ログ] タブにクエリによって返されたログイベントのテーブルが表示されます。表の上にあるのは、クエリに一致するレコードの数に関するメッセージです。これは、一致する 71,101 レコードのうち 10,000 レコードを表示するというメッセージと似ています。

7. [パターン] タブを選択します。
8. テーブルにクエリで検出されたパターンが表示されるようになりました。クエリには `pattern` コマンドが含まれていなかったため、このタブには、[ログ] タブのテーブルに表示された 10,000 個のログイベントで検出されたパターンのみが表示されます。

パターンごとに、次の情報が表示されます。

- [パターン] では、各動的トークンが `<string-number>` として表示されます。### は、トークンが表すデータのタイプの説明です。## は、他の動的トークンと比較して、このトークンがパターン内のどこに表示されるかを示します。
- [イベント数] は、クエリされたログイベントにパターンが表示された回数です。[イベント数] 列の見出しを選択して、パターンを頻度でソートします。
- [イベント比率] は、このパターンを含むクエリされたログイベントの割合です。
- [重要度タイプ] は、次のいずれかになります。
 - パターンに `Error` という単語が含まれている場合は、`ERROR` です。
 - パターンに `Warn` という単語が含まれているが、`Error` が含まれていない場合は `WARN` です。
 - パターンに `Warn` または `Error` が含まれていない場合は `INFO` です。

[Severity info] 列の見出しを選択して、重要度別にパターンをソートします。

9. 次に、クエリを変更します。クエリの `| sort @timestamp desc` 行を `| pattern @message` に置き換えて、次のような完全なクエリにします。

```
fields @timestamp, @message, @logStream, @log
| pattern @message
```

10. [Run query] (クエリの実行) を選択します。

クエリが終了すると、[ログ] タブに結果は表示されません。ただし、[パターン] タブには、クエリされたログイベントの合計数に応じて、リストされたパターンの数が多く表示される可能性があります。

- クエリに `pattern` を含めるかどうかにかかわらず、クエリが返すパターンをさらに検査することができます。これを行うには、いずれかのパターンについて [検査] 列のアイコンを選択します。

[パターン検査] ペインが表示され、以下が表示されます。

- [パターン]。パターン内のトークンを選択して、そのトークンの値を分析します。
- クエリされた時間範囲におけるパターンの出現回数を示すヒストグラム。これにより、パターンの発生急増など、興味深い傾向を特定できます。
- [ログサンプル] タブには、選択したパターンに一致するログイベントがいくつか表示されます。
- [トークンの値] タブには、選択している場合は、選択した動的トークンの値が表示されます。

Note

トークンごとに最大 10 個のトークン値がキャプチャされます。トークン数は正確ではない可能性があります。CloudWatch Logs は確率カウンターを使用して、絶対値ではなくトークン数を生成します。

- [関連パターン] タブには、検査するパターンとほぼ同じ時間に頻繁に発生した他のパターンが表示されます。例えば、ERROR メッセージのパターンに通常、追加の詳細を含む INFO とマークされた別のログイベントが伴っていた場合、そのパターンがここに表示されます。

パターンコマンドの詳細

このセクションでは、`pattern` コマンドとその用途について詳しく説明します。

- 前のチュートリアルでは、`sort` コマンドの後に `pattern` コマンドが含まれている場合、クエリは有効ではないため、`pattern` コマンドを追加したときに `sort` コマンドを削除しました。`sort` の前に `pattern` があるのは有効です。

`pattern` 構文の詳細については、「[pattern](#)」を参照してください。

- クエリで `pattern` を使用する場合、`@message` は `pattern` コマンドで選択されたフィールドの 1 つである必要があります。
- `pattern` コマンドの前に `filter` コマンドを含めると、フィルタリングされたログイベントのセットのみをパターン分析の入力として使用できます。

- `parse` コマンドから派生したフィールドなど、特定のフィールドのパターン結果を表示するには、`pattern @fieldname` を使用します。
- `stats` コマンドを使用したクエリなど、ログ以外の出力を持つクエリは、パターン結果を返しません。

CloudWatch Logs Insights クエリの保存と再実行

作成したクエリは、後で再度実行できるように保存できます。保存したクエリは、フォルダ構造が保持されるため、整理された状態を保つことができます。アカウントごとに、リージョンあたり最大 1000 件保存できます。

クエリは、ユーザー固有のレベルではなく、リージョン固有のレベルに保存されます。クエリを作成して保存すると、同じリージョンで CloudWatch Logs にアクセスできる他のユーザーは、リージョンで保存されたすべてのクエリとそのフォルダ構造を表示できます。

クエリを保存するには、アクセス許可 `logs:PutQueryDefinition` を持つロールにログインする必要があります。保存されたクエリのリストを表示するには、アクセス許可 `logs:DescribeQueryDefinitions` を持つロールにログインする必要があります。

Note

パラメータを使用してクエリを作成および保存できます。名前付きプレースホルダーを持つ再利用可能なテンプレートです。同じクエリの数回のバリエーションを異なる値で保存する代わりに、1 つのテンプレートを作成し、実行時に異なるパラメータ値を指定します。この機能は現在、Logs Insights クエリ言語を使用するクエリでのみサポートされています。詳細については、[「パラメータを使用した保存済みクエリの使用」](#)を参照してください。

Console

クエリを保存するには

1. CloudWatch コンソールの <https://console.aws.amazon.com/cloudwatch/> を開いてください。
2. ナビゲーションペインで、[Logs] (ログ)、[Logs Insights] (ログのインサイト) の順に選択します。
3. クエリエディタで、クエリを作成します。

4. [保存] を選択します。
5. クエリの名前を入力します。
6. (オプション) クエリを保存するフォルダを選択します。[新規作成] を選択して、フォルダを作成します。新しいフォルダを作成した場合、フォルダ名にスラッシュ (/) 文字を使用してフォルダ構造を定義できます。たとえば、新しいフォルダに **folder-level-1/folder-level-2** という名前を付けると、**folder-level-1** という最上位フォルダが作成され、そのフォルダ内に **folder-level-2** という別のフォルダが作成されます。クエリは **folder-level-2** に保存されます。
7. (オプション) クエリのロググループまたはクエリテキストを変更します。
8. (オプション) クエリでパラメータを使用するには、以下の追加ステップに従います。
 - a. クエリにパラメータを追加します。{{parameter}} 構文 (パラメータ名の前後に二重括弧) を使用して、静的値をプレースホルダーに置き換えます。

例: 静的な値を持つ元のクエリ:

```
fields @timestamp, @message
| filter level = "Error"
| filter applicationName = "OrderService"
```

パラメータでクエリを更新しました。

```
fields @timestamp, @message
| filter level = {{logLevel}}
| filter applicationName = {{applicationName}}
```

- b. クエリで使用されるパラメータを定義します。プレースホルダーパラメータごとに、以下を指定します。
 - 名前: プレースホルダー名と完全に一致する必要があります (例: logLevel、applicationName)。
 - デフォルト値 (オプション): パラメータ値が指定されていない場合に使用する値。
 - 説明 (オプション): パラメータの目的について説明します。
- c. パラメータを含むクエリを実行するには、\$プレフィックス付きのクエリ名を使用し、キーと値のペアとしてパラメータ名を渡します。詳細については、「保存済みクエリを実行するには」を参照してください。

9. [保存] を選択します。

AWS CLI

クエリを保存するには、`aws logs put-query-definition` を使用します。

```
aws logs put-query-definition \
  --name "ErrorsByLevel" \
  --query-string "fields @timestamp, @message | filter level = \"ERROR\"" \
  --log-group-names "/aws/lambda/my-function" \
  --region us-east-1
```

(オプション) パラメータを含むクエリを保存するには、`--parameters` オプションを追加し、クエリ文字列に `{{parameterName}}` プレースホルダーを使用します。

```
aws logs put-query-definition \
  --name "ErrorsByLevel" \
  --query-string "fields @timestamp, @message | filter level = {{logLevel}} | filter applicationName = {{applicationName}}" \
  --parameters '[{"name": "logLevel", "defaultValue": "ERROR", "description": "Log level to filter"}, {"name": "applicationName", "defaultValue": "OrderService", "description": "Application name to filter"}]' \
  --log-group-names "/aws/lambda/my-function" \
  --region us-east-1
```

クエリをフォルダに保存するには、クエリ名の前にフォルダパスを付けます。

```
aws logs put-query-definition \
  --name "my-folder/ErrorsByLevel" \
  --query-string "fields @timestamp, @message | filter level = {{logLevel}}" \
  --parameters '[{"name": "logLevel", "defaultValue": "ERROR", "description": "Log level to filter"}]' \
  --log-group-names "/aws/lambda/my-function" \
  --region us-east-1
```

API

クエリを保存するには、[PutQueryDefinition](#) を呼び出します。

```
{
  "name": "ErrorsByLevel",
  "queryString": "fields @timestamp, @message | filter level = \"ERROR\"",
```

```
"logGroupNames": ["/aws/lambda/my-function"]
}
```

(オプション) パラメータを含むクエリを保存するには、parametersフィールドを含め、クエリ文字列に{{parameterName}}プレースホルダーを使用します。

```
{
  "name": "ErrorsByLevel",
  "queryString": "fields @timestamp, @message | filter level = {{logLevel}} | filter applicationName = {{applicationName}}",
  "logGroupNames": ["/aws/lambda/my-function"],
  "parameters": [
    {
      "name": "logLevel",
      "defaultValue": "ERROR",
      "description": "Log level to filter"
    },
    {
      "name": "applicationName",
      "defaultValue": "OrderService",
      "description": "Application name to filter"
    }
  ]
}
```

Tip

PutQueryDefinition で保存したクエリー用のフォルダを作成することができます。保存したクエリ用のフォルダを作成するには、スラッシュ (/) を使用して、目的のクエリ名の前に目的のフォルダ名を付加します: **<folder-name>/<query-name>**。このリソースの更新の詳細については、「[PutQueryDefinition](#)」を参照してください。

Console

保存されたクエリを実行するには

1. CloudWatch コンソールの <https://console.aws.amazon.com/cloudwatch/> を開いてください。
2. ナビゲーションペインで、[Logs] (ログ)、[Logs Insights] (ログのインサイト) の順に選択します。

3. 右側の [クエリ] を選択します。
4. 保存済みクエリリストからクエリを選択します。クエリテキストがクエリエディタに表示されます。
5. (オプション) パラメータでクエリを使用するには:
 - a. 保存済みクエリのサイドパネルで、クエリ名の横にある + アイコンを選択します。
 - b. パラメータを含むクエリがクエリエディタに表示されます。たとえば、 の横にある + アイコンを選択すると `ErrorsByLevel`、クエリエディタに次のように入力されます。
`$ErrorsByLevel(level=, applicationName=)`
 - c. パラメータの値 (レベル、 `applicationName`) を指定し、クエリを実行します。例:
`$ErrorsByLevel(level= "ERROR", applicationName= "OrderService")`
6. [Run] (実行) を選択します。

AWS CLI

パラメータを使用して保存されたクエリを実行するには

`$QueryName()` 構文 `start-query` で を使用します。

```
aws logs start-query \  
  --log-group-names "/aws/lambda/my-function" \  
  --start-time 1707566400 --end-time 1707570000 \  
  --query-string '$ErrorsByLevel(level= "ERROR", applicationName= "OrderService")' \  
  --region us-east-1
```

API

パラメータを使用して保存されたクエリを実行するには

`queryString` フィールドの `$QueryName()` 構文を使用して [StartQuery](#) を呼び出します。

```
{  
  "logGroupNames": ["/aws/lambda/my-function"],  
  "startTime": 1707566400,  
  "endTime": 1707570000,  
  "queryString": "$ErrorsByLevel(level=\"ERROR\", applicationName= \"OrderService  
  \")"  
}
```

保存したクエリの新しいバージョンを保存するには

1. CloudWatch コンソールの <https://console.aws.amazon.com/cloudwatch/> を開いてください。
2. ナビゲーションペインで、[Logs] (ログ)、[Logs Insights] (ログのインサイト) の順に選択します。
3. 右側の [クエリ] を選択します。
4. [保存されたクエリ] リストからクエリを選択します。クエリエディタに表示されます。
5. クエリを修正します。作業を確認するために実行する必要がある場合は、[クエリの実行] を選択します。
6. 新しいバージョンを保存する準備ができたなら、[アクション]、[名前を付けて保存] の順に選択します。
7. クエリの名前を入力します。
8. (オプション) クエリを保存するフォルダを選択します。[新規作成] を選択して、フォルダを作成します。新しいフォルダを作成した場合、フォルダ名にスラッシュ (/) 文字を使用してフォルダ構造を定義できます。たとえば、新しいフォルダに **folder-level-1/folder-level-2** という名前を付けると、**folder-level-1** という最上位フォルダが作成され、そのフォルダ内に **folder-level-2** という別のフォルダが作成されます。クエリは **folder-level-2** に保存されます。
9. (オプション) クエリのロググループまたはクエリテキストを変更します。
10. [Save] を選択します。

クエリを削除するには、logs:DeleteQueryDefinition アクセス許可を持つロールにログインする必要があります。

保存したクエリを編集または削除するには

1. CloudWatch コンソールの <https://console.aws.amazon.com/cloudwatch/> を開いてください。
2. ナビゲーションペインで、[Logs] (ログ)、[Logs Insights] (ログのインサイト) の順に選択します。
3. 右側の [クエリ] を選択します。
4. [保存されたクエリ] リストからクエリを選択します。クエリエディタに表示されます。
5. [アクション]、[編集]、または [アクション]、[削除] を選択します。

パラメータを使用した保存済みクエリの使用

パラメータを含む保存されたクエリは、名前付きプレースホルダーを持つ再利用可能なクエリテンプレートです。ほぼ同一のクエリの複数のコピーを維持する代わりに、テンプレートを保存し、クエリの実行時に異なるパラメータ値を指定できます。パラメータは CloudWatch Logs Insights クエリ言語でのみサポートされます。

仕組み

クエリを保存する場合、プレースホルダーはクエリ実行時に指定できる値を識別します。プレースホルダーは `{{parameterName}}` 構文を使用します。以下は、`logLevel` と `applicationName` の 2 つのパラメータ `ErrorsByLevel` を持つ という名前の保存済みクエリの例です。

```
fields @timestamp, @message
| filter level = {{logLevel}}
| filter applicationName = {{applicationName}}
```

保存済みクエリを実行するには、というプレフィックスが付いたクエリ名を使用してクエリを呼び出し、パラメータ値を渡します。CloudWatch Logs Insights クエリエンジンは、各プレースホルダーを置き換えます。パラメータにデフォルト値が含まれている場合、他の値が指定されていない場合はそれらの値が使用されます。

```
# Run query by using query name and passing parameter values explicitly
$ErrorsByLevel(logLevel = "WARN", applicationName = "OrderService")

# Run query without specifying parameter values - default values are used in this case.
$ErrorsByLevel()
```

スペースまたは特殊文字を含む保存されたクエリ名は、バックティックで囲む必要があります。

```
$`Errors By Level`(logLevel = "WARN")
```

パラメータを含む保存済みクエリのサンプル

パラメータとして結果制限を追加する

クエリ名: パラメータ `ErrorsByLevel` 付き `logLevel` (デフォルト: "ERROR")、`applicationName` (デフォルト: "OrderService")、および `maxResults` (デフォルト: 50)

```
fields @timestamp, @message, @logStream
| filter level = {{logLevel}}
| filter applicationName = {{applicationName}}
| sort @timestamp desc
| limit {{maxResults}}
```

```
# Run the query using the query name and passing parameter values
$ErrorsByLevel(logLevel = "WARN", applicationName = "OrderService", maxResults = 100)
```

パラメータで複数の保存済みクエリを使用する

次の例では、`ErrorsByLevel`として定義されている 2 番目の保存済みクエリ `sort @timestamp desc | limit {{count}}` (パラメータ、`count`デフォルト) を使用します。20。CloudWatch Logs Insights クエリエンジンは、各クエリを実行する前に展開します。

```
# Using multiple queries with parameters in sequence
$ErrorsByLevel(logLevel = "WARN", applicationName = "OrderService")
| $RecentN(count = 10)

# Each of the queries is expanded, resulting in the following query when it is run.
fields @timestamp, @message
| filter level = "WARN"
| filter applicationName = "OrderService"
| sort @timestamp desc
| limit 10
```

クォータとエラー処理

Note

保存された各クエリには、最大 20 個のパラメータを含めることができます。

展開されたクエリ文字列は 10,000 文字を超えることはできません。パラメータ名は文字またはアンダースコアで始まる必要があります。保存されたクエリは、別の保存されたクエリを参照できません (ネストされた呼び出しはサポートされていません)。

一般的なエラー

エラー	原因
パラメータは CWLI クエリ言語でのみサポートされています	パラメータは CloudWatch Logs Insights クエリ言語でのみサポートされます。
queryString にない必須パラメータ	のパラメータ名には、クエリ文字列{{placeholder}} に一致するもの--parameters がありません。
パラメータ数が最大 20 を超えています	現在、保存されたクエリは 20 個のパラメータのみをサポートしています。
重複するパラメータ名	クエリ定義には、に重複するパラメータがありますparameters 。

Note

パラメータを使用して保存されたクエリを作成または更新するには、アクセスlogs:PutQueryDefinition許可が必要です。これを実行するには、logs:StartQueryと が必要ですlogs:DescribeQueryDefinitions。

クエリをダッシュボードに追加する、またはクエリ結果をエクスポートする

クエリの実行後に、クエリを CloudWatch ダッシュボードに追加したり、クエリ結果をクリップボードにコピーしたりできます。

ダッシュボードに追加したクエリは、ダッシュボードをロードおよび更新するたびに再実行されます。これらのクエリは、100 件の同時 CloudWatch Logs Insights クエリの制限にカウントされます。

クエリ結果をダッシュボードに追加するには

1. CloudWatch コンソールの <https://console.aws.amazon.com/cloudwatch/> を開いてください。

2. ナビゲーションペインで、[Logs] (ログ)、[Logs Insights] (ログのインサイト) の順に選択します。
3. 1 つ以上のロググループを選択し、クエリを実行します。
4. [ダッシュボードに追加] を選択します。
5. ダッシュボードを選択するか、[新規作成] を選択して、クエリ結果用のダッシュボードを作成します。
6. クエリ結果に使用するウィジェットの種類を選択します。
7. ウィジェットの名前を入力します。
8. [ダッシュボードに追加] を選択します。

クエリ結果をクリップボードにコピーするか、クエリ結果をダウンロードするには

1. CloudWatch コンソールの <https://console.aws.amazon.com/cloudwatch/> を開いてください。
2. ナビゲーションペインで、[Logs] (ログ)、[Logs Insights] (ログのインサイト) の順に選択します。
3. 1 つ以上のロググループを選択し、クエリを実行します。
4. [結果のエクスポート] を選択し、必要なオプションを選択します。

実行中のクエリまたはクエリ履歴を表示する

現在進行中のクエリや最近のクエリ履歴を表示できます。

現在実行中のクエリには、ダッシュボードに追加したクエリも含まれます。ダッシュボードに追加されたクエリを含め、アカウントあたり 100 件の同時 CloudWatch Logs Insights クエリに制限されます。さらに、OpenSearch Service PPL または OpenSearch Service SQL のいずれかに対して 15 OpenSearch の同時クエリを実行できます。

最近のクエリ履歴を表示するには

1. CloudWatch コンソールの <https://console.aws.amazon.com/cloudwatch/> を開いてください。
2. ナビゲーションペインで、[Logs] (ログ)、[Logs Insights] (ログのインサイト) の順に選択します。
3. CloudWatch Logs コンソールに新しいデザインを使用する場合は、[History (履歴)] を選択します。古いデザインを使用している場合は、[アクション]、[このアカウントのクエリ履歴を表示] の順に選択します。

最近のクエリが一覧表示されます。クエリを選択して [実行] を選択すると、それらのいずれかを再度実行できます。

CloudWatch Logs では、[Status (ステータス)] の下に、現在実行中のクエリが [In progress (進行中)] と表示されます。

でクエリ結果を暗号化する AWS Key Management Service

デフォルトでは、CloudWatch Logs は、デフォルトの CloudWatch Logs サーバー側の暗号化方法を使用して CloudWatch Logs Insights クエリの保存済み結果を暗号化します。代わりに、AWS KMS キーを使用してこれらの結果を暗号化することもできます。AWS KMS キーを暗号化結果に関連付けると、CloudWatch Logs はそのキーを使用して、アカウント内のすべてのクエリの保存された結果を暗号化します。

後でクエリ結果からキーの関連付けを解除すると、CloudWatch Logs は、その後のクエリに対してデフォルトの暗号化方法を適用します。しかし、キーが関連付けられていたときに実行されたクエリは、そのキーで暗号化されたままになります。CloudWatch Logs は引き続きキーを参照できるため、KMS キーの関連付けが解除された後も CloudWatch Logs はそれらの結果を返すことができます。ただし、キーを後で無効にすると、CloudWatch Logs はそのキーで暗号化されたクエリ結果を読み取ることができなくなります。

Important

CloudWatch Logs は、対称 KMS キーのみをサポートします。クエリ結果の暗号化に非対称キーを使用しないでください。詳細については、「[対称キーと非対称キーの使用](#)」を参照してください。

制限

- 以下の手順を実行するには、`kms:CreateKey`、`kms:GetKeyPolicy`、および `kms:PutKeyPolicy` アクセス許可が必要です。
- キーとクエリ結果を関連付けた後、または関連付けを解除した後、オペレーションが有効になるまで最大 5 分かかることがあります。
- 関連付けられたキーへの CloudWatch Logs のアクセスを取り消した場合、または関連付けられた KMS キーを削除した場合、CloudWatch Logs 内の暗号化されたデータを取得できなくなります。

- CloudWatch コンソールを使用してキーを関連付けることはできません。AWS CLI または CloudWatch Logs API を使用する必要があります。

ステップ 1: を作成する AWS KMS key

KMS キーを作成するには、次の [create-key](#) コマンドを使用します。

```
aws kms create-key
```

出力には、キーのキー ID と Amazon リソースネーム (ARN) が含まれます。出力例を次に示します。

```
{
  "KeyMetadata": {
    "Origin": "AWS_KMS",
    "KeyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
    "Description": "",
    "KeyManager": "CUSTOMER",
    "Enabled": true,
    "CustomerMasterKeySpec": "SYMMETRIC_DEFAULT",
    "KeyUsage": "ENCRYPT_DECRYPT",
    "KeyState": "Enabled",
    "CreationDate": 1478910250.94,
    "Arn": "arn:aws:kms:us-west-2:123456789012:key/6f815f63-e628-448c-8251-e40cb0d29f59",
    "AWSAccountId": "123456789012",
    "EncryptionAlgorithms": [
      "SYMMETRIC_DEFAULT"
    ]
  }
}
```

ステップ 2: KMS キーでアクセス許可を設定する

デフォルトでは、すべての KMS キーはプライベートです。リソースの所有者のみがその CMK を使用してデータを暗号化および復号できます。ただし、リソース所有者は、他のユーザーとリソースにキーへのアクセス許可を付与することができます。このステップでは、CloudWatch Logs サービスプリンシパルに、キーを使用するアクセス許可を付与します。このサービスプリンシパルは、キーが保存されているリージョンと同じ AWS リージョンに存在する必要があります。

ベストプラクティスとして、キーの使用は、指定した AWS アカウントのみに制限することをお勧めします。

まず、[get-key-policy](#) コマンドを使用して、KMS キーのデフォルトポリシーを `policy.json` として保存します。

```
aws kms get-key-policy --key-id key-id --policy-name default --output text > ./policy.json
```

テキストエディタで `policy.json` ファイルを開き、以下のいずれかのステートメントから太字のセクションを追加します。既存のステートメントと新しいステートメントをカンマで区切ります。これらのステートメントでは、Condition セクションを使用して AWS KMS キーのセキュリティを強化します。詳細については、「[AWS KMS キーと暗号化コンテキスト](#)」を参照してください。

この例の Condition セクションでは、AWS KMS キーの使用を、指定されたアカウントの CloudWatch Logs Insights クエリ結果に制限します。

JSON

```
{
  "Version": "2012-10-17",
  "Id": "key-default-1",
  "Statement": [
    {
      "Sid": "Enable IAM User Permissions",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:root"
      },
      "Action": "kms:*",
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "logs.region.amazonaws.com"
      },
      "Action": [
        "kms:Encrypt*",
        "kms:Decrypt*",
        "kms:ReEncrypt*",
```

```

        "kms:GenerateDataKey*",
        "kms:Describe*"
    ],
    "Resource": "*",
    "Condition": {
        "ArnEquals": {
            "aws:SourceArn": "arn:aws:logs:us-east-1:111122223333:query-
result:*"
        },
        "StringEquals": {
            "aws:SourceAccount": "111122223333"
        }
    }
}
]
}

```

最後に、次の [put-key-policy](#) コマンドを使用して更新されたポリシーを追加します。

```
aws kms put-key-policy --key-id key-id --policy-name default --policy file://
policy.json
```

ステップ 3: KMS キーをクエリ結果に関連付ける

KMS キーをアカウントのクエリ結果に関連付けるには

次のように、[disassociate-kms-key](#) コマンドを使用します。

```
aws logs associate-kms-key --resource-identifier "arn:aws:logs:region:account-id:query-
result:*" --kms-key-id "key-arn"
```

ステップ 4: アカウントのクエリ結果からキーの関連付けを解除する

クエリ結果に関連付けられた KMS キーの関連付けを解除するには、次の [disassociate-kms-key](#) コマンドを使用します。

```
aws logs disassociate-kms-key --resource-identifier "arn:aws:logs:region:account-
id:query-result:*"
```

CloudWatch Logs Insights クエリ結果から自然言語の概要を生成する

アプリケーションの動作を理解するにはログデータを分析することが重要ですが、大量のログエントリを解釈するには時間がかかる場合があります。CloudWatch Logs Insights は、複雑なクエリ結果を明確で簡潔な要約に変換する自然言語の要約機能を提供できるようになりました。この機能は、問題をすばやく特定し、ログデータから実用的なインサイトを得るのに役立ちます。

仕組み

CloudWatch Logs Insights は、Amazon Bedrock を使用してクエリ結果から人間が読み取れる概要を生成できます。この機能は、すべての CloudWatch Logs Insights クエリ言語をサポートし、ログデータから明確で実用的なインサイトを提供します。

リージョンの可用性とデータ処理

Important

この機能を使用すると、クエリ結果が別の AWS リージョンで処理される場合があります。例えば、米国東部 (バージニア北部) でクエリを実行すると、要約が米国西部 (オレゴン) で発生する可能性があります。

次の表は、クエリ結果機能が使用可能な AWS リージョン さまざまな地域で処理可能な処理の一覧です。

サポートされている CloudWatch Logs の地域	処理される可能性があるリージョン
[米国 (US)]	US East (N. Virginia) Region
	米国東部 (オハイオ) リージョン
	米国西部 (オレゴン) リージョン
欧州	欧州 (フランクフルト) リージョン
	欧州 (アイルランド) リージョン
	欧州 (パリ) リージョン

サポートされている CloudWatch Logs の地域	処理される可能性があるリージョン
アジアパシフィック	欧州 (ストックホルム) リージョン
	欧州 (ロンドン) リージョン
	US East (N. Virginia) Region
	米国東部 (オハイオ) リージョン
	米国西部 (オレゴン) リージョン
南米	US East (N. Virginia) Region
	米国東部 (オハイオ) リージョン
	米国西部 (オレゴン) リージョン

開始方法

自然言語の概要を生成するには

- CloudWatch Logs Insights のクエリを実行します。
- クエリが完了したら、[結果の要約] を選択します。

アクセス許可

以下のいずれかが必要です。

- CloudWatchLogsFullAccess アクセス許可
- CloudWatchLogsReadOnlyAccess アクセス許可
- cloudwatch:GenerateQueryResultsSummary、logs:GetQueryResults、logs:DescribeQueryResults アクションを含むカスタム IAM ポリシー

データプライバシー

クエリ結果は安全に処理され、CloudWatch Logs Insights または Amazon Bedrock のトレーニングや改善には使用されません。フィードバックボタンを使用してクエリ結果の概要に関するフィード

バックを提供する場合、フィードバックは CloudWatch Logs Insights で提供される機能に対する満足度を示します。

スケジュールされたクエリによるログ分析の自動化

スケジュールされたクエリを使用すると、CloudWatch Logs Insights クエリの実行を定期的に自動化できます。手動でクエリを実行してログデータを分析する代わりに、スケジュールされたクエリを自動的に実行し、Amazon S3 バケットや Amazon EventBridge イベントバスなどの送信先に結果を配信するように設定できます。この自動化は、定期的なレポートの生成、傾向のモニタリング、ログ分析結果に基づくダウンストリームプロセスのトリガーに最適です。

スケジュールされたクエリは、CloudWatch Logs Insights で使用できる 3 つのクエリ言語すべてをサポートします。

- [Logs Insights クエリ言語 \(Logs Insights QL\)](#)
- [OpenSearch Service Piped Processing Language \(PPL\)](#)
- [OpenSearch Service 構造化クエリ言語 \(SQL\)](#)

内容

- [スケジュールされたクエリ of の概念を理解する](#)
- [スケジュール式リファレンス](#)
- [ベストプラクティス](#)
- [スケジュールされたクエリ of の開始方法](#)
- [スケジュールされたクエリ of の S3 送信先 of の設定](#)
- [スケジュールされたクエリ of のトラブルシューティング](#)

スケジュールされたクエリ of の概念を理解する

スケジュールされたクエリを作成する前に、クエリの実行方法と結果 of の配信先に影響するこれらの主要な概念を理解してください。

IAM ロール of の分離

スケジュールされたクエリには、クエリを実行するためのロールと、Amazon S3 または Amazon EventBridge イベントバスに結果を配信するためのロール of の 2 つ of の異なる IAM ロールが必要です。この分離が存在する理由を理解することで、アクセス許可を正しく設定し、セキュリティと運用上 of の利点を活用できます。

2 ロールアーキテクチャは、データアクセスとデータ配信の責任を分けます。クエリ実行ロールはログデータにアクセスしてクエリを実行し、送信先配信ロールは選択した送信先に結果を書き込みます。この分離は最小特権の原則に従います。各ロールには、特定の関数に必要なアクセス許可のみがあります。

クエリ実行ロール

CloudWatch Logs がユーザーに代わって CloudWatch Logs Insights クエリを実行できるようにします。このロールには、ロググループにアクセスしてクエリを実行するためのアクセス許可が必要ですが、送信先リソースへのアクセスは必要ありません。必要なアクセス許可:

- logs:StartQuery
- logs:StopQuery
- logs:GetQueryResults
- logs:DescribeLogGroups
- logs:Unmask データのマスクを解除する必要がある場合

KMS で暗号化されたロググループの場合: ロググループの暗号化に使用される KMS キーのアクセスkms:Decryptkms:DescribeKey許可。これらのアクセス許可も追加する必要があります。

信頼関係の要件: クエリ実行ロールには、CloudWatch Logs サービス (logs.amazonaws.com) がロールを引き受けることを許可する信頼ポリシーが含まれている必要があります。この信頼関係がないと、スケジュールされたクエリはアクセス許可エラーで失敗します。

クエリ実行ロールの信頼ポリシーの例:

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

クエリ実行ロールのアクセス許可ポリシーの例:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:StartQuery",
        "logs:StopQuery",
        "logs:GetQueryResults",
        "logs:DescribeLogGroups"
      ],
      "Resource": "*"
    }
  ]
}
```

送信先配信ロール

CloudWatch Logs が選択した宛先にクエリ結果を配信できるようにします。このロールには、最小特権の原則に従って、特定の送信先サービスのアクセス許可のみが必要です。必要なアクセス許可は、送信先タイプによって異なります。

信頼関係の要件: 送信先配信ロールには、CloudWatch Logs サービス (logs.amazonaws.com) がロールを引き受けることを許可する信頼ポリシーも含める必要があります。

S3 送信先配信ロールのアクセス許可ポリシーの例:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObject"
      ],
      "Resource": "arn:aws:s3:::your-scheduled-query-results-bucket/*"
    }
  ]
}
```

この分離は、オペレーションに実用的な利点をもたらします。セキュリティの観点から、結果の配信先を変更する必要がある場合は、クエリ実行のアクセス許可を変更せずに送信先配信ロールのみを変更します。コンプライアンスと監査のために、機密ログデータにアクセスするロールと外部システムに書き込むロールを明確に追跡できます。これにより、ログ分析インフラストラクチャがセキュリティのベストプラクティスに従っていることを簡単に実証できます。

クロスリージョンおよびクロスアカウントの使用

スケジュールされたクエリは特定のリージョンで作成され、そのリージョンで実行されます。ただし、ロググループをクエリし、リージョンとアカウント間で結果を配信できます。1 つ以上の AWS アカウントをモニタリングアカウントとして設定し、複数のソースアカウントにリンクする必要があります。モニタリングアカウントは、ソース AWS アカウントから生成されたオブザーバビリティデータを表示して操作できる中央アカウントです。ソースアカウントは、そのアカウントに存在するリソースのオブザーバビリティデータを生成する個々の AWS アカウントです。ソースアカウントは、オブザーバビリティデータをモニタリングアカウントと共有します。そのため、すべてのリンクされたアカウントのロググループを使用して、モニタリングアカウントからスケジュールされたクエリを設定できます。

クロスリージョンロググループのクエリ

スケジュールされたクエリは、任意のリージョンのロググループにアクセスできます。完全な ARN 形式を使用してロググループを指定します。arn:aws:logs:region:account-id:log-group:log-group-name クエリ実行ロールには、すべてのターゲットリージョンのロググループの logs:StartQuery と logs:GetQueryResults アクセス許可が必要です。

Important

ロググループをクエリする場合、またはリージョン間で結果を提供する場合、ログデータはリージョンの境界を越えます。以下の点を考慮してください。

- データレジデンシー要件 - クロスリージョンデータ転送が組織のデータガバナンスポリシーと規制要件に準拠していることを確認します。
- データ転送コスト - クロスリージョンデータ転送には追加料金が発生します
- ネットワークレイテンシー - 離れたリージョンのロググループにアクセスするクエリでは、レイテンシーが高くなる可能性があります

最適なパフォーマンスとコスト効率を得るには、プライマリロググループと同じリージョンにスケジュールされたクエリを作成します。

代替アプローチ: [CloudWatch Logs の一元管理](#)を使用して、複数のアカウントとリージョンのログデータを一元的なモニタリングアカウントにレプリケートします。これにより、すべての一元化されたログにアクセスするスケジュールされたクエリを 1 つのリージョンで作成できるため、リージョン間のクエリを回避し、IAM アクセス許可の管理を簡素化できます。

スケジュール式とタイムゾーン処理

定義したスケジュールによって、クエリの実行日時と実行頻度が決まります。適切なスケジュール式を選択すると、結果を受け取るタイミングとクエリするデータの量に影響します。式タイプを理解すると、シンプルさと精度のどちらかを選択できます。

Cron 式はタイミングを正確に制御するため、正確な時間、曜日、または曜日を指定できます。特定の営業時間にクエリを実行したり、運用スケジュールに合わせる必要がある場合は、cron 式を使用します。コンソールでは、簡単なカレンダーオプションを使用してクエリをスケジュールすることもできます。

cron 式

特定の時間にクエリを実行します。形式: cron(minute hour day-of-month month day-of-week year)。例:

- cron(0 9 * * ? *) - 毎日午前 9 時 UTC
- cron(0 18 ? * MON-FRI *) - 平日の午後 6 時 UTC
- cron(0 0 1 * ? *) - 毎月 1 日午前 0 時 UTC
- cron(0 12 ? * SUN *) - 毎週日曜日の正午 UTC
- cron(30 8 1 1 ? *) - 1 月 1 日午前 8:30 UTC

スケジュールされたクエリはすべて、ローカルのタイムゾーンや AWS リソースの場所に関係なく、UTC で実行されます。これは、営業時間や時間的制約のある分析のクエリをスケジュールするときに特に重要です。たとえば、米国東部標準時で事業を運営していて、東部標準時午前 9 時に日次レポートが必要な場合は、UTC オフセット (夏時間中は 14:00 UTC、それ以外の場合は 13:00 UTC) を考慮する必要があります。クエリが意図した時間に実行されるように、UTC を念頭に置いてスケジュール式を計画します。

クエリ言語の選択

スケジュールされたクエリは 3 つの異なるクエリ言語をサポートしており、クエリの作成方法とチームがクエリを簡単に維持できるかどうかの両方に影響します。適切な言語は、分析要件とチームの既存のスキルによって異なります。

主にログデータをフィルタリングおよび集約する場合、CloudWatch Logs Insights クエリ言語は最も単純な構文を提供します。複数のステップでデータを再形成または強化する必要がある複雑なデータ変換の場合、PPL のパイプラインアプローチにより、ロジックに従うのが容易になります。データベースオペレーションに似た結合や複雑な集計を実行する必要がある場合、SQL はデータベース経験豊富なチームがすばやく採用できる使い慣れた構文を提供します。

CloudWatch Logs Insights クエリ言語 (CWLI)

直感的な構文を使用したログ分析専用です。以下に最適です。

- テキストベースのログ分析とフィルタリング
- 時系列の集計と統計
- ログ分析を初めて使用するチーム

OpenSearch Service Piped Processing Language (PPL)

強力なデータ変換機能を備えたパイプラインベースのクエリ言語。以下に最適です。

- 複雑なデータ変換とエンリッチメント
- マルチステップデータ処理ワークフロー
- パイプラインベースの処理に精通しているチーム

OpenSearch Service 構造化クエリ言語 (SQL)

使い慣れたデータベーススタイルのクエリ用の標準 SQL 構文。以下に最適です。

- 複雑な結合と集計
- ビジネスインテリジェンスとレポート
- SQL の経験が豊富なチーム

送信先の選択とユースケース

クエリ結果を送信する場所によって、クエリ結果で何ができるかが決まります。この選択により、長期的な分析の構築、自動応答のトリガー、またはその両方を行うかどうかにかかわらず、ダウンスト

リームワークフロー全体が形成されます。各送信先タイプの長所を理解することは、ユースケースに適したアーキテクチャを設計するのに役立ちます。

Amazon S3 の送信先は、ストレージとバッチ処理用に最適化されています。クエリ結果を数か月または数年間保持したり、経時的な傾向を分析したり、分析プラットフォームにデータをフィードしたりする必要がある場合、Amazon S3 は無制限の保持で費用対効果の高いストレージを提供します。EventBridge の送信先は、リアルタイムオートメーション用に最適化されています。クエリ結果がアラートの送信、ワークフローの開始、システムの更新などの即時アクションをトリガーする場合、EventBridge は、アプリケーションが即座に応答できるイベントとして結果を提供します。デフォルトでは、すべてのクエリ完了イベントはイベントとしてデフォルトのイベントバスに自動的に送信され、ダウンストリーム処理システム、Lambda 関数、またはその他のイベント駆動型アーキテクチャとの統合が可能になります。結果は、クエリが正常に実行された場合にのみ送信先に発行されます。

Amazon S3 の送信先

クエリ結果を JSON ファイルとして保存し、長期保存とバッチ処理を行います。最適:

- 履歴分析とデータアーカイブ
- データレイクと分析プラットフォームとの統合
- コンプライアンスと監査の要件
- 大きな結果セットのコスト効率の高いストレージ

EventBridge の送信先

クエリ結果をリアルタイム処理と自動化のイベントとして送信します。イベントで送信された queryId を使用してクエリ結果を取得できるのは、結果を 30 日間保存する場合のみです。最適:

- クエリ結果への自動応答のトリガー
- サーバーレスワークフローおよび Lambda 関数との統合
- リアルタイムアラートおよび通知システム
- イベント駆動型アーキテクチャとマイクロサービス

クエリ結果の形式と構造

Amazon S3 送信先の場合 - クエリ結果は GetQueryResults API レスポンスと同じ構造で JSON 形式で配信されます。Amazon EventBridge がスケジュールされたクエリ結果の形式を理解すると、ダウンストリーム処理および統合ワークフローを設計するのに役立ちます。

クエリ結果は、次の構造で JSON 形式で配信されます。

```
{
  "version": "0",
  "id": "be72061b-eca2-e068-a7e1-83e01d6fe807",
  "detail-type": "Scheduled Query Completed",
  "source": "aws.logs",
  "account": "123456789012",
  "time": "2025-11-18T11:31:48Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:logs:us-east-1:123456789012:scheduled-query:477b4380-b098-474e-9c5e-e10a8cc2e6e7"
  ],
  "detail": {
    "queryId": "2038fd57-ab4f-4018-bb2f-61d363f4a004",
    "queryString": "fields @timestamp, @message, @logStream, @log\n| sort\n@timestamp desc\n| limit 10000",
    "logGroupIdentifiers": [
      "/aws/lambda/my-function"
    ],
    "status": "Complete",
    "startTime": 1763465460,
    "statistics": {
      "recordsMatched": 0,
      "recordsScanned": 0,
      "estimatedRecordsSkipped": 0,
      "bytesScanned": 0,
      "estimatedBytesSkipped": 0,
      "logGroupsScanned": 1
    }
  }
}
```

主要要素は次のとおりです。

- **statistics** - 一致したレコード、スキャンされたレコード、処理されたバイト数、スキップされた推定データを含むパフォーマンスメトリクスをクエリする
- **startTime** - クエリ実行が開始されたとき (Unix タイムスタンプ)
- **queryString** - 実行された実際のクエリ
- **queryId** - 結果を取得できるクエリのクエリ ID

- `logGroupIdentifiers` - クエリされたロググループのリスト
- `status` - クエリ実行ステータス (完了、失敗など)

スケジュール式リファレンス

これらのリファレンステーブルを使用して、スケジュールされたクエリのスケジュール式を作成します。すべての時刻は協定世界時 (UTC) です。

Cron 式の構文

形式: `cron(minute hour day-of-month month day-of-week year)`

ユースケース	Cron 式	説明	いつ を使用 する
日次スケジュール	<code>cron(0 9 * * ? *)</code>	毎日午前 9:00 UTC	日次レポート
	<code>cron(0 */6 * * ? *)</code>	6 時間ごと (00:00、06:00、12:00、18:00 UTC)	頻繁なモニタリング
	<code>cron(30 2 * * ? *)</code>	毎日午前 2:30 UTC	オフピーク分析
営業時間	<code>cron(0 9-17 ? * MON-FRI *)</code>	月曜日から金曜日の UTC の午前 9 時から午後 5 時まで 1 時間ごと	ビジネスモニタリング
	<code>cron(0 18 ? * MON-FRI *)</code>	平日の午後 6:00 UTC	営業日の終了
	<code>cron(0 8,12,17 ? * MON-FRI *)</code>	平日 UTC の午前 8 時、正午、午後 5 時	主要なビジネス時間
週次スケジュール	<code>cron(0 12 ? * SUN *)</code>	毎週日曜日の正午 UTC	週次概要

ユースケース	Cron 式	説明	いつ を使用 する
	<code>cron(0 9 ? * MON *)</code>	毎週月曜日の午前 9:00 UTC	週開始レ ポート
	<code>cron(0 23 ? * FRI *)</code>	毎週金曜日の午後 11:00 UTC	週終了 クリーン アップ
月別スケジュール	<code>cron(0 0 1 * ? *)</code>	毎月 1 日午前 0 時 UTC	月次レ ポート
	<code>cron(0 9 L * ? *)</code>	UTC の毎月の最終日午前 9:00	月末処理
	<code>cron(0 10 1 1,4,7,10 ? *)</code>	各四半期の初日の午前 10:00 UTC	四半期分 析
高頻度	<code>cron(* /15 * * * ? *)</code>	15 分ごと	リアルタ イムモニ タリング
	<code>cron(0,30 * * * ? *)</code>	30 分ごと (:00 および :30)	頻繁な チェック
	<code>cron(0 */2 * * ? *)</code>	2 時間毎	定期的な 間隔
特殊なケース	<code>cron(30 8 1 1 ? *)</code>	1 月 1 日午前 8:30 UTC	年次レ ポート
	<code>cron(0 6 * * SAT,SUN *)</code>	UTC 午前 6:00 の週末	週末処理
	<code>cron(0 0 ? * MON#1 *)</code>	毎月第 1 月曜日の午前 0 時 UTC	月別計画

Cron 式フィールドリファレンス

フィールド	値	ワイルドカード	例
分 (1)	0-59	* , - /	0 (最上位)、 */15 (15 分ごと)、 0,30 (1 時間に 2 回)
時間 (2 番目)	0-23	* , - /	9 (午前 9 時)、 */2 (2 時間ごと)、 9-17 (営業時間)
Day-of-month (3)	1~31、L、W	* , - / ?	1 (1 日目)、 L (最終日)、 ? (day-of-weekを使用する場合)
月 (4 日)	1-12 または JAN-DEC	* , - /	1 (1 月)JAN、、 1,4,7,10 (四半期)
Day-of-week (5)	1-7 または SUN-SAT	* , - / ? # L	MON-FRI (平日)SUN、、 MON#1 (第 1 月曜日)
年 (6)	1970-2199	* , - /	* (毎年)、 2024 (特定の年)、 2024-2026 (範囲)

ワイルドカード文字と特殊文字

* (アスタリスク)

フィールド内のすべての値と一致します。例: hour フィールド*のは 1 時間ごとを意味します。

? (疑問符)

特定の値はありません。他の が指定されている場合、day-of-monthまたはday-of-week を使用します。例: day-of-month?で指定するときはday-of-weekMON-FRIで を使用します。

- (ダッシュ)

値の範囲。例: MON-FRI (月曜日から金曜日)、 9-17 (午前 9 時から午後 5 時)。

, (カンマ)

複数の特定の値。例: MON,WED,FRI (月曜日、水曜日、金曜日)、 8,12,17 (午前 8 時、正午、午後 5 時)。

/ (スラッシュ)

ステップ値または増分。例: 0/15 分単位は 0 分から 15 分ごと (0、15、30、45) を意味します。時間*/2単位は 2 時間ごとを意味します。

L (最新)

月の最終日または平日の最後の出現日。例: day-of-monthLは月の最終日を意味します。は月の最終金曜日FRILを意味します。

W (平日)

最も近い平日。例: は、月の 15 日に最も近い平日15Wを意味します。

(n 回目の出現)

月の平日の N 回目の出現。例: MON#1 は月の最初の月曜日を意味し、は月の第 2 金曜日FRI#2を意味します。

一般的なパターンとベストプラクティス

- ビジネスアプリケーションの場合: 週末や営業時間外にクエリが実行されないように、MON-FRIと 営業時間 (例: 9-17) を使用します。
- 高頻度モニタリングの場合: のような増分 */15 (15 分ごと) を使用しますが、クエリの同時実行数の制限に注意してください。
- リソース効率のため: UTC などの早朝の時間を使用して、オフピーク時にリソースを大量に消費するクエリ2-6をスケジュールします。
- 月別レポートの場合: 日の最終日Lには を使用し、日の最初の日1には のような特定の日付を使用して、一貫したタイミングを確保します。

ベストプラクティス

次のベストプラクティスに従って、信頼性が高く効率的なスケジュールされたクエリオペレーションを確保します。

クエリの最適化

- スケジュール前にクエリを手動でテストし、パフォーマンスと結果を検証する
- クエリの早い段階でフィルターインデックスを使用してデータ処理を減らす
- 大量のロググループのタイムアウトを避けるために時間範囲を制限する

- クエリの複雑さと実行時間の制限を考慮する

スケジュール計画

- 次のスケジュールされた実行の前にクエリが完了するようにすることで、実行が重複しないようにする
- 時間範囲を設定するときは、ログ取り込みの遅延を考慮する
- 特定の時間に cron 式を使用する
- スケジュールを分散して、クエリの同時実行数の制限に達しないようにする

モニタリングとメンテナンス

- 実行履歴を定期的にモニタリングして、障害やパフォーマンスの問題を特定する
- セキュリティを維持するために IAM ロールを定期的に確認および更新する
- 本番環境にデプロイする前に送信先アクセシビリティをテストする

Authorization

- スケジュールされたクエリのすべての APIs、ロググループなどの入力に含まれるリソースではなく、スケジュールされたクエリリソースで承認します。それに応じて IAM ポリシーを設定する
- APIs で渡された実行ロールを使用してロググループの認可を管理する

スケジュールされたクエリの開始方法

スケジュールされたクエリを作成するときは、クエリの実行方法と結果の配信場所を定義するいくつかの主要なコンポーネントを設定します。これらのコンポーネントを理解することは、効果的な自動ログ分析を設定するのに役立ちます。

スケジュールされた各クエリは、次の主要なコンポーネントで構成されます。

クエリ設定

分析に使用する CloudWatch Logs Insights クエリ文字列、ターゲットロググループ、およびクエリ言語。

スケジュール式

クエリが実行されるタイミングを定義する cron 式または頻度カレンダー。タイムゾーン設定を指定して、クエリが正しい現地時刻に実行されるようにできます。コンソールには、「毎週火曜日の 15:10 にクエリを 5 分間実行し、すぐに UTC で無期限に有効」など、人間が読めるようにスケジュールの説明が表示されます。

時間範囲

各クエリ実行のルックバック期間。実行時間と開始時間のオフセットで定義されます。これにより、各クエリ実行が分析する履歴データの量を決定します。

実行スケジュールのプレビュー

コンソールには、正確な日付と時刻 (15:10、UTC、2025/10/28 2025/11/04 15:10、UTC、2025/11/11 15:10、UTC など) を含む次の 3 つのスケジュールされたクエリ実行が表示され、スケジュールが正しく設定されていることを確認するのに役立ちます。

送信先

ここで、クエリ結果は正常に実行された後に配信されます。サポートされている送信先には Amazon S3 バケットが含まれ、デフォルトでは結果メタデータがデフォルトのイベントバスに送信されます。

実行ロール

CloudWatch Logs がクエリを実行し、指定された送信先に結果を配信するために引き受ける IAM ロール。

スケジュールされたクエリを作成する前に、必要なアクセス許可とリソースが設定されていることを確認してください。

スケジュールされたクエリの作成

CloudWatch Logs Insights クエリを自動的に実行し、選択した送信先に結果を配信するスケジュールされたクエリを作成します。

前提条件

スケジュールされたクエリを作成する前に、以下があることを確認してください。

- ロググループ - 分析するデータを含む 1 つ以上のロググループ
- 実行 IAM ロール - 次のアクセス許可を持つ IAM ロール。
 - logs:StartQuery - CloudWatch Logs Insights クエリを開始するアクセス許可
 - logs:GetQueryResults - クエリ結果を取得するアクセス許可
 - logs:DescribeLogGroups - ロググループ情報にアクセスするためのアクセス許可。これは、ロググループ検出のプレフィックススペースのロググループにのみ必要です。

- 送信先アクセス許可 - 選択した送信先に対する追加の IAM アクセス許可:
 - Amazon S3 の送信先の場合: `s3:PutObject`
- AWS CLI および API の使用 - CloudWatch Logs APIsを呼び出すアクセス許可を持つ AWS 認証情報を設定

IAM ポリシーの詳細な例については、「」を参照してください[Amazon CloudWatch Logs の Identity and Access Management](#)。また、アカウントごとにスケジュールされたクエリは 1,000 件のみです。

Console

スケジュールされたクエリを作成するには (コンソール)

1. で CloudWatch Logs コンソールを開きますか? <https://us-east-1.console.aws.amazon.com/cloudwatch/homeregion=us-east-1#logsV2:logs-insights>。
2. ナビゲーションペインで、Logs Insights を選択します。
3. スケジュールされたクエリの作成を選択します。
4. クエリ定義セクションで、次の操作を行います。
 - a. クエリ言語 で、リストから使用するクエリ言語を選択します。
 - b. クエリ文字列には、ボックスに CloudWatch Logs Insights クエリを入力します。
 - c. ロググループで、リストからクエリするロググループを選択します。
5. 「スケジュール設定」セクションで、次の操作を行います。
 - a. スケジュール式では、クエリが実行されるタイミングを設定します。事前定義されたオプションから選択するか、カスタム cron 式を入力します。
 - b. 「作成時に有効」で、スケジュールがいつアクティブになるかを指定します。すぐに開始するか、YYYY/MM/DD 形式を使用して特定の日時から開始するかを選択します。
 - c. 時間範囲には、各クエリ実行のルックバック期間を指定します。実行時間からクエリまでの時間を定義する時間を分単位で入力します。
 - d. 「無期限に続行する」で、スケジュールの終了時刻を指定します。無期限に実行するか、YYYY/MM/DD 形式を使用して特定の日時まで実行するかを選択します。
6. コンソールには、設定に基づいて次の 3 つのスケジュールされたクエリ実行が表示され、クエリが実行される正確な日時が UTC で表示されます。
7. 「クエリ結果を S3 に投稿する - オプション」セクション (S3 送信先を使用している場合):

- a. S3 バケットの場合、送信先バケットが同じアカウントにある場合はこの AWS アカウントを選択し、バケットが別の AWS アカウントにある場合は別のアカウントを選択し、バケット所有アカウントのアカウント ID を入力として指定します。
 - b. Amazon S3 URI の場合は、結果が保存される Amazon S3 バケットとプレフィックス (例:) を入力します `s3://my-bucket/query-results/`。このアカウントを選択した場合は、Amazon S3 の参照を選択して移動し、既存の Amazon S3 の場所を選択できます。
 - c. (オプション) KMS キー ARN には、カスタマーマネージド AWS KMS キーの ARN を入力して、SSE-KMS を使用してクエリ結果を暗号化します。キーは、送信先の Amazon S3 バケットと同じ AWS リージョンにある必要があります。
8. Amazon Amazon S3 ロールセクションで、次のいずれかのオプションを選択します。
- a. デフォルトのアクセス許可を持つ新しいロールの自動作成を選択すると、CloudWatch Logs がクエリ結果を Amazon S3 に配信するために必要なアクセス許可を持つ IAM ロールが自動的に設定されます。
 - b. 既存のロールを使用して、CloudWatch Logs がクエリ結果を Amazon S3 に配信するために必要なポリシーを持つ既存の IAM ロールを選択します。検索フィールドを使用して、リストから適切な IAM ロールを検索して選択します。
9. スケジュールされたクエリ実行の IAM ロールセクションで、次のいずれかのオプションを選択します。
- a. デフォルトのアクセス許可を持つ新しいロールを自動作成を選択して、CloudWatch Logs がスケジュールされたクエリを実行するために必要なアクセス許可を持つ IAM ロールを自動的にセットアップします。
 - b. 既存のロールを使用して、CloudWatch Logs がスケジュールされたクエリを実行するために必要なポリシーを持つ既存の IAM ロールを選択します。検索フィールドを使用して、リストから適切な IAM ロールを検索して選択します。
10. スケジュールの作成を選択して、スケジュールされたクエリを作成します。

AWS CLI

スケジュールされたクエリを作成するには (AWS CLI)

- `create-scheduled-query` コマンドを使用して、新しいスケジュールされたクエリを作成します。


```
aws logs create-scheduled-query \  
  --name "ErrorAnalysisQuery" \  
  --query-language "CWLI" \  
  --query-string "fields @timestamp, @message | filter @message like /ERROR/ |  
stats count() by bin(5m)" \  
  --schedule-expression "cron(8 * * * ? *)" \  
  --execution-role-arn "arn:aws:iam::123456789012:role/  
CloudWatchLogsScheduledQueryRole" \  
  --log-group-identifiers "/aws/lambda/my-function" "/aws/apigateway/my-api" \  
  --state "ENABLED"
```

API

スケジュールされたクエリを作成するには (API)

- `CreateScheduledQuery` アクションを使用して、新しいスケジュールされたクエリを作成します。次の例では、1 時間ごとに実行されるスケジュールされたクエリを作成します。

```
{  
  "name": "ErrorAnalysisQuery",  
  "queryLanguage": "CWLI",  
  "queryString": "fields @timestamp, @message | filter @message like /ERROR/ |  
stats count() by bin(5m)",  
  "scheduleExpression": "cron(8 * * * ? *)",  
  "executionRoleArn": "arn:aws:iam::123456789012:role/  
CloudWatchLogsScheduledQueryRole",  
  "logGroupIdentifiers": ["/aws/lambda/my-function", "/aws/apigateway/my-  
api"],  
  "state": "ENABLED"  
}
```

スケジュールされたクエリを作成したら、スケジュールされたクエリページから `ListScheduledQueries` API を使用してクエリを表示および管理できます。この API には、スケジュールされたすべてのクエリの名前、作成日、最終実行のステータス、最終トリガー時間、繰り返し頻度が表示されます。

スケジュールされたクエリの表示と管理

クエリごとに以下の情報を使用できます。

名前

スケジュールされたクエリに割り当てた一意の名前。詳細設定と実行履歴を表示するには、名前を選択します。

作成日

スケジュールされたクエリが作成された日付。YYYY-MM-DD 形式で表示されます。

最終実行のステータス

最新のクエリ実行の実行ステータス。可能な値は以下のとおりです:

- 完了 - クエリが正常に実行され、結果が設定されたすべての送信先に配信されました。
- Failed - クエリの実行または結果の配信に失敗しました。エラーの詳細については、実行履歴を確認してください。
- 無効なクエリ - クエリが無効で構文上の問題があります
- タイムアウト - クエリがタイムアウトしました。クエリは 60 分後に自動的にタイムアウトします

最後にトリガーされた時刻

クエリが最後に実行された日時。YYYY-MM-DD HH:MM:SS 形式で表示されます。クエリがまだ実行されていない場合は「Never」と表示されます。

ごとに繰り返す

クエリのスケジュール頻度。cron 式を使用するクエリのカスタム、またはより簡単なスケジュールの特定の頻度の説明を表示します。

スケジュールされたクエリページには、スケジュールされたすべてのクエリの概要が表示され、現在のステータスと実行履歴が表示されるため、スケジュールされたすべてのクエリを一元的に表示、モニタリング、管理できます。この情報を使用して、クエリのパフォーマンスのモニタリング、問題の特定、自動ログ分析ワークフローの管理を行います。

Console

スケジュールされたクエリを表示するには (コンソール)

- で CloudWatch Logs コンソールを開きますか? <https://us-east-1.console.aws.amazon.com/cloudwatch/homeregion=us-east-1#logsV2:logs-insights>。
- CloudWatch Logs コンソールで、スケジュールされたクエリ、スケジュールされたクエリの表示を選択します。

AWS CLI

スケジュールされたクエリを一覧表示するには (AWS CLI)

- `list-scheduled-queries` コマンドを使用して、スケジュールされたすべてのクエリを一覧表示します。

```
aws logs list-scheduled-queries --max-results 10
```

API

スケジュールされたクエリを一覧表示するには (API)

- `ListScheduledQueries` アクションを使用して、スケジュールされたすべてのクエリを取得します。

```
{
  "maxResults": 10
}
```

スケジュールされたクエリページヘッダーには、アカウントでスケジュールされたクエリの合計数が表示され、使用状況を追跡し、自動ログ分析ワークフローを効果的に管理できます。

スケジュールされたクエリ実行履歴の表示

実行履歴を使用して、スケジュールされたクエリのパフォーマンスをモニタリングし、クエリの実行または結果配信に関する問題をトラブルシューティングします。

実行履歴には、成功した実行、失敗、送信先処理結果など、各クエリ実行のステータスが表示されます。この情報を使用して、パターンの特定、問題の診断、クエリが正常に実行されていることの検証を行うことができます。

Console

実行履歴を表示するには (コンソール)

1. CloudWatch Logs コンソールで、スケジュールされたクエリ、スケジュールされたクエリの表示を選択します。
2. 検査するスケジュールされたクエリを選択します。

3. [Execution history (実行履歴の表示)] タブを選択します。

AWS CLI

実行履歴を表示するには (AWS CLI)

1. `get-scheduled-query-history` コマンドを使用して、スケジュールされたクエリの実行履歴を取得します。

```
aws logs get-scheduled-query-history \
  --identifier "DailyErrorMonitoring" \
  --start-time 1743379200 \
  --end-time 1743465600 \
  --max-results 10
```

2. 実行ステータスでフィルタリングするには、`--execution-statuses` パラメータを追加します。

```
aws logs get-scheduled-query-history \
  --identifier "DailyErrorMonitoring" \
  --start-time 1743379200 \
  --end-time 1743465600 \
  --max-results 1 \
  --execution-statuses "SUCCEEDED"
```

API

実行履歴を表示するには (API)

- `GetScheduledQueryHistory` アクションを使用して実行履歴を取得します。

```
{
  "identifier": "DailyErrorMonitoring",
  "startTime": 1743379200,
  "endTime": 1743465600,
  "maxResults": 10,
  "executionStatuses": ["SUCCEEDED", "FAILED"]
}
```

実行履歴には以下が表示されます。

- 実行ステータス - 実行中、完了、失敗、タイムアウト、または InvalidQuery
- トリガー時間 - クエリが実行された時刻
- 送信先 - S3 や EventBridge など、設定された各送信先の処理ステータス
- エラーメッセージ - クエリの実行または送信先処理の失敗に関する詳細

スケジュールされたクエリの更新

スケジュールされたクエリ設定を変更して、要件の進化に応じてクエリ文字列、スケジュール、送信先、または実行ロールを変更します。

クエリ文字列、スケジュール式、送信先、実行ロールなど、スケジュールされたクエリの任意の側面を更新できます。変更は、今後の実行ですぐに有効になります。

Console

スケジュールされたクエリを更新するには (コンソール)

1. CloudWatch Logs コンソールで、スケジュールされたクエリ、スケジュールされたクエリの表示を選択します。
2. 更新するスケジュールされたクエリを選択します。
3. [編集] を選択します。
4. 必要に応じて設定を変更します。
5. [Save changes] (変更の保存) をクリックします。

AWS CLI

スケジュールされたクエリを更新するには (AWS CLI)

- update-scheduled-query コマンドを使用して、既存のスケジュールされたクエリを変更します。

```
aws logs update-scheduled-query \
  --identifier "arn:aws:logs:us-east-1:111122223333:scheduled-
query:5e0c0228-1c29-4d26-904f-59f1f1ba3c8f" \
  --description "Monitor for ERROR level logs daily" \
  --query-language "LogsQL" \
```

```
--query-string "fields @timestamp, @message | filter @message like /ERROR/"
\
--log-group-identifiers "/aws/lambda/my-function-1" "/aws/lambda/my-
function-2"
```

API

スケジュールされたクエリを更新するには (API)

1. UpdateScheduledQuery アクションを使用して、スケジュールされたクエリ設定を変更します。

```
{
  "identifier": "arn:aws:logs:us-east-1:111122223333:scheduled-
query:5e0c0228-1c29-4d26-904f-59f1f1ba3c8f",
  "queryString": "fields @timestamp, @message | filter @message like /WARNING|
ERROR/ | stats count() by bin(5m)",
  "scheduleExpression": "cron(0 */2 * * ? *)",
  "state": "ENABLED"
}
```

2. 複数の設定パラメータを一度に更新するには:

```
{
  "identifier": "arn:aws:logs:us-east-1:111122223333:scheduled-
query:5e0c0228-1c29-4d26-904f-59f1f1ba3c8f",
  "queryString": "fields @timestamp, @message, @level | filter @level =
'ERROR'",
  "scheduleExpression": "cron(0 8,12,16 * * ? *)",
  "executionRoleArn": "arn:aws:iam::111122223333:role/
UpdatedScheduledQueryRole",
  "logGroupIdentifiers": ["/aws/lambda/my-function", "/aws/lambda/another-
function"],
  "destinationConfiguration": {
    "s3Configuration": {
      "destinationIdentifier": "s3://111122223333-sqn-results-bucket/
processed-results",
      "roleArn": "arn:aws:iam::111122223333:role/Admin"
    }
  }
}
```

スケジュールされたクエリの S3 送信先の設定

スケジュールされたクエリ結果を長期的な保持と分析のために JSON ファイルとして保存する送信先として Amazon S3 を設定します。

Amazon S3 を送信先として使用すると、クエリ結果は指定されたバケットとプレフィックスに JSON ファイルとして保存されます。このオプションは、結果のアーカイブ、バッチ分析の実行、または S3 データを処理する他の AWS サービスとの統合に最適です。

クエリ結果は、スケジュールされたクエリと同じ AWS アカウントの Amazon S3 バケット、または別の AWS アカウントのバケットに配信できます。オプションで、カスターマネージド AWS KMS キー (SSE-KMS) を使用してクエリ結果を暗号化することもできます。

同じアカウントの Amazon S3 バケットに結果を配信する

送信先 Amazon S3 バケットがスケジュールされたクエリと同じ AWS アカウントにある場合、コンソールから直接バケットを参照して選択できます。

同じアカウントの Amazon S3 送信先を設定するには (コンソール)

1. S3 へのクエリ結果の投稿セクションの S3 バケットで、このアカウントを選択します。
2. Amazon S3 URI の場合は、結果が保存される Amazon S3 バケットとプレフィックス (たとえば `s3://my-bucket/query-results/`) を入力するか、Amazon S3 を参照を選択して移動し、既存の Amazon S3 の場所を選択します。
3. (オプション) カスターマネージド AWS KMS キーを使用して結果を暗号化するには、KMS AWS KMS キー ARN フィールドにキーの ARN を入力します。キーは、送信先の Amazon S3 バケットと同じ AWS リージョンにある必要があります。AWS KMS キーを指定しない場合、バケットのデフォルトの暗号化設定が適用されます。
4. Amazon S3 にクエリ結果を投稿するための IAM ロールセクションで、デフォルトのアクセス許可を持つ新しいロールの自動作成を選択して必要なアクセス許可を自動的に設定するか、既存のロールを使用して必要なポリシーを持つ既存の IAM ロールを選択します。

送信先配信 IAM ロールには、次のアクセス許可が必要です。

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

```
{
  "Effect": "Allow",
  "Action": "s3:PutObject",
  "Resource": "arn:aws:s3::my-bucket/prefix/*",
  "Condition": {
    "StringEquals": {
      "aws:ResourceAccount": "111122223333"
    }
  }
}
```

別のアカウントの Amazon S3 バケットに結果を配信する

スケジュールされたクエリ結果を別の AWS アカウントの Amazon S3 バケットに配信できます。クロスアカウントバケットを使用する場合は、Amazon S3 URI とバケット所有アカウントのアカウント ID を指定する必要があります。

クロスアカウント Amazon S3 の送信先を設定するには (コンソール)

1. S3 へのクエリ結果の投稿セクションの S3 バケットで、別のアカウントを選択し、バケット所有アカウントのアカウント ID を入力として指定します。
2. Amazon S3 URI の場合は、送信先バケットの完全な Amazon S3 URI と、他のアカウントのプレフィックス (例:) を入力します `s3://cross-account-bucket/query-results/`。
3. (オプション) カスタマーマネージド AWS KMS キーで結果を暗号化するには、KMS AWS KMS キー ARN フィールドにキーの ARN を入力します。キーは、送信先の Amazon S3 バケットと同じ AWS リージョンにある必要があります。
4. Amazon S3 にクエリ結果を投稿するための IAM ロールセクションで、デフォルトのアクセス許可を持つ新しいロールを自動作成して必要なアクセス許可を自動的に設定するか、既存のロールを使用して必要なポリシーを持つ既存の IAM ロールを選択します。

クロスアカウント配信には、両側に対するアクセス許可が必要です。ソースアカウントの宛先配信 IAM ロールには、次のアクセス許可が必要です。

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



```

    "Effect": "Allow",
    "Action": "s3:PutObject",
    "Resource": "arn:aws:s3:::cross-account-bucket/prefix/*",
    "Condition": {
      "StringEquals": {
        "aws:ResourceAccount": "123456789012"
      }
    }
  }
]
}

```

送信先アカウントの Amazon S3 バケットポリシーは、オブジェクトを書き込むアクセス許可をソースアカウントの IAM ロールに付与する必要があります。

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowScheduledQueryRolePutObject",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:role/my-s3-delivery-role"
      },
      "Action": "s3:PutObject",
      "Resource": "arn:aws:s3:::cross-account-bucket/prefix/*"
    }
  ]
}

```

カスターマネージド AWS KMS キーによる結果の暗号化

オプションでカスターマネージド AWS KMS キーを指定して、SSE-KMS を使用して Amazon S3 に配信されるクエリ結果を暗号化できます。AWS KMS キーは、スケジュールされたクエリと同じアカウントまたは別のアカウントにあることができます。

AWS KMS キーを指定すると、スケジュールされたクエリはそのキーを使用して SSE-KMS を介して結果を暗号化します。AWS KMS キーを指定しない場合、バケットのデフォルトの暗号化設定が適用されます。バケットがカスターマネージドキーを使用したデフォルトの SSE-KMS 暗号化で設定されている場合、送信先配信 IAM ロールには引き続きそのキーに対する `kms:GenerateDataKey` アクセス許可が必要です。

送信先配信 IAM ロールには、AWS KMS キーに対する `kms:GenerateDataKey` アクセス許可が必要です。次の例は、カスタマーマネージド AWS KMS キーを持つ Amazon S3 送信先に必要なアクセス許可を示しています。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "s3:PutObject",
      "Resource": "arn:aws:s3:::my-bucket/prefix/*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceAccount": "111122223333"
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": "kms:GenerateDataKey",
      "Resource": "arn:aws:kms:us-east-1:111122223333:key/key-id",
      "Condition": {
        "StringEquals": {
          "kms:ViaService": "s3.us-east-1.amazonaws.com"
        },
        "StringLike": {
          "kms:EncryptionContext:aws:s3:arn": "arn:aws:s3:::my-bucket*"
        }
      }
    }
  ]
}
```

AWS KMS キーが送信先配信 IAM ロールとは異なるアカウントにある場合、AWS KMS キー所有者アカウントのキーポリシーはロールに明示的にアクセスを許可する必要があります。

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

```
    "Principal": {
      "AWS": "arn:aws:iam::111122223333:role/my-s3-delivery-role"
    },
    "Action": "kms:GenerateDataKey",
    "Resource": "*",
    "Condition": {
      "StringEquals": {
        "kms:ViaService": "s3.us-east-1.amazonaws.com"
      },
      "StringLike": {
        "kms:EncryptionContext:aws:s3:arn": "arn:aws:s3:::my-bucket*"
      }
    }
  }
]
```

Note

AWS KMS キーと宛先配信 IAM ロールが同じアカウントにある場合、AWS KMS キーポリシーにデフォルトの「IAM ポリシーを有効にする」ルートステートメントが含まれている場合は、IAM ID ポリシーだけで十分です。明示的な AWS KMS キーポリシーの付与は、キーポリシーが IAM に委任されない場合にのみ必要です。

Amazon S3 にクエリ結果を投稿するための IAM ロールは、スケジュールされたクエリ実行の IAM ロールとは別に設定する必要があります。この分離により、実行ロールがクエリを実行でき、Amazon S3 ロールが結果配信を特に処理するきめ細かなアクセスコントロールが可能になります。どちらのロールにも、CloudWatch Logs サービス (logs.amazonaws.com) がロールを引き受けることを許可する信頼ポリシーが含まれている必要があります。

スケジュールされたクエリのトラブルシューティング

これらのトラブルシューティングトピックを使用して、スケジュールされたクエリに関する一般的な問題を解決します。

クエリの実行がアクセス許可エラーで失敗する

スケジュールされたクエリの実行または送信先への結果の配信を妨げるアクセス許可エラーを解決します。

アクセス許可エラーは、実行ロールにロググループからの読み取りまたは送信先リソースへの書き込みに必要なアクセス許可がない場合に発生します。

アクセス許可エラーを解決するには

1. 実行ロールにlogs:StartQuery、ターゲットロググループの、logs:GetQueryResults、およびアクセスlogs:DescribeLogGroups許可があることを確認します。
2. 実行ロールに、送信先リソース (S3 バケットs3:PutObjectなど) に対する書き込みアクセス許可があることを確認します。
3. 信頼ポリシーで CloudWatch Logs が実行ロールを引き受けることが許可されていることを確認します。ロールは、信頼ポリシーでログサービスプリンシパル (logs.amazonaws.com) を信頼する必要があります。

一般的な原因には、IAM アクセス許可の欠落、ポリシー内のリソース ARNs の誤り、信頼ポリシーの設定の問題などがあります。

アクセス許可エラーを防ぐには、スケジュールされたクエリを本番環境にデプロイする前に、実行ロールとテストアクセス許可を作成するときに最小特権の原則を使用します。

クエリのタイムアウト

スケジュールされたクエリが最大実行時間制限を超えたときに発生するタイムアウトエラーを解決します。

クエリのタイムアウトは、クエリが指定されたデータ範囲の処理に 60 分以上かかる場合に発生します。多くの場合、大規模なデータセットや複雑なクエリロジックが原因です。

タイムアウトエラーを解決するには

1. 開始時間のオフセットを小さくして時間範囲を短縮し、実行あたりの処理データを減らします。
2. クエリの早い段階でフィルターを追加してクエリを最適化し、処理されるデータ量を減らします。フィルターインデックスを使用してデータスキャンサイズを縮小します。
3. 複雑なクエリをよりシンプルで焦点を絞ったクエリに分割することを検討してください。

一般的な原因には、大量のロググループのクエリ、大量のロググループの処理、適切なフィルタリングなしでの複雑な集計の使用などがあります。

タイムアウトを防ぐには、CloudWatch Logs Insights で予想されるデータボリュームを使用してクエリを手動でテストし、スケジューリング前にパフォーマンスを最適化します。

送信先処理が失敗する

スケジュールされたクエリ結果を設定された送信先に配信できない場合に発生する障害を解決します。

送信先処理の失敗は、ターゲット Amazon S3 バケットまたは EventBridge イベントバスにアクセスできないか、正しく設定されていない場合に発生します。

クエリ結果が送信先に発行されない障害を解決するには

1. 指定された Amazon S3 バケットが存在し、アクセス可能であることを確認します。
2. 正しい URIs の送信先設定を確認します。
3. 実行ロールに、送信先に書き込むために必要なアクセス許可があることを確認します。

一般的な原因には、宛先リソースの削除または名前の変更、宛先 URIs 誤り、ネットワーク接続の問題などがあります。

送信先の障害を防ぐには、送信先設定を定期的に検証し、送信先リソースの可用性をモニタリングします。

無効なクエリエラー

スケジュールされたクエリ文字列の構文エラーとロジックエラーを解決して、正常に実行できないようにします。

無効なクエリエラーは、クエリ文字列に構文エラーが含まれているか、存在しないフィールドを参照しているか、サポートされていないクエリ言語機能を使用している場合に発生します。

無効なクエリエラーを解決するには

1. CloudWatch Logs Insights でクエリを手動でテストして、構文とロジックを検証します。
2. 参照されるすべてのログフィールドがターゲットロググループに存在することを確認します。
3. 使用しているクエリ言語機能がスケジュールされたクエリでサポートされていることを確認します。

一般的な原因には、フィールド名のタイプミス、誤ったクエリ構文、スケジュールされた実行環境でサポートされていないクエリ機能の使用などがあります。

無効なクエリエラーを防ぐには、スケジュールする前にクエリをインタラクティブにテストし、フィールド検出機能を使用してフィールド名を確認します。

クエリ同時実行エラー

スケジュールされたクエリが Cloudwatch Logs インサイトクエリと同じクォータを使用していると同時実行エラーが見られる場合、以下に留意すべき重要な点がいくつかあります。同時実行数の制限に達しないように、スケジュールを分散することをお勧めします。

- **クォータ:** AWS アカウントごとに最大 100 個の CloudWatch Logs Insights クエリを同時に実行できます。
- **ダッシュボード:** CloudWatch ダッシュボードに追加されたクエリは、ダッシュボードがロードまたは更新されたときに実行されるため、この同時実行数の制限にもカウントされます。
- **OpenSearch Service PPL/SQL:** AWS アカウントごとに最大 15 個の OpenSearch PPL または OpenSearch SQL クエリを同時に実行できます。
- **クロスアカウントクエリ:** 同時実行クォータは、単一クエリとクロスアカウントクエリの両方に適用されます。CloudWatch クロスアカウントオブザーバビリティを使用する場合、リンクされたソースアカウントに対してモニタリングアカウントで開始されたクエリも、モニタリングアカウントの同時実行制限にカウントされます。
- **低頻度アクセスロググループ:** 低頻度アクセスログクラスのロググループの場合、同時 Logs Insights クエリの最大数は 5 に制限されます。

ログ異常検出

ログデータの異常を検出するには、継続的モニタリング用のログ異常ディテクターを作成するか、CloudWatch Logs Insights クエリの [anomaly detection](#) コマンドを使用してオンデマンド分析を行うという 2 つの方法があります。

ログ異常ディテクターは、ロググループに取り込まれたログイベントをスキャンし、ログデータ内の異常を自動的に検出します。異常検出では、機械学習とパターン認識を使用して、一般的なログコンテンツのベースラインを確立します。オンデマンド分析では、CloudWatch Logs Insights クエリで `anomaly detection` コマンドを使用し、時系列データの異常なパターンを特定できます。クエリベースの異常検出の詳細については、「[CloudWatch Logs Insights での異常検出の使用](#)」を参照してください。

ロググループの異常ディテクターを作成すると、そのグループの過去 2 週間分のログイベントを使用してトレーニングが行われます。トレーニングの所要時間は最大 15 分間です。トレーニングが完了すると、受信ログの分析が開始されて異常が識別されます。CloudWatch Logs コンソールに異常が表示されたら、調査を行います。

CloudWatch Logs パターン認識では、ログ内の静的コンテンツと動的コンテンツを識別することによって、ログパターンを抽出します。多数のログイベントをいくつかのパターンに圧縮することができるため、パターンは大きなログセットを分析する際に役立ちます。

例えば、次の 3 つのログイベントの例を参照してください。

```
2023-01-01 19:00:01 [INFO] Calling DynamoDB to store for ResourceID: 12342342k124-12345
2023-01-01 19:00:02 [INFO] Calling DynamoDB to store for ResourceID: 324892398123-1234R
2023-01-01 19:00:03 [INFO] Calling DynamoDB to store for ResourceID: 3ff231242342-12345
```

前のサンプルでは、3 つのログイベントはすべて 1 つのパターンに従っています。

```
<Date-1> <Time-2> [INFO] Calling DynamoDB to store for resource id <ResourceID-3>
```

パターン内のフィールドはトークンと呼ばれます。リクエスト ID やタイムスタンプなど、パターン内の異なるフィールドは動的トークンと呼ばれます。動的トークンのそれぞれ異なる値は、トークン値と呼ばれます。

CloudWatch Logs が動的トークンの表すデータのタイプを推測できる場合、トークンは `<string-number>` として表示されます。### は、トークンが表すデータのタイプの説明です。# は、他の動的トークンと比較して、このトークンがパターン内のどこに表示されるかを示します。

CloudWatch Logs は、名前を含むログイベントのコンテンツの分析に基づいて、その名前の文字列部分を割り当てます。

CloudWatch Logs が動的トークンの表すデータのタイプを推測できない場合は、トークンは <トークン - ##> として表示され、## は他の動的トークンと比較して、このトークンがパターン内のどこに表示されるかを示します。

動的トークンの一般的な例には、エラーコード、IP アドレス、タイムスタンプ、リクエスト ID があります。

異常検出口は、これらのパターンを使用して異常を検出します。異常ディテクターのモデルトレーニング期間が終了したら、ログは既知の傾向に照らして評価されます。異常ディテクターは、大幅な変動を異常とみなし、フラグを立てます。

この章では、異常検出の有効化の方法、異常の表示方法、ログ異常ディテクターのアラームおよび異常ディテクターが発行するメトリクスの作成方法について説明します。また、異常ディテクターとその結果を暗号化する方法についても説明します AWS Key Management Service。

ログ異常ディテクターを作成しても料金は発生しません。

異常とパターンの重要度と優先度

ログ異常ディテクターによって検出される各異常には、優先度が割り当てられます。検出された各パターンには重要度が割り当てられます。

- 優先度は自動的に計算され、パターンの重要度レベルと想定値からの偏差量の両方に基づきます。例えば、特定のトークン値が突然 500% 増加すると、その重要度が NONE であっても、その異常が HIGH 優先度として指定される可能性があります。
- 重要度は、FATAL、ERROR、WARN などのパターンで見つかったキーワードのみに基づいています。これらのキーワードが見つからない場合、パターンの重要度は NONE としてマークされます。

異常可視性期間

異常ディテクターを作成するときは、その異常の最大可視性期間を指定します。これは、異常がコンソールに表示され、[ListAnomalies](#) API オペレーションによって返される日数です。この期間が経過した後も異常が発生し続けると、自動的に通常の動作として受け入れられ、異常検出モデルは異常としてフラグ付けするのを停止します。

異常ディテクターの作成時に可視性期間を調整しない場合、デフォルトで 21 日を使用されます。

異常の抑制

異常が検出されたら、一時的または永続的に抑制することを選択できます。異常を抑制すると、指定した期間中、異常ディテクターはこの発生を異常としてフラグ付けしなくなります。異常を抑制する場合、その特定の異常のみを抑制するか、異常が検出されたパターンに関連するすべての異常を抑制するかを選択できます。

抑制された異常はコンソールで表示できます。また、抑制を停止することもできます。

よくある質問

自分のデータ AWS を使用して機械学習アルゴリズムをトレーニングしたり、他の顧客向けにトレーニング AWS したりしていますか？

いいえ。トレーニングによって作成された異常検出モデルは、ロググループのログイベントに基づいており、そのロググループとその AWS アカウント内でのみ使用されます。

異常検出にはどのようなタイプのログイベントが適していますか？

ログ異常検出は、アプリケーションログや、ほとんどのログエントリが一般的なパターンに適合するその他のタイプのログに適しています。INFO、ERROR、DEBUG などのログレベルまたは重要度キーワードを含むイベントを持つロググループは、ログ異常検出に特に適しています。

ログ異常検出は、CloudTrail Logs などの非常に長い JSON 構造を持つログイベントには適していません。パターン分析では、ログラインの最初の 1500 文字までしか分析されないため、その制限を超える文字はスキップされます。

VPC フローログなどの監査ログやアクセスログも、異常検出で成功しません。異常検出はアプリケーションの問題を検出することを目的としているため、ネットワークやアクセスの異常には適さない可能性もあります。

異常ディテクターが特定のロググループに適しているかどうかを判断するには、CloudWatch Logs パターン分析を使用してグループのログイベントの中からパターン数を見つけます。パターン数が約 300 以下の場合、異常検出が適切に機能する可能性があります。パターン分析の詳細については、「[パターン分析](#)」を参照してください。

何が異常としてフラグ付けされるのですか？

次の状況が発生すると、ログイベントに異常としてフラグが立てられる可能性があります。

- ロググループで以前には見られなかったパターンを持つログイベント。

- 既知のパターンに対する大きな変化。
- 通常の値の個別のセットを持つ動的トークンの新しい値。
- 動的トークンの値の発生回数の大きな変化。

上記の項目はすべて異常としてフラグ付けされる可能性があります。すべてがアプリケーションのパフォーマンスが低いことを意味するわけではありません。例えば、通常よりも多くの 200 成功値が異常としてフラグ付けされる場合もあります。このような場合は、問題を示していないとして、これらの異常を抑制することを検討してください。

マスキングされている機密データはどうなりますか？

機密データとしてマスクされたログイベントの部分は、異常検出のためにスキャンされることはありません。機密データのマスキングの詳細については、「[Help protect sensitive log data with masking](#)」を参照してください。

CloudWatch Logs Insights での異常検出の使用

継続的モニタリング用のログ異常ディテクターを作成するだけでなく、CloudWatch Logs Insights クエリで `anomaly` コマンドを使用して、ログデータの異常なパターンをオンデマンドで特定することもできます。このコマンドは既存の `pattern` 機能を拡張し、機械学習を使用して、パターン頻度の変化、新しいパターン、トークンのバリエーションなど、5 種類の異常を検出します。

`anomaly` コマンドは、特に次の場合に便利です。

- 異常なパターンを特定するための履歴ログデータのアドホック分析
- 異常動作の特定の期間の調査
- Lambda 関数などのアプリケーションの実行に関する問題のモニタリング

クエリにおける `anomaly` コマンドの使用方法の詳細については、「[異常](#)」を参照してください。

このクエリベースの異常検出は、以下のセクションで説明する継続的な異常ディテクターを補完し、リアルタイムのモニタリング機能とオンデマンド分析機能の両方を提供します。

ロググループの異常検出を有効にする

次の手順を使用して、CloudWatch コンソールを使用して、ロググループをスキャンして異常を検出するログ異常ディテクターを作成します。

プログラムで異常検知器を作成することもできます。詳細については、
「[CreateLogAnomalyDetector](#)」を参照してください。

ログ異常ディテクターを作成するには

1. CloudWatch コンソールの <https://console.aws.amazon.com/cloudwatch/> を開いてください。
2. [ログ]、[ログ異常] を選択します。
3. [異常ディテクターを作成] を選択します。
4. この異常ディテクターを作成するロググループを選択します。
5. [異常検出器の名前] にディテクターの名前を入力します。
6. (オプション) [評価頻度] をデフォルトの 5 分から変更します。ロググループが新しいログを受信する頻度に従って、この値を設定します。例えば、ロググループが 10 分ごとに新しいログイベントをバッチで受信する場合は、評価頻度を 15 分に設定するのが適切です。
7. (オプション) 特定の単語または文字列を含むログイベントでのみ異常を検索するように異常ディテクターを設定するには、[パターンをフィルタリング] を選択します。

次に、[異常検出フィルターパターン] にパターンを入力します。パターン構文の詳細については、「[メトリクスフィルター、サブスクリプションフィルター、フィルターログイベント、およびライブテールのフィルターパターン構文](#)」を参照してください。

(オプション) フィルターパターンをテストするには、[イベントメッセージをログ記録] にいくつかのログメッセージを入力し、[パターンをテスト] を選択します。

8. (オプション) 異常可視性期間をデフォルトから変更するか、AWS KMS キーをこの異常ディテクターに関連付けるには、詳細設定を選択します。
 - a. 異常可視性期間をデフォルトから変更するには、[異常の最大表示期間 (日数)] に新しい値を入力します。
 - b. AWS KMS キーをこの異常ディテクターに関連付けるには、KMS キー ARN に ARN を入力します。キーを割り当てると、このディテクターによって検出された異常情報は、保管時にキーで暗号化されます。検出された異常に関する情報を取得するには、このキーと異常ディテクターへのアクセス許可が必要です。

また、CloudWatch Logs サービスプリンシパルにキーを使用するアクセス許可があることを確認する必要があります。詳細については、「[で異常ディテクターとその結果を暗号化する AWS KMS](#)」を参照してください。

9. [異常検出を有効にする] を選択します。

異常ディテクターが作成され、ロググループが取り込むログイベントに基づいてモデルのトレーニングが開始されます。約 15 分後、異常検出がアクティブになり、異常の検出を開始して表面化させます。

検出された異常を表示する

1 つ以上のログ異常ディテクターを作成したら、CloudWatch コンソールを使用して、検出された異常を表示できます。

プログラムで異常を表示できます。詳細については、「[ListAnomalies](#)」を参照してください。

すべてのログ異常ディテクターによって検出された異常を表示するには

1. CloudWatch コンソールの <https://console.aws.amazon.com/cloudwatch/> を開いてください。
2. [ログ]、[ログ異常] を選択します。

[ログの異常] テーブルが表示されます。[ログ異常] の横にある上部の数値は、テーブルにリストされているログ異常の数を表します。テーブルの各行に表示される情報は、次のとおりです。

- [異常] 列には、異常の簡単な概要が表示されます。これらの概要は CloudWatch Logs によって生成されます。
- 異常の優先度。優先度は、ログイベントの変化量、ログイベントで発生する Exception などのキーワードに基づくなどして自動的に計算されます。
- 異常の根拠となるログパターン。パターンの詳細については、「[ログ異常検出](#)」を参照してください。
- [異常ログのトレンド] には、パターンに一致するログの量を示すヒストグラムが表示されます。
- [最終検出時刻] には、この異常が最後に検出された時刻が表示されます。
- [初回検出時刻] には、この異常が最初に検出された時刻が表示されます。
- [異常ディテクター] には、この異常に関連するログイベントを含むロググループの名前が表示されます。この名前を選択すると、ロググループの詳細ページを表示できます。

3. 1 つの異常をさらに検査するには、その行のラジオボタンを選択します。

[パターン検査] ペインが表示され、以下が表示されます。

- この異常の根拠となるパターン。パターン内のトークンを選択して、そのトークンの値を分析します。
- クエリされた時間範囲における異常の発生回数を示すヒストグラム。
- [ログサンプル] タブには、異常の一部であるログイベントがいくつか表示されます。
- [トークンの値] タブには、選択している場合は、選択した動的トークンの値が表示されます。

 Note

トークンごとに最大 10 個のトークン値がキャプチャされます。トークン数は正確ではない可能性があります。CloudWatch Logs は確率カウンターを使用して、絶対値ではなくトークン数を生成します。

4. 異常を抑制するには、その行のラジオボタンを選択し、以下を実行します。
 - a. [アクション]、[異常を抑制] を選択します。
 - b. 次に、異常を抑制する期間を指定します。
 - c. このパターンに関連するすべての異常を抑制するには、[抑制パターン] を選択します。
 - d. [異常を抑制] を選択します。

単一のロググループで検出された異常を表示するには

1. CloudWatch コンソールの <https://console.aws.amazon.com/cloudwatch/> を開いてください。
2. [Logs] (ログ)、[Log groups] (ロググループ) を選択します。
3. ロググループの名前を選択し、[異常検出] タブを選択します。

[異常検出] テーブルが表示されます。[ログ異常] の横にある上部の数値は、テーブルにリストされているログ異常の数を表します。テーブルの各行に表示される情報は、次のとおりです。

- [異常] 列には、異常の簡単な概要が表示されます。これらの概要は CloudWatch Logs によって生成されます。
- 異常の優先度。優先度は、ログイベントの変化量、ログイベントで発生する Exception などのキーワードに基づくなどして自動的に計算されます。
- 異常の根拠となるログパターン。パターンの詳細については、「[ログ異常検出](#)」を参照してください。

- [異常ログのトレンド] には、パターンに一致するログの量を示すヒストグラムが表示されます。
 - [最終検出時刻] には、この異常が最後に検出された時刻が表示されます。
 - [初回検出時刻] には、この異常が最初に検出された時刻が表示されます。
4. 1 つの異常をさらに検査するには、その行のラジオボタンを選択します。

[パターン検査] ペインが表示され、以下が表示されます。

- この異常の根拠となるパターン。パターン内のトークンを選択して、そのトークンの値を分析します。
- クエリされた時間範囲における異常の発生回数を示すヒストグラム。
- [ログサンプル] タブには、異常の一部であるログイベントがいくつか表示されます。
- [トークンの値] タブには、選択している場合は、選択した動的トークンの値が表示されます。

 Note

トークンごとに最大 10 個のトークン値がキャプチャされます。トークン数は正確ではない可能性があります。CloudWatch Logs は確率カウンターを使用して、絶対値ではなくトークン数を生成します。

5. 異常を抑制するには、その行のラジオボタンを選択し、以下を実行します。
- a. [アクション]、[異常を抑制] を選択します。
 - b. 次に、異常を抑制する期間を指定します。
 - c. このパターンに関連するすべての異常を抑制するには、[抑制パターン] を選択します。
 - d. [異常を抑制] を選択します。

ログ異常ディテクターにアラームを作成する

ロググループのログ異常ディテクターのアラームを作成できます。指定した期間中にロググループで指定された数の異常が検出されたときにアラームの状態が ALARM になるように指定できます。フィルターを使用して、指定された優先順位の異常のみがアラームにカウントされるようにすることもできます。

ログ異常ディテクターのアラームを作成するには

1. CloudWatch コンソールの <https://console.aws.amazon.com/cloudwatch/> を開いてください。
2. ナビゲーションペインで、[ログ]、[ログ異常] の順に選択します。

ログ異常ディテクターのテーブルが表示されます。

3. アラームを設定する異常ディテクターのラジオボタンを選択し、[アラームを作成] を選択します。

CloudWatch アラーム作成ウィザードが表示されます。[LogAnomalyDetector] フィールドには、選択した異常ディテクターの名前が表示されます。[メトリクス名] フィールドに AnomalyCount が表示されます。

4. (オプション) このアラームを異常優先度でフィルタリングするには、次のいずれかを実行します。
 - アラームカウントを優先度の高い異常のみにするには、[LogAnomalyPriority] に **HIGH** と入力します。
 - アラームカウントを優先度の高い異常と中程度の異常のみにするには、[LogAnomalyPriority] に **MEDIUM** と入力します。

優先度の詳細については、「[異常とパターンの重要度と優先度](#)」を参照してください。

5. アラームに静的またはメトリクスの異常検出しきい値を使用するかどうかを選択します。この選択により、アラームしきい値の設定方法が決まります。静的しきい値とは、アラームしきい値が選択した静的な定数であることを意味します。異常検出しきい値とは、CloudWatch が通常の値の範囲を決定し、実際の数がこのバンドのしきい値を超えた場合にアラームがトリガーされることを意味します。ログ異常検出アラームで [異常検出] を選択する必要はありません。メトリクス異常検出の詳細については、「[Using CloudWatch anomaly detection](#)」を参照してください。
6. [**your-metric-name** が次の時...] で、[より大きい]、[以上]、[以下]、または [より低い] から選択します。次に、[... よりも] で、しきい値の数値を指定します。[期間] で指定された時間内に異常ディテクターがこの数を超えるアラームを検出した場合、アラームは ALARM 状態になります。
7. [Additional configuration (追加設定)] を選択します。[Datapoints to alarm (アラームを発生させるデータポイント数)] で、アラームをトリガーするために ALARM 状態を維持する必要がある評価期間 (データポイント) の数を指定します。2 つの値が一致する場合は、該当する数の連続した期間でしきい値を超過したときに ALARM 状態に移行するアラームを作成します。

N 個中 M 個のアラームを作成するには、2 番目の値よりも小さい数字を最初の値に指定します。詳細については、「[Evaluating an alarm](#)」を参照してください。

8. [Missing data treatment] (欠落データの処理) で、一部のデータポイントが欠落している際のアラームによる対処方法を選択します。詳細については、「[Configuring how CloudWatch alarms treat missing data](#)」を参照してください。
9. [次へ] を選択します。
10. [通知] で、[通知の追加] を選択し、その後アラームが ALARM、OK、INSUFFICIENT_DATA のいずれかの状態に遷移するときに通知するための Amazon SNS トピックを指定します。
 - a. (オプション) 同じアラーム状態または異なるアラーム状態について複数の通知を送信するには、[Add notification] (通知の追加) を選択します。

 Note

[アラーム] 状態になったときだけでなく、[データ不足] 状態になったときにもアクションを実行するようにアラームを設定することをお勧めします。これは、データソースに接続する Lambda 関数に関する多くの問題により、アラームが [データ不足] に遷移する可能性があるためです。

- b. (オプション) Amazon SNS の通知を送信しない場合は、[削除] を選択します。
11. (オプション) アラームで Amazon EC2 Auto Scaling、Amazon EC2、チケット、または のアクションを実行する場合は AWS Systems Manager、適切なボタンを選択し、アラームの状態とアクションを指定します。

 Note

アラームは、ALARM 状態になった時にのみ、Systems Manager のアクションを実行できます。Systems Manager のアクションの詳細については、「[Configuring CloudWatch to create OpsItems](#)」(OpsItems を作成するように CloudWatch を設定する) および「[Incident creation](#)」(インシデントの作成) を参照してください。

12. [次へ] を選択します。
13. [名前と説明] にアラームの名前と説明を入力し、[次へ] を選択してください。アラーム名には UTF-8 文字のみを使用する必要があり、ASCII 制御文字は使用できません。説明にはマークダウン形式を含めることができます。マークダウン形式は、CloudWatch コンソールのアラーム

の [詳細] タブにのみ表示されます。マークダウンは、ランブックや他の内部リソースへのリンクを追加するのに役立ちます。

 Tip

アラーム名には UTF-8 文字のみを使用する必要があります。ASCII 制御文字を含めることはできません。

14. [プレビューと作成] で、アラームの情報と条件が正しいことを確認し、[アラームの作成] を選択します。

ログ異常ディテクターによって発行されるメトリクス

CloudWatch Logs は AnomalyCount メトリクスを CloudWatch メトリクスに発行します。このメトリクスは、AWS/Logs 名前空間に公開されます。

AnomalyCount メトリクスは、次のディメンションで発行されます。

- LogAnomalyDetector – 異常ディテクターの名前
- LogAnomalyPriority – 異常の優先度レベル

で異常ディテクターとその結果を暗号化する AWS KMS

CloudWatch Logs では、異常検出データは常に暗号化されます。デフォルトでは、CloudWatch Logs は保管中のデータに対してサーバー側の暗号化を使用します。別の方法として、この暗号化 AWS Key Management Service に 使用することもできます。その場合、暗号化は AWS KMS キーを使用して行われます。AWS KMS を使用した暗号化は、KMS キーを異常ディテクターに関連付けることで異常ディテクターレベルで有効になります。

 Important

CloudWatch Logs は、対称 KMS キーのみをサポートします。非対称キーを使用してロググループのデータを暗号化しないでください。詳細については、「[対称キーと非対称キーの使用](#)」を参照してください。

制限

- 以下の手順を実行するには、`kms:CreateKey`、`kms:GetKeyPolicy`、および `kms:PutKeyPolicy` アクセス許可が必要です。
- キーと異常ディテクターを関連付けるか、あるいは関連付けを解除すると、オペレーションが有効になるまで最大 5 分かかることがあります。
- 関連付けられたキーへの CloudWatch Logs のアクセスを取り消したり、関連付けられた KMS キーを削除した場合、CloudWatch Logs 内の暗号化されたデータを取得できなくなります。

ステップ 1: AWS KMS キーを作成する

KMS キーを作成するには、次の [create-key](#) コマンドを使用します。

```
aws kms create-key
```

出力には、キーのキー ID と Amazon リソースネーム (ARN) が含まれます。出力例を次に示します。

```
{
  "KeyMetadata": {
    "Origin": "AWS_KMS",
    "KeyId": "key-default-1",
    "Description": "",
    "KeyManager": "CUSTOMER",
    "Enabled": true,
    "CustomerMasterKeySpec": "SYMMETRIC_DEFAULT",
    "KeyUsage": "ENCRYPT_DECRYPT",
    "KeyState": "Enabled",
    "CreationDate": 1478910250.94,
    "Arn": "arn:aws:kms:us-west-2:123456789012:key/key-default-1",
    "AWSAccountId": "123456789012",
    "EncryptionAlgorithms": [
      "SYMMETRIC_DEFAULT"
    ]
  }
}
```

ステップ 2: KMS キーでアクセス許可を設定する

デフォルトでは、すべての AWS KMS キーはプライベートです。リソースの所有者のみがその CMK を使用してデータを暗号化および復号できます。ただし、リソース所有者は、他のユーザーとリソースに KMS キーへのアクセス許可を付与することができます。このステップでは、CloudWatch Logs サービスプリンシパルに、キーを使用するアクセス許可を付与します。このサービスプリンシパルは、KMS キーが保存されているリージョンと同じ AWS リージョンにある必要があります。

ベストプラクティスとして、KMS キーの使用は、指定した AWS アカウントまたは異常ディテクターのように制限することをお勧めします。

まず、[get-key-policy](#) コマンドを使用して、KMS キーのデフォルトポリシーを `policy.json` として保存します。

```
aws kms get-key-policy --key-id key-id --policy-name default --output text > ./policy.json
```

テキストエディタで `policy.json` ファイルを開き、以下のいずれかのステートメントから太字のセクションを追加します。既存のステートメントと新しいステートメントをカンマで区切ります。これらのステートメントでは、Condition セクションを使用して AWS KMS キーのセキュリティを強化します。詳細については、「[AWS KMS キーと暗号化コンテキスト](#)」を参照してください。

この例の Condition セクションでは、AWS KMS キーの使用を指定したアカウントのみに制限していますが、これを使用できる異常ディテクターに制限はありません。

JSON

```
{
  "Version": "2012-10-17",
  "Id": "key-default-1",
  "Statement": [
    {
      "Sid": "EnableIAMUserPermissions",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:root"
      },
      "Action": "kms:*",
      "Resource": "*"
    },
  ],
}
```

```

{
  "Sid": "AllowCloudWatchLogsEncryption",
  "Effect": "Allow",
  "Principal": {
    "Service": "logs.us-east-1.amazonaws.com"
  },
  "Action": [
    "kms:Encrypt",
    "kms:Decrypt",
    "kms:GenerateDataKey*"
  ],
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "aws:SourceAccount": "123456789012"
    },
    "StringLike": {
      "kms:EncryptionContext:aws:logs:arn": "arn:aws:logs:us-east-1:123456789012:anomaly-detector:*"
    },
    "ArnLike": {
      "aws:SourceArn": "arn:aws:logs:us-east-1:123456789012:anomaly-detector:*"
    }
  }
},
{
  "Sid": "AllowCloudWatchLogsDescribeKey",
  "Effect": "Allow",
  "Principal": {
    "Service": "logs.us-east-1.amazonaws.com"
  },
  "Action": "kms:DescribeKey",
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "aws:SourceAccount": "123456789012"
    }
  }
},
{
  "Sid": "AllowCloudWatchLogsReEncryption",
  "Effect": "Allow",
  "Principal": {

```

```

        "Service": "logs.us-east-1.amazonaws.com"
    },
    "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*"
    ],
    "Resource": "*",
    "Condition": {
        "StringEquals": {
            "aws:SourceAccount": "123456789012"
        },
        "StringLike": {
            "kms:EncryptionContext:aws-crypto-ec:aws:logs:arn":
"arn:aws:logs:us-east-1:123456789012:anomaly-detector:*"
        },
        "ArnLike": {
            "aws:SourceArn": "arn:aws:logs:us-
east-1:123456789012:anomaly-detector:*"
        }
    }
},
{
    "Sid": "AllowCloudWatchLogsDescribeKeyForReEncryption",
    "Effect": "Allow",
    "Principal": {
        "Service": "logs.us-east-1.amazonaws.com"
    },
    "Action": "kms:DescribeKey",
    "Resource": "*",
    "Condition": {
        "StringEquals": {
            "aws:SourceAccount": "123456789012"
        }
    }
}
]
}

```

最後に、次の [put-key-policy](#) コマンドを使用して更新されたポリシーを追加します。

```
aws kms put-key-policy --key-id key-id --policy-name default --policy file://  
policy.json
```

ステップ 3: KMS キーを異常ディテクターに関連付ける

KMS キーを異常ディテクターに関連付けるには、コンソールで作成するか、AWS CLI または APIs を使用します。

ステップ 4: 異常ディテクターからキーの関連付けを解除する

キーが異常ディテクターに関連付けられていると、キーを更新できません。キーを削除する唯一の方法は、異常ディテクターを削除してから再作成することです。

CloudWatch Logs Live Tail を使用したトラブルシューティング

CloudWatch Logs Live Tail を使用すると、新しいログイベントが取り込まれたときにストリーミングリストを表示して、インシデントのトラブルシューティングをすばやく行うことができます。取り込まれたログをほぼリアルタイムで表示、フィルタリング、強調表示できるため、問題をすばやく検出して解決することができます。指定した用語に基づいてログをフィルタリングしたり、特定の用語を含むログを強調表示したりすることで、探しているものをすぐに見つけることができます。

Live Tail セッションでは、セッションの使用時間ごとに 1 分間隔でコストが発生します。料金の詳細については、「[Amazon CloudWatch 料金表](#)」の [ログ] のタブを参照してください。

Live Tail は、標準ログクラスのロググループでのみサポートされています。ログクラスの詳細については、「[ログクラス](#)」を参照してください。

以下のセクションでは、コンソールおよび AWS CLI で Live Tail を使用方法について説明します。Live Tail セッションをプログラムで開始することもできます。詳細については、「[StartLiveTail](#)」を参照してください。SDK の例については、[AWS「SDK を使用してライブテールセッションを開始する」](#)を参照してください。

AWS Toolkit for Visual Studio Code で Live Tail を使用することもできます。VS Code コマンドパレットからライブテールセッションを開始するには、AWS Toolkit for Visual Studio Code ユーザーガイドの[Amazon CloudWatch Logs Live Tail セクション](#)を参照してください。

Live Tail 機能は、すべての commercial AWS [Regions](#) で使用できます。中国リージョンまたは AWS GovCloud (米国) リージョンでは利用できません。

Note

StartLiveTail API は SDK ホストプレフィックスインジェクションを使用してリクエストをルーティングします。2026 年 4 月 1 日より前にリリースされた SDK バージョンは streaming-logs.*Region*.amazonaws.com、VPC エンドポイントをサポートしていないにルーティングされます。2026 年 4 月 1 日以降にリリースされた SDK バージョンは、VPC エンドポイントをサポートするにルーティング stream-logs.*Region*.amazonaws.com されます。この API の VPC エンドポイントを設定するには、[CloudWatch Logs の VPC エンドポイントの作成](#)を参照してください。

を使用して Live Tail セッションを開始する AWS CLI

`start-live-tail` AWS CLI コマンドは、ターミナル内の 1 つ以上のロググループの Live Tail ストリーミングセッションを開始します。Live Tail セッションは最大 3 時間持続します。1 秒あたり 500 件を超えるログイベントがフィルターと一致する場合、ログイベントがログイベント全体のサンプルとして表示され、リアルタイムのテーリングエクスペリエンスを提供します。`start-live-tail` コマンドの詳細については、「[start-live-tail](#)」を参照してください。

`start-live-tail` は、次の 2 つのモードで使用できます。

- `print-only` – これはデフォルトモードです
- `interactive`

print-only

`print-only` モードでは、ログイベントはターミナルでストリーミングされます。新しいイベントが 1 秒ごとに下部に追加され、Linux での `tail -f` と同様のほぼリアルタイムのテーリングエクスペリエンスが作成されます。

`print-only` モードで Live Tail セッションを開始するには、次のコマンドを入力します。

```
aws logs start-live-tail --log-group-identifiers arn:aws:logs:us-east-1:111111222222:log-group:my-logs
```

`print-only` モードを使用する場合、他の Linux コマンドを使用してパイプ処理を行い、分析機能を向上させることもできます。次の例では、`error` キーワードを使用してログイベントをフィルタリングし、これらのイベントの 2 列目と 4 列目を印刷して、特定の情報を抽出しやすくします。

```
aws logs start-live-tail --log-group-identifiers arn:aws:logs:us-east-1:111111222222:log-group:my-logs --mode print-only | grep "error" | awk '{print $2, $4}'
```

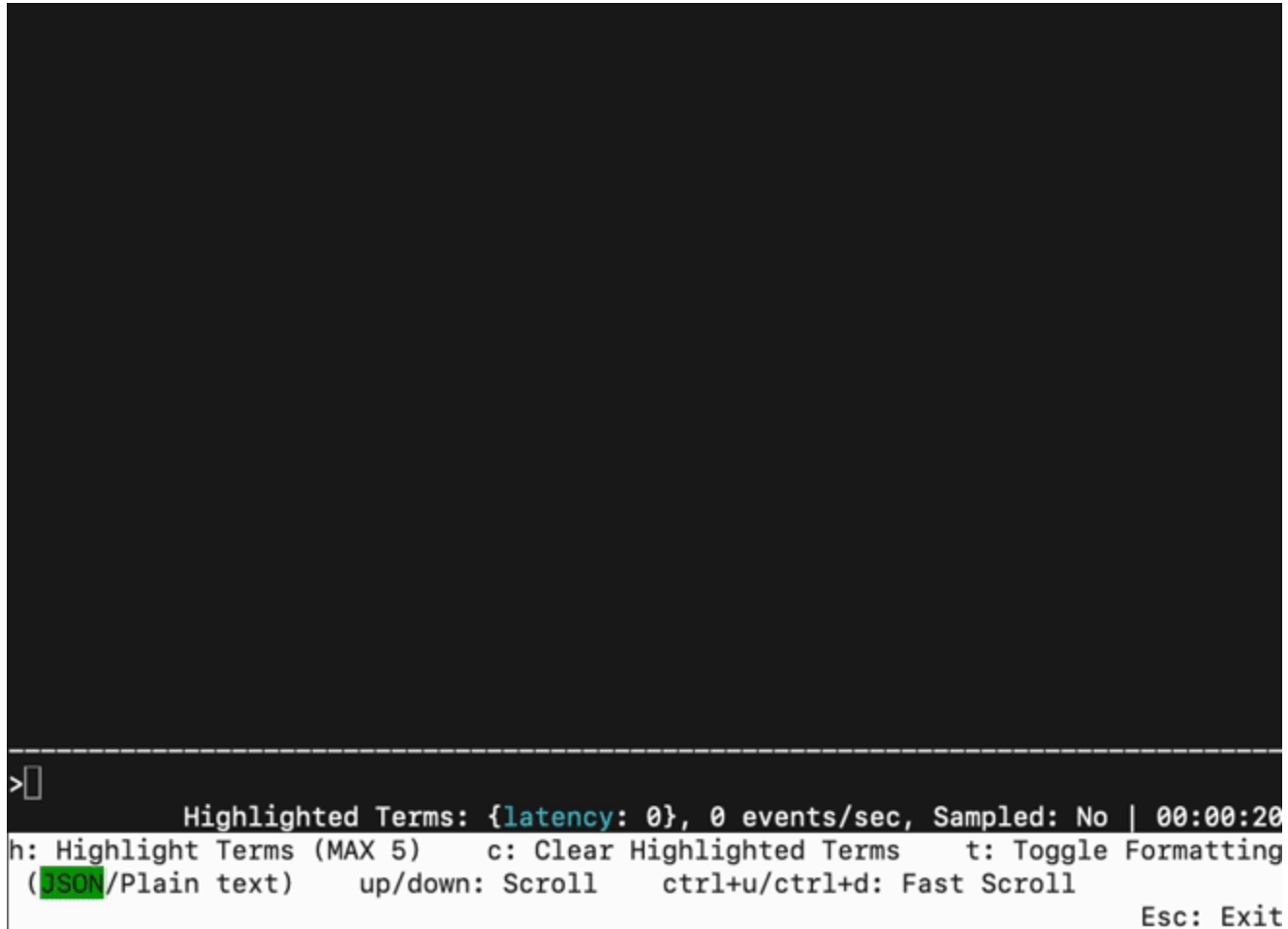
interactive

`interactive` モードでは、用語を強調表示し、出力ログイベントの形式を JSON とプレーンテキストの間で切り替えることができます。Interactive モードでは、セッションの所要時間、セッションのサンプリングの有無、現在強調表示されている用語、およびそれらの発生回数など、Live Tail セッションに関する情報も表示されます。

Interactive モードで Live Tail セッションを開始するには、次のコマンドを入力します。

```
aws logs start-live-tail --log-group-identifiers arn:aws:logs:us-east-1:111111222222:log-group:my-logs --mode interactive
```

Live Tail セッションが開始されます。次のビデオは、セッションの例の一部を示しています。



ストリーミングログで用語を強調表示するには、h を押してから用語を入力します。以下は、用語 latency が強調表示された後の画面を示しています。

強調表示された用語をクリアするには、c を押し、強調表示を停止する用語を表す数値を入力します。

t を押して、JSON とプレーンテキストの間で受信イベントの表示形式を切り替えることができます。このトグル機能はベストエフォートであり、ログイベント形式が互換性がある場合にのみ発生します。

スクロールするには上矢印キーと下矢印キーを使用し、より速くスクロールするには CTRL+u と CTRL+d を使用します。

次の図は、Live Tail セッション中の latency 用語の強調表示を示しています。

```
2024-06-27 12:34:56 [INFO] User login successful
2024-06-27 12:34:56 [ERROR] Disk space exhausted
2024-06-27 12:34:56 [WARN] Unauthorized access attempt
2024-06-27 12:34:56 [WARN] Disk space running low
2024-06-27 12:34:56 [INFO] User logout successful
2024-06-27 12:34:56 [WARN] High latency in network.
2024-06-27 12:34:57 [ERROR] Database connection failed
2024-06-27 12:34:57 [INFO] Database connection established
2024-06-27 12:34:57 [WARN] SSL certificate is about to expire
2024-06-27 12:34:57 [INFO] Scheduled task started
2024-06-27 12:34:57 [WARN] Network latency detected.
2024-06-27 12:34:57 [WARN] Outdated library version
2024-06-27 12:34:58 [INFO] New user registered
2024-06-27 12:34:58 [INFO] Database query executed
2024-06-27 12:34:58 [INFO] File uploaded successfully
2024-06-27 12:34:58 [WARN] Memory usage is high
2024-06-27 12:34:59 [ERROR] Unable to connect to server
[INFO] Connection established with the server
[WARN] SSL certificate is about to expire
[INFO] Scheduled task started
```

Instruction Toolbar
Press h to highlight a term and c to clear

Highlighted Terms: {latency: 2}, 0 events/sec, Sampled: No | 00:08:21
h: Highlight Terms (MAX 5) c: Clear Highlighted Terms t: Toggle Formatting
(JSON/Plain text) up/down: Scroll ctrl+u/ctrl+d: Fast Scroll Esc: Exit

コンソールで Live Tail セッションを開始する

CloudWatch コンソールを使用して Live Tail セッションを開始します。以下の手順では、ナビゲーションペインの [Live tail] を選択して Live Tail セッションを開始する方法について説明します。Live Tail セッションは、ロググループのページ、または CloudWatch Logs Insights のページから開始することもできます。

Live Tail で表示されるロググループの機密データを、データ保護ポリシーを使用してマスクしている場合、Live Tail セッションでは、機密データは常にマスクされて表示されます。ロググループの機密

データのマスキングに関する詳細は、「[機密性の高いログデータをマスキングで保護する](#)」を参照してください。

⚠ Important

ネットワークセキュリティチームがウェブソケットの使用を許可しない場合は、現在 CloudWatch コンソールの Live Tail の部分にアクセスすることはできません。Live Tail は、AWS CLI または APIs で使用できます。詳細については、「[を使用して Live Tail セッションを開始する AWS CLI](#)」および「[StartLiveTail](#)」を参照してください。

Live Tail セッションを開始するには

1. CloudWatch コンソールの <https://console.aws.amazon.com/cloudwatch/> を開いてください。
2. ナビゲーションペインで [ログ]、[Live tail] の順に選択します。
3. [ロググループを選択] で、Live Tail セッションでイベントを表示するロググループを選択します。ロググループは 10 個まで選択できます。
4. (オプション) ロググループを 1 つのみ選択する場合は、ログイベントを表示するログストリームを 1 つ以上選択すれば、Live Tail セッションをさらに絞り込むことができます。それには、[ログストリームを選択] で、ドロップダウンリストからログストリームの名前を選択します。あるいは、[ログストリームを選択] の 2 番目のボックスにログストリーム名のプレフィックスを入力すれば、このプレフィックスに一致する名前を持つすべてのログストリームが選択されます。
5. (オプション) 特定の単語やその他の文字列を含むログイベントのみを表示するときは、その単語または文字列を Add filter patterns に入力します。

例えば、Warning という語を含むログイベントのみを表示するときは、**Warning** と入力します。フィルターフィールドでは、大文字と小文字が区別されます。このフィールドには、次に示す複数の用語とパターン演算子を含めることができます。

- **error 404** は、error と 404 の両方を含むログイベントのみを表示します。
- **?Error ?error** は、Error または error を含むログイベントを表示します。
- **-INFO** は、INFO を含まないログイベントをすべて表示します。
- **{ \$.eventType = "UpdateTrail" }** は、イベントタイプフィールドの値が UpdateTrail である JSON ログイベントをすべて表示します。

正規表現を使用してフィルタリングすることもできます。

- **%ERROR%** は regex を使用して、ERROR キーワードを含むすべてのログイベントを表示します。
- **{ \$.names = %Steve% }** は Steve が "name" プロパティ内にいる場合、regex を使用して JSON ログイベントを表示します。
- **[w1 = %abc%, w2]** は regex を使用して、最初の単語が abc の場合にスペースで区切られたログイベントを表示します。

パターン構文の詳細については、[「フィルターパターン構文」](#)を参照してください。

6. (オプション) 表示されたログイベントの一部を強調表示するには、検索する用語を入力し、[Live Tail] で強調表示します。強調表示する用語は 1 度に 1 つずつ入力します。複数の用語を追加して強調表示すると、用語ごとに異なる色が割り当てられます。指定した用語を含むログイベントの左側に強調表示のインジケータが表示されます。また、メインウィンドウでログイベントを展開してログイベント全体を表示すると、用語自体の下にも表示されます。

フィルタリングと強調表示を併用することで、問題をすばやくトラブルシューティングできます。例えば、イベントをフィルタリングして、Error を含むイベントのみを表示し、さらに、404 を含むイベントを強調表示することもできます。

7. セッションを開始するには、[フィルターを適用] を選択します

一致するログイベントがウィンドウに表示されます。以下の情報も表示されます。

- timer には、Live Tail セッションの実行時間が表示されます。
 - events/sec には、設定したフィルターに一致するログイベントが 1 秒間にいくつ取り込まれたかが表示されます。
 - 多くのイベントがフィルターに一致したために、セッションのスクロールが速くなりすぎることを防ぐため、CloudWatch Logs には一致するイベントの一部しか表示されない場合があります。その場合は、画面に表示されているイベントが一致するイベントの何割であるのかが % で表示されます。
8. イベントのフローを一時停止して、現在表示されている内容を調べるには、イベントウィンドウの任意の場所をクリックします。
 9. セッション中は、以下を使って各ログイベントの詳細を確認できます。

- メインウィンドウにログイベントのテキスト全体を表示するには、そのログイベントの横にある矢印をクリックします。
- サイドウィンドウにログイベントのテキスト全体を表示するには、そのログイベントの横にある虫眼鏡の [+] をクリックします。イベントフローが一時停止し、サイドウィンドウが表示されます。

サイドウィンドウにログイベントのテキストを表示すると、そのテキストをメインウィンドウの他のイベントと比較するのに便利です。

10. Live Tail セッションを停止するには、[停止] をクリックします。
11. セッションを再開するには、[フィルター] パネルを使用してフィルター条件を変更し、[フィルターを適用] をクリックします。次に、[Start (開始)] を選択します。

クロスアカウントクロスリージョンログの一元化

Amazon CloudWatch Logs データー一元化は と連携して AWS Organizations 、クロスアカウントおよびクロスリージョン一元化ルールを使用して、複数のメンバーアカウントから 1 つのデータリポジトリにログデータを収集します。複数のアカウントおよび AWS リージョン から組織内の一元化されたアカウントにログデータを自動的にレプリケートするルールを定義します。この機能により、ログの統合が効率化され、AWS インフラストラクチャ全体の一元的なモニタリング、分析、コンプライアンスが向上します。

CloudWatch Logs データー一元化は、障害耐性を高めるために、送信先アカウント内でのルール設定中にバックアップリージョンを設定する機能など、運用上およびセキュリティ上の要件を満たすための設定の柔軟性を提供します。さらに、元々カスタマーマネージド KMS キーで暗号化されたデータを処理するために、ソースアカウントからコピーされたロググループの暗号化動作を完全に制御できます。

Note

CloudWatch Logs の一元化機能は、一元化ルールの作成後にソースアカウントに到着した新しいログデータのみを処理します。履歴ログデータ (ルールの作成前に存在していたログ) は一元化されません。

データー一元化の概念

CloudWatch Logs データー一元化の使用を開始する前に、以下の概念を理解してください。

一元化ルール

ソースアカウントとリージョンからのログデータを送信先アカウントとリージョンにレプリケートする方法を定義する設定。ルールは、ソース条件と送信先設定を指定します。

ソースアカウント

ログデータの送信元の AWS アカウント。ソースアカウントからのログイベントは、定義した一元化ルールに基づいて送信先アカウントにレプリケートされます。

送信先アカウント

レプリケートされたログデータが保存される送信先 AWS アカウント。このアカウントは、ログの分析とモニタリングのための一元的な場所として機能します。

バックアップリージョン

障害耐性とディザスタリカバリの目的でログデータをレプリケートできる、送信先アカウント内のオプションのセカンダリリージョン。

CloudWatch Logs での暗号化

ロググループのデータは常に CloudWatch Logs で暗号化されます。デフォルトでは、CloudWatch Logs は 256 ビットの Advanced Encryption Standard Galois/Counter Mode (AES-GCM) によるサーバー側の暗号化を使用して、保管中のログデータを暗号化します。別の方法として、この暗号化に AWS Key Management Service を使用することもできます。詳細については、[CloudWatch Logs Encryption ドキュメント](#)を参照してください。

- 一元化中の暗号化の仕組み: CloudWatch Logs の一元化は、取り込み時にログデータをソースアカウントから宛先アカウントにアクティブにコピーします。このプロセス中、データは AWS 所有のサービスキーを使用して転送中に暗号化されたままになります。ソースロググループと宛先ロググループの保管中のデータは、選択した暗号化方法 (カスタマー管理または AWS 所有の KMS キー) を使用して暗号化されます。送信先ロググループでカスタマーマネージド KMS キーを使用している場合は、一元化サービスがアクセスできるように `kms LogsManaged = true` キーに タグを追加します。
- KMS アクセス許可が必要な場合:
 - ソースアカウントでカスタマーマネージド KMS キーを使用している場合、CloudWatch Logs では次のシナリオ例で [KMS アクセス許可](#)が必要です。
 - スループット管理: 一元化スループット制限に達すると、帯域幅が利用可能になるまで、ログデータはカスタマーマネージド KMS キーで暗号化されて一時的に保存されます。
 - データ保護と秘匿化: ソースロググループでデータ保護ポリシーが有効になっている場合、CloudWatch Logs は raw ログデータにアクセスして一元化するための復号アクセス許可を必要とします。

Important

一元化ルールは、AWS Organizations 管理アカウントまたは委任された管理者によって管理されます。カスタマーマネージド KMS で暗号化されたロググループを一元化から除外するには、ルール設定を「KMS AWS キーで暗号化されたロググループを一元化しないでください」に設定します。

ログの一元化の設定

CloudWatch Logs Centralization を設定するには、ソースアカウントのロググループから送信先アカウントのロググループへのログデータのフローを定義する一元化ルールを設定する必要があります。

一元化ルールが有効になり、ログイベントが送信先アカウントにレプリケートされたら、強化されたフィルタリング機能を使用して、一元化されたロググループにメトリクス、サブスクリプション、アカウントフィルターを作成できます。これらのフィルターは、特定のソースアカウントとリージョンからのログイベントをターゲットにすることができ、ソースアカウントとリージョンの情報をメトリクスディメンションとして出力できます。詳細については、「[フィルターを使用したログイベントからのメトリクスの作成](#)」を参照してください。

前提条件

- AWS Organizations をセットアップする必要があり、送信元アカウントと送信先アカウントの両方が組織に属している必要があります。
- ログデータへのアクセスを提供するには、CloudWatch、管理アカウント、送信先アカウントに対して信頼されたアクセスを有効にする必要があります。

Note

コンソールを使用して信頼されたアクセスを有効にすることをお勧めします。これにより、必要なサービスにリンクされたロール (SLR) が自動的に作成されます。信頼されたアクセスが他の方法で有効になっている場合は、サービスにリンクされたロールを個別に作成する必要があります。

送信先ロググループ名をカスタマイズする

一元化ルールを作成するときに、属性を使用して送信先ロググループ名の構成方法をカスタマイズできます。これらの属性は、ロググループの作成時に自動的に実際の値に置き換えられるため、送信先アカウントでログを階層的に整理できます。デフォルトでは、`${source.logGroup}` 属性のみが使用され、送信先アカウントの同じ名前のすべてのロググループがマージされます。変数を解決できない場合は、階層内の親変数から値を継承します。

使用可能な属性

送信先ロググループ名パターンでは、次の属性を使用できます。

送信先ロググループ名の属性

属性	説明
<code>\${source.accountId}</code>	ログが発生した AWS アカウント ID。
<code>\${source.region}</code>	ログが発生した AWS リージョン。
<code>\${source.logGroup}</code>	ソースアカウントの元のロググループ名。
<code>\${source.org.id}</code>	ソースアカウントの AWS Organizations ID。
<code>\${source.org.ouId}</code>	ソースアカウントの組織単位 ID
<code>\${source.org.rootId}</code>	組織のルート ID
<code>\${source.org.path}</code>	アカウントからルートへの完全な組織パス

例

元のロググループ構造を保持する

パターン: `/centralized/${source.accountId}${source.logGroup}`

結果: `/centralized/123456789012/aws/lambda/my-function`

アカウントとリージョン別に整理する

パターン: `/centralized/${source.accountId}/${source.region}`

結果: `/centralized/123456789012/us-east-1`

組織構造別に整理する

パターン: `/logs/${source.org.id}/${source.org.ouId}/${source.accountId}`

結果: `/logs/o-abc123/ou-xyz-12345678/123456789012`

シンプルなフラット構造

パターン: `/centralized-logs`

結果: `/centralized-logs`

ベストプラクティス

- ソースアカウント ID を含めると、どのアカウントログの出所かを簡単に識別できます。
- 複数のリージョンから一元化する場合は、ソースリージョンを含めます。
- 構造の送信先ロググループ名は 512 文字未満にする必要があります。CloudWatch Logs では、ロググループ名の最大長は 512 文字です。

一元化ルールを作成

次の手順を使用して、ソースアカウントから送信先アカウントにログデータをレプリケートする一元化ルールを作成します。

一元化ルールを作成するには

1. 組織の管理アカウントまたは委任管理者アカウントの CloudWatch コンソールに移動します。
2. [設定] を選択します。
3. [組織] タブに移動します。
4. [ルールの設定] を選択します。
5. 次のフィールドを設定してソースの詳細を指定し、[次へ] を選択します。
 - a. 一元化ルール名: 一元化ルールの一意の名前を入力します。
 - b. ソースアカウント: テレメトリデータを一元化するアカウントを選択するソース選択基準を定義します。選択基準には以下を含むことができます。
 - 組織のメンバーアカウントのリスト
 - 組織の組織単位のリスト
 - 組織全体

選択基準は 2 つのモードで指定できます。

- ビルダー: ソース選択基準を生成するためのクリックベースのエクスペリエンス
- エディタ: ソース選択基準を提供する自由形式のテキストボックス

ソース選択基準でサポートされている構文:

- サポートされているキー: OrganizationId | OrganizationUnitId | AccountId | *

- サポートされている演算子: = | IN | OR
 - c. ソースリージョン: 一元化するテレメトリデータを検索するリージョンのリストを選択します。
6. 次のフィールドを設定して送信先の詳細を指定し、[次へ] を選択します。
- a. 送信先アカウント: テレメトリデータの中央送信先として機能する組織内のアカウントを選択します。
 - b. 送信先リージョン: 一元化されたテレメトリデータのコピーを保存するプライマリリージョンを選択します。
 - c. バックアップリージョン: 必要に応じて、一元化されたテレメトリデータの 2 番目のコピーを保存するリージョンを選択します。
7. 次のフィールドを設定してテレメトリデータを指定し、[次へ] を選択します。
- a. ロググループ: 次のオプションのいずれかを選択します。
 - すべてのロググループ: ソースアカウントのすべてのロググループからのログを一元化します。
 - ロググループのフィルタリング: ソースアカウントのロググループのサブセットからのログを一元化し、選択基準を一致させます。選択基準は 2 つのモードで指定できます。
 - ビルダー: 選択条件を生成するためのクリックベースのエクスペリエンス
 - エディタ: 選択基準を提供する自由形式のテキストボックス
- ログのフィルタリングに使用できる選択基準は 2 つあります。
- ロググループの選択基準: 一元化するソースロググループを指定する選択基準。
 - サポートされているキー: LogGroupName | *
 - サポートされている演算子: = | != | IN | NOT IN | AND | OR | LIKE | NOT LIKE
 - データソース選択基準: 一元化するデータソースを指定する選択基準。
 - サポートされているキー: DataSourceName | DataSourceType
 - サポートされている演算子: = | != | IN | NOT IN | AND | OR | LIKE | NOT LIKE
- ロググループ選択基準とデータソース選択基準の両方を指定する場合、ログイベントを一元化するには両方の基準と一致する必要があります。
- b. KMS 暗号化ロググループ

⚠ Important

CloudWatch 一元化ルールは、その一元化ルールで指定された KMS キーで CloudWatch Logs によるその使用が許可されていない場合、ソースアカウントから送信先ロググループにログを配信できません。送信先ロググループでカスタマーマネージド KMS キーを使用している場合は、タグ `LogsManaged = true` を kms キーに追加します。詳細については、「[ステップ 2: KMS キーでアクセス許可を設定する](#)」を参照してください。

以下のオプションのいずれかを選択してください。

- 送信先固有のカスタマーマネージド KMS キーを使用してカスタマーマネージド KMS キーで暗号化されたソースロググループを一元化する: カスタマーマネージド KMS キーで暗号化されたソースロググループからのログイベントを送信先アカウントのカスタマーマネージド KMS キーで暗号化された送信先ロググループに一元化します。

この設定を選択すると、以下も設定する必要があります。

- 送信先暗号化キー ARN: 新しく作成された送信先ロググループに関連付ける、送信先アカウントとプライマリ送信先リージョンのカスタマーマネージド KMS キーの ARN。
- バックアップ先暗号化キー ARN (バックアップリージョンが選択されている場合): 新しく作成された送信先ロググループに関連付ける、送信先アカウントとバックアップ先リージョンのカスタマーマネージド KMS キーの ARN。
- 暗号化されていない送信先ロググループへの一元化をスキップする (オプション): カスタマーマネージド KMS キーなしでロググループが既に存在する場合、CloudWatch は暗号化を更新できません。このオプションを選択すると、カスタマーマネージド KMS キーで暗号化されたソースロググループから、カスタマーマネージド KMS キーに関連付けられていない送信先ロググループへのログイベントの一元化がスキップされます。
- AWS 所有の KMS キーを使用して送信先アカウントのカスタマーマネージド KMS キーで暗号化されたロググループを一元化する: カスタマーマネージド KMS キーで暗号化されたソースロググループからのログイベントを、AWS 所有の KMS キーを使用して暗号化された新しく作成された送信先ロググループに一元化します。
- カスタマーマネージド KMS キーで暗号化されたロググループを一元化しない: カスタマーマネージド KMS キーで暗号化されたソースロググループからのログイベントの一元化をスキップします。

8. 一元化ルールを確認し、オプションで直前の編集を行い、一元化ポリシーの作成を選択します。

一元化ルールの変更

既存の一元化ルールを変更するには、次の手順に従います。

一元化ルールを変更するには

1. 組織の管理アカウントまたは委任管理者アカウントの CloudWatch コンソールに移動します。
2. [設定] を選択します。
3. [組織] タブに移動します。
4. [ルールの管理] を選択します。
5. ルールを選択してから、[編集] を選択します。
6. 必要に応じてルール設定を更新し、[次へ] を選択して各ステップに進みます。
7. ステップ 4「確認して設定」で、[一元化ポリシーの更新] を選択します。

一元化ルールの表示

既存の一元化ルールの詳細を表示するには、次の手順に従います。

一元化ルールを表示するには

1. 組織の管理アカウントまたは委任管理者アカウントの CloudWatch コンソールに移動します。
2. [設定] を選択します。
3. [組織] タブに移動します。
4. [ルールの管理] を選択します。
5. 既存のすべての一元化ルールのリストを表示し、特定のルール名を選択して詳細を表示します。

一元化ルールの削除

既存の一元化ルールを削除するには、次の手順に従います。

一元化ルールを削除するには

1. 組織の管理アカウントまたは委任管理者アカウントの CloudWatch コンソールに移動します。

2. [設定] を選択します。
3. [組織] タブに移動します。
4. [ルール管理] を選択します。
5. 削除するルールを選択し、[削除] を選択します。
6. 削除を確認し、[Delete] (削除) を選択します。

一元化ルールのモニタリングとトラブルシューティング

CloudWatch メトリクス、CloudWatch Logs コンソール、および AWS CloudTrail ログを使用して、一元化ルールのステータスとパフォーマンスをモニタリングできます。これにより、ログデータが正常にレプリケートされ、一元化の設定に伴う問題を特定していることを確認しやすくなります。

CloudWatch Logs は以下を提供します。

1. 一元化ルールあたりのルールの正常性
 - a. [設定] を選択します。
 - b. [組織] タブに移動します。
 - c. [ルール管理] を選択します。
2. を使用して API コールをログに記録します。AWS CloudTrail
3. CloudWatch は、レプリケートされたログイベント、エラー、スロットリングなど、一元化のメトリクスも発行します。これらのメトリクスとそのディメンションの詳細については、「」を参照してください [一元化メトリクスとディメンション](#)。

一元化ルールのヘルスステータス

個々の一元化ルールには、正しく動作しているか否かを示すヘルスステータスがあります。ルールのヘルスは、コンソールを使用するか、API を使用してプログラムで確認できます。

ルールのヘルスステータスは次のとおりです。

- HEALTHY: ルールが正常に動作し、ログデータが設定どおりにレプリケートされている
- UNHEALTHY: ルールで問題が発生し、データが正しくレプリケートされていない可能性がある
- PROVISIONING: 組織の一元化がセットアップ中である

ルールが UNHEALTHY としてマークされると、FailureReason フィールドには、対処する必要がある特定の問題に関する詳細が表示されます。

による一元化 API コールのモニタリング AWS CloudTrail

AWS CloudTrail は、一元化サービスに対して行われた API コールをログに記録します。これにより、設定の変更を追跡し、のメンバーであるアカウントの問題をトラブルシューティングできます AWS Organizations。

一元化の主な CloudTrail イベントには以下が含まれます。

- CreateCentralizationRuleForOrganization: 新しい一元化ルールが作成された場合
- UpdateCentralizationRuleForOrganization: 既存のルールが変更された場合
- DeleteCentralizationRuleForOrganization: ルールが削除された場合
- GetCentralizationRuleForOrganization: ルールの詳細が取得された場合
- ListCentralizationRulesForOrganization: ルールがリストされた場合

CloudTrail ログを使用して、一元化設定の変更を監査し、パフォーマンスの問題やレプリケーションの失敗にそれらに関連付けることができます。

モニタリングの推奨事項

一元化が正しく機能するように、CloudWatch メトリクスに供給した主要な一元化メトリクスに CloudWatch アラームを設定することをお勧めします。このプロアクティブモニタリングは、問題を早期に検出し、組織全体で信頼性の高いログの一元化を維持するのに役立ちます。

モニタリングする主なメトリクスは次のとおりです。

- IncomingCopiedBytes: 送信先アカウントに正常にレプリケートされるログデータの量を監視します。このメトリクスが突然低下または欠落した場合、一元化の問題を示している可能性があります。
- CentralizationError: 一元化プロセスでエラーが発生した場合にアラームを設定して、問題をすばやく特定して解決します。
- CentralizationThrottled: ログレプリケーションのパフォーマンスに影響を与える可能性のあるスロットリングイベントをモニタリングします。

利用可能な一元化メトリクスとそのディメンションの完全なリストについては、「」を参照してください [一元化メトリクスとディメンション](#)。

ログが想定どおりに一元化されていない場合は、ログの一元化を妨げる可能性がある以下の一般的なシナリオを確認してください。

履歴ログデータ

CloudWatch Logs の一元化機能は、一元化ルールの作成後にソースアカウントに到着した新しいログデータのみを処理します。履歴ログデータ (ルールの作成前に存在していたログ) は一元化されません。

KMS キーのアクセス許可

一元化ルールで指定された KMS キーが CloudWatch Logs に使用を許可していない場合、一元化ルールはソースアカウントから送信先ロググループにログを配信できません。KMS キーポリシーが CloudWatch Logs に必要なアクセス許可を付与していることを確認します。詳細については、「[ステップ 2: KMS キーでアクセス許可を設定する](#)」を参照してください。

カスタマーマネージド KMS キー設定

ルールの作成中にカスタマーマネージド KMS キーで暗号化されたロググループを一元化しないことを選択した場合、カスタマーマネージド KMS キーで暗号化されたソースロググループからのログイベントはスキップされ、一元化されません。

送信先暗号化の不一致

送信先ロググループが、一元化ルールで指定されているものとは異なる KMS 暗号化設定で既に存在しており、競合解決が SKIP に設定されている場合、レコードは削除され、DestinationEncryptionMismatchエラーが発生します。たとえば、これは送信先にデフォルトの暗号化があるが、ルールでカスタマーマネージド KMS キーが指定されている場合に発生します。

信頼されたアクセスが有効になっていない

ログデータへのアクセスを提供するには、で CloudWatch AWS Organizations に対して信頼されたアクセスを有効にする必要があります。

ソース選択基準

一元化ルールのソース選択基準が正しく設定されていることを確認します。

- アカウントとリージョン: ログが発生するソースアカウントとリージョンがルールに含まれていることを確認します。ルールで指定されていないアカウントまたはリージョンからのロググループは一元化されません。

- ロググループフィルター: ロググループフィルターを設定した場合、指定された条件に一致するロググループのみが一元化されます。ロググループの選択基準に、一元化が予想されるロググループが含まれていることを確認します。
- 組織のメンバーシップ: 送信元アカウントと送信先アカウントの両方が同じ AWS Organizations 組織に属している必要があります。組織外のアカウントは、一元化に参加できません。

ロググループのクォータ制限に達しました

送信先アカウントがロググループのクォータ制限に達した場合、一元化のために新しいロググループを作成することはできません。送信先アカウントに、すべての送信元アカウントの集中ロググループに対応するのに十分なクォータがあることを確認します。必要に応じて、クォータの引き上げをリクエストできます。

ログストリーム名の長さの制限を超えました

ログストリーム名には最大長の制限があります。一元化がログストリームを送信先アカウントにレプリケートすると、ログストリーム名にサフィックスが追加されます。結果のログストリーム名が最大許容長を超えると、レコードは削除され、エラーInvalidLogStreamがお客様のアカウントに出力されます。

ルールのヘルスステータス

コンソールまたは GetCentralizationRuleForOrganization API を使用して、一元化ルールのヘルスステータスを確認します。ルールが UNHEALTHY としてマークされている場合は、問題に関する特定の詳細について FailureReason フィールドを確認してください。

一元化の問題を診断するには、コンソールで一元化ルールのヘルスステータスを確認し、CloudWatch メトリクスでエラーとスロットリングをチェックし、API コールの失敗の AWS CloudTrail ログを調べます。一元化メトリクスの詳細については、「」を参照してください [一元化メトリクスとディメンション](#)。

ロググループとログストリームの操作

ログストリームは、同じソースを共有する一連のログイベントです。CloudWatch Logs でのログの各ソースで各ログストリームが構成されます。

ロググループは、保持、モニタリング、アクセス制御について同じ設定を共有するログストリームのグループです。ロググループを定義して、各グループに入れるストリームを指定できます。1つのロググループに属することができるログストリーミングの数に制限はありません。

複数のアカウントとリージョンのログデータを統合する必要がある組織の場合、CloudWatch Logs Centralization を使用して、ロググループを中央アカウントに自動的にレプリケートできます。詳細については、「[クロスアカウントクロスリージョンログの一元化](#)」を参照してください。

このセクションの手順を使用して、ロググループおよびログストリームを処理できます。

CloudWatch Logs にロググループを作成します。

Amazon CloudWatch Logs ユーザーガイドで前のセクションに記載されている手順を使用して、CloudWatch Logs エージェントを Amazon EC2 インスタンスにインストールすると、そのプロセスの一環としてロググループが作成されます。CloudWatch コンソールで、直接ロググループを作成することもできます。

ロググループを作成するには

1. CloudWatch コンソールの <https://console.aws.amazon.com/cloudwatch/> を開いてください。
2. ナビゲーションペインで、ログ管理を選択します。
3. [Actions (アクション)] を選択し、[Create log group (ロググループの作成)] を選択します。
4. ロググループの名前を入力し、[Create log group (ロググループの作成)] を選択します。

Tip

ロググループ、ダッシュボード、アラームは、ナビゲーションペインの [お気に入りと最近使ったコンテンツ] メニューからお気に入りに登録できます。[最近アクセスしたサービス] 列で、お気に入りに登録するロググループにカーソルを合わせ、その横にある星の記号を選択します。

ロググループへのログの送信

CloudWatch Logs は、複数の AWS サービスからログイベントを自動的に受信します。次のいずれかの方法を使用して、CloudWatch Logs に他のログイベントを送信することもできます。

- CloudWatch エージェント — 統合された CloudWatch エージェントは、メトリックとログの両方を CloudWatch Logs に送信できます。CloudWatch エージェントのインストールと使用については、「Amazon CloudWatch ユーザーガイド」の「[CloudWatch エージェントを使用して Amazon EC2 インスタンスとオンプレミスサーバーからメトリックとログを収集する](#)」を参照してください。
- AWS CLI - [put-log-events](#) は、ログイベントのバッチを CloudWatch Logs にアップロードします。
- プログラムによる - [PutLogEvents](#) API を使用して、ログイベントのバッチをプログラムにより CloudWatch Logs にアップロードできます。

CloudWatch Logs に送信されたログデータを表示する

CloudWatch Logs エージェントによって CloudWatch Logs に送信されたログデータは、ストリームごとに表示およびスクロールできます。表示するログデータの時間範囲を指定できます。

ログデータを表示するには

1. CloudWatch コンソールの <https://console.aws.amazon.com/cloudwatch/> を開いてください。
2. ナビゲーションペインで、ログ管理を選択します。
3. [Log Groups] で、ストリームを表示するロググループを選択します。
4. ロググループのリストで、表示するロググループの名前を選択します。
5. ログストリームのリストで、表示するログストリームの名前を選択します。
6. ログデータの表示方法を変更するには、次のいずれかを実行します。
 - 1つのログイベントを展開するには、そのログイベントの横にある矢印を選択します。
 - すべてのログイベントを展開してプレーンテキストとして表示するには、ログイベントのリストの上で、[Text] を選択します。
 - ログイベントをフィルターするには、検索フィールドに目的の検索フィルターを入力します。詳細については、「[フィルターを使用したログイベントからのメトリクスの作成](#)」を参照してください。

- 指定した日時範囲のログデータを表示するには、検索フィルターの隣の日付と時刻の横にある矢印を選択します。日付と時間の範囲を指定するには、[Absolute (絶対)] を選択します。事前定義された分、時間、日数、または週数を選択するには、[Relative (相対)] を選択します。UTC とローカルタイムゾーンを切り替えることもできます。

フィルターパターンを使用してログデータを検索する

ログデータは、[メトリクスフィルター](#)、[サブスクリプションフィルター](#)、[フィルターログイベント](#)、および[ライブテールのフィルターパターン構文](#)を使用して検索できます。ロググループ内のすべてのログストリームを検索するか、を使用して特定のログストリームを検索 AWS CLI することもできます。各検索を実行すると、最大で、見つかったデータの最初のページと、データの次のページを取得するか検索を続行するためのトークンが返されます。結果が返されない場合は、検索を続行できません。

クエリを実行する時間範囲を設定し、検索範囲を制限することができます。広い範囲から開始して関心のあるログ行が収まっている場所を確認した後、時間範囲を短縮し、関心のある時間範囲のログまでビューを絞り込みます。

ログから抽出したメトリクスを直接、対応するログに移動することもできます。

CloudWatch のクロスアカウントオブザーバビリティでモニタリングアカウントとして設定されたアカウントにサインインしている場合、このモニタリングアカウントにリンクされているソースアカウントのログイベントを検索してフィルタリングできます。詳細については、「[CloudWatch のクロスアカウントオブザーバビリティ](#)」を参照してください。

コンソールを使用してログエントリを検索する

コンソールを使用して、指定した基準を満たすログエントリを検索することができます。

コンソールを使用してログを検索するには

- CloudWatch コンソールの <https://console.aws.amazon.com/cloudwatch/> を開いてください。
- ナビゲーションペインで、ログ管理を選択します。
- [ロググループ] で、検索するログストリームを含むロググループの名前を選択します。
- [ログストリーム] で、検索するログストリームの名前を選択します。
- [Log Events (ログイベント)] で、使用するフィルター構文を入力します。

コンソールを使用してすべてのログエントリで時間範囲を検索するには

1. CloudWatch コンソールの <https://console.aws.amazon.com/cloudwatch/> を開いてください。
2. ナビゲーションペインで、ログ管理を選択します。
3. [ロググループ] で、検索するログストリームを含むロググループの名前を選択します。
4. [ロググループの検索] を選択します。
5. [Log Events (ログイベント)] で、日付と時刻の範囲を選択し、フィルター構文を入力します。

を使用してログエントリを検索する AWS CLI

を使用して、指定された条件を満たすログエントリを検索できます AWS CLI。

を使用してログエントリを検索するには AWS CLI

コマンドプロンプトで、次の [filter-log-events](#) コマンドを実行します。結果を指定したフィルターパターンに限定するには `--filter-pattern` を使用し、結果を指定したログストリームに限定するには `--log-stream-names` を使用します。

```
aws logs filter-log-events --log-group-name my-group [--log-stream-names LIST_OF_STREAMS_TO_SEARCH] [--filter-pattern VALID_METRIC_FILTER_PATTERN]
```

を使用して特定の時間範囲のログエントリを検索するには AWS CLI

コマンドプロンプトで、次の [filter-log-events](#) コマンドを実行します。

```
aws logs filter-log-events --log-group-name my-group [--log-stream-names LIST_OF_STREAMS_TO_SEARCH] [--start-time 1482197400000] [--end-time 1482217558365] [--filter-pattern VALID_METRIC_FILTER_PATTERN]
```

メトリクスからログへのピボット

コンソールの他の部分から、特定のログエントリに移動することができます。

ダッシュボードウィジェットからログに移動するには

1. CloudWatch コンソールの <https://console.aws.amazon.com/cloudwatch/> を開いてください。
2. ナビゲーションペインで、ダッシュボードを選択します。

3. ダッシュボードを選択します。
4. ウィジェットで [View logs] アイコンを選択し、[View logs in this time range] を選択します。メトリクスフィルターが複数ある場合は、リストから 1 つ選択します。メトリクスフィルターをリストに表示しきれない場合は、[More metric filters] を選択し、メトリクスフィルターを選択するか検索します。

メトリクスからログに移動するには

1. CloudWatch コンソールの <https://console.aws.amazon.com/cloudwatch/> を開いてください。
2. ナビゲーションペインで [Metrics (メトリクス)] を選択してください。
3. [All metrics] タブの検索フィールドに、メトリクスの名前を入力して Enter キーを押します。
4. 検索結果から 1 つ以上のメトリクスを選択します。
5. [Actions]、[View logs] の順に選択します。メトリクスフィルターが複数ある場合は、リストから 1 つ選択します。メトリクスフィルターをリストに表示しきれない場合は、[More metric filters] を選択し、メトリクスフィルターを選択するか検索します。

トラブルシューティング

[Search takes too long to complete]

ログデータが多い場合、検索の完了に時間がかかる場合があります。検索の速度を上げるには、次を実行します：

- を使用している場合は AWS CLI、検索を目的のログストリームのみで制限できます。例えば、ロググループに 1000 個のログストリームがあり、関連性があることがわかっているログストリームを 3 つだけ表示する場合は、AWS CLI を使用して、ロググループ内のこれら 3 つのログストリームのみで検索を制限できます。
- 時間範囲を短く、細かくして検索対象のデータ量を減らし、クエリの速度を上げます。

Change log data retention in CloudWatch Logs

デフォルトでは、ログデータは CloudWatch Logs に無期限に保存されます。ただし、ロググループにログデータを保存する期間を設定できます。現在の保持設定より古いデータはすべて削除されます。各ロググループのログの保持期間は、いつでも変更できます。

Note

CloudWatch Logs は、保持設定に達したログイベントをすぐに削除しません。通常、ログイベントが削除されるまでに最大 72 時間かかりますが、まれにそれ以上かかる場合もあります。

つまり、有効期限を過ぎているが実際には削除されていないログイベントが含まれている場合に、ロググループを長い保持設定に変更すると、新しい保持期間に達してからこれらのログイベントが削除されるまでに最大 72 時間かかります。ログデータを完全に削除するには、前の保持期間が終了してから 72 時間が経過するか、古いログイベントが削除されることを確認するまで、ロググループを低い保持設定にしておきます。

ログイベントが保持設定に達すると、削除対象としてマークされます。削除対象としてマークされた後は、後で実際に削除されない場合でも、アーカイブストレージのコストが追加されることはありません。また、削除対象としてマークされたこれらのログイベントは、API を使用して `storedBytes` の値を取得し、ロググループが保存しているバイト数を確認する場合にも含まれません。

ログの保持設定を変更するには

1. CloudWatch コンソールの <https://console.aws.amazon.com/cloudwatch/> を開いてください。
2. ナビゲーションペインで、[ログ]、[ロググループ] の順に選択します。
3. 更新するロググループを見つけます。
4. そのロググループの [保持] 列で、現在の保持設定 (例: [失効しない]) を選択します。
5. [保持期間の設定] の [次の期間経過後にイベントを失効] で、ログ保持値を選択し、[保存] を選択します。

ロググループの削除からの保護

オプションで削除保護を有効にして、重要なロググループが誤って削除されるのを防ぐことができます。削除保護の詳細については、「」を参照してください [ロググループの削除からの保護](#)。

ロググループの削除からの保護

削除保護の有効化

新しいロググループを作成するとき、または既存のロググループで削除保護を有効にすることができます。ロググループの作成時に、「削除保護を有効にする」を選択するか、パラメータを渡します `--deletion-protection-enabled`。デフォルトでは、削除保護は有効になっていません。

既存のロググループの削除保護を有効または無効にするには (コンソール)

1. CloudWatch コンソールの <https://console.aws.amazon.com/cloudwatch/> を開いてください。
2. ナビゲーションペインで、ログ管理を選択します。
3. 保護するロググループを選択します。
4. アクション、削除保護の編集を選択します。
5. ダイアログボックスで、変更を確認して送信します。

を使用している場合 AWS CLI、既存のロググループで削除保護を有効にするには:

```
aws logs put-log-group-deletion-protection \  
--log-group-identifier "/my-application/logs" \  
--deletion-protection-enabled
```

既存のロググループの削除保護を削除するには:

```
aws logs put-log-group-deletion-protection \  
--log-group-identifier "/my-application/logs" \  
--no-deletion-protection-enabled
```

エラー処理

削除保護が有効になっているロググループを削除しようとする、と、「削除保護が有効になっているロググループは削除できませんValidationException」というメッセージが表示されます。削除保護を最初に無効にします。」

Amazon CloudWatch Logs のロググループにタグを付ける

Amazon CloudWatch Logs で作成したロググループに、独自のメタデータをタグ形式で割り当てることができます。タグは、ロググループに対して定義するキーと値のペアです。タグの使用は、AWS リソースを管理し、請求データを含むデータを整理するためのシンプルで強力な方法です。

Note

ロググループや送信先などの CloudWatch Logs のリソースへのアクセスを制御するには、タグを使用します。ロググループとログストリームの間には階層的な関係があるため、ログストリームへのアクセスはロググループレベルで制御されます。リソースへのアクセスを制御するタグの使用の詳細については、[タグを使用した Amazon Web Services のリソースへのアクセスの制御](#)を参照してください。

内容

- [タグの基本](#)
- [タグ付けを使用したコストの追跡](#)
- [タグの制限](#)
- [を使用したロググループのタグ付け AWS CLI](#)
- [CloudWatch Logs API を使用したロググループのタグ付け](#)

タグの基本

AWS CloudFormation AWS CLIまたは CloudWatch Logs API を使用して、次のタスクを完了します。

- ロググループの作成時にタグを追加する
- 既存のロググループにタグを追加する
- ロググループのタグを一覧表示する
- ロググループからタグを削除する

タグを使用すると、ロググループを分類できます。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。たとえば、所有者と、関連するアプリケーションに基づいてロググループを追跡するのに役立つタグのセットを定義できます。次にいくつかのタグの例を示します。

- プロジェクト: プロジェクト名
- 所有者: 名前
- 目的: 負荷テスト
- アプリケーション: アプリケーション名
- 環境: 本稼働

タグ付けを使用したコストの追跡

タグを使用して、AWS コストを分類および追跡できます。ロググループを含む 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

タグには次の制限があります。

基本制限

- タグの最大数はロググループごとに 50 です。
- タグのキーと値では、大文字と小文字が区別されます。
- 削除されたロググループ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aws:このプレフィックスで始まるタグ AWS AWS を作成するため、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

- タグ値の長さは 0 ~ 255 文字 (Unicode) にする必要があります。
- 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 (_ . / = + - @)。

を使用したロググループのタグ付け AWS CLI

AWS CLI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。

[create-log-group](#)

ロググループを作成します。ロググループの作成時に、オプションでタグを追加できます。

[tag-resource](#)

指定された CloudWatch Logs リソースに 1 つまたは複数のタグ (キーと値のペア) を割り当てます。

[list-tags-for-resource](#)

CloudWatch Logs リソースに関連付けられているタグを表示します。

[untag-resource](#)

指定された CloudWatch Logs リソースから 1 つまたは複数のタグを削除します。

CloudWatch Logs API を使用したロググループのタグ付け

CloudWatch Logs 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。

[CreateLogGroup](#)

ロググループを作成します。ロググループの作成時に、オプションでタグを追加できます。

[TagResource](#)

指定された CloudWatch Logs リソースに 1 つまたは複数のタグ (キーと値のペア) を割り当てます。

[ListTagsForResource](#)

CloudWatch Logs リソースに関連付けられているタグを表示します。

[UntagResource](#)

指定された CloudWatch Logs リソースから 1 つまたは複数のタグを削除します。

を使用して CloudWatch Logs のログデータを暗号化する AWS Key Management Service

ロググループのデータは常に CloudWatch Logs で暗号化されます。デフォルトでは、CloudWatch Logs は 256 ビットの Advanced Encryption Standard Galois/Counter Mode (AES-GCM) によるサーバー側の暗号化を使用して、保管中のログデータを暗号化します。別の方法として、この暗号化には AWS Key Management Service を使用できます。その場合、暗号化は AWS KMS キーを使用して行われます。を使用した暗号化 AWS KMS は、ロググループの作成時または作成後に、KMS キーをロググループに関連付けることで、ロググループレベルで有効になります。

Important

CloudWatch Logs は、`kms:EncryptionContext:aws:logs:arn` をキーとして使用し、ロググループの ARN をそのキーの値として使用して、暗号化コンテキストをサポートするようになりました。KMS キーで暗号化したロググループがあり、そのキーが 1 つのアカウントとロググループで使用されるように制限する場合は、新しい KMS キーを割り当て、そのための条件を IAM ポリシーに含める必要があります。詳細については、「[AWS KMS キーと暗号化コンテキスト](#)」を参照してください。

Important

CloudWatch Logs がサポートするようになりました。`kms:ViaService` これにより、ログはユーザーに代わって AWS KMS 呼び出しを行うことができます。これを、キーポリシーまたは IAM で CloudWatch Logs を呼び出すロールに追加する必要があります。詳細については、「[kms:ViaService](#)」を参照してください。

KMS キーをロググループと関連付けると、ロググループの新たに取り込まれたすべてのデータは、このキーを使用して暗号化されます。このデータは、保持期間を通じて暗号化形式で保存されます。CloudWatch Logs は、リクエストがあればいつでもこのデータを復号化します。暗号化されたデータがリクエストされた場合、CloudWatch Logs に KMS キーのアクセス許可が必要です。

後でロググループから KMS キーの関連付けを解除すると、CloudWatch Logs は CloudWatch Logs デフォルトの暗号化方法を使用して、新たに取り込まれたデータを暗号化します。以前に取り込まれたデータのうち KMS キーで暗号化されたものは、すべて KMS キーで暗号化されたままになります。CloudWatch Logs は引き続きキーを参照できるため、KMS キーの関連付けが解除された後も、CloudWatch Logs はそのデータを返すことができます。ただし、キーを後で無効にすると、CloudWatch Logs はそのキーで暗号化されたログを読み取ることができなくなります。

Important

CloudWatch Logs は、対称 KMS キーのみをサポートします。非対称キーを使用してロググループのデータを暗号化しないでください。詳細については、「[対称キーと非対称キーの使用](#)」を参照してください。

制限

- 以下の手順を実行するには、`kms:CreateKey`、`kms:GetKeyPolicy`、および `kms:PutKeyPolicy` アクセス許可が必要です。
- キーとロググループを関連付けまたは関連付け解除すると、オペレーションが有効になるまで最大 5 分かかることがあります。
- 関連付けられたキーへの CloudWatch Logs のアクセスを取り消したり、関連付けられた KMS キーを削除した場合、CloudWatch Logs 内の暗号化されたデータを取得できなくなります。
- CloudWatch コンソールを使用して KMS キーを既存のロググループと関連付けることはできません。

ステップ 1: AWS KMS キーを作成する

KMS キーを作成するには、次の [create-key](#) コマンドを使用します。

```
aws kms create-key
```

出力には、キーのキー ID と Amazon リソースネーム (ARN) が含まれます。出力例を次に示します。

```
{
  "KeyMetadata": {
    "Origin": "AWS_KMS",
    "KeyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
    "Description": "",
    "KeyManager": "CUSTOMER",
    "Enabled": true,
    "CustomerMasterKeySpec": "SYMMETRIC_DEFAULT",
    "KeyUsage": "ENCRYPT_DECRYPT",
    "KeyState": "Enabled",
    "CreationDate": 1478910250.94,
    "Arn": "arn:aws:kms:us-west-2:123456789012:key/6f815f63-e628-448c-8251-e40cb0d29f59",
    "AWSAccountId": "123456789012",
    "EncryptionAlgorithms": [
      "SYMMETRIC_DEFAULT"
    ]
  }
}
```

ステップ 2: KMS キーでアクセス許可を設定する

デフォルトでは、すべての AWS KMS キーはプライベートです。リソースの所有者のみがその CMK を使用してデータを暗号化および復号できます。ただし、リソース所有者は、他のユーザーとリソースに KMS キーへのアクセス許可を付与することができます。このステップでは、このキーを使用するための CloudWatch Logs サービスプリンシパルと呼び出し元ロール権限を付与します。このサービスプリンシパルは、KMS キーが保存されているリージョンと同じ AWS リージョンにある必要があります。

ベストプラクティスとして、KMS キーの使用を、指定した AWS アカウントまたはロググループのみに制限することをお勧めします。

まず、[get-key-policy](#) コマンドを使用して、KMS キーのデフォルトポリシーを `policy.json` として保存します。

```
aws kms get-key-policy --key-id key-id --policy-name default --output text > ./policy.json
```

テキストエディタで `policy.json` ファイルを開き、以下のいずれかのステートメントから太字のセクションを追加します。既存のステートメントと新しいステートメントをカンマで区切ります。これらのステートメントでは、Conditionセクションを使用して AWS KMS キーのセキュリティを強化します。詳細については、「[AWS KMS キーと暗号化コンテキスト](#)」を参照してください。

この例の Condition セクションでは、キーを 1 つのロググループ ARN に制限しています。

JSON

```
{
  "Version": "2012-10-17",
  "Id": "key-default-1",
  "Statement": [
    {
      "Sid": "Enable IAM User Permissions",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::123456789012:root"
      },
      "Action": "kms:*",
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "logs.us-east-1.amazonaws.com"
      },
      "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:Describe*"
      ],
      "Resource": "*",
      "Condition": {
        "ArnEquals": {
          "kms:EncryptionContext:aws:logs:arn": "arn:aws:logs:us-east-1:111122223333:log-group:log-group-name"
        }
      }
    }
  ]
}
```

```
]
}
```

この例の Condition セクションは AWS KMS キーの使用を指定したアカウントに制限しますが、これを使用できるロググループに制限はありません。

JSON

```
{
  "Version": "2012-10-17",
  "Id": "key-default-1",
  "Statement": [
    {
      "Sid": "Enable IAM User Permissions",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::123456789012:root"
      },
      "Action": "kms:*",
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "logs.us-east-1.amazonaws.com"
      },
      "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:Describe*"
      ],
      "Resource": "*",
      "Condition": {
        "ArnLike": {
          "kms:EncryptionContext:aws:logs:arn": "arn:aws:logs:us-east-1:123456789012:"
        }
      }
    }
  ]
}
```



```
]
}
```

ステップ 3: 呼び出し元の IAM プリンシパルにアクセス許可を設定する

CloudWatch Logs APIs を呼び出す IAM プリンシパル (ユーザーまたはロール) にアクセス許可を追加します。必要なアクセス許可は、プリンシパルが実行する必要があるオペレーションによって異なります。これらのアクセス許可は、AWS KMS キーポリシーまたはロール自体の IAM を通じて追加できます。CloudWatch Logs は `kms:ViaService` を使用して、お客様に代わって を呼び出し AWS KMS ます。詳細については、「[kms:ViaService](#)」を参照してください。

KMS キーをロググループに関連付けるためのアクセス許可

`kmsKeyId` パラメータ `CreateLogGroup` を使用して を呼び出すか、 を呼び出す IAM プリンシパルには `AssociateKmsKey`、指定された KMS キーに対する `kms:DescribeKey` アクセス許可が必要です。呼び出し元にこのアクセス許可がない場合、API コールは で失敗します `AccessDeniedException`。

次のキーポリシーステートメントの例では、KMS キーをロググループに関連付けるために必要な最小限のアクセス許可を付与します。

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::account_id:role/role_name"
  },
  "Action": [
    "kms:Describe*"
  ],
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "kms:ViaService": [
        "logs.region.amazonaws.com"
      ]
    }
  }
}
```

暗号化されたログデータの読み取りと書き込みのためのアクセス許可

IAM プリンシパルが暗号化されたログデータを読み書きする必要がある場合 (カスタマーマネージド KMS キーで暗号化されたロググループ `StartQuery` で `PutLogEvents` `GetLogEvents`、`FilterLogEvents`、`PutLogEvents` を呼び出す場合など)、プリンシパルには追加の AWS KMS アクセス許可が必要です。次のキーポリシーステートメントの例では、キーの関連付けと暗号化されたログデータの読み取りまたは書き込みの両方に必要なアクセス許可の完全なセットを付与します。

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::account_id:role/role_name"
  },
  "Action": [
    "kms:Encrypt",
    "kms:ReEncrypt*",
    "kms:Decrypt",
    "kms:GenerateDataKey*",
    "kms:Describe*"
  ],
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "kms:ViaService": [
        "logs.region.amazonaws.com"
      ]
    }
  }
}
```

ベストプラクティスとして、AWS KMS 暗号化されたロググループとやり取りするロールのみにポリシーの範囲を設定します。

または、キーの関連付けと IAM を介した暗号化されたログデータの読み取りまたは書き込みの両方のためのアクセス許可の完全なセットを付与するには、発信者ロールに次のポリシーを追加します。これは、既存のロールポリシーに追加すること、追加の個別のポリシーとしてロールにアタッチすることもできます。この方法を使用する場合は、ベストプラクティスとして、ログの暗号化に使用される AWS KMS キーのみにポリシーの範囲を設定します。詳細については、[「IAM ポリシーの編集」](#)を参照してください。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kms:Encrypt",
        "kms:ReEncrypt*",
        "kms:Decrypt",
        "kms:GenerateDataKey*",
        "kms:Describe*"
      ],
      "Condition": {
        "StringEquals": {
          "kms:ViaService": [
            "logs.us-east-1.amazonaws.com"
          ]
        }
      },
      "Resource": "arn:aws:kms:us-east-1:444455556666:key/key_id"
    }
  ]
}
```

最後に、次の [put-key-policy](#) コマンドを使用して更新されたポリシーを追加します。

```
aws kms put-key-policy --key-id key-id --policy-name default --policy file://
policy.json
```

ステップ 4: KMS キーをロググループに関連付ける

KMS キーとロググループは、ロググループの作成時または作成後に関連付けることができます。

ロググループに KMS キーがすでに関連付けられているかどうかを確認するには、次の [describe-log-groups](#) コマンドを使用します。

```
aws logs describe-log-groups --log-group-name-prefix "log-group-name-prefix"
```

出力に `kmsKeyId` フィールドが含まれている場合、ロググループはそのフィールドの値に対して表示されるキーに関連付けられます。

ロググループの作成時に KMS キーをロググループに関連付けるには

次のように、[create-log-group](#) コマンドを使用します。

```
aws logs create-log-group --log-group-name my-log-group --kms-key-id "key-arn"
```

KMS キーを既存のロググループに関連付けるには

次のように、[associate-kms-key](#) コマンドを使用します。

```
aws logs associate-kms-key --log-group-name my-log-group --kms-key-id "key-arn"
```

ステップ 5: ロググループからキーの関連付けを解除する

ロググループに関連付けられた KMS キーの関連付けを解除するには、次の [disassociate-kms-key](#) コマンドを使用します。

```
aws logs disassociate-kms-key --log-group-name my-log-group
```

AWS KMS キーと暗号化コンテキスト

AWS Key Management Service キーと暗号化されたロググループのセキュリティを強化するために、CloudWatch Logs はログデータの暗号化に使用される暗号化コンテキストの一部としてロググループ ARNs を配置するようになりました。暗号化コンテキストは、追加の認証済みデータとして使用されるキーと値のペアのセットです。暗号化コンテキストを使用すると、IAM ポリシー条件を使用して、アカウントとロググループごとに AWS KMS AWS キーへのアクセスを制限できます。詳細については、[暗号化コンテキスト](#)および [IAM JSON ポリシー要素: 条件](#)を参照してください。

暗号化されたロググループごとに異なる KMS キーを使用することをお勧めします。

前に暗号化したロググループがあり、そのロググループを変更して、そのグループでのみ機能する新しい KMS キーを使用する場合は、次の手順に従います。

暗号化されたロググループを変更して、KMS キーの使用をそのグループのみに制限するには

1. 次のコマンドを入力して、ロググループの現在のキーの ARN を見つけます。

```
aws logs describe-log-groups
```

出力には以下の行が含まれます。ARN を書きとめておきます。ステップ 7 で使用する必要があります。

```
...  
"kmsKeyId": "arn:aws:kms:us-west-2:123456789012:key/01234567-89ab-  
cdef-0123-456789abcdef"  
...
```

2. 以下のコマンドを入力して、新しい KMS キーを作成します。

```
aws kms create-key
```

3. 以下のコマンドを入力して、新しいキーのポリシーを `policy.json` ファイルに保存します。

```
aws kms get-key-policy --key-id new-key-id --policy-name default --output text > ./  
policy.json
```

4. テキストエディタを使用して `policy.json` を開き、Condition 式をポリシーに追加します。

JSON

```
{  
  "Version": "2012-10-17",  
  "Id": "key-default-1",  
  "Statement": [  
    {  
      "Sid": "Enable IAM User Permissions",  
      "Effect": "Allow",  
      "Principal": {  
        "AWS": "arn:aws:iam::111122223333:root"  
      },  
      "Action": "kms:*",  
      "Resource": ""  
    },  
    {  
      "Effect": "Allow",  
      "Principal": {  
        "Service": "logs.us-east-1.amazonaws.com"  
      },  
      "Action": "logs:CreateLogGroup",  
      "Resource": "arn:aws:logs:us-east-1:111122223333:log-group:aws-kms-logs"  
    }  
  ]  
}
```

```

    },
    "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:Describe*"
    ],
    "Resource": "*",
    "Condition": {
        "ArnLike": {
            "kms:EncryptionContext:aws:logs:arn": "arn:aws:logs:us-east-1:111122223333:log-group:LOG-GROUP-NAME"
        }
    }
}

```

5. 次のコマンドを入力して、更新されたポリシーを新しい KMS キーに追加します。

```
aws kms put-key-policy --key-id new-key-ARN --policy-name default --policy file://policy.json
```

6. 以下のコマンドを入力して、そのポリシーをロググループに関連付けます。

```
aws logs associate-kms-key --log-group-name my-log-group --kms-key-id new-key-ARN
```

CloudWatch Logs で、すべての新しいデータが新しいキーを使用して暗号化されるようになりました。

7. 次に、Decrypt を除くすべてのアクセス許可を古いキーから取り消します。まず、以下のコマンドを入力して古いポリシーを取得します。

```
aws kms get-key-policy --key-id old-key-ARN --policy-name default --output text
> ./policy.json
```

8. テキストエディタを使用して policy.json を開き、Action リストから kms:Decrypt を除くすべての値を削除します。

JSON

```
{
  "Version": "2012-10-17",
  "Id": "key-default-1",
  "Statement": [
    {
      "Sid": "Enable IAM User Permissions",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:root"
      },
      "Action": "kms:*",
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "logs.region.amazonaws.com"
      },
      "Action": [
        "kms:Decrypt"
      ],
      "Resource": "*"
    }
  ]
}
```

9. 次のコマンドを入力して、更新されたポリシーを古いキーに追加します。

```
aws kms put-key-policy --key-id old-key-ARN --policy-name default --policy file://
policy.json
```

機密性の高いログデータをマスキングで保護する

ロググループのデータ保護ポリシーを使用することで、CloudWatch Logs に取り込まれた機密データを保護できます。これらのポリシーを使うことで、アカウントのロググループが取り込んだログイベントに表示される機密データを、監査およびマスクできます。

データ保護ポリシーを作成すると、デフォルトでは、ユーザーの選択したデータ識別子に一致する機密データがすべての出力ポイント (CloudWatch Logs Insights、メトリクスフィルター、サブスクリプションフィルターなど) でマスクされます。マスクされていないデータを閲覧できるのは、logs:Unmask IAMアクセス許可を持つユーザーのみです。

アカウントのすべてのロググループに対してデータ保護ポリシーを作成できます。また、個々のロググループのデータ保護ポリシーも作成できます。アカウント全体に対するポリシーを作成すると、既存のロググループと今後作成するロググループの両方に、ポリシーが適用されます。

アカウント全体に対するデータ保護ポリシーを作成し、1つのロググループに対するポリシーも作成すると、そのロググループには両方のポリシーが適用されます。いずれかのポリシーで指定されたマネージドデータ識別子は、すべてそのロググループで監査およびマスクされます。

Note

機密データのマスキングは、標準ログクラスと低頻度アクセスログクラスの両方のロググループでサポートされています。ログクラスの詳細については、「[ログクラス](#)」を参照してください。

各ロググループで設定できるロググループレベルのデータ保護ポリシーは1つのみです。ただしそのポリシーでは、監査およびマスキングの対象となるマネージドデータ識別子を複数指定できます。データ保護ポリシーの文字数の上限は、30,720 文字です。

Important

機密データは、ロググループに取り込まれるときに検出され、マスクされます。データ保護ポリシーを設定しても、それ以前にロググループに取り込まれたログイベントはマスクされません。

CloudWatch Logs は多くのマネージドデータ識別子をサポートしており、財務データ、個人健康情報 (PHI)、個人を特定できる情報 (PII) を保護するために選択できる事前設定されたデータタイプを提供しています。CloudWatch Logs のデータ保護では、パターンマッチングと機械学習モデルを活用して機密データを検出できます。マネージドデータ識別子の種類によっては、検出は、機密データに密接に関連する特定のキーワードの検出結果にも依存します。また、カスタムデータ識別子を使用して、特定のユースケースに合わせたデータ識別子を作成することもできます。

選択されたデータ識別子に一致する機密データが検出されると、CloudWatch にメトリクスが発行されます。これは LogEventsWithFindings メトリクスで、AWS/Logs 名前空間で発行されます。このメトリクスは CloudWatch アラームを作成するために使用でき、グラフやダッシュボードで視覚化できます。データ保護によって発行されたメトリクスは無料で提供されるメトリクスなので、料金はかかりません。CloudWatch Logs が CloudWatch に送信するメトリクスの詳細については、「[CloudWatch メトリクスによるモニタリング](#)」を参照してください。

各マネージドデータ識別子は、特定の国またはリージョンのクレジットカード番号、AWS シークレットアクセスキー、パスポート番号など、特定のタイプの機密データを検出するように設計されています。データ保護ポリシーを作成する際に、これらの識別子を使用してロググループが取り込んだログを分析し、検出された場合にアクションを実行するように設定できます。

CloudWatch Logs データ保護では、マネージドデータ識別子を使用して、次のカテゴリの機密データを検出できます。

- プライベートキーや AWS シークレットアクセスキーなどの認証情報
- クレジットカード番号などの財務情報
- 運転免許証や社会保障番号などの個人を特定できる情報 (PII)
- 健康保険または医療識別番号などの保護対象保健情報 (PHI)
- IP アドレスや MAC アドレスなどのデバイス識別子

保護できるデータの種類の詳細については、「[保護できるデータの種類](#)」を参照してください。

目次

- [データ保護ポリシーを理解する](#)
 - [データ保護ポリシーとは](#)
 - [データ保護ポリシーの構成の仕組み](#)
 - [データ保護ポリシーの JSON プロパティ](#)
 - [ポリシーステートメントの JSON プロパティ](#)
 - [ポリシーステートメントオペレーションの JSON プロパティ](#)
- [データ保護ポリシーの作成または操作に必要な IAM 権限](#)
 - [アカウントレベルのデータ保護ポリシーに必要なアクセス権限](#)
 - [1つのロググループのデータ保護ポリシーに必要なアクセス権限](#)
 - [データ保護ポリシーのサンプル](#)
- [アカウント全体のデータ保護ポリシーを作成する](#)

- [コンソール](#)
- [AWS CLI](#)
 - [AWS CLI または API オペレーションのデータ保護ポリシー構文](#)
- [1 つのロググループ用のデータ保護ポリシーを作成する](#)
- [コンソール](#)
- [AWS CLI](#)
 - [AWS CLI または API オペレーションのデータ保護ポリシー構文](#)
- [データをマスクせずに表示する](#)
- [監査結果レポート](#)
 - [で保護された バケットに監査結果を送信するために必要なキーポリシー AWS KMS](#)
- [保護できるデータの種類の種類](#)
 - [機密データの種類の種類の CloudWatch Logs マネージドデータ識別子](#)
 - [認証情報](#)
 - [認証情報データタイプのデータ識別子 ARN](#)
 - [デバイス識別子](#)
 - [デバイスデータタイプのデータ識別子 ARN](#)
 - [財務情報](#)
 - [財務データタイプのデータ識別子 ARN](#)
 - [保護対象保健情報 \(PHI\)](#)
 - [保護対象の医療情報 \(PHI\) データタイプのデータ識別子 ARN](#)
 - [個人を特定できる情報 \(PII\)](#)
 - [運転免許証識別番号のキーワード](#)
 - [国民識別番号のキーワード](#)
 - [パスポート番号のキーワード](#)
 - [納税者識別と参照番号のキーワード](#)
 - [個人を特定できる情報 \(PII\) のデータ識別子 ARN](#)
 - [カスタムデータ識別子](#)
 - [SNS カスタムデータ識別子とは](#)
 - [カスタムデータ識別子の制約](#)
 - [コンソールでのカスタムデータ識別子の使用](#)

- [データ保護ポリシーでカスタムデータ識別子を使用する](#)

データ保護ポリシーを理解する

トピック

- [データ保護ポリシーとは](#)
- [データ保護ポリシーの構成の仕組み](#)

データ保護ポリシーとは

CloudWatch はデータ保護ポリシーを使用して、スキャンする機密データと、そのデータを保護するために実行するアクションを選択します。目的の機密データを選択するには、[データ識別子](#)を使用します。そうすることで、CloudWatch Logs データ保護が機械学習とパターンマッチングを使用して機密データを検出ようになります。検出されたデータ識別子に基づいてアクションを実行するには、Audit (監査) および De-identify (匿名化) 操作を定義できます。これらの操作は、検出された (または検出されなかった) 機密データをログに記録し、ログイベントが表示されるときに機密データをマスクすることを可能にします。

データ保護ポリシーの構成の仕組み

次の図に示すように、データ保護ポリシードキュメントには次の要素が含まれています。

- ドキュメントの最上部に記載されるポリシー全体の情報 (任意)
- 監査および匿名化アクションを定義する 1 つのステートメント

CloudWatch Logs ロググループごとに定義できるデータ保護ポリシーは 1 つだけです。データ保護ポリシーには、1 つまたは複数の拒否または識別解除ステートメントと 1 つの監査ステートメントのみを含めることができます。

データ保護ポリシーの JSON プロパティ

データ保護ポリシーでは、識別のために以下の基本ポリシー情報が必要です。

- Name – ポリシーの名前。
- Description (オプション) – ポリシーの説明。
- Version – ポリシー言語のバージョン。現在のバージョンは 2021-06-01. です。
- Statement – データ保護ポリシーアクションを指定するステートメントのリスト。

```
{
  "Name": "CloudWatchLogs-PersonalInformation-Protection",
  "Description": "Protect basic types of sensitive data",
  "Version": "2021-06-01",
  "Statement": [
    ...
  ]
}
```

ポリシーステートメントの JSON プロパティ

ポリシーステートメントは、データ保護オペレーションの検出コンテキストを設定します。

- Sid (オプション) – ステートメント識別子。
- DataIdentifier - CloudWatch ログがスキャンする必要がある機密データ。名前、住所、電話番号などです。
- Operation – 後続アクション (Audit または De-identify のいずれか)。CloudWatch Logs は、機密データが検出されたときにこれらのアクションを実行します。

```
{
  ...
  "Statement": [
    {
      "Sid": "audit-policy",
      "DataIdentifier": [
        "arn:aws:dataprotection::aws:data-identifier/Address"
      ],
      "Operation": {
        "Audit": {
          "FindingsDestination": {}
        }
      }
    }
  ],
}
```

ポリシーステートメントオペレーションの JSON プロパティ

ポリシーステートメントは、次のデータ保護オペレーションのいずれかを設定します。

- Audit – ロギングを中断することなく、メトリクスと結果レポートを発行します。一致する文字列は、CloudWatch Logs が CloudWatch の AWS/Logs 名前空間に発行する LogEventsWithFindings

メトリクスをインクリメントします。これらのメトリクスは、アラームを作成するために使用できます。

結果レポートの例については、「[監査結果レポート](#)」を参照してください。

CloudWatch Logs が CloudWatch に送信するメトリクスの詳細については、「[CloudWatch メトリクスによるモニタリング](#)」を参照してください。

- De-identify – ロギングを中断することなく機密データをマスクします。

データ保護ポリシーの作成または操作に必要な IAM 権限

ロググループのデータ保護ポリシーにアクセスできるようにするには、次の表で表示されている特定のアクセス権限を持っている必要があります。アクセス権限は、アカウント全体のデータ保護ポリシーと、単一のロググループに適用されるデータ保護ポリシーとで異なります。

アカウントレベルのデータ保護ポリシーに必要なアクセス権限

Note

Lambda 関数内でこれらの操作のいずれかを実行する場合、Lambda 実行ロールとアクセス許可の境界には次の権限も含める必要があります。

運用	IAM 権限が必要です	[リソース]
監査先を指定しないデータ保護ポリシーを作成する	logs:PutAccountPolicy	*
	logs:PutDataProtectionPolicy	*
監査先として CloudWatch Logs を使用してデータ保護ポリシーを作成する	logs:PutAccountPolicy	*
	logs:PutDataProtectionPolicy	*

運用	IAM 権限が必要です	[リソース]
監査先として Firehose を使用してデータ保護ポリシーを作成する	logs:CreateLogDelivery	*
	logs:PutResourcePolicy	*
	logs:DescribeResourcePolicies	*
	logs:DescribeLogGroups	*
	logs:PutAccountPolicy	*
	logs:PutDataProtectionPolicy	*
	logs:CreateLogDelivery	*
	firehose:TagDeliveryStream	arn:aws:logs:::deliverystream/ <i>YOUR_DELIVERY_STREAM</i>
監査先として Amazon S3 を使用してデータ保護ポリシーを作成する	logs:PutAccountPolicy	*
	logs:PutDataProtectionPolicy	*
	logs:CreateLogDelivery	*
	s3:GetBucketPolicy	arn:aws:s3::: <i>YOUR_BUCKET</i>

運用	IAM 権限が必要です	[リソース]
	s3:PutBucketPolicy	arn:aws:s3::: <i>YOUR_BUCKET</i>
指定したロググループのマスクされたログイベントのマスクを外す	logs:Unmask	arn:aws:logs:::log-group:*
既存のデータ保護ポリシーを表示する	logs:GetDataProtectionPolicy	*
データ保護ポリシーを削除する	logs>DeleteAccountPolicy	*
	logs>DeleteDataProtectionPolicy	*

いずれかのデータ保護監査ログがすでに宛先に送信されている場合、同じ宛先にログを送信する他のポリシーに必要なのは logs:PutDataProtectionPolicy および logs>CreateLogDelivery 権限のみです。

1 つのロググループのデータ保護ポリシーに必要なアクセス権限

Note

Lambda 関数内でこれらの操作のいずれかを実行する場合、Lambda 実行ロールとアクセス許可の境界には次の権限も含める必要があります。

運用	IAM 権限が必要です	[リソース]
監査先を指定しないデータ保護ポリシーを作成する	logs:PutDataProtectionPolicy	arn:aws:logs:::log-group: <i>YOUR_LOG_GROUP</i> :*

運用	IAM 権限が必要です	[リソース]
監査先として CloudWatch Logs を使用してデータ保護ポリシーを作成する	logs:PutDataProtectionPolicy logs:CreateLogDelivery logs:PutResourcePolicy logs:DescribeResourcePolicies logs:DescribeLogGroups	arn:aws:logs::log-group: <i>YOUR_LOG_GROUP</i> :* * * *
監査先として Firehose を使用してデータ保護ポリシーを作成する	logs:PutDataProtectionPolicy logs:CreateLogDelivery firehose:TagDeliveryStream	arn:aws:logs::log-group: <i>YOUR_LOG_GROUP</i> :* * arn:aws:logs::deliverystream/ <i>YOUR_DELIVERY_STREAM</i>
監査先として Amazon S3 を使用してデータ保護ポリシーを作成する	logs:PutDataProtectionPolicy logs:CreateLogDelivery s3:GetBucketPolicy s3:PutBucketPolicy	arn:aws:logs::log-group: <i>YOUR_LOG_GROUP</i> :* * arn:aws:s3::: <i>YOUR_BUCKET</i> arn:aws:s3::: <i>YOUR_BUCKET</i>

運用	IAM 権限が必要です	[リソース]
マスクされたログイベントをマスク解除する	logs:Unmask	arn:aws:logs::log-group: <i>YOUR_LOG_GROUP</i> :*
既存のデータ保護ポリシーを表示する	logs:GetDataProtectionPolicy	arn:aws:logs::log-group: <i>YOUR_LOG_GROUP</i> :*
データ保護ポリシーを削除する	logs>DeleteDataProtectionPolicy	arn:aws:logs::log-group: <i>YOUR_LOG_GROUP</i> :*

いずれかのデータ保護監査ログがすでに宛先に送信されている場合、同じ宛先にログを送信する他のポリシーに必要なのは logs:PutDataProtectionPolicy および logs:CreateLogDelivery 権限のみです。

データ保護ポリシーのサンプル

次のサンプルポリシーにより、3 種類の監査先すべてに監査結果を送信できるデータ保護ポリシーを、ユーザーが作成、表示、削除できます。ユーザーはマスクされていないデータを確認することはできません。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowLogDeliveryConfiguration",
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogDelivery",
        "logs:PutResourcePolicy",
        "logs:DescribeLogGroups",
        "logs:DescribeResourcePolicies"
      ],
    }
  ],
}
```

```
        "Resource": "*"
    },
    {
        "Sid": "AllowDataProtectionAndBucketConfiguration",
        "Effect": "Allow",
        "Action": [
            "logs:GetDataProtectionPolicy",
            "logs:DeleteDataProtectionPolicy",
            "logs:PutDataProtectionPolicy",
            "s3:PutBucketPolicy",
            "firehose:TagDeliveryStream",
            "s3:GetBucketPolicy"
        ],
        "Resource": [
            "arn:aws:firehose:us-east-1:111122223333:deliverystream/delivery-  
stream-name",
            "arn:aws:s3:::amzn-s3-demo-destination-bucket",
            "arn:aws:logs:us-east-1:111122223333:log-group:log-group-name:*"
        ]
    }
]
```

アカウント全体のデータ保護ポリシーを作成する

CloudWatch Logs コンソールまたは AWS CLI コマンドを使用して、アカウント内のすべてのロググループの機密データをマスクするデータ保護ポリシーを作成できます。作成すると、現在のロググループと今後作成するロググループの両方に影響します。

Important

機密データは、ロググループに取り込まれるときに検出され、マスクされます。データ保護ポリシーを設定しても、それ以前にロググループに取り込まれたログイベントはマスクされません。

トピック

- [コンソール](#)
- [AWS CLI](#)

コンソール

コンソールを使用してアカウント全体のデータ保護ポリシーを作成するには

1. CloudWatch コンソールの <https://console.aws.amazon.com/cloudwatch/> を開いてください。
2. ナビゲーションペインで [設定] を選択します。リストの一番下付近にあります。
3. [ログ] タブを選択します。
4. [設定] を選択します。
5. [マネージドデータ識別子] で、使用しているすべてのロググループで監査およびマスクするデータの種類の選択します。選択ボックスに入力して、必要な識別子を見つけることができます。

ログデータやビジネスに関連するデータ識別子のみを選択することをお勧めします。多くの種類のデータを選択すると、誤検出につながる可能性があります。

保護できるデータの種類の詳細については、「[保護できるデータの種類](#)」を参照してください。

6. (オプション) カスタムデータ識別子を使用して他のタイプのデータを監査およびマスクする場合は、[カスタムデータ識別子を追加] を選択します。次に、データ型の名前と、ログイベントでそのタイプのデータを検索するために使用する正規表現を入力します。詳細については、「[カスタムデータ識別子](#)」を参照してください。

単一のデータ保護ポリシーには、最大 10 個のカスタムデータ識別子を含めることができます。カスタムデータ識別子を定義する各正規表現は、200 文字以下である必要があります。

7. (オプション) 監査結果の送信先となるサービスを 1 つまたは複数選択します。監査結果をこれらのサービスのいずれにも送信しないことを選択した場合でも、選択した機密データタイプは引き続きマスクされます。
8. [Activate data protection] (データ保護をアクティブにする) を選択します。

AWS CLI

を使用してデータ保護ポリシー AWS CLI を作成するには

1. テキストエディタを使用して DataProtectionPolicy.json という名前のポリシーファイルを作成します。ポリシーの構文については、次のセクションを参照してください。
2. 次のコマンドを入力します。

```
aws logs put-account-policy \  
--policy-name TEST_POLICY --policy-type "DATA_PROTECTION_POLICY" \  

```

```
--policy-document file://policy.json \  
--scope "ALL" \  
--region us-west-2
```

AWS CLI または API オペレーションのデータ保護ポリシー構文

AWS CLI コマンドまたは API オペレーションで使用する JSON データ保護ポリシーを作成する場合、ポリシーには 2 つの JSON ブロックを含める必要があります。

- 最初のブロックには、DataIdentifier 配列と Audit アクションを含む Operation プロパティの両方が含まれている必要があります。DataIdentifier 配列には、マスクする機密データの種類が表示されます。利用できるすべてのオプションについての詳細は、「[保護できるデータの種類](#)」を参照してください。

Audit アクションを含む Operation プロパティは、機密データ用語を検索するために必要です。この Audit アクションには FindingsDestination オブジェクトが含まれている必要があります。オプションで FindingsDestination オブジェクトを使用して、監査結果レポートの送信先を 1 つ、または複数リストできます。ロググループ、Amazon Data Firehose ストリーム、S3 バケットなどの送信先を指定する場合、それらは既に存在している必要があります。監査結果レポートの例については、「[監査結果レポート](#)」を参照してください。

- 2 番目のブロックには、DataIdentifier 配列と Deidentify アクションを含む Operation プロパティの両方が含まれている必要があります。DataIdentifier 配列は、ポリシーの最初のブロックにある DataIdentifier 配列と完全に一致する必要があります。

Deidentify アクションを含む Operation プロパティが実際にデータをマスクするものであり、そのアクションには "MaskConfig": {} オブジェクトが含まれている必要があります。"MaskConfig": {} オブジェクトは空である必要があります。

以下は、マネージドデータ識別子のみを使用するデータ保護ポリシーの例です。このポリシーは、E メールアドレスと米国の運転免許証をマスクします。

カスタムデータ識別子を指定するポリシーの詳細については、「[データ保護ポリシーでカスタムデータ識別子を使用する](#)」を参照してください。

```
{  
  "Name": "data-protection-policy",  
  "Description": "test description",  
  "Version": "2021-06-01",
```

```

    "Statement": [{
      "Sid": "audit-policy",
      "DataIdentifier": [
        "arn:aws:dataprotection::aws:data-identifier/EmailAddress",
        "arn:aws:dataprotection::aws:data-identifier/DriversLicense-US"
      ],
      "Operation": {
        "Audit": {
          "FindingsDestination": {
            "CloudWatchLogs": {
              "LogGroup": "EXISTING_LOG_GROUP_IN_YOUR_ACCOUNT",
            },
            "Firehose": {
              "DeliveryStream": "EXISTING_STREAM_IN_YOUR_ACCOUNT"
            },
            "S3": {
              "Bucket": "EXISTING_BUCKET"
            }
          }
        }
      },
    },
    {
      "Sid": "redact-policy",
      "DataIdentifier": [
        "arn:aws:dataprotection::aws:data-identifier/EmailAddress",
        "arn:aws:dataprotection::aws:data-identifier/DriversLicense-US"
      ],
      "Operation": {
        "Deidentify": {
          "MaskConfig": {}
        }
      }
    }
  ]
}

```

1 つのロググループ用のデータ保護ポリシーを作成する

CloudWatch Logs コンソールまたは AWS CLI コマンドを使用して、機密データをマスクするデータ保護ポリシーを作成できます。

各ロググループに 1 つのデータ保護ポリシーを割り当てることができます。各データ保護ポリシーで、複数の種類の情報を監査できます。各データ保護ポリシーには、監査ステートメントを 1 つ含めることができます。

トピック

- [コンソール](#)
- [AWS CLI](#)

コンソール

コンソールを使用してデータ保護ポリシーを作成するには

1. CloudWatch コンソールの <https://console.aws.amazon.com/cloudwatch/> を開いてください。
2. ナビゲーションペインで、[ログ]、[ロググループ] の順に選択します。
3. ロググループの名前を選択します。
4. [Actions] (アクション)、[Create data protection policy] (データ保護ポリシーを作成) を選択します。
5. [マネージドデータ識別子] で、このロググループで監査およびマスクするデータの種類の種類を選択します。選択ボックスに入力して、必要な識別子を見つけることができます。

ログデータやビジネスに関連するデータ識別子のみを選択することをお勧めします。多くの種類のデータを選択すると、誤検出につながる可能性があります。

マネージドデータ識別子を使用して保護できるデータの種類の詳細については、「[保護できるデータの種類](#)」を参照してください。

6. (オプション) カスタムデータ識別子を使用して他のタイプのデータを監査およびマスクする場合は、[カスタムデータ識別子を追加] を選択します。次に、データ型の名前と、ログイベントでそのタイプのデータを検索するために使用する正規表現を入力します。詳細については、「[カスタムデータ識別子](#)」を参照してください。

単一のデータ保護ポリシーには、最大 10 個のカスタムデータ識別子を含めることができます。カスタムデータ識別子を定義する各正規表現は、200 文字以下である必要があります。

7. (オプション) 監査結果の送信先となるサービスを 1 つまたは複数選択します。監査結果をこれらのサービスのいずれにも送信しないことを選択した場合でも、選択した機密データタイプは引き続きマスクされます。
8. [Activate data protection] (データ保護をアクティブにする) を選択します。

AWS CLI

を使用してデータ保護ポリシー AWS CLI を作成するには

1. テキストエディタを使用して DataProtectionPolicy.json という名前のポリシーファイルを作成します。ポリシーの構文については、次のセクションを参照してください。
2. 次のコマンドを入力します。

```
aws logs put-data-protection-policy --log-group-identifier "my-log-group" --policy-document file:///Path/DataProtectionPolicy.json --region us-west-2
```

AWS CLI または API オペレーションのデータ保護ポリシー構文

AWS CLI コマンドまたは API オペレーションで使用する JSON データ保護ポリシーを作成する場合、ポリシーには 2 つの JSON ブロックを含める必要があります。

- 最初のブロックには、DataIdentifier 配列と Audit アクションを含む Operation プロパティの両方が含まれている必要があります。DataIdentifier 配列には、マスクする機密データの種別が表示されます。利用できるすべてのオプションについての詳細は、「[保護できるデータの種別](#)」を参照してください。

Audit アクションを含む Operation プロパティは、機密データ用語を検索するために必要です。この Audit アクションには FindingsDestination オブジェクトが含まれている必要があります。オプションで FindingsDestination オブジェクトを使用して、監査結果レポートの送信先を 1 つ、または複数リストできます。ロググループ、Amazon Data Firehose ストリーム、S3 バケットなどの送信先を指定する場合、それらは既に存在している必要があります。監査結果レポートの例については、「[監査結果レポート](#)」を参照してください。

- 2 番目のブロックには、DataIdentifier 配列と Deidentify アクションを含む Operation プロパティの両方が含まれている必要があります。DataIdentifier 配列は、ポリシーの最初のブロックにある DataIdentifier 配列と完全に一致する必要があります。

Deidentify アクションを含む Operation プロパティが実際にデータをマスクするものであり、そのアクションには "MaskConfig": {} オブジェクトが含まれている必要があります。"MaskConfig": {} オブジェクトは空である必要があります。

E メールアドレスと米国の運転免許証をマスクするデータ保護ポリシーの例を次に示します。

```

{
  "Name": "data-protection-policy",
  "Description": "test description",
  "Version": "2021-06-01",
  "Statement": [{
    "Sid": "audit-policy",
    "DataIdentifier": [
      "arn:aws:dataprotection::aws:data-identifier/EmailAddress",
      "arn:aws:dataprotection::aws:data-identifier/DriversLicense-US"
    ],
    "Operation": {
      "Audit": {
        "FindingsDestination": {
          "CloudWatchLogs": {
            "LogGroup": "EXISTING_LOG_GROUP_IN_YOUR_ACCOUNT",
          },
          "Firehose": {
            "DeliveryStream": "EXISTING_STREAM_IN_YOUR_ACCOUNT"
          },
          "S3": {
            "Bucket": "EXISTING_BUCKET"
          }
        }
      }
    }
  },
  {
    "Sid": "redact-policy",
    "DataIdentifier": [
      "arn:aws:dataprotection::aws:data-identifier/EmailAddress",
      "arn:aws:dataprotection::aws:data-identifier/DriversLicense-US"
    ],
    "Operation": {
      "Deidentify": {
        "MaskConfig": {}
      }
    }
  }
]
}

```


データをマスクせずに表示する

データをマスクせずに表示するには、ユーザーに `logs:Unmask` アクセス許可が必要です。このアクセス許可を持つユーザーは、次の方法でデータをマスクせずに表示できます。

- ログストリームでイベントを表示するときは、[Display] (表示)、[Unmask] (マスク解除) を選択します。
- `unmask(@message)` コマンドを含む CloudWatch Logs Insights クエリを使用します。次のクエリ例では、ストリーム内の最新の 20 件のログイベントがマスクされずに表示されます。

```
fields @timestamp, @message, unmask(@message)
| sort @timestamp desc
| limit 20
```

CloudWatch Logs Insights コマンドの詳細については、「[CloudWatch Logs Insights 言語のクエリ構文](#)」を参照してください。

- `unmask` パラメータを含む [GetLogEvents](#) オペレーションまたは [FilterLogEvents](#) オペレーションを使用します。

CloudWatchLogsFullAccess ポリシーには `logs:Unmask` アクセス許可が含まれています。CloudWatchLogsFullAccess を持たないユーザーにカスタム IAM ポリシーをアタッチして、そのユーザーに `logs:Unmask` を許可できます。詳細については、「[ユーザーへのアクセス許可の追加 \(コンソール\)](#)」を参照してください。

監査結果レポート

CloudWatch Logs、Amazon S3、または Firehose に監査レポートを書き込むように CloudWatch Logs のデータ保護監査ポリシーを設定した場合、これらの検出結果レポートは以下の例と類似します。CloudWatch Logs は、機密データが含まれるログイベントごとに 1 つの結果レポートを書き込みます。

```
{
  "auditTimestamp": "2023-01-23T21:11:20Z",
  "resourceArn": "arn:aws:logs:us-west-2:111122223333:log-group:/aws/lambda/MyLogGroup:*",
  "dataIdentifiers": [
    {
      "name": "EmailAddress",
      "count": 2,
```

```
    "detections": [  
      {  
        "start": 13,  
        "end": 26  
      },  
      {  
        "start": 30,  
        "end": 43  
      }  
    ]  
  }  
]
```

レポート内のフィールドは、以下のとおりです。

- resourceArn フィールドには、機密データが見つかったロググループが表示されます。
- dataIdentifiers オブジェクトには、監査している機密データのタイプの 1 つに関する結果が表示されます。
- name フィールドには、このセクションで報告されている機密データのタイプが特定されています。
- count フィールドには、ログイベントでこのタイプの機密データが出現する回数が表示されます。
- start および end フィールドには、機密データがログイベントのどこで出現しているかが、出現ごとに文字数で表示されます。

上記の例は、1 つのログイベントで 2 件の E メールアドレスが見つかったレポートです。最初の E メールアドレスは、ログイベントの 13 文字目から始まり、26 文字目で終わります。2 番目のメールアドレスは 30 文字目から 43 文字目までとなっています。このログイベントには 2 件の E メールアドレスがありますが、LogEventsWithFindings メトリクスの値は 1 つしかインクリメントされていません。これは、メトリクスが数えているのが機密データの出現回数ではなく、機密データが含まれるログイベントの数だからです。

で保護された バケットに監査結果を送信するために必要なキーポリシー AWS KMS

Amazon S3 バケット内のデータを保護するには、Amazon S3 マネージドキーを使用したサーバー側の暗号化 (SSE-S3)、または KMS キーを使用したサーバー側の暗号化 (SSE-KMS) のいずれかを有効にします。詳細については、Amazon S3 ユーザーガイドの「[サーバー側の暗号化を使用したデータの保護](#)」を参照してください。

SSE-S3 で保護されているバケットに監査結果を送信した場合、追加の設定は必要ありません。Amazon S3 が暗号化キーを処理します。

SSE-KMS で保護されているバケットに監査結果を送信すると、ログ配信アカウントが S3 バケットに書き込めるように、KMS キーのキーポリシーを更新する必要があります。SSE-KMS での使用に必要なキーポリシーについては、「Amazon CloudWatch Logs ユーザーガイド」の「[Amazon S3](#)」を参照してください。

保護できるデータの種類

このセクションには、CloudWatch Logs データ保護ポリシーで保護できるデータの種類の情報が記載されています。CloudWatch Logs マネージドデータ識別子には、財務データ、個人健康情報 (PHI)、個人を特定できる情報 (PII) を保護するために、事前設定されたデータタイプが用意されています。また、カスタムデータ識別子を使用して、特定のユースケースに合わせたデータ識別子を作成することもできます。

目次

- [機密データの種類の CloudWatch Logs マネージドデータ識別子](#)
 - [認証情報](#)
 - [認証情報データタイプのデータ識別子 ARN](#)
 - [デバイス識別子](#)
 - [デバイスデータタイプのデータ識別子 ARN](#)
 - [財務情報](#)
 - [財務データタイプのデータ識別子 ARN](#)
 - [保護対象保健情報 \(PHI\)](#)
 - [保護対象の医療情報 \(PHI\) データタイプのデータ識別子 ARN](#)
 - [個人を特定できる情報 \(PII\)](#)
 - [運転免許証識別番号のキーワード](#)
 - [国民識別番号のキーワード](#)
 - [パスポート番号のキーワード](#)
 - [納税者識別と参照番号のキーワード](#)
 - [個人を特定できる情報 \(PII\) のデータ識別子 ARN](#)
 - [カスタムデータ識別子](#)

- [カスタムデータ識別子の制約](#)
- [コンソールでのカスタムデータ識別子の使用](#)
- [データ保護ポリシーでカスタムデータ識別子を使用する](#)

機密データの種類の CloudWatch Logs マネージドデータ識別子

このセクションには、マネージドデータ識別子を使用して保護できるデータの種類の種類、およびそれぞれの種類のデータに関連する国と地域に関する情報が記載されています。

機密データの種類のによっては、CloudWatch Logs のデータ保護機能でデータと密接に関連するキーワードがないかスキャンされ、そのキーワードが見つかった場合にのみ該当するとみなされます。キーワードが特定のタイプのデータの近くにある必要がある場合は、通常、キーワードはデータから 30 文字以内 (包括的) になければなりません。

キーワードにスペースが含まれている場合、CloudWatch Logs のデータ保護では、スペースを含まないキーワードのバリエーションや、スペースではなくアンダースコア (_) またはハイフン (-) を含むキーワードのバリエーションが自動的に照合されます。場合によっては、CloudWatch Logs ではキーワードの一般的なバリエーションに対処するためにキーワードを省略形にしたり、省略形のときは元に戻したりします。

次のテーブルでは、CloudWatch Logs でマネージドデータ識別子を使用して検出できる認証情報、デバイス、財務、医療、および保護対象保健情報 (PHI) の種類の一覧を示しています。これらは、個人を特定できる情報 (PII) としても認定される可能性のある特定のタイプのデータに加えたものです。

言語や地域に依存しないサポート対象の識別子

識別子	Category
Address	個人
AwsSecretKey	認証情報
CreditCardExpiration	金融
CreditCardNumber	金融
CreditCardSecurityCode	金融

識別子	Category
EmailAddress	個人
IpAddress	個人
LatLong	個人
Name	個人
OpenSshPrivateKey	認証情報
PgpPrivateKey	認証情報
PkcsPrivateKey	認証情報
PuttyPrivateKey	認証情報
VehicleIdentificationNumber	個人

地域に依存するデータ識別子には、識別子名に続けてハイフンと 2 文字のコード (ISO 3166-1 alpha-2) が必要です。例えば、DriversLicense-US。

2 文字の国コードまたは地域コードを含む必要があるサポート対象の識別子

識別子	Category	国と言語
BankAccountNumber	金融	DE、ES、FR、GB、IT、US
CepCode	個人	BR
Cnpj	個人	BR
cpfCode	個人	BR
DriversLicense	個人	AT、AU、BE、 BG、CA、CY、 CZ、DE、DK、EE、ES、FI、 FR、GB、GR、 HR、HU、IE、IT、LT、LU、

識別子	Category	国と言語
		LV、MT、NL、 PL、PT、RO、SE、SI、SK、 US
DrugEnforcementAge ncyNumber	健康	US
ElectoralRollNumber	個人	GB
HealthInsuranceCardNumber	健康	EU
HealthInsuranceClaimNumber	健康	US
HealthInsuranceNumber	健康	FR
HealthcareProcedureCode	健康	US
IndividualTaxIdentification Number	個人	US
inSeeCode	個人	FR
MedicareBeneficiaryNumber	健康	US
NationalDrugCode	健康	US
NationalIdentificationNumber	個人	DE、ES、IT
NationalInsuranceNumber	個人	GB
NationalProviderId	健康	米国
NhsNumber	健康	GB
nieNumber	個人	ES
nifNumber	個人	ES
PassportNumber	個人	CA、DE、ES、 FR、GB、IT、US

識別子	Category	国と言語
PermanentResidenceNumber	個人	CA
PersonalHealthNumber	健康	CA
PhoneNumber	個人	BR、DE、ES、FR、GB、IT、US
PostalCode	個人	CA
rgNumber	個人	BR
SocialInsuranceNumber	個人	CA
SSN	個人	es-US
TaxID	個人	DE、ES、FR、GB
ZipCode	個人	米国

認証情報

CloudWatch Logs のデータ保護では、次の種類の認証情報を検出できます。

データの種類	データ識別子 ID	キーワードが必須	国とリージョン
AWS シークレットアクセスキー	AwsSecretKey	aws_secret_access_key , credentials , secret access key, secret key, set-awscredential	すべて
OpenSSH プライベートキー	OpenSSHPrivateKey	なし	すべて
PGP プライベートキー	PgpPrivateKey	なし	すべて

データの種類	データ識別子 ID	キーワードが必須	国とリージョン
Pkcs プライベートキー	PkcsPrivateKey	なし	すべて
PuTTY プライベートキー	PuttyPrivateKey	なし	すべて

認証情報データタイプのデータ識別子 ARN

以下は、データ保護ポリシーに追加できるデータ識別子の Amazon リソースネーム (ARN) のリストを示しています。

認証情報データ識別子 ARN

```
arn:aws:dataprotection::aws:data-identifier/AwsSecretKey
```

```
arn:aws:dataprotection::aws:data-identifier/OpenSshPrivateKey
```

```
arn:aws:dataprotection::aws:data-identifier/PgpPrivateKey
```

```
arn:aws:dataprotection::aws:data-identifier/PkcsPrivateKey
```

```
arn:aws:dataprotection::aws:data-identifier/PuttyPrivateKey
```

デバイス識別子

CloudWatch Logs のデータ保護では、次の種類のデバイス識別子を検出できます。

データの種類	データ識別子 ID	キーワードが必須	国とリージョン
IP アドレス	IpAddress	なし	すべて

デバイスデータタイプのデータ識別子 ARN

以下は、データ保護ポリシーに追加できるデータ識別子の Amazon リソースネーム (ARN) のリストを示しています。

デバイスデータ識別子 ARN

```
arn:aws:dataprotection::aws:data-identifier/IpAddress
```

財務情報

CloudWatch Logs のデータ保護では、次の種類の財務情報を検索できます。

データ保護ポリシーを設定すると、CloudWatch Logs では、ロググループがどの地理的位置にあるかに関係なく、指定したデータ識別子がないかスキャンします。この表の国と地域列の情報は、データ識別子に 2 文字の国コードを追加して、それらの国や地域に適したキーワードを検出する必要がありますかどうかを示しています。

データの種類	データ識別子 ID	キーワードが必須	国とリージョン	注意事項
銀行口座番号	BankAccountNumber	はい。国によって適用されるキーワードは異なります。詳細については、このセクションの後半にある銀行口座番号のキーワードの表を参照してください。	フランス、ドイツ、イタリア、スペイン、英国、米国	国コードなどの要素を含む、最大 34 文字の英数字で構成される国際銀行口座番号 (IBAN) が含

データの種類	データ識別子 ID	キーワードが必須	国とリージョン	注意事項
				まれます。
クレジットカードの有効期限	CreditCardExpiration	exp d, exp m, exp y, expiration , expiry	すべて	

データの種類	データ識別子 ID	キーワードが必須	国とリージョン	注意事項
クレジットカード番号	CreditCardNumber	account number, american express, amex, bank card, card, card number, card num, cc #, ccn, check card, credit, credit card#, dankort, debit, debit card, diners club, discover, electron, japanese card bureau, jcb, mastercard , mc, pan, payment account number, payment card number, pcn, union pay, visa	すべて	検出では、データは Luhn チェック式に従った 13～19 桁のシーケンスである必要があります。American Express、Dankort、Diner's Club、Discover、Electron、Japanese Card Bureau (JCB)、Mastercard、UnionPay、Visa のいずれかの種

データの種類	データ識別子 ID	キーワードが必須	国とリージョン	注意事項
				類のクレジットカードに 対する標準の カード番号 プレフィックス が使用されま す。
クレジットカード認証コード	CreditCardSecurityCode	card id, card identification code, card identification number , card security code, card validation code , card validation number , card verification data , card verification value, cvc, cvc2, cvv, cvv2, elo verification code	すべて	

銀行口座番号のキーワード

銀行口座番号には、次のキーワードを使用します。これには、国コードなどの要素を含む、最大 34 文字の英数字で構成される国際銀行口座番号 (IBAN) が含まれます。

Country	キーワード
フランス	account code, account number, accountno# , accountnumber# , bban, code bancaire, compte bancaire, customer account id, customer account number, customer bank account id, iban, numéro de compte
ドイツ	account code, account number, accountno# , accountnumber# , bankleitzahl , bban, customer account id, customer account number, customer bank account id, geheimzahl , iban, kartenum mer , kontonummer , kreditkartenummer , sepa
イタリア	account code, account number, accountno# , accountnumber# , bban, codice bancario, conto bancario, customer account id, customer account number, customer bank account id, iban, numero di conto
スペイン	account code, account number, accountno# , accountnumber# , bban, código cuenta, código cuenta bancaria, cuenta cliente id, customer account ID, customer account number, customer bank account id, iban, número cuenta bancaria cliente, número cuenta cliente
英国	account code, account number, accountno# , accountnumber# , bban, customer account ID, customer account number, customer bank account id, iban, sepa
アメリカ	bank account, bank acct, checking account, checking acct, deposit account, deposit acct, savings account, savings acct, chequing account, chequing acct

CloudWatch Logs では、クレジットカード発行会社がパブリックテストのために予約している、次のシーケンスの出現はレポートされません。

```
1220000000000003, 2222405343248877, 2222990905257051, 2223007648726984,  
2223577120017656,  
30569309025904, 34343434343434, 3528000700000000, 3530111333300000, 3566002020360505,  
36148900647913,
```

```
36700102000000, 371449635398431, 378282246310005, 378734493671000, 38520000023237,
4012888888881881,
4111111111111111, 4222222222222, 4444333322221111, 4462030000000000, 4484070000000000,
49118300000000,
4917300800000000, 4917610000000000, 4917610000000000003, 5019717010103742,
5105105105105100,
5111010030175156, 5185540810000019, 5200828282828210, 5204230080000017,
5204740009900014, 5420923878724339,
5454545454545454, 5455330760000018, 55069004900000436, 55069004900000444,
55069005100000234, 5506920809243667,
5506922400634930, 5506927427317625, 5553042241984105, 5555553753048194,
555555555554444, 5610591081018250,
6011000990139424, 6011000400000000, 6011111111111117, 630490017740292441,
630495060000000000,
6331101999990016, 6759649826438453, 6799990100000000019, and 76009244561.
```

財務データタイプのデータ識別子 ARN

以下は、データ保護ポリシーに追加できるデータ識別子の Amazon リソースネーム (ARN) のリストを示しています。

財務データ識別子 ARN

```
arn:aws:dataprotection::aws:data-identifier/BankAccountNumber-DE
```

```
arn:aws:dataprotection::aws:data-identifier/BankAccountNumber-ES
```

```
arn:aws:dataprotection::aws:data-identifier/BankAccountNumber-FR
```

```
arn:aws:dataprotection::aws:data-identifier/BankAccountNumber-GB
```

```
arn:aws:dataprotection::aws:data-identifier/BankAccountNumber-IT
```

```
arn:aws:dataprotection::aws:data-identifier/BankAccountNumber-US
```

```
arn:aws:dataprotection::aws:data-identifier/CreditCardExpiration
```

```
arn:aws:dataprotection::aws:data-identifier/CreditCardNumber
```

```
arn:aws:dataprotection::aws:data-identifier/CreditCardSecurityC
ode
```

保護対象保健情報 (PHI)

CloudWatch Logs のデータ保護では、次の種類の保護対象保健情報 (PHI) が検出できます。

データ保護ポリシーを設定すると、CloudWatch Logs では、ロググループがどの地理的位置にあるかに関係なく、指定したデータ識別子がないかスキャンします。この表の国と地域列の情報は、データ識別子に 2 文字の国コードを追加して、それらの国や地域に適したキーワードを検出する必要がありますかどうかを示しています。

データの種類	データ識別子 ID	キーワードが必須	国とリージョン
麻薬取締局 (DEA) 登録番号	DrugEnforcementAgencyNumber	dea number, dea registration	アメリカ
健康保険証番号 (EHIC)	HealthInsuranceCardNumber	assicurazione sanitaria numero, carta assicurazione numero, carte d'assurance maladie , carte européenne d'assurance maladie , ceam, ehic, ehic#, finlandehicnumber# , gesundheitskarte , hälsokort , health card, health card number, health insurance card, health insurance number, insurance card number, krankenversicherungskarte , krankenversicherungnummer , medical account number, numero conto medico, numéro d'assuran	欧州連合

データの種類	データ識別子 ID	キーワードが必須	国とリージョン
		ce maladie , numéro de carte d'assurance , numéro de compte medical, número de cuenta médica, número de seguro de salud, número de tarjeta de seguro, sairaanho itokortin , sairausva kuutuskortti , sairausvakuutusnumero , sjukförsäkringsnummer, sjukförsäkringskort , suomi ehic-numero , tarjeta de salud, terveyskortti , tessera sanitaria assicurazione numero , versicherungsnummer	
健康保険請求番号 (HICN)	HealthInsuranceClaimNumber	health insurance claim number, hic no, hic no., hic number, hic#, hich, hichn#, hichno#	アメリカ
健康保険または医療識別番号	HealthInsuranceNumber	carte d'assuré social, carte vitale, insurance card	フランス

データの種類	データ識別子 ID	キーワードが必須	国とリージョン
ヘルスケア共通手順コーディングシステム (HCPCS) コード	HealthcareProcedureCode	current procedural terminology , hcpcs, healthcare common procedure coding system	アメリカ
メディケア受給者番号 (MBN)	MedicareBeneficiaryNumber	mbi, medicare beneficiary	アメリカ
全米医薬品コード (NDC)	NationalDrugCode	national drug code, ndc	アメリカ
国家プロバイダー識別子 (NPI)	NationalProviderId	hipaa, n.p.i., national provider, npi	アメリカ
国民保健サービス (NHS) 番号	NhsNumber	national health service, NHS	グレートブリテン
個人健康管理番号	PersonalHealthNumber	canada healthcare number, msp number, care number, phn, soins de santé	カナダ

保護対象の医療情報 (PHI) データタイプのデータ識別子 ARN

保護対象保健情報 (PHI) データ保護ポリシーで利用できるデータ識別子 Amazon リソースネーム (ARN) を次に示します。

PHI データ識別子 ARN

```
arn:aws:dataprotection::aws:data-identifier/DrugEnforcementAgencyNumber-US
```

```
arn:aws:dataprotection::aws:data-identifier/HealthcareProcedureCode-US
```

```
arn:aws:dataprotection::aws:data-identifier/HealthInsuranceCardNumber-EU
```

```
arn:aws:dataprotection::aws:data-identifier/HealthInsuranceClaimNumber-US
```

```
arn:aws:dataprotection::aws:data-identifier/HealthInsuranceNumber-FR
```

```
arn:aws:dataprotection::aws:data-identifier/MedicareBeneficiaryNumber-US
```

```
arn:aws:dataprotection::aws:data-identifier/NationalDrugCode-US
```

```
arn:aws:dataprotection::aws:data-identifier/NationalInsuranceNumber-GB
```

```
arn:aws:dataprotection::aws:data-identifier/NationalProviderId-US
```

```
arn:aws:dataprotection::aws:data-identifier/NhsNumber-GB
```

```
arn:aws:dataprotection::aws:data-identifier/PersonalHealthNumber-CA
```

個人を特定できる情報 (PII)

CloudWatch Logs のデータ保護では、次の種類の個人を特定できる情報 (PII) を検出できます。

データ保護ポリシーを設定すると、CloudWatch Logs では、ロググループがどの地理的位置にあるかに関係なく、指定したデータ識別子がないかスキャンします。この表の国と地域列の情報は、デー

タ識別子に 2 文字の国コードを追加して、それらの国や地域に適したキーワードを検出する必要がありますかどうかを示しています。

データの種類	データ識別子 ID	キーワードが必須	国とリージョン	注意事項
生年月日	DateOfBirth	dob, date of birth, birthdate , birth date, birthday, b-day, bday	いずれか	Support には、すべての数字や数字と月の名前の組み合わせなど、ほとんどの日付形式が含まれます。日付コンポーネントは、スペース、スラッシュ (/)、またはハイフン (-) で区切ることが

データの種類	データ識別子 ID	キーワードが必須	国とリージョン	注意事項
				できます。
Código de Endereçamento Postal (CEP)	CepCode	cep, código de endereçamento postal, codigo de endereçamento postal	ブラジル	
Cadastro Nacional da Pessoa Jurídica (CNPJ)	Cnpj	cadastro nacional da pessoa jurídica, cadastro nacional da pessoa juridica, cnpj	ブラジル	
Cadastro de Pessoas Físicas (CPF)	CpfCode	Cadastro de pessoas físicas, cadastro de pessoas físicas, cadastro de pessoa física, cadastro de pessoa fisica, cpf	ブラジル	
運転免許証識別番号	DriversLicense	はい。国によって適用されるキーワードは異なります。詳細については、このセクションの後半にある運転免許証識別番号の表を参照してください。	多くの国。詳細については、運転免許証識別番号の表を参照してください。	

データの種類	データ識別子 ID	キーワードが必須	国とリージョン	注意事項
選挙人名簿番号	Electoral RollNumber	electoral #, electoral number, electoral roll #, electoral roll no., electoral roll number, electoral rollno	英国	
個人納税者識別	IndividualTaxIdentificationNumber	はい。国によって適用されるキーワードは異なります。詳細については、このセクションの後半にある個人納税者識別番号の表を参照してください。	ブラジル、フランス、ドイツ、スペイン、英国	
国立統計経済研究所 (INSEE)	InseeCode	はい。国によって適用されるキーワードは異なります。詳細については、このセクションの後半にある国民識別番号のキーワードの表を参照してください。	フランス	

データの種類	データ識別子 ID	キーワードが必須	国とリージョン	注意事項
国民識別番号	NationalIdentificationNumber	はい。詳細については、このセクションの後半にある国民識別番号のキーワードの表を参照してください。	ドイツ、イタリア、スペイン	これには、Documento Nacional de Identidad (DNI) 識別子 (スペイン)、Codice fiscale codes (イタリア)、国民 ID カード番号 (ドイツ語) が含まれます。

データの種類	データ識別子 ID	キーワードが必須	国とリージョン	注意事項
国民保険番号 (NINO)	NationalInsuranceNumber	insurance no., insurance number, insurance# , national insurance number, nationalinsurance# , nationalinsurance number , nin, nino	英国	–
Número de identidad de extranjero (NIE)	NieNumber	はい。国によって適用されるキーワードは異なります。詳細については、このセクションの後半にある個人納税者識別番号の表を参照してください。	スペイン	
Número de Identificación Fiscal (NIF)	NifNumber	はい。国によって適用されるキーワードは異なります。詳細については、このセクションの後半にある個人納税者識別番号の表を参照してください。	スペイン	

データの種類	データ識別子 ID	キーワードが必須	国とリージョン	注意事項
パスポート番号	PassportNumber	はい。国によって適用されるキーワードは異なります。詳細については、このセクションの後半にあるパスポート番号のキーワードの表を参照してください。	カナダ、フランス、ドイツ、イタリア、スペイン、英国、米国	
本籍地	Permanent Residence Number	carte résident permanent , numéro carte résident permanent , numéro résident permanent , permanent resident card, permanent resident card number, permanent resident no, permanent resident no., permanent resident number, pr no, pr no., pr non, pr number, résident permanent no., résident permanent non	カナダ	

データの種類	データ識別子 ID	キーワードが必須	国とリージョン	注意事項
Phone number (電話番号)	PhoneNumber	ブラジル: キーワードには、cel、celular、fone、residencial、numero residencial、telefone も含まれます。 その他: cell、contact、fax、fax number、mobile、phone、number、tel、telephone、telephone number	ブラジル、カナダ、フランス、ドイツ、イタリア、スペイン、英国、米国	これには、米国の通話料無料の番号とファックス番号が含まれます。キーワードがデータの近くにある場合、番号に国コードを含める必要はありません。キーワードがデータの近くにならない場合は、番

データの種類	データ識別子 ID	キーワードが必須	国とリージョン	注意事項
				号に国コードを含める必要があります。
郵便番号	PostalCode	なし	カナダ	
Registro Geral (RG)	RgNumber	はい。国によって適用されるキーワードは異なります。詳細については、このセクションの後半にある個人納税者識別番号の表を参照してください。	ブラジル	
社会保険番号 (SIN)	SocialInsuranceNumber	canadian id, numéro d'assurance sociale, social insurance number, sin	カナダ	
社会保障番号 (SSN)	Ssn	スペイン – número de la seguridad social、social security no.、social security no、número de la seguridad social、social security number、socialsecurityno# 、ssn、ssn# 米国 – social security、ss#、ssn	スペイン、米国	

データの種類	データ識別子 ID	キーワードが必須	国とリージョン	注意事項
納税者識別番号または参照番号	TaxId	はい。国によって適用されるキーワードは異なります。詳細については、このセクションの後半にある個人納税者識別番号の表を参照してください。	フランス、ドイツ、スペイン、英国	これには、TIN (フランス)、Steueridentifikationsnummer (ドイツ)、CIF (スペイン)、TRN と UTR (英国) が含まれます。
ZIP コード	ZipCode	zip code, zip+4	アメリカ	米国の郵便番号。

データの種類	データ識別子 ID	キーワードが必須	国とリージョン	注意事項
郵送先住所	Address	なし	オーストラリア、カナダ、フランス、ドイツ、イタリア、スペイン、英国、米国	キーワードは必要ありませんが、検出では、住所には都市または場所の名前および ZIP コードまたは郵便番号を含める必要があります。
電子メールアドレス	EmailAddress	なし	いずれか	

データの種類	データ識別子 ID	キーワードが必須	国とリージョン	注意事項
全地球測位システム (GPS) 座標	LatLong	coordinate , coordinates , lat long, latitude longitude , location, position	いずれか	CloudWatch Logs では、緯度と経度の座標がペアとして保存され、それらが十進角 (DD) 形式の場合 (例: 41.948614 , -87.655311)、GPS 座標を検出できます。サポートには、度 10 進分 (DDM) 形式 (例: 41°56.9168'N

データの種類	データ識別子 ID	キーワードが必須	国とリージョン	注意事項
				87°39.3187'W)、または、度、分、秒 (DMS) 形式 (例: 41°56'55.0104"N 87°39'19.1196"W) の座標は含まれません。
フルネーム	Name	なし	いずれか	CloudWatch Logs で検出できるのはフルネームのみです。Support はラテン文字セットに限定されます。

データの種類	データ識別子 ID	キーワードが必須	国とリージョン	注意事項
車両識別番号 (VIN)	VehicleIdentificationNumber	Fahrgestellnummer , niv, numarul de identificare , numarul seriei de sasiu, serie sasiu, numer VIN, Número de Identificação do Veículo, Número de Identificación de Automóviles , numéro d'identification du véhicule, vehicle identification number, vin, VIN numeris	いずれか	CloudWatch Logs では、17 文字の シーケ ンスで 構成さ れ、ISO 3779 および 3780 規格 に従っ た VIN を検出 できま す。こ れら の規 格は、 世界中 で使用 するた めに設 計され ていま す。

運転免許証識別番号のキーワード

さまざまなタイプの運転免許証識別番号を検出するために、CloudWatch Logs では番号に密接に関連するキーワードが必要です。次のテーブルに、CloudWatch Logs で特定の国およびリージョンが認識されるキーワードのリストを示します。

国またはリージョン	キーワード
オーストラリア	dl# dl:, dl :, dlno# driver licence, driver license, driver permit, drivers lic., drivers licence, driver's licence, drivers license, driver's license, drivers permit, driver's permit, drivers permit number, driving licence, driving license, driving permit
オーストリア	führerschein, fuhrerschein, fuhrerschein republik österreich, fuhrerschein republik osterreich
ベルギー	fuehrerschein, fuehrerschein- nr, fuehrerscheinnummer, fuhrerschein, fuhrerschein, fuhrerschein- nr, fuhrerschein- nr, fuhrersch einnummer, fuhrerscheinnummer, numéro permis conduire, permis de conduire, rijbewijs, rijbewijsnummer
ブルガリア	превозно средство, свидетелство за управление на моторно, свидетелство за управление на мпс, сумпс, шофьорска книжка
カナダ	dl#, dl:, dlno#, driver licence, driver licences, driver license, driver licenses, driver permit, drivers lic., drivers licence, driver's licence, drivers licences, driver's licences, drivers license, driver's license, drivers licenses, driver's licenses, drivers permit, driver's permit,

国またはリージョン	キーワード
	drivers permit number, driving licence, driving license, driving permit, permis de conduire
クロアチア	vozačka dozvola
キプロス	άδεια οδήγησης
チェコ共和国	číslo licence, číslo licence řidiče, číslo řidičskéh o průkazu, ovladače lic., povolení k jízdě, povolení řidiče, řidiči povolení, řidičský průkaz, řidičský průkaz
デンマーク	kørekort, kørekortnummer
エストニア	juhi litsentsi number, juhiloa number, juhiluba, juhiluba number
フィンランド	ajokortin numero, ajokortti, förare lic., körkort, körkort nummer, kuljettaja lic., permis de conduire
フランス	permis de conduire
ドイツ	fuehrerschein, fuehrerschein- nr, fuehrersc heinnummer, fuhrerschein, fuhrerschein, fuhrerschein- nr, fuhrerschein- nr, fuhrersch einnummer, fuhrerscheinnummer
ギリシャ	δεια οδήγησης, adeia odigisis
ハンガリー	illesztőprogramok lic, jogosítvány, jogsí, licencszám, vezető engedély, vezetői engedély
アイルランド	ceadúnas tiomána
イタリア	patente di guida, patente di guida numero, patente guida, patente guida numero

国またはリージョン	キーワード
ラトビア	autovadītāja apliecība, licences numurs, vadītāja apliecība, vadītāja apliecības numurs, vadītāja atļauja, vadītāja licences numurs, vadītāji lic.
リトアニア	vairuotojo pažymėjimas
ルクセンブルグ	fahrerlaubnis, fährerschäin
マルタ	licenzja tas-sewqan
オランダ	permis de conduire, rijbewijs, rijbewijsnummer
ポーランド	numer licencyjny, prawo jazdy, zezwolenie na prowadzenie
ポルトガル	carta de condução, carteira de habilitação, carteira de motorist, carteira habilitação, carteira motorist, licença condução, licença de condução, número de licença, número licença, permissão condução, permissão de condução
ルーマニア	numărul permisului de conducere, permis de conducere
スロバキア	číslo licencie, číslo vodičského preukazu, ovládače lic., povolenia vodičov, povolenie jazdu, povolenie na jazdu, povolenie vodiča, vodičský preukaz
スロベニア	vozniško dovoljenje

国またはリージョン	キーワード
スペイン	carnet conducir, el carnet de conducir, licencia conducir, licencia de manejo, número carnet conducir, número de carnet de conducir, número de permiso conducir, número de permiso de conducir, número licencia conducir, número permiso conducir, permiso conducción, permiso conducir, permiso de conducción
スウェーデン	ajokortin numero, dlno# ajokortti, drivere lic., förare lic., körkort, körkort nummer, körkortsn ummer, kuljettajat lic.
英国	dl#, dl:, dlno#, driver licence, driver licences, driver license, driver licenses, driver permit, drivers lic., drivers licence, driver's licence, drivers licences, driver's licences, drivers license, driver's license, drivers licenses, driver's licenses, drivers permit, driver's permit, drivers permit number, driving licence, driving license, driving permit
アメリカ	dl#, dl:, dlno#, driver licence, driver licences, driver license, driver licenses, driver permit, drivers lic., drivers licence, driver's licence, drivers licences, driver's licences, drivers license, driver's license, drivers licenses, driver's licenses, drivers permit, driver's permit, drivers permit number, driving licence, driving license, driving permit

国民識別番号のキーワード

さまざまなタイプの国民識別番号を検出するために、CloudWatch Logs では番号に密接に関連するキーワードが必要です。これには、Documento Nacional de Identidad (DNI) 識別子 (スペイン)、フ

ランス国立統計経済研究所 (INSEE) コード、ドイツの国民 ID カード番号、Registro Geral (RG) 番号 (ブラジル) が含まれます。

次のテーブルに、CloudWatch Logs で特定の国およびリージョンが認識されるキーワードのリストを示します。

国またはリージョン	キーワード
ブラジル	registro geral, rg
フランス	assurance sociale, carte nationale d'identité, cni, code sécurité sociale, French social security number, fssn#, insee, insurance number, national id number, nationalid#, numéro d'assurance, sécurité sociale, sécurité sociale non., sécurité sociale numéro, social, social security, social security number, socialsecuritynumber, ss#, ssn, ssn#
ドイツ	ausweisnummer, id number, identification number, identity number, insurance number, personal id, personalausweis
イタリア	codice fiscale, dati anagrafici, ehic, health card, health insurance card, p. iva, partita i.v.a., personal data, tax code, tessera sanitaria
スペイン	dni, dni#, dninúmero#, documento nacional de identidad, identidad único, identidadúnico#, insurance number, national identification number, national identity, nationalid#, nationalidno#, número nacional identidad, personal identification number, personal identity no, unique identity number, uniqueid#

パスポート番号のキーワード

さまざまなタイプのパスポート番号を検出するために、CloudWatch Logs では番号に密接に関連するキーワードが必要です。次のテーブルに、CloudWatch Logs で特定の国およびリージョンが認識されるキーワードのリストを示します。

国またはリージョン	キーワード
カナダ	passport, passport#, passport, passport#, passportno, passportno#
フランス	numéro de passeport, passeport, passeport #, passeport #, passeportn °, passeport n °, passeportNon, passeport non
ドイツ	ausstellungsdatum, ausstellungsort, geburtsdatum, passport, passports, reisepass, reisepass-nr, reisepassnummer
イタリア	italian passport number, numéro passeport , numéro passeport italien, passaporto, passaporto italiana, passaporto numero, passport number, repubblica italiana passaporto
スペイン	españa pasaporte, libreta pasaporte, número pasaporte, pasaporte, passport, passport book, passport no, passport number, spain passport
英国	passeport #, passeport n °, passeportNon, passeport non, passeportn °, passport #, passport no, passport number, passport#, passportid
アメリカ	passport, travel document

納税者識別と参照番号のキーワード

さまざまなタイプの納税者識別番号と参照番号を検出するために、CloudWatch Logs では番号に密接に関連するキーワードが必要です。次のテーブルに、CloudWatch Logs で特定の国およびリージョンが認識されるキーワードのリストを示します。

国またはリージョン	キーワード
ブラジル	cadastro de pessoa física, cadastro de pessoa fisica, cadastro de pessoas físicas, cadastro de pessoas fisicas, cadastro nacional da pessoa jurídica, cadastro nacional da pessoa juridica, cnpj, cpf
フランス	numéro d'identification fiscale, tax id, tax identification number, tax number, tin, tin#
ドイツ	identifikationsnummer, steuer id, steueridentifikationsnummer, steuernummer, tax id, tax identification number, tax number
スペイン	cif, cif número, cifnúmero#, nie, nif, número de contribuyente, número de identidad de extranjero, número de identificación fiscal, número de impuesto corporativo, personal tax number, tax id, tax identification number, tax number, tin, tin#
英国	paye, tax id, tax id no., tax id number, tax identification, tax identification#, tax no., tax number, tax reference, tax#, taxid#, temporary reference number, tin, trn, unique tax reference, unique taxpayer reference, utr
アメリカ	個別の納税者識別番号、itin、i.t.i.n。

個人を特定できる情報 (PII) のデータ識別子 ARN

次の表は、データ保護ポリシーに追加できる個人を特定できる情報 (PII) データ識別子の Amazon リソースネーム (ARN) のリストを示しています。

PII データ識別子 ARN

```
arn:aws:dataprotection::aws:data-identifier/Address
```

```
arn:aws:dataprotection::aws:data-identifier/CepCode-BR
```

```
arn:aws:dataprotection::aws:data-identifier/Cnpj-BR
```

```
arn:aws:dataprotection::aws:data-identifier/CpfCode-BR
```

```
arn:aws:dataprotection::aws:data-identifier/DriversLicense-AT
```

```
arn:aws:dataprotection::aws:data-identifier/DriversLicense-AU
```

```
arn:aws:dataprotection::aws:data-identifier/DriversLicense-BE
```

```
arn:aws:dataprotection::aws:data-identifier/DriversLicense-BG
```

```
arn:aws:dataprotection::aws:data-identifier/DriversLicense-CA
```

```
arn:aws:dataprotection::aws:data-identifier/DriversLicense-CY
```

```
arn:aws:dataprotection::aws:data-identifier/DriversLicense-CZ
```

```
arn:aws:dataprotection::aws:data-identifier/DriversLicense-DE
```

```
arn:aws:dataprotection::aws:data-identifier/DriversLicense-DK
```

```
arn:aws:dataprotection::aws:data-identifier/DriversLicense-EE
```

```
arn:aws:dataprotection::aws:data-identifier/DriversLicense-ES
```

```
arn:aws:dataprotection::aws:data-identifier/DriversLicense-FI
```

```
arn:aws:dataprotection::aws:data-identifier/DriversLicense-FR
```

```
arn:aws:dataprotection::aws:data-identifier/DriversLicense-GB
```

PII データ識別子 ARN

arn:aws:dataprotection::aws:data-identifier/DriversLicense-GR

arn:aws:dataprotection::aws:data-identifier/DriversLicense-HR

arn:aws:dataprotection::aws:data-identifier/DriversLicense-HU

arn:aws:dataprotection::aws:data-identifier/DriversLicense-IE

arn:aws:dataprotection::aws:data-identifier/DriversLicense-IT

arn:aws:dataprotection::aws:data-identifier/DriversLicense-LT

arn:aws:dataprotection::aws:data-identifier/DriversLicense-LU

arn:aws:dataprotection::aws:data-identifier/DriversLicense-LV

arn:aws:dataprotection::aws:data-identifier/DriversLicense-MT

arn:aws:dataprotection::aws:data-identifier/DriversLicense-NL

arn:aws:dataprotection::aws:data-identifier/DriversLicense-PL

arn:aws:dataprotection::aws:data-identifier/DriversLicense-PT

arn:aws:dataprotection::aws:data-identifier/DriversLicense-RO

arn:aws:dataprotection::aws:data-identifier/DriversLicense-SE

arn:aws:dataprotection::aws:data-identifier/DriversLicense-SI

arn:aws:dataprotection::aws:data-identifier/DriversLicense-SK

arn:aws:dataprotection::aws:data-identifier/DriversLicense-US

arn:aws:dataprotection::aws:data-identifier/ElectoralRollNumber-GB

arn:aws:dataprotection::aws:data-identifier/EmailAddress

PII データ識別子 ARN

arn:aws:dataprotection::aws:data-identifier/IndividualTaxIdentificationNumber-US

arn:aws:dataprotection::aws:data-identifier/InseeCode-FR

arn:aws:dataprotection::aws:data-identifier/LatLong

arn:aws:dataprotection::aws:data-identifier/Name

arn:aws:dataprotection::aws:data-identifier/NationalIdentificationNumber-DE

arn:aws:dataprotection::aws:data-identifier/NationalIdentificationNumber-ES

arn:aws:dataprotection::aws:data-identifier/NationalIdentificationNumber-IT

arn:aws:dataprotection::aws:data-identifier/NieNumber-ES

arn:aws:dataprotection::aws:data-identifier/NifNumber-ES

arn:aws:dataprotection::aws:data-identifier/PassportNumber-CA

arn:aws:dataprotection::aws:data-identifier/PassportNumber-DE

arn:aws:dataprotection::aws:data-identifier/PassportNumber-ES

arn:aws:dataprotection::aws:data-identifier/PassportNumber-FR

arn:aws:dataprotection::aws:data-identifier/PassportNumber-GB

arn:aws:dataprotection::aws:data-identifier/PassportNumber-IT

arn:aws:dataprotection::aws:data-identifier/PassportNumber-US

arn:aws:dataprotection::aws:data-identifier/PermanentResidenceNumber-CA

PII データ識別子 ARN

arn:aws:dataprotection::aws:data-identifier/PhoneNumber-BR

arn:aws:dataprotection::aws:data-identifier/PhoneNumber-DE

arn:aws:dataprotection::aws:data-identifier/PhoneNumber-ES

arn:aws:dataprotection::aws:data-identifier/PhoneNumber-FR

arn:aws:dataprotection::aws:data-identifier/PhoneNumber-GB

arn:aws:dataprotection::aws:data-identifier/PhoneNumber-IT

arn:aws:dataprotection::aws:data-identifier/PhoneNumber-US

arn:aws:dataprotection::aws:data-identifier/PostalCode-CA

arn:aws:dataprotection::aws:data-identifier/RgNumber-BR

arn:aws:dataprotection::aws:data-identifier/SocialInsuranceNumber-CA

arn:aws:dataprotection::aws:data-identifier/Ssn-ES

arn:aws:dataprotection::aws:data-identifier/Ssn-US

arn:aws:dataprotection::aws:data-identifier/TaxId-DE

arn:aws:dataprotection::aws:data-identifier/TaxId-ES

arn:aws:dataprotection::aws:data-identifier/TaxId-FR

arn:aws:dataprotection::aws:data-identifier/TaxId-GB

arn:aws:dataprotection::aws:data-identifier/VehicleIdentificationNumber

arn:aws:dataprotection::aws:data-identifier/ZipCode-US

カスタムデータ識別子

トピック

- [SNS カスタムデータ識別子とは](#)
- [カスタムデータ識別子の制約](#)
- [コンソールでのカスタムデータ識別子の使用](#)
- [データ保護ポリシーでカスタムデータ識別子を使用する](#)

SNS カスタムデータ識別子とは

カスタムデータ識別子 (CDI) を使用すると、データ保護ポリシーで利用できる独自のカスタム正規表現を定義できます。カスタムデータ識別子を使用すると、[マネージドデータ識別子](#)では提供できないビジネス固有の個人を特定できる情報 (PII) のユースケースをターゲットにすることができます。たとえば、カスタムデータ識別子を使用すると、会社固有の従業員 ID を検索できます。カスタムデータ識別子は、マネージドデータ ID と組み合わせて使用できます。

カスタムデータ識別子の制約

CloudWatch Logs カスタムデータ識別子には、以下の制限があります。

- 各データ保護ポリシーに使用できるカスタムデータ識別子は最大 10 個です。
- カスタムデータ識別子名に使用できるのは 128 文字までです。以下の文字がサポートされています。
 - 英数字: (a-zA-Z0-9)
 - 記号: ('_' | '-' | '.')
- RegEx の最大長は 200 文字です。以下の文字がサポートされています。
 - 英数字: (a-zA-Z0-9)
 - 記号: ('_' | '#' | '=' | '@' | '/' | ';' | ':' | '-' | '|')
 - RegEx 予約文字: ('\' | '\$' | '?' | '[' | ']' | '{' | '}' | '|' | '\' | '*' | '+' | '!')
- カスタムデータ識別子は、マネージドデータ識別子と同じ名前を共有することはできません。
- カスタムデータ識別子は、アカウントレベルのデータ保護ポリシーまたはロググループレベルのデータ保護ポリシーで指定できます。マネージドデータ識別子と同様に、アカウントレベルのポリシーで定義されたカスタムデータ識別子は、ロググループレベルのポリシーで定義されたカスタムデータ識別子との組み合わせにより機能します。

コンソールでのカスタムデータ識別子の使用

CloudWatch コンソールを使用してデータ保護ポリシーを作成または編集する場合、カスタムデータ識別子を指定するには、データ識別子の名前と正規表現を入力します。例えば、名前には **Employee_ID** と入力し、正規表現として **EmployeeID-\d{9}** を入力します。この正規表現は、EmployeeID- の後に 9 つの数値を持つログイベントを検出してマスクします。例: EmployeeID-123456789

データ保護ポリシーでカスタムデータ識別子を使用する

AWS CLI または AWS API を使用してカスタムデータ識別子を指定する場合は、データ保護ポリシーの定義に使用される JSON ポリシーにデータ識別子名と正規表現を含める必要があります。次のデータ保護ポリシーは、会社固有の従業員 ID を含むログイベントを検出およびマスクします。

1. データ保護ポリシー内で Configuration ブロックを作成します。
2. カスタムデータ識別子の Name を入力します。例えば、**EmployeeId**。
3. カスタムデータ識別子の Regex を入力します。例えば、**EmployeeID-\d{9}**。この正規表現は、EmployeeID- の後に 9 桁の EmployeeID- を含むログイベントと一致します。例: EmployeeID-123456789
4. ポリシーステートメントで、以下のカスタムデータ識別子を参照します。

```
{
  "Name": "example_data_protection_policy",
  "Description": "Example data protection policy with custom data identifiers",
  "Version": "2021-06-01",
  "Configuration": {
    "CustomDataIdentifier": [
      {
        "Name": "EmployeeId",
        "Regex": "EmployeeId-\\d{9}"
      }
    ]
  },
  "Statement": [
    {
      "Sid": "audit-policy",
      "DataIdentifier": [
        "EmployeeId"
      ],
      "Operation": {
        "Audit": {
          "FindingsDestination": {
            "S3": {
              "Bucket": "EXISTING_BUCKET"
            }
          }
        }
      }
    }
  ]
}
```

```
    }
  }
}
},
{
  "Sid": "redact-policy",
  "DataIdentifier": [
    "EmployeeId"
  ],
  "Operation": {
    "Deidentify": {
      "MaskConfig": {
      }
    }
  }
}
]
```

5. (オプション) 必要に応じて、Configuration ブロックにさらに カスタムデータ識別子を追加します。現在、データ保護ポリシーでは最大 10 個のカスタムデータ識別子を使用できます。

取り込み中のログの変換

ログ変換とエンリッチメントを使用すると、CloudWatch Logs への取り込み時に、すべてのログを一貫したコンテキスト豊富な形式で正規化できます。AWS WAF や Amazon Route 53 などの一般的な AWS サービス用に事前設定されたテンプレートを使用してログに構造を追加したり、[Grok](#) などのネイティブパーサーを使用してカスタムトランスフォーマーを構築したりできます。既存の属性の名前を変更し、アカウント ID、リージョンなどのメタデータをログに追加することもできます。

ログ変換は、アプリケーション全体のログクエリを簡素化および短縮し、ログへのアラートの作成を簡素化するのに役立ちます。この機能は、VPC フローログ、Route 53、などの主要な AWS ログソース用のout-of-the-box変換テンプレートを使用して、一般的なログタイプの変換を提供します Amazon RDS for PostgreSQL。事前設定された変換テンプレートを使用するか、ニーズに合わせてカスタムトランスフォーマーを作成できます。

ログ変換は、形式や属性名が大きく異なるさまざまなソースから出力されるログを管理するのに役立ちます。

トランスフォーマーを作成すると、取り込まれたログイベントが変換され、標準形式で保存されます。これらの変換されたログを活用し、次の機能を使用して分析エクスペリエンスを加速できます。

- [フィールドインデックス](#)
- [CloudWatch Logs Insights 検出フィールド](#)
- [メトリクスフィルター](#)を使用したアラームの柔軟性
- [サブスクリプションフィルター](#)による転送
- [Contributor Insights](#) を使用してログイベントからメトリクスデータを作成すると、Contributor Insights ルールにおいてログイベント評価をログイベントの変換前または変換後に行うかを選択できます。

変換はログの取り込み中にのみ行われます。既に取り込まれたログイベントを変換することはできません。変換は元に戻せません。元のログと変換されたログはどちらも、同じ保持ポリシーで CloudWatch Logs に保存されます。ログ変換とエンリッチメント機能は、既存の標準ログクラスの取り込み料金に含まれています。ログストレージコストは、変換後のログサイズに基づいており、元のログボリュームを超える可能性があります。

⚠ Important

ログイベントが変換されたら、CloudWatch Logs Insights クエリを使用して、変換されたログのバージョンを参照する必要があります。[GetLogEvents](#) アクションと [FilterLogEvents](#) アクションは、変換される前のログイベントの元のバージョンのみを返します。

⚠ Important

PutLogEvents では最大 1MB のログイベントは 1 つしか発生しませんが、ログ変換は 512 KB 未満のサイズのログイベントのみを処理できます。512 KB を超えるログイベントは変換に失敗し、エラーが発生します。PutLogEvents の合計サイズは引き続き 512 KB を超える可能性があります。

さまざまな形式への変換に加えて、アカウント ID、リージョン、キーワードといった追加のコンテキストでログを強化することもできます。これらは、ロググループ名と静的キーワードから抽出されます。

ログ変換は、形式や属性名が大きく異なるさまざまなソースから出力されるログを管理するのに役立ちます。

ログ変換とエンリッチメントは、標準ログクラスのロググループでのみサポートされます。

個々のロググループのトランスフォーマーを作成できます。また、アカウント内のすべてのロググループまたは多数のロググループに適用されるアカウントレベルのトランスフォーマーを作成することもできます。ロググループにロググループレベルのトランスフォーマーがある場合、そのトランスフォーマーはそのロググループに適用されるアカウントレベルのトランスフォーマーを上書きします。

トピック

- [ログトランスフォーマーの作成と管理](#)
- [設定可能なパーサータイプのプロセッサ](#)
- [AWS 販売ログ用の組み込みプロセッサ](#)
- [文字列ミューテーションプロセッサ](#)
- [JSON ミューテーションプロセッサ](#)
- [データ型コンバータプロセッサ](#)

- [変換メトリクスとエラー](#)

ログトランスフォーマーの作成と管理

ログトランスフォーマーには、論理パイプラインにある 1 つ以上のプロセッサが含まれています。各プロセッサは、トランスフォーマー設定にリストされている順序でログイベントに適用されます。

一部のプロセッサはパーサータイプです。各トランスフォーマーには少なくとも 1 つのパーサーが必要です。トランスフォーマーの最初のプロセッサはパーサーである必要があります。

一部のパーサーは、特定タイプの AWS で提供されるログ用に設定された組み込みパーサーです。

その他のプロセッサタイプは、文字列ミューテーター、JSON ミューテーター、データプロセッサです。

個々のロググループのトランスフォーマーを作成できます。また、アカウント内のすべてのロググループまたは多数のロググループに適用されるアカウントレベルのトランスフォーマーを作成することもできます。ロググループにロググループレベルのトランスフォーマーがある場合、そのトランスフォーマーはそのロググループに適用されるアカウントレベルのトランスフォーマーを上書きします。アカウント内のリージョンには、最大 20 個のアカウントレベルのトランスフォーマーを設定できます。

トランスフォーマーを作成するときは、以下のガイドラインに従う必要があります。

- AWS 販売ログのタイプに事前設定済みのパーサーを含める場合は、トランスフォーマーにリストされている最初のプロセッサである必要があります。トランスフォーマーに含めることができるプロセッサは 1 つだけです。
- トランスフォーマーに含めることができる grok プロセッサは 1 つだけです。
- トランスフォーマーには少なくとも 1 つのパーサータイプのプロセッサが必要です。最大 5 つのパーサータイプのプロセッサを含めることができます。この 5 つの制限には、組み込みパーサーと設定可能なパーサーの両方が含まれます。
- トランスフォーマーには最大 20 個のプロセッサを含めることができます。
- トランスフォーマーに含めることができる addKeys プロセッサは 1 つだけです。
- トランスフォーマーに含めることができる copyValue プロセッサは 1 つだけです。
- 各トランスフォーマーは、ログイベントから最大 200 個のフィールドを抽出できます。
- 各ログイベントは 512KB 未満である必要があります。ログイベントの合計サイズは引き続き 512KB を超える可能性があります。

トピック

- [アカウントレベルのトランスフォーマーポリシーを作成する](#)
- [アカウントレベルのトランスフォーマーポリシーの編集または削除](#)
- [log-group-level ログトランスフォーマーを最初から作成する](#)
- [既存のトランスフォーマーをコピーして log-group-level トランスフォーマーを作成する](#)
- [log-group-level トランスフォーマーを編集する](#)
- [log-group-level トランスフォーマーを削除する](#)

アカウントレベルのトランスフォーマーポリシーを作成する

このセクションのステップを使用して、アカウント内のすべてのロググループ、または同じ文字列 (プレフィックス) で始まるロググループ名を持つ複数のロググループに適用されるトランスフォーマーポリシーを作成します。1 つのリージョンには最大 20 個のアカウントレベルのトランスフォーマーポリシーを設定できます。

同じプレフィックスが使用されている同一のリージョン、あるいは別のトランスフォーマーポリシーに 1 つのプレフィックスが含まれている同一のリージョンに 2 つのトランスフォーマーポリシーを作成することはできません。例えば、文字列プレフィックス /aws/lambda に対して 1 つのトランスフォーマーポリシーを作成する場合、プレフィックス /aws を使用して別のトランスフォーマーポリシーを作成することはできません。ただし、/aws/lambda 用に 1 つのトランスフォーマーを設定し、/aws/waf 用に別のトランスフォーマーを設定することができます。

アカウントレベルのトランスフォーマーポリシーを作成するには

1. CloudWatch コンソールの <https://console.aws.amazon.com/cloudwatch/> を開いてください。
2. ナビゲーションペインで、[設定] を選択し、[ログ] を選択します。
3. [アカウントのトランスフォーマーポリシー] セクションで、[トランスフォーマーポリシーの作成] を選択します。
4. [トランスフォーマーポリシー名] に新しいポリシーの名前を入力します。
5. [ロググループを選択] で、次のいずれかの操作を行います。
 - [すべての標準ロググループ] を選択して、トランスフォーマーポリシーをアカウントのすべての標準クラスロググループに適用します。
 - [プレフィックス一致でロググループを選択] を選択し、同じ文字列で始まる名前を持つロググループのサブセットにポリシーを適用します。次に、[選択基準] にこれらのロググループのプレフィックスを入力します。

6. [パーサーの選択] エリアで、[パーサー]を使用してトランスフォーマーに含めるパーサーを選択します。

あるタイプの AWS 販売済みログ用に事前設定されたパーサーである場合は、その設定を指定する必要はありません。

別のパーサーの場合は、その設定を指定する必要があります。詳細については、「[設定可能なパーサータイプのプロセッサ](#)」でそのプロセッサの情報を参照してください。

7. 別のプロセッサを追加するには、[プロセッサの選択] を選択します。次に、[プロセッサ] ボックスで目的のプロセッサを選択し、設定パラメータを入力します。

プロセッサは、トランスフォーマーに追加する順序でログイベントで動作することに注意してください。

8. (オプション) プロセッサを追加するには、[+ プロセッサ] を選択し、前のステップを繰り返します。
9. (オプション) これまでにサンプルログイベントで構築したトランスフォーマーは、いつでもテストできます。これを行うには、[トランスフォーマーのプレビュー] セクションで次のいずれかを実行します。
 - [ロググループの選択] で最大 5 つのロググループを選択し、[最新のログイベントのロード] を選択します。次に、[トランスフォーマーのテスト] を選択します。
 - ログイベントを [サンプルログイベント] に直接コピーし、[トランスフォーマーのテスト] を選択します。

変換されたバージョンのログが表示されます。

10. プロセッサの追加が完了し、サンプルログのテストに満足したら、[保存] を選択します。
11. 終了したら、[Create] (作成) を選択します。

アカウントレベルのトランスフォーマーポリシーの編集または削除

このセクションのステップを使用して、アカウントレベルのトランスフォーマーポリシーを編集または削除します。

アカウントレベルのトランスフォーマーポリシーを編集または削除するには

1. CloudWatch コンソールの <https://console.aws.amazon.com/cloudwatch/> を開いてください。
2. ナビゲーションペインで、[設定] を選択し、[ログ] を選択します。

3. [トランスフォーマーアカウントポリシー] セクションで、[管理] を選択します。
4. 管理するトランスフォーマーポリシーでボタンを選択し、[編集] または [削除] を選択します。

ポリシーを編集する場合は、[設定可能なパーサータイプのプロセッサ](#) のステップ 5～11 を参照してオプションを確認してください。

log-group-level ログトランスフォーマーを最初から作成する

以下のステップを使用して、log-group-level トランスフォーマーを最初から作成します。

コンソールを使用してロググループのログトランスフォーマーを作成するには

1. CloudWatch コンソールの <https://console.aws.amazon.com/cloudwatch/> を開いてください。
2. ナビゲーションペインで、[ログ]、[ロググループ] の順に選択します。
3. トランスフォーマーを作成するロググループを選択します。
4. [トランスフォーマー] タブを選択します。タブリストを表示するには、タブリストを右にスクロールする必要がある場合があります。
5. [トランスフォーマーの作成] を選択します。
6. [パーサーの選択] ボックスで、トランスフォーマーに含めるパーサーを選択します。

あるタイプの AWS 販売済みログ用に事前設定されたパーサーである場合は、その設定を指定する必要はありません。

別のパーサーの場合は、その設定を指定する必要があります。詳細については、「[設定可能なパーサータイプのプロセッサ](#)」でそのプロセッサの情報を参照してください。

7. 別のプロセッサを追加するには、[+ プロセッサの追加] を選択します。次に、[プロセッサの選択] ボックスで目的のプロセッサを選択し、設定パラメータを入力します。

プロセッサは、トランスフォーマーに追加する順序でログイベントで動作することに注意してください。

8. (オプション) これまでにサンプルログイベントで構築したトランスフォーマーは、いつでもテストできます。このためには、以下を実行します。

- [変換プレビュー] セクションで、[サンプルログをロード] を選択してこのトランスフォーマーが属するロググループからサンプルログイベントをロードするか、ログイベントをテキストボックスに貼り付けます。

[トランスフォーマーのテスト] を選択します。変換されたバージョンのログが表示されます。

9. プロセッサの追加が完了し、サンプルログのテストに満足したら、[保存] を選択します。

を使用してログトランスフォーマーを最初から AWS CLI 作成するには

- aws logs put-transformer コマンドを使用します。parseJSON を最初のプロセッサとして使用する場合は、@message をソースフィールドとして使用してログイベント全体を解析する必要があります。最初の JSON 解析後、後続のプロセッサで特定のフィールドを操作できます。以下は、parseJSON および addKeys プロセッサを含むトランスフォーマーを作成する例です。

```
aws logs put-transformer \
  --transformer-config '[{"parseJSON":{"source":"@message"}}, {"addKeys":
{"entries":[{"key":"metadata.transformed_in","value":"CloudWatchLogs"},
{"key":"feature","value":"Transformation"}]}], {"trimString":{"withKeys":
["status"]}]}' \
  --log-group-identifier my-log-group-name
```

既存のトランスフォーマーをコピーして log-group-level トランスフォーマーを作成する

コンソールを使用して、既存のトランスフォーマーの JSON 設定をコピーできます。その後、そのコードを使用して、を使用して同一のトランスフォーマーを作成するか AWS CLI、最初に設定を変更できます。

既存のトランスフォーマーをコピーしてログトランスフォーマーを作成するには

- CloudWatch コンソールの <https://console.aws.amazon.com/cloudwatch/> を開いてください。
- ナビゲーションペインで、[ログ]、[ロググループ] の順に選択します。
- コピーするトランスフォーマーがあるロググループを選択します。
- [変換] タブを選択します。タブリストを表示するには、タブリストを右にスクロールする必要があります。
- [トランスフォーマーの管理] を選択します。

6. [トランスフォーマーのコピー] を選択します。これにより、トランスフォーマー JSON がクリップボードにコピーされます。
7. ファイルを作成し、トランスフォーマー設定に貼り付けます。この例では、ファイル `CopiedTransformer.json` を呼び出します。
8. AWS CLI を使用して、その設定で新しいトランスフォーマーを作成します。

```
aws logs put-transformer --log-group-identifier my-log-group-name \
--transformer-config file://CopiedTransformer.json
```

log-group-level トランスフォーマーを編集する

既存のログトランスフォーマーを編集するには、次の手順に従います。

ログトランスフォーマーを編集するには

1. CloudWatch コンソールの <https://console.aws.amazon.com/cloudwatch/> を開いてください。
2. ナビゲーションペインで、[ログ]、[ロググループ] の順に選択します。
3. 編集するトランスフォーマーがあるロググループを選択します。
4. [変換] タブを選択します。タブリストを表示するには、タブリストを右にスクロールする必要があります。
5. [トランスフォーマーの管理] を選択します。
6. [パーサー] および [プロセッサ] セクションで、変更を加えます。
7. 別のプロセッサを追加するには、[+ プロセッサの追加] を選択します。次に、[プロセッサ] ボックスで目的のプロセッサを選択し、設定パラメータを入力します。

プロセッサは、トランスフォーマーに追加する順序でログイベントで動作することに注意してください。

8. (オプション) これまでにサンプルログイベントで構築したトランスフォーマーは、いつでもテストできます。このためには、以下を実行します。
 - [変換プレビュー] セクションで、[サンプルログをロード] を選択してこのトランスフォーマーが属するロググループからサンプルログイベントをロードするか、ログイベントをテキストボックスに貼り付けます。

[変換のテスト] を選択します。変換されたバージョンのログが表示されます。

9. プロセッサの追加が完了し、サンプルログのテストに満足したら、[保存] を選択します。

log-group-level トランスフォーマーを削除する

ログトランスフォーマーを削除するには、次の手順に従います。

ログトランスフォーマーを削除するには

1. CloudWatch コンソールの <https://console.aws.amazon.com/cloudwatch/> を開いてください。
2. ナビゲーションペインで、[ログ]、[ロググループ] の順に選択します。
3. 編集するトランスフォーマーがあるロググループを選択します。
4. [変換] タブを選択します。タブリストを表示するには、タブリストを右にスクロールする必要があります。
5. [削除] を選択します。
6. 確認ボックスで [ポリシーの削除] を選択します。

設定可能なパーサータイプのプロセッサ

このセクションでは、ログイベントトランスフォーマーで利用できる設定可能なデータパーサープロセッサについて説明します。

目次

- [parseJSON](#)
- [grok](#)
 - [Grok の例](#)
 - [例 1: grok を使用して非構造化ログからフィールドを抽出する](#)
 - [例 2: grok を parseJSON と組み合わせて使用し、JSON ログイベントからフィールドを抽出する](#)
 - [例 3: FIELD_NAME に点線注釈が付いた Grok パターン](#)
 - [サポートされている grok パターン](#)
 - [一般的なログ形式の例](#)
 - [Apache ログの例](#)
 - [NGINX ログの例](#)
 - [Syslog Protocol \(RFC 5424\) ログの例](#)
- [CSV](#)
- [parseKeyValue](#)

parseJSON

parseJSON プロセッサは JSON ログイベントを解析し、抽出された JSON キーと値のペアを送信先に挿入します。送信先を指定しない場合、プロセッサはキーと値のペアをルートノードに配置します。parseJSON を最初のプロセッサとして使用する場合は、@message をソースフィールドとして使用してログイベント全体を解析する必要があります。最初の JSON 解析後、後続のプロセッサで特定のフィールドを操作できます。

元の @message コンテンツは変更されず、新しいキーがメッセージに追加されます。

フィールド	説明	必須?	デフォルト	制限
ソース	解析されるログイベントのフィールドへのパス。ドット表記を使用して子フィールドにアクセスします。例: store.book	いいえ	@message	最大長: 128 ネストされたキーの最大深度: 3
destination	解析された JSON の送信先フィールド	いいえ	Parent JSON node	最大長: 128 ネストされたキーの最大深度: 3

例

取り込まれたログイベントは次のようになります。

```
{
  "outer_key": {
    "inner_key": "inner_value"
  }
}
```

次に、この parseJSON プロセッサがある場合:

```
[
  {
    "parseJSON": {
      "destination": "new_key"
    }
  }
]
```

```
    }
  }
]
```

変換されたログイベントは次のとおりです。

```
{
  "new_key": {
    "outer_key": {
      "inner_key": "inner_value"
    }
  }
}
```

grok

grok プロセッサを使用すると、パターンマッチングを使用して非構造化データを解析および構造化できます。このプロセッサは、ログメッセージからフィールドを抽出することもできます。

フィールド	説明	必須?	デフォルト	制限	注意事項
ソース	Grok マッチングを適用するフィールドのパス	いいえ	@message	最大長: 128 ネストされたキーの最大深度: 3	
match	ログイベントと照合する grok パターン	はい		最大長: 512 最大 grok パターン: 20 一部の grok パターンタイプには個別の使用制限があります。次のパターンの任意の組み合わせを最大 5 回使用できます: {URI、URIP	サポートされているすべての Grok パターンを表示する

フィールド	説明	必須?	デフォルト	制限	注意事項
				ARAM、URIP ATHPARAM、 SPACE、DAT A、GREEDYD ATA、GREED YDATA_MUL TILINE} Grok パターンは型 変換をサポートし ていません。 一般的なログ 形式パターン (APACHE_A CCESS_LOG 、NGINX_AC CESS_LOG、 SYSLOG5424) では、共通のロ グパターンの後 に DATA、GREE DYDATA、また は GREEDYDAT A_MULTILINE パ ターンのみを含め ることができま す。	

Grok パターンの構造

サポートされている grok パターン構造は次のとおりです。

%{PATTERN_NAME:FIELD_NAME}

- **PATTERN_NAME**: 特定タイプのデータを一致させるための事前定義された正規表現を指します。定義済みの [grok パターン](#) のみがサポートされています。カスタムパターンの作成は許可されません。
- **FIELD_NAME**: 抽出された値に名前を割り当てます。**FIELD_NAME** はオプションですが、この値を指定しない場合、抽出されたデータは変換されたログイベントから削除されます。**FIELD_NAME** がドット表記 (「parent.child」など) を使用している場合、JSON パスと見なされます。
- **型変換**: 明示的な型変換はサポートされていません。[TypeConverter プロセッサ](#) を使用して、grok によって抽出された任意の値のデータ型を変換します。

より複雑なマッチング式を作成するには、複数の grok パターンを組み合わせることができます。ログイベントに合わせて最大 20 個の grok パターンを組み合わせることができます。例えば、`%{NUMBER:timestamp} [%{NUMBER:db} %{IP:client_ip}: %{NUMBER:client_port}] %{GREEDYDATA:data}` のパターンの組み合わせを使用して、次のような Redis スローログエントリからフィールドを抽出できます。

```
1629860738.123456 [0 127.0.0.1:6379] "SET" "key1" "value1"
```

Grok の例

例 1: grok を使用して非構造化ログからフィールドを抽出する

サンプルログ:

```
293750 server-01.internal-network.local OK "[Thread-000] token generated"
```

使用するトランスフォーマー:

```
[
  {
    "grok": {
      "match": "%{NUMBER:version} %{HOSTNAME:hostname} %{NOTSPACE:status}
%{QUOTEDSTRING:logMsg}"
    }
  }
]
```

出力:

```
{
  "version": "293750",
```

```
{
  "hostname": "server-01.internal-network.local",
  "status": "OK",
  "logMsg": "[Thread-000] token generated"
}
```

サンプルログ:

```
23/Nov/2024:10:25:15 -0900 172.16.0.1 200
```

使用するトランスフォーマー:

```
[
  {
    "grok": {
      "match": "%{HTTPDATE:timestamp} %{IPORHOST:clientip} %{NUMBER:response_status}"
    }
  }
]
```

出力:

```
{
  "timestamp": "23/Nov/2024:10:25:15 -0900",
  "clientip": "172.16.0.1",
  "response_status": "200"
}
```

例 2: grok を parseJSON と組み合わせて使用し、JSON ログイベントからフィールドを抽出する

サンプルログ:

```
{
  "timestamp": "2024-11-23T16:03:12Z",
  "level": "ERROR",
  "logMsg": "GET /page.html HTTP/1.1"
}
```

使用するトランスフォーマー:

```
[
  {
```

```
    "parseJSON": {}
  },
  {
    "grok": {
      "source": "logMsg",
      "match": "%{WORD:http_method} %{NOTSPACE:request} HTTP/"
%{NUMBER:http_version}"
    }
  }
]
```

出力:

```
{
  "timestamp": "2024-11-23T16:03:12Z",
  "level": "ERROR",
  "logMsg": "GET /page.html HTTP/1.1",
  "http_method": "GET",
  "request": "/page.html",
  "http_version": "1.1"
}
```

例 3: FIELD_NAME に点線注釈が付いた Grok パターン

サンプルログ:

```
192.168.1.1 GET /index.html?param=value 200 1234
```

使用するトランスフォーマー:

```
[
  {
    "grok": {
      "match": "%{IP:client.ip} %{WORD:method} %{URIPATHPARAM:request.uri}"
%{NUMBER:response.status} %{NUMBER:response.bytes}"
    }
  }
]
```

出力:

```
{
```

```
"client": {
  "ip": "192.168.1.1"
},
"method": "GET",
"request": {
  "uri": "/index.html?param=value"
},
"response": {
  "status": "200",
  "bytes": "1234"
}
}
```

サポートされている grok パターン

次の表に、grok プロセッサでサポートされているパターンを示します。

一般的な grok パターン

Grok パターン	説明	最大パターン制限	例
USERNAME または USER	小文字 (a~z)、大文字 (A~Z)、数字 (0~9)、ドット (.)、アンダースコア (_)、またはハイフン (-) を使用できる 1 つ以上の文字との一致	20	入力: user123.name-TEST パターン: %{USERNAME E:name} 出力: {"name": "user123.name-TEST"}
INT	オプションのプラス記号またはマイナス記号の後に 1 つ以上の数字が続くものとの一致。	20	入力: -456 パターン: %{INT:num} 出力: {"num": "-456"}
BASE10NUM	整数または浮動小数点数とオプションの符号と小数点との一致	20	入力: -0.67 パターン: %{BASE10NUM:num} 出力: {"num": "-0.67"}

Grok パターン	説明	最大パターン制限	例
BASE16NUM	10 進数と 16 進数をオプションの記号 (+ または -) とオプションの 0x プレフィックスとの一致	20	入力: +0xA1B2 パターン: %{BASE16NUM:num} 出力: {"num": "+0xA1B2"}
POSINT	1 つ以上の数字 (1~9 の後に 0~9) で構成され、先頭にゼロのない正の整数全体との一致	20	入力: 123 パターン: %{POSINT:num} 出力: {"num": "123"}
NONNEGINT	ゼロと先頭にゼロがある数字を含むすべての整数 (1 つ以上の数字 0~9 で構成) との一致。	20	入力: 007 パターン: %{NONNEGINT:num} 出力: {"num": "007"}
WORD	文字、数字、アンダースコアを含む 1 つ以上の単語文字 (\w) で構成される単語全体との一致	20	入力: user_123 パターン: %{WORD:user} 出力: {"user": "user_123"}
NOTSPACE	空白以外の 1 つ以上の文字との一致。	5	入力: hello_world123 パターン: %{NOTSPACE:msg} 出力: {"msg": "hello_world123"}

Grok パターン	説明	最大パターン制限	例
SPACE	0 個以上の空白類文字との一致。	5	入力: " " パターン: %{SPACE:extra} 出力: {"extra": " "}
DATA	任意の文字 (改行文字を除く) に 0 回以上一致。最短マッチを実行。	5	入力: abc def ghi パターン: %{DATA:x} %{DATA:y} 出力: {"x": "abc", "y": "def ghi"}
GREEDYDATA	任意の文字 (改行文字を除く) に 0 回以上一致。最長マッチを実行。	5	入力: abc def ghi パターン: %{GREEDYDATA:x} %{GREEDYDATA:y} 出力: {"x": "abc def", "y": "ghi"}
GREEDYDATA_MULTILINE	任意の文字 (改行文字を含む) に 0 回以上一致。最長マッチを実行。	1	入力: abc def ghi パターン: %{GREEDYDATA_MULTILINE:data} 出力: {"data": "abc\ndef\nghi"}

Grok パターン	説明	最大パターン制限	例
QUOTEDSTRING	エスケープ文字付きの、引用符で囲まれた文字列 (一重引用符または二重引用符) との一致。	20	入力: "Hello, world!" パターン: %{QUOTEDSTRING:msg} 出力: {"msg": "Hello, world!"}
UUID	標準の UUID 形式との一致。16 進数文字 8 文字の後に 16 進数文字 4 文字の 3 グループが続き、16 進数文字 12 文字で終わり、すべてハイフンで区切られます。	20	入力: 550e8400-e29b-41d4-a716-446655440000 パターン: %{UUID:id} 出力: {"id": "550e8400-e29b-41d4-a716-446655440000"}
URN	URN (Uniform Resource Name) 構文との一致。	20	入力: urn:isbn:0451450523 パターン: %{URN:urn} 出力: {"urn": "urn:isbn:0451450523"}

AWS Grok パターン

パターン	説明	最大パターン制限	例
ARN	AWS Amazon リソースネーム (ARNs) に一致し、パーティション (aws、aws-cn、または aws-us-gov)、サービ	5	入力: arn:aws:iam:us-east-1:123456789012:user/johndoe

パターン	説明	最大パターン制限	例
	ス、リージョン、アカウント ID、スラッシュで区切られた最大 5 つの階層リソース識別子をキャプチャします。コロン間で情報が欠落している ARN には一致しません。		パターン: %{ARN:arn} 出力: {"arn": "arn:aws:iam:us-east-1:123456789012:user/john doe"}

ネットワーク grok パターン

Grok パターン	説明	最大パターン制限	例
CISCOMAC	4-4-4 の 16 進形式の MAC アドレスとの一致。	20	入力: 0123.4567.89AB パターン: %{CISCOMAC:MacAddress} 出力: {"MacAddress": "0123.4567.89AB"}
WINDOWSMA C	ハイフン付きの 16 進形式の MAC アドレスとの一致。	20	入力: 01-23-45-67-89-AB パターン: %{WINDOWS MAC:MacAddress} 出力: {"MacAddress": "01-23-45-67-89-AB"}
COMMONMAC	コロン付きの 16 進形式の MAC アドレスとの一致。	20	入力: 01:23:45:67:89:AB パターン: %{COMMONM AC:MacAddress} 出力: {"MacAddress": "01:23:45:67:89:AB"}

Grok パターン	説明	最大パターン制限	例
MAC	CISCOMAC、WINDOWSMA C、または COMMONMAC grok パターンのいずれかとの 一致	20	入力: 01:23:45:67:89:AB パターン: %{MAC:m1} 出力: {"m1": "01 :23:45:67:89:AB"}
IPv6	圧縮フォームや IPv4 マップ IPv6 アドレスを含む IPv6 ア ドレスとの一致。	5	入力: 2001:db8: 3333:4444:5555:666 6:7777:8888 パターン: %{IPv6:ip} 出力: {"ip": "2001:db8 :3333:4444:5555:66 66:7777:8888"}
IPv4	IPv4 アドレスとの一致。	20	入力: 192.168.0.1 パターン: %{IPv4:ip} 出力: {"ip": "192.168. 0.1"}
IP	%{IPv6} でサポートされて いる IPv6 アドレスまたは %{IPv4} でサポートされてい る IPv4 アドレスのいずれかと の一致。	5	入力: 192.168.0.1 パターン: %{IP:ip} 出力: {"ip": "192.168. 0.1"}

Grok パターン	説明	最大パターン制限	例
HOSTNAME または HOST	サブドメインを含むドメイン名との一致	5	入力: server-01 .internal-network. local パターン: %{HOST:host} 出力: {"host": "server-01.internal-network.local"}
IPORHOST	ホスト名または IP アドレスのいずれかとの一致	5	入力: 2001:db8: 3333:4444:5555:666 6:7777:8888 パターン: %{IPORHOST:ip} 出力: {"ip": "2001:db8: :3333:4444:5555:66 66:7777:8888"}
HOSTPORT	%{IPORHOST} パターンでサポートされている IP アドレスまたはホスト名にコロンとポート番号が続くものに一致し、出力でポートを「PORT」としてキャプチャします。	5	入力: 192.168.1.1:8080 パターン: %{HOSTPORT:ip} 出力: {"ip": "192.168.1.1:8080", "PORT": "8080"}

Grok パターン	説明	最大パターン制限	例
URIHOST	%{IPORHOST} パターンでサポートされている IP アドレスまたはホスト名との一致。オプションでコロンとポート番号が続き、ポートが存在する場合はポートを「port」としてキャプチャします。	5	入力: example.com:443 10.0.0.1 パターン: %{URIHOST}:host} %{URIHOST:ip} 出力: {"host": "example.com:443", "port": "443", "ip": "10.0.0.1"}

パス grok パターン

Grok パターン	説明	最大パターン制限	例
UNIXPATH	クエリパラメータを含む可能性のある URL パスとの一致。	20	入力: /search?q=regex パターン: %{UNIXPATH}:path} 出力: {"path": "/search?q=regex"}
WINPATH	Windows ファイルパスとの一致。	5	入力: C:\Users\John\Documents\file.txt パターン: %{WINPATH}:path} 出力: {"path": "C:\\Users\\John\\Documents\\file.txt"}

Grok パターン	説明	最大パターン制限	例
PATH	URL または Windows ファイルパスのいずれかとの一致	5	入力: /search?q=regex パターン: %{PATH:path} 出力: {"path":"/search?q=regex"}
TTY	ターミナルと擬似ターミナルの Unix デバイスパスとの一致。	20	入力: /dev/tty1 パターン: %{TTY:path} 出力: {"path":"/dev/tty1"}
URIPROTO	文字との一致。オプションでプラス (+) 文字と追加文字、またはプラス (+) 文字が続きます。	20	入力: web+transformer パターン: %{URIPROTO:protocol} 出力: {"protocol":"web+transformer"}
URIPATH	URI のパスコンポーネントとの一致	20	入力: /category/sub-category/product_name パターン: %{URIPATH:path} 出力: {"path":"/category/sub-category/product_name"}

Grok パターン	説明	最大パターン制限	例
URIPARAM	URL クエリパラメータとの一致	5	入力: ?param1=value1&param2=value2 パターン: %{URIPARAM:url} 出力: {"url": "?param1=value1&param2=value2"}
URIPATHPARAM	URI パスとの一致。オプションでクエリパラメータが続きます。	5	入力: /category/sub-category/product?id=12345&color=red パターン: %{URIPATHPARAM:path} 出力: {"path": "/category/sub-category/product?id=12345&color=red"}
[URI]	完全な URI との一致	5	入力: https://user:password@example.com/path/to/resource?param1=value1&param2=value2 パターン: %{URI:uri} 出力: {"path": "https://user:password@example.com/path/to/resource?param1=value1&param2=value2"}

日付と時刻の grok パターン

Grok パターン	説明	最大パターン制限	例
MONTH	完全な英月名または省略された英月名の単語全体との一致	20	入力: Jan パターン: <code>%{MONTH:month}</code> 出力: <code>{"month": "Jan"}</code> 入力: January パターン: <code>%{MONTH:month}</code> 出力: <code>{"month": "January"}</code>
MONTHNUM	1～12 の月番号との一致。1 桁の月ではオプションで先頭に 0 が付きます。	20	入力: 5 パターン: <code>%{MONTHNUM:month}</code> 出力: <code>{"month": "5"}</code> 入力: 05 パターン: <code>%{MONTHNUM:month}</code> 出力: <code>{"month": "05"}</code>
MONTHNUM	01 から 12 までの 2 桁の月番号との一致。	20	入力: 05 パターン: <code>%{MONTHNUM2:month}</code> 出力: <code>{"month": "05"}</code>

Grok パターン	説明	最大パターン制限	例
MONTHDAY	1 から 31 までの日付との一致。オプションで先頭に 0 が付きます。	20	入力: 31 パターン: <code>%{MONTHDAY:monthDay}</code> 出力: <code>{"monthDay": "31"}</code>
YEAR	2 桁または 4 桁の年との一致	20	入力: 2024 パターン: <code>%{YEAR:year}</code> 出力: <code>{"year": "2024"}</code> 入力: 24 パターン: <code>%{YEAR:year}</code> 出力: <code>{"year": "24"}</code>
DAY	完全な日付名または省略された日付名との一致。	20	入力: Tuesday パターン: <code>%{DAY:day}</code> 出力: <code>{"day": "Tuesday"}</code>
HOURL	24 時間形式の時間との一致。オプションで先頭にゼロ (0)0 ~ 23 が付きます。	20	入力: 22 パターン: <code>%{HOURL:hour}</code> 出力: <code>{"hour": "22"}</code>

Grok パターン	説明	最大パターン制限	例
MINUTE	分 (00 ~ 59) との一致。	20	入力: 59 パターン: %{MINUTE: min} 出力: {"min": "59"}
SECOND	秒 (0)0 ~ 60 を表す数値との一致。オプションで小数点またはコロン、小数分の 1 桁以上が続きます。	20	入力: 3 パターン: %{SECOND: second} 出力: {"second": "3"} 入力: 30.5 パターン: %{SECOND: minSec} 出力: {"minSec": "30.5"} 入力: 30:5 パターン: %{SECOND: minSec} 出力: {"minSec": "30:5"}
TIME	時間、分、秒を (H)H:mm:(s)s のフォーマットで表した時間形式との一致。秒にはうるう秒 (0)0 ~ 60 が含まれます。	20	入力: 09:45:32 パターン: %{TIME: time} 出力: {"time": "09:45:32"}

Grok パターン	説明	最大パターン制限	例
DATE_US	(M)M/(d)d/(yy)yy または (M)M-(d)d-(yy)yy の形式の日付との一致。	20	入力: 11/23/2024 パターン: %{DATE_US:date} 出力: {"date": "11/23/2024"} 入力: 1-01-24 パターン: %{DATE_US:date} 出力: {"date": "1-01-24"}
DATE_EU	(d)d/(M)M/(yy)yy、(d)d-(M)M-(yy)yy、または (d)d.(M)M.(yy)yyの形式の日付との一致。	20	入力: 23/11/2024 パターン: %{DATE_EU:date} 出力: {"date": "23/11/2024"} 入力: 1.01.24 パターン: %{DATE_EU:date} 出力: {"date": "1.01.24"}

Grok パターン	説明	最大パターン制限	例
ISO8601_TIMEZONE	UTC オフセット 'Z' またはタイムゾーンオフセットとの一致。オプションで [+](H)H(:)mm 形式のコロンと一致させます。	20	入力: +05:30 パターン: %{ISO8601_TIMEZONE:tz} 出力: {"tz":"+05:30"} 入力: -530 パターン: %{ISO8601_TIMEZONE:tz} 出力: {"tz":"-530"} 入力: Z パターン: %{ISO8601_TIMEZONE:tz} 出力: {"tz":"Z"}
ISO8601_SECONDS	秒 (0)0 ~ 60 を表す数値との一致。オプションで小数点またはコロン、小数秒の 1 桁以上が続きます。	20	入力: 60 パターン: %{ISO8601_SECONDS:second} 出力: {"second":"60"}

Grok パターン	説明	最大パターン制限	例
TIMESTAMP_ISO8601	ISO8601 日時形式 (yy)yy-(M)M-(d)dT(H)H:mm:((s)s)(Z [-+](H)H:mm) との一致。オプションで、秒とタイムゾーンと一致させます。	20	入力: 2023-05-15T14:30:00+05:30 パターン: %{TIMESTAMP_ISO8601 1:timestamp} 出力: {"timestamp": "2023-05-15T14:30:00+05:30"} 入力: 23-5-1T1:25+5:30 パターン: %{TIMESTAMP_ISO8601 1:timestamp} 出力: {"timestamp": "23-5-1T1:25+5:30"} 入力: 23-5-1T1:25Z パターン: %{TIMESTAMP_ISO8601 1:timestamp} 出力: {"timestamp": "23-5-1T1:25Z"}

Grok パターン	説明	最大パターン制限	例
DATE	<code>%{DATE_US}</code> を使用した米国形式または <code>%{DATE_EU}</code> を使用した欧州形式の日付のいずれかとの一致	20	入力: 11/29/2024 パターン: <code>%{DATE:date}</code> 出力: <code>{"date": "11/29/2024"}</code> 入力: 29.11.2024 パターン: <code>%{DATE:date}</code> 出力: <code>{"date": "29.11.2024"}</code>
DATESTAMP	<code>%{DATE}</code> の後に <code>%{TIME}</code> パターンが続き、スペースまたはハイフンで区切られたものとの一致。	20	入力: 29-11-2024 14:30:00 パターン: <code>%{DATESTAMP:dateTime}</code> 出力: <code>{"dateTime": "29-11-2024 14:30:00"}</code>
TZ	一般的なタイムゾーンの略語 (PST、PDT、MST、MDT、CST、CDT、EST、EDT、UTC) との一致。	20	入力: PDT パターン: <code>%{TZ:tz}</code> 出力: <code>{"tz": "PDT"}</code>

Grok パターン	説明	最大パターン制限	例
DATESTAMP_RFC822	次の形式の日付と時刻との一致: Day MonthName (D)D (YY)YY (H)H:mm:(s)s Timezone	20	入力: Monday Jan 5 23 1:30:00 CDT パターン: %{DATESTAMP_RFC822:dateTime} 出力: {"dateTime": "Monday Jan 5 23 1:30:00 CDT"} 入力: Mon January 15 2023 14:30:00 PST パターン: %{DATESTAMP_RFC822:dateTime} 出力: {"dateTime": "Mon January 15 2023 14:30:00 PST"}

Grok パターン	説明	最大パターン制限	例
DATESTAMP_RFC2822	以下の RFC2822 日時形式との一致: Day、(d)d MonthName (yy)yy (H)H:mm:(s)s Z[+-(H)H:mm	20	入力: Mon, 15 May 2023 14:30:00 +0530 パターン: %{DATESTAMP_RFC2822:dateTime} 出力: {"dateTime": "Mon, 15 May 2023 14:30:00 +0530"} 入力: Monday, 15 Jan 23 14:30:00 Z パターン: %{DATESTAMP_RFC2822:dateTime} 出力: {"dateTime": "Monday, 15 Jan 23 14:30:00 Z"}
DATESTAMP_OTHER	次の形式の日付と時刻との一致: Day MonthName (d)d (H)H:mm:(s)s Timezone (yy)yy	20	入力: Mon May 15 14:30:00 PST 2023 パターン: %{DATESTAMP_OTHER:dateTime} 出力: {"dateTime": "Mon May 15 14:30:00 PST 2023"}

Grok パターン	説明	最大パターン制限	例
DATESTAMP_EVENTLOG	区切り文字なしのコンパクトな日時形式との一致: (yy)yyMM(d)d(H)Hm(s)s	20	入力: 20230515143000 パターン: %{DATESTAMP_EVENTLOG:dateTime} 出力: {"dateTime": "20230515143000"}

ログ grok パターン

Grok パターン	説明	最大パターン制限	例
logLevel	Alert/ALE RT、Trace/TRACE、Debug/DEBUG、Notice/NOTICE、Info/INFO、Warn/Warning/WARN/WARNING、Err/Error/ERROR、Critical/CRITICAL、Fatal/FATAL、Severe/SEVERE、Emerg/Emergency/EMERGENCY など、さまざまな大文字と小文字の標準ログレベルとの一致。	20	入力: INFO パターン: %{LOGLEVEL:logLevel} 出力: {"logLevel": "INFO"}

Grok パターン	説明	最大パターン制限	例
HTTPDATE	ログファイルでよく使用される日付と時刻の形式との一致。形式: (d)d/MonthName/(yy)yy: (H)H:mm:(s)s Timezone MonthName: 完全または省略された英語の月名 (例: 「Jan」または「January」) タイムゾーン: %{INT} grok パターンとの一致	20	入力: 23/Nov/2024:14:30:00 +0640 パターン: %{HTTPDATE:date} 出力: {"date":"23/Nov/2024:14:30:00+0640"}
SYSLOGTIMESTAMP	MMonthName (d)d (H)H:mm:(s)s を持つ日付形式との一致。MonthName: 完全な、または省略された英語の月名との一致 (例: 「Jan」または「January」)	20	入力: Nov 29 14:30:00 パターン: %{SYSLOGTIMESTAMP:dateTime} 出力: {"dateTime":"Nov 29 14:30:00"}
PROG	文字、数字、ドット、アンダースコア、スラッシュ、パーセント記号、ハイフンの文字列で構成されるプログラム名との一致。	20	入力: user.profile/settings-page パターン: %{PROG:program} 出力: {"program":"user.profile/settings-page"}

Grok パターン	説明	最大パターン制限	例
SYSLOGPROGRAM	PROG grok パターンと一致。オプションでプロセス ID が角括弧で囲まれます。	20	入力: user.profile/settings-page[1234] パターン: %{SYSLOGPROGRAM:programWithId} 出力: {"programWithId": "user.profile/settings-page[1234]", "program": "user.profile/settings-page", "pid": "1234"}
SYSLOGHOST	%{HOST} または %{IP} パターンのいずれかと一致。	5	入力: 2001:db8:3333:4444:5555:6666:7777:8888 パターン: %{SYSLOGHOST:ip} 出力: {"ip": "2001:db8:3333:4444:5555:6666:7777:8888"}

Grok パターン	説明	最大パターン制限	例
SYSLOGFACILITY	10 進形式の syslog 優先度との一致。値は角括弧 (<>) で囲む必要があります。	20	入力: <13.6> パターン: %{SYSLOGFACILITY:syslog} 出力: {"syslog": "<13.6>", "facility": "13", "priority": "6"}

一般的なログ grok パターン

事前定義されたカスタム grok パターンを使用して、Apache、NGINX、Syslog Protocol (RFC 5424) のログ形式を一致させることができます。これらの特定のパターンを使用する場合、それらは一致する設定の最初のパターンである必要があり、他のパターンの前に配置することはできません。また、1 つの DATA でのみ追跡できます。GREEDYDATA または GREEDYDATA_MULTILINE パターン。

Grok パターン	説明	最大パターン制限
APACHE_ACCESS_LOG	Apache アクセスログとの一致	1
NGINX_ACCESS_LOG	NGINX アクセスログとの一致	1
SYSLOG5424	Syslog Protocol (RFC 5424) ログとの一致	1

以下は、これらの一般的なログ形式パターンを使用するための有効および無効な例を示しています。

```
"%{NGINX_ACCESS_LOG} %{DATA}" // Valid
"%{SYSLOG5424}%{DATA:logMsg}" // Valid
"%{APACHE_ACCESS_LOG} %{GREEDYDATA:logMsg}" // Valid
"%{APACHE_ACCESS_LOG} %{SYSLOG5424}" // Invalid (multiple common log patterns used)
"%{NGINX_ACCESS_LOG} %{NUMBER:num}" // Invalid (Only GREEDYDATA and DATA patterns are supported with common log patterns)
"%{GREEDYDATA:logMsg} %{SYSLOG5424}" // Invalid (GREEDYDATA and DATA patterns are supported only after common log patterns)
```

一般的なログ形式の例

Apache ログの例

サンプルログ:

```
127.0.0.1 - - [03/Aug/2023:12:34:56 +0000] "GET /page.html HTTP/1.1" 200 1234
```

トランスフォーマー:

```
[
  {
    "grok": {
      "match": "%{APACHE_ACCESS_LOG}"
    }
  }
]
```

出力:

```
{
  "request": "/page.html",
  "http_method": "GET",
  "status_code": 200,
  "http_version": "1.1",
  "response_size": 1234,
  "remote_host": "127.0.0.1",
  "timestamp": "2023-08-03T12:34:56Z"
}
```

NGINX ログの例

サンプルログ:

```
192.168.1.100 - Foo [03/Aug/2023:12:34:56 +0000] "GET /account/login.html HTTP/1.1"
200 42 "https://www.amazon.com/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/92.0.4515.131 Safari/537.36"
```

トランスフォーマー:

```
[
  {
    "grok": {
      "match": "%{NGINX_ACCESS_LOG}"
    }
  }
]
```

出力:

```
{
  "request": "/account/login.html",
  "referrer": "https://www.amazon.com/",
  "agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
  Gecko) Chrome/92.0.4515.131 Safari/537.36",
  "http_method": "GET",
  "status_code": 200,
  "auth_user": "Foo",
  "http_version": "1.1",
  "response_size": 42,
  "remote_host": "192.168.1.100",
  "timestamp": "2023-08-03T12:34:56Z"
}
```

Syslog Protocol (RFC 5424) ログの例

サンプルログ:

```
<165>1 2003-10-11T22:14:15.003Z mymachine.example.com evntslog - ID47
[exampleSDID@32473 iut="3" eventSource= "Application" eventID="1011"]
[examplePriority@32473 class="high"]
```

トランスフォーマー:

```
[
  {
    "grok": {
      "match": "%{SYSLOG5424}"
    }
  }
]
```

出力:

```
{
  "pri": 165,
  "version": 1,
  "timestamp": "2003-10-11T22:14:15.003Z",
  "hostname": "mymachine.example.com",
  "app": "evntslog",
  "msg_id": "ID47",
  "structured_data": "exampleSDID@32473 iut=\"3\" eventSource= \"Application\" eventID= \"1011\"",
  "message": "[examplePriority@32473 class=\"high\"]"
}
```

CSV

csv プロセッサは、ログイベントからカンマ区切り値 (CSV) を解析して列に分けます。

フィールド	説明	必須?	デフォルト	制限
ソース	解析されるログイベントのフィールドへのパス	いいえ	@message	最大長: 128 ネストされたキーの最大深度: 3
delimiter	元のカンマ区切り値ログイベントの各列を区切るために使用される文字	いいえ	,	最大長: 1。値が \t または \s の場合を除く。

フィールド	説明	必須?	デフォルト	制限
quoteCharacter	データの単一の列のテキスト修飾子として使用される文字	いいえ	"	最大長: 1
columns	変換されたログイベントの列に使用する名前のリスト。	いいえ	[column_1, column_2]	最大 CSV 列: 100 最大長: 128 ネストされたキーの最大深度: 3
destination	変換されたキーと値のペアを配置する親フィールド	いいえ	Root node	最大長: 128 ネストされたキーの最大深度: 3

delimiter を \t に設定するとタブ文字の各列が区切られ、\t に設定すると 1 つの空白文字の各列が区切られます。

例

取り込まれたログイベントの一部は次のようになります。

```
'Akua Mansa':28:'New York: USA'
```

csv プロセッサのみを使用するとします。

```
[
  {
    "csv": {
      "delimiter": ":",
      "quoteCharacter": ""
    }
  }
]
```

変換されたログイベントは次のとおりです。

```
{
  "column_1": "Akua Mansa",
  "column_2": "28",
  "column_3": "New York: USA"
}
```

例 2

取り込まれたログイベントは次のようになります。

```
{
  "timestamp": "2024-11-23T16:03:12Z",
  "type": "user_data",
  "logMsg": "'Akua Mansa':28:'New York: USA'"
}
```

イベントを JSON として解析し、CSV プロセッサを使用して JSON フィールドを解析し、列名と宛先を指定するとします。

```
[
  {
    "parseJSON": {}
  },
  {
    "csv": {
      "source": "logMsg",
      "delimiter": ":",
      "quoteCharacter": "'",
      "columns": ["name", "age", "location"],
      "destination": "msg"
    }
  }
]
```

変換されたログイベントは次のとおりです。

```
{
  "timestamp": "2024-11-23T16:03:12Z",
  "logMsg": "'Akua Mansa':28:'New York: USA'",
  "type": "user_data",
  "msg": {
    "name": "Akua Mansa",
```



```

    "age": "28",
    "location": "New York: USA"
  }
}
```

parseKeyValue

parseKeyValue プロセッサを使用して、指定されたフィールドをキーと値のペアに解析します。以下のオプションを使用して、フィールド情報を解析するようにプロセッサをカスタマイズできます。

フィールド	説明	必須?	デフォルト	制限
ソース	解析されるログイベントのフィールドへのパス	いいえ	@message	最大長: 128 ネストされたキーの最大深度: 3
destination	抽出されたキーと値のペアを配置する送信先フィールド	いいえ		最大長: 128
FieldDelimiter	元のログイベントのキーと値のペア間で使用されるフィールド区切り文字の文字列	いいえ	&	最大長: 128
keyValueDelimiter	変換されたログイベントの各ペアのキーと値の間に使用する区切り文字の文字列	いいえ	=	最大長: 128
nonMatchValue	キーと値のペアが正常に分割されない場合に、結果の値フィールドに挿入する値。	いいえ		最大長: 128
keyPrefix	変換されたすべてのキーにプレフィックスを追加する場合は、ここで指定します。	いいえ		最大長: 128
overwriteIfExists	送信先キーが既に存在する場合に値を上書きするかどうか	いいえ	false	

例

ログイベントの例を以下に示します。

```
key1:value1!key2:value2!key3:value3!key4
```

次のプロセッサ設定を使用しているとします。

```
[
  {
    "parseKeyValue": {
      "destination": "new_key",
      "fieldDelimiter": "!",
      "keyValueDelimiter": ":",
      "nonMatchValue": "defaultValue",
      "keyPrefix": "parsed_"
    }
  }
]
```

変換されたログイベントは次のとおりです。

```
{
  "new_key": {
    "parsed_key1": "value1",
    "parsed_key2": "value2",
    "parsed_key3": "value3",
    "parsed_key4": "defaultValue"
  }
}
```

AWS 販売ログ用の組み込みプロセッサ

このセクションでは、ログを提供する AWS サービスで利用できる組み込みプロセッサについて説明します。

目次

- [parseWAF](#)
- [parsePostgres](#)

- [parseCloudfront](#)
- [parseRoute53](#)
- [parseVPC](#)
- [parseToOCSF](#)

parseWAF

このプロセッサを使用して AWS WAF、提供されたログを解析 `httpRequest.headers` し、の内容を取得し、対応する値を使用して各ヘッダー名から JSON キーを作成します。また、`labels` でも同じことを行います。これらの変換により、AWS WAF ログのクエリが大幅に簡単になります。AWS WAF ログ形式の詳細については、[「ウェブ ACL トラフィックのログ例」](#)を参照してください。

このプロセッサは、`@message` を入力としてのみ受け入れます。

Important

このプロセッサを使用する場合は、トランスフォーマーの最初のプロセッサである必要があります。

例

ログイベントの例を以下に示します。

```
{
  "timestamp": 1576280412771,
  "formatVersion": 1,
  "webaclId": "arn:aws:wafv2:ap-southeast-2:111122223333:regional/webacl/
STMTest/1EXAMPLE-2ARN-3ARN-4ARN-123456EXAMPLE",
  "terminatingRuleId": "STMTest_SQLi_XSS",
  "terminatingRuleType": "REGULAR",
  "action": "BLOCK",
  "terminatingRuleMatchDetails": [
    {
      "conditionType": "SQL_INJECTION",
      "sensitivityLevel": "HIGH",
      "location": "HEADER",
      "matchedData": ["10", "AND", "1"]
    }
  ]
}
```

```
],
"httpSourceName": "-",
"httpSourceId": "-",
"ruleGroupList": [],
"rateBasedRuleList": [],
"nonTerminatingMatchingRules": [],
"httpRequest": {
  "clientIp": "1.1.1.1",
  "country": "AU",
  "headers": [
    { "name": "Host", "value": "localhost:1989" },
    { "name": "User-Agent", "value": "curl/7.61.1" },
    { "name": "Accept", "value": "*/*" },
    { "name": "x-stm-test", "value": "10 AND 1=1" }
  ],
  "uri": "/myUri",
  "args": "",
  "httpVersion": "HTTP/1.1",
  "httpMethod": "GET",
  "requestId": "rid"
},
"labels": [{ "name": "value" }]
}
```

プロセッサ設定は次のとおりです。

```
[
  {
    "parseWAF": {}
  }
]
```

変換されたログイベントは次のとおりです。

```
{
  "httpRequest": {
    "headers": {
      "Host": "localhost:1989",
      "User-Agent": "curl/7.61.1",
      "Accept": "*/*",
      "x-stm-test": "10 AND 1=1"
    },
    "clientIp": "1.1.1.1",
```

```
    "country": "AU",
    "uri": "/myUri",
    "args": "",
    "httpVersion": "HTTP/1.1",
    "httpMethod": "GET",
    "requestId": "rid"
  },
  "labels": { "name": "value" },
  "timestamp": 1576280412771,
  "formatVersion": 1,
  "webaclId": "arn:aws:wafv2:ap-southeast-2:111122223333:regional/webacl/
STMTest/1EXAMPLE-2ARN-3ARN-4ARN-123456EXAMPLE",
  "terminatingRuleId": "STMTest_SQLi_XSS",
  "terminatingRuleType": "REGULAR",
  "action": "BLOCK",
  "terminatingRuleMatchDetails": [
    {
      "conditionType": "SQL_INJECTION",
      "sensitivityLevel": "HIGH",
      "location": "HEADER",
      "matchedData": ["10", "AND", "1"]
    }
  ],
  "httpSourceName": "-",
  "httpSourceId": "-",
  "ruleGroupList": [],
  "rateBasedRuleList": [],
  "nonTerminatingMatchingRules": []
}
```

parsePostgres

このプロセッサを使用して、Amazon RDS for PostgreSQL 提供されたログを解析し、フィールドを抽出して、JSON 形式に変換します。RDS for PostgreSQL ログ形式の詳細については、[「RDS for PostgreSQL データベースログファイル」](#)を参照してください。

このプロセッサは、@message を入力としてのみ受け入れます。

Important

このプロセッサを使用する場合は、トランスフォーマーの最初のプロセッサである必要があります。

例

ログイベントの例を以下に示します。

```
2019-03-10 03:54:59 UTC:10.0.0.123(52834):postgres@logtestdb:[20175]:ERROR: column
"wrong_column_name" does not exist at character 8
```

プロセッサ設定は次のとおりです。

```
[
  {
    "parsePostgres": {}
  }
]
```

変換されたログイベントは次のとおりです。

```
{
  "logTime": "2019-03-10 03:54:59 UTC",
  "srcIp": "10.0.0.123(52834)",
  "userName": "postgres",
  "dbName": "logtestdb",
  "processId": "20175",
  "logLevel": "ERROR"
}
```

parseCloudfront

このプロセッサを使用して、Amazon CloudFront 提供されたログを解析し、フィールドを抽出して、JSON 形式に変換します。エンコードされたフィールド値はデコードされます。整数と倍精度の値は、そのように扱われます。Amazon CloudFront ログ形式の詳細については、[「標準ログの設定と使用 \(アクセスログ\)」](#)を参照してください。

このプロセッサは、@message を入力としてのみ受け入れます。

Important

このプロセッサを使用する場合は、トランスフォーマーの最初のプロセッサである必要があります。

例

ログイベントの例を以下に示します。

```
2019-12-04 21:02:31 LAX1 392 192.0.2.24 GET
d111111abcdef8.cloudfront.net /index.html 200 - Mozilla/5.0%20(Windows
%20NT%2010.0;%20Win64;%20x64)%20AppleWebKit/537.36%20(KHTML,
%20like%20Gecko)%20Chrome/78.0.3904.108%20Safari/537.36 - - Hit
SOX4xwn4XV6Q4rgb7XiVG0Hms_BG1TAC4KyHmureZmBNrjGdRLiNIQ==
d111111abcdef8.cloudfront.net https 23 0.001 - TLSv1.2 ECDHE-RSA-AES128-GCM-
SHA256 Hit HTTP/2.0 - - 11040 0.001 Hit text/html 78 - -
```

プロセッサ設定は次のとおりです。

```
[
  {
    "parseCloudfront": {}
  }
]
```

変換されたログイベントは次のとおりです。

```
{
  "date": "2019-12-04",
  "time": "21:02:31",
  "x-edge-location": "LAX1",
  "sc-bytes": 392,
  "c-ip": "192.0.2.24",
  "cs-method": "GET",
  "cs(Host)": "d111111abcdef8.cloudfront.net",
  "cs-uri-stem": "/index.html",
  "sc-status": 200,
  "cs(Referer)": "-",
  "cs(User-Agent)": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/78.0.3904.108 Safari/537.36",
  "cs-uri-query": "-",
  "cs(Cookie)": "-",
  "x-edge-result-type": "Hit",
  "x-edge-request-id": "SOX4xwn4XV6Q4rgb7XiVG0Hms_BG1TAC4KyHmureZmBNrjGdRLiNIQ==",
  "x-host-header": "d111111abcdef8.cloudfront.net",
  "cs-protocol": "https",
  "cs-bytes": 23,
```

```
{
  "time-taken": 0.001,
  "x-forwarded-for": "-",
  "ssl-protocol": "TLSv1.2",
  "ssl-cipher": "ECDHE-RSA-AES128-GCM-SHA256",
  "x-edge-response-result-type": "Hit",
  "cs-protocol-version": "HTTP/2.0",
  "fle-status": "-",
  "fle-encrypted-fields": "-",
  "c-port": 11040,
  "time-to-first-byte": 0.001,
  "x-edge-detailed-result-type": "Hit",
  "sc-content-type": "text/html",
  "sc-content-len": 78,
  "sc-range-start": "-",
  "sc-range-end": "-"
}
```

parseRoute53

このプロセッサを使用して、Amazon Route 53 Public Data Plane 提供されたログを解析し、フィールドを抽出して、JSON 形式に変換します。エンコードされたフィールド値はデコードされます。このプロセッサは Amazon Route 53 Resolver ログをサポートしていません。

このプロセッサは、@message を入力としてのみ受け入れます。

Important

このプロセッサを使用する場合は、トランスフォーマーの最初のプロセッサである必要があります。

例

ログイベントの例を以下に示します。

```
1.0 2017-12-13T08:15:50.235Z Z123412341234 example.com AAAA NOERROR TCP IAD12 192.0.2.0
198.51.100.0/24
```

プロセッサ設定は次のとおりです。

```
[
```



```
{
  "parseRoute53": {}
}
```

変換されたログイベントは次のとおりです。

```
{
  "version": 1.0,
  "queryTimestamp": "2017-12-13T08:15:50.235Z",
  "hostZoneId": "Z123412341234",
  "queryName": "example.com",
  "queryType": "AAAA",
  "responseCode": "NOERROR",
  "protocol": "TCP",
  "edgeLocation": "IAD12",
  "resolverIp": "192.0.2.0",
  "ednsClientSubnet": "198.51.100.0/24"
}
```

parseVPC

このプロセッサを使用し、Amazon VPC で提供されるログを解析し、フィールドを抽出して JSON 形式に変換します。エンコードされたフィールド値はデコードされます。

このプロセッサは、@message を入力としてのみ受け入れます。

Important

このプロセッサを使用する場合は、トランスフォーマーの最初のプロセッサである必要があります。

例

ログイベントの例を以下に示します。

```
2 123456789010 eni-abc123de 192.0.2.0 192.0.2.24 20641 22 6 20 4249 1418530010
1418530070 ACCEPT OK
```

プロセッサ設定は次のとおりです。

```
[
  {
    "parseVPC": {}
  }
]
```

変換されたログイベントは次のとおりです。

```
{
  "version": 2,
  "accountId": "123456789010",
  "interfaceId": "eni-abc123de",
  "srcAddr": "192.0.2.0",
  "dstAddr": "192.0.2.24",
  "srcPort": 20641,
  "dstPort": 22,
  "protocol": 6,
  "packets": 20,
  "bytes": 4249,
  "start": 1418530010,
  "end": 1418530070,
  "action": "ACCEPT",
  "logStatus": "OK"
}
```

parseToOCSF

parseToOCSF プロセッサは、ログを Open Cybersecurity Schema Framework (OCSF) イベントに変換します。OCSF は、セキュリティデータの共通スキーマを提供するオープン標準であり、さまざまなセキュリティツールやプラットフォーム間の相互運用性と分析を高めることができます。

このプロセッサは、さまざまな AWS サービスのログ形式をダウンストリーム分析用の一貫したスキーマに標準化する必要があるセキュリティ分析ワークフローに特に役立ちます。

パラメータ

eventSource (必須)

変換するログイベントを生成する AWS サービスまたはプロセスを指定します。次の値を指定できます。

- CloudTrailCloudTrail のログ

- Route53Resolver - Route 53 Resolver のログ
- VPCFlow - Amazon VPC Flow Logs
- EKSAudit - Amazon EKS の監査ログ
- AWSWAF - AWS WAF ログ

ocsfVersion (必須)

変換されたログイベントに使用する OCSF スキーマのバージョンを指定します。現在サポートされているバージョン: V1.1, V1.5

mappingVersion (オプション)

OCSF 変換マッピングバージョンを指定します。ログを OCSF 形式に変換するときに適用される変換ロジックを制御します。指定しない場合、はポリシー作成時に利用可能な最新バージョンを使用します。新しいマッピングバージョンがリリースされると、既存のポリシーは自動的にアップグレードされません。現在の最新バージョン: v1.5.0。

注: ocsfVersionが の場合、サポートされていませんV1.1。

source (オプション)

解析するログイベントのフィールドへのパス。省略すると、ログメッセージ全体が解析されます。

例

次の例は、parseToOCSF を使用して VPC Flow Logs を OCSF 形式に変換する方法を示しています。

```
{
  "parseToOCSF": {
    "eventSource": "VPCFlow",
    "ocsfVersion": "V1.1"
  }
}
```

次の例は、一貫した変換動作のために特定のマッピングバージョンを指定する方法を示しています。

```
{
  "parseToOCSF": {
```

```
"eventSource": "CloudTrail",
"ocsfVersion": "V1.5",
"mappingVersion": "v1.5.0"
}
}
```

文字列ミューテーションプロセッサ

このセクションでは、ログイベントトランスフォーマーで使える文字列ミューテーションプロセッサについて説明します。

目次

- [lowerCaseString](#)
- [upperCaseString](#)
- [splitString](#)
- [substituteString](#)
- [trimString](#)

lowerCaseString

lowerCaseString プロセッサは文字列をその小文字バージョンに変換します。

フィールド	説明	必須?	デフォルト	制限
withKeys	小文字に変換するキーのリスト	はい		最大エントリ: 10

例

ログイベントの例を以下に示します。

```
{
  "outer_key": {
    "inner_key": "INNER_VALUE"
  }
}
```

```
}
```

トランスフォーマーの設定は次のようになります。lowerCaseString と parseJSON を併用します。

```
[
  {
    "parseJSON": {}
  },
  {
    "lowerCaseString": {
      "withKeys": ["outer_key.inner_key"]
    }
  }
]
```

変換されたログイベントは次のとおりです。

```
{
  "outer_key": {
    "inner_key": "inner_value"
  }
}
```

upperCaseString

upperCaseString プロセッサは文字列をその大文字バージョンに変換します。

フィールド	説明	必須?	デフォルト	制限
withKeys	大文字に変換するキーのリスト	はい		最大エントリ: 10

例

ログイベントの例を以下に示します。

```
{
```

```
"outer_key": {
  "inner_key": "inner_value"
}
}
```

トランスフォーマーの設定は次のようになります。upperCaseString と parseJSON を併用します。

```
[
  {
    "parseJSON": {}
  },
  {
    "upperCaseString": {
      "withKeys": ["outer_key.inner_key"]
    }
  }
]
```

変換されたログイベントは次のとおりです。

```
{
  "outer_key": {
    "inner_key": "INNER_VALUE"
  }
}
```

splitString

splitString プロセッサは、区切り文字を使用してフィールドを配列に分割する文字列ミューテーションプロセッサの一種です。

フィールド	説明	必須?	デフォルト	制限
エントリ	エントリの配列。配列内の各項目には、source フィールドと delimiter フィールドが含まれている必要があります。	はい		最大エントリ: 10

フィールド	説明	必須?	デフォルト	制限
ソース	分割するフィールド値のキー	はい		最大長: 128
delimiter	フィールド値を分割する区切り文字の文字列	はい		最大長: 128

例 1

ログイベントの例を以下に示します。

```
{
  "outer_key": {
    "inner_key": "inner_value"
  }
}
```

トランスフォーマーの設定は次のようになります。splitString と parseJSON を併用します。

```
[
  {
    "parseJSON": {}
  },
  {
    "splitString": {
      "entries": [
        {
          "source": "outer_key.inner_key",
          "delimiter": "_"
        }
      ]
    }
  }
]
```

変換されたログイベントは次のとおりです。

```
{
  "outer_key": {
```

```
    "inner_key": [
      "inner",
      "value"
    ]
  }
}
```

例 2

文字列を分割する区切り文字は複数文字にすることができます。

ロギイベントの例を以下に示します。

```
{
  "outer_key": {
    "inner_key": "item1, item2, item3"
  }
}
```

トランスフォーマーの設定は次のとおりです。

```
[
  {
    "parseJSON": {}
  },
  {
    "splitString": {
      "entries": [
        {
          "source": "outer_key.inner_key",
          "delimiter": ", "
        }
      ]
    }
  }
]
```

変換されたロギイベントは次のとおりです。

```
{
  "outer_key": {
    "inner_key": [
```



```
    "item1",  
    "item2",  
    "item3"  
  ]  
}  
}
```

substituteString

substituteString プロセッサは文字列ミューテーションプロセッサの一種で、キーの値を正規表現と照合し、すべての一致を置換文字列に置き換えます。

フィールド	説明	必須?	デフォルト	制限
エントリ	エントリの配列。配列内の各項目には、source、from、to フィールドが含まれている必要があります。	はい		最大エントリ: 10
ソース	変更するフィールドのキー	はい		最大長: 128 ネストされたキーの最大深度: 3
送信元	置き換える正規表現の文字列。 [and] などの特殊な正規表現文字は、二重引用符を使用する場合は\\、一重引用符を使用する場合や AWS マネジメントコンソールから設定する場合は \ を使用してエスケープする必要があります。詳細については、Oracle のウェブサイトの「 Class Pattern 」を参照してください。 パターンを (...) でラップして番号付きキャプチャグループを作成し、to フィールドで参照できる	はい		最大長: 128

フィールド	説明	必須?	デフォルト	制限
	(?P<group_name>...) の名前付きキャプチャグループを作成できます。			
まで	キャプチャグループへの from バックリファレンスの一致ごとに置き換えられる文字列を使用できます。\$1 などの番号付きグループには \$n という形式を使用し、\${my_group} .> などの名前付きグループには \${group_name} を使用します。	はい		最大長: 128 バックリファレンスの最大数: 10 重複するバックリファレンスの最大数: 2

例 1

ログイベントの例を以下に示します。

```
{
  "outer_key": {
    "inner_key1": "[]",
    "inner_key2": "123-345-567",
    "inner_key3": "A cat takes a catnap."
  }
}
```

トランスフォーマーの設定は次のようになります。substituteString と parseJSON を併用します。

```
[
  {
    "parseJSON": {}
  },
  {
    "substituteString": {
      "entries": [
        {
          "source": "outer_key.inner_key1",
```

```

        "from": "\\[\\]",
        "to": "value1"
    },
    {
        "source": "outer_key.inner_key2",
        "from": "[0-9]{3}-[0-9]{3}-[0-9]{3}",
        "to": "xxx-xxx-xxx"
    },
    {
        "source": "outer_key.inner_key3",
        "from": "cat",
        "to": "dog"
    }
]
}
]

```

変換されたログイベントは次のとおりです。

```

{
  "outer_key": {
    "inner_key1": "value1",
    "inner_key2": "xxx-xxx-xxx",
    "inner_key3": "A dog takes a dognap."
  }
}

```

例 2

ログイベントの例を以下に示します。

```

{
  "outer_key": {
    "inner_key1": "Tom, Dick, and Harry",
    "inner_key2": "arn:aws:sts::123456789012:assumed-role/MyImportantRole/MySession"
  }
}

```

トランスフォーマーの設定は次のようになります。substituteString と parseJSON を併用します。

```
[
  {
    "parseJSON": {}
  },
  {
    "substituteString": {
      "entries": [
        {
          "source": "outer_key.inner_key1",
          "from": "(\\w+), (\\w+), and (\\w+)",
          "to": "$1 and $3"
        },
        {
          "source": "outer_key.inner_key2",
          "from": "^arn:aws:sts::(?P<account_id>\\d{12}):assumed-role/(?P<role_name>[\\w+=,\\.@-]+)/(?P<role_session_name>[\\w+=,\\.@-]+)$",
          "to": "${account_id}:${role_name}:${role_session_name}"
        }
      ]
    }
  }
]
```

変換されたログイベントは次のとおりです。

```
{
  "outer_key": {
    "inner_key1": "Tom and Harry",
    "inner_key2": "123456789012:MyImportantRole:MySession"
  }
}
```

trimString

trimString プロセッサは、キーの先頭と末尾から空白を削除します。

フィールド	説明	必須?	デフォルト	制限
withKeys	キーのリストをトリムするには	はい		最大エントリ: 10

例

ログイベントの例を以下に示します。

```
{
  "outer_key": {
    "inner_key": "  inner_value  "
  }
}
```

トランスフォーマーの設定は次のようになります。trimString と parseJSON を併用します。

```
[
  {
    "parseJSON": {}
  },
  {
    "trimString": {
      "withKeys":["outer_key.inner_key"]
    }
  }
]
```

変換されたログイベントは次のとおりです。

```
{
  "outer_key": {
    "inner_key": "inner_value"
  }
}
```

JSON ミューテーションプロセッサ

このセクションでは、ログイベントトランスフォーマーで使える JSON ミューテーションプロセッサについて説明します。

目次

- [addKeys](#)
- [DeleteKeys](#)

- [moveKeys](#)
- [renameKeys](#)
- [copyValue](#)
- [listToMap](#)

addKeys

addKeys プロセッサを使用して、ログイベントに新しいキーと値のペアを追加します。

フィールド	説明	必須?	デフォルト	制限
エントリ	エントリの配列。配列内の各項目には、key、value、overwrite IfExists フィールドを含めることができます。	はい		最大エントリ: 5
key	追加する新しいエントリのキー	はい		最大長: 128 ネストされたキーの最大深度: 3
値	追加する新しいエントリの値	はい		最大長: 256
overwrite IfExists	これを true に設定すると、key が既にイベントに存在する場合、既存の値は上書きされます。デフォルト値は false です。	いいえ	false	無制限

例

ログイベントの例を以下に示します。

```
{
  "outer_key": {
    "inner_key": "inner_value"
  }
}
```

```
}

```

トランスフォーマーの設定は次のようになります。addKeys と parseJSON を併用します。

```
[
  {
    "parseJSON": {}
  },
  {
    "addKeys": {
      "entries": [
        {
          "key": "outer_key.new_key",
          "value": "new_value"
        }
      ]
    }
  }
]
```

変換されたログイベントは次のとおりです。

```
{
  "outer_key": {
    "inner_key": "inner_value",
    "new_key": "new_value"
  }
}
```

DeleteKeys

deleteKeys プロセッサを使用して、ログイベントからフィールドを削除します。これらのフィールドには、キーと値のペアを含めることができます。

フィールド	説明	必須?	デフォルト	制限
withKeys	削除するキーのリスト。	はい	無制限	最大エントリ: 5

例

ログイベントの例を以下に示します。

```
{
  "outer_key": {
    "inner_key": "inner_value"
  }
}
```

トランスフォーマーの設定は次のようになります。deleteKeys と parseJSON を併用します。

```
[
  {
    "parseJSON": {}
  },
  {
    "deleteKeys": {
      "withKeys": ["outer_key.inner_key"]
    }
  }
]
```

変換されたログイベントは次のとおりです。

```
{
  "outer_key": {}
}
```

moveKeys

moveKeys プロセッサを使用して、あるフィールドから別のフィールドにキーを移動します。

フィールド	説明	必須?	デフォルト	制限
エントリ	エントリの配列。配列内の各項目には、source、target、overwrite IfExists フィールドを含めることができます。	はい		最大エントリ: 5

フィールド	説明	必須?	デフォルト	制限
ソース	移動するキー	はい		最大長: 128 ネストされたキーの最大深度: 3
target	移動先のキー	はい		最大長: 128 ネストされたキーの最大深度: 3
overwriteIfExists	これを true に設定すると、key が既にイベントに存在する場合、既存の値は上書きされます。デフォルト値は false です。	いいえ	false	無制限

例

ログイベントの例を以下に示します。

```
{
  "outer_key1": {
    "inner_key1": "inner_value1"
  },
  "outer_key2": {
    "inner_key2": "inner_value2"
  }
}
```

トランスフォーマーの設定は次のようになります。moveKeys と parseJSON を併用します。

```
[
  {
    "parseJSON": {}
  },
  {
    "moveKeys": {
      "entries": [
```

```
        {
            "source": "outer_key1.inner_key1",
            "target": "outer_key2"
        }
    ]
}
```

変換されたログイベントは次のとおりです。

```
{
  "outer_key1": {},
  "outer_key2": {
    "inner_key2": "inner_value2",
    "inner_key1": "inner_value1"
  }
}
```

renameKeys

renameKeys プロセッサを使用して、ログイベントのキーの名前を変更します。

フィールド	説明	必須?	デフォルト	制限
エントリ	エントリの配列。配列内の各項目には、key、target、overwrite IfExists フィールドを含めることができます。	はい	無制限	最大エントリ: 5
key	名前を変更するキー	はい	無制限	最大長: 128
target	新しいキー名	はい	無制限	最大長: 128 ネストされたキーの最大深度: 3
overwrite IfExists	これを true に設定すると、key が既にイベントに存在する場合、	いいえ	false	無制限

フィールド	説明	必須?	デフォルト	制限
	既存の値は上書きされます。デフォルト値は false です。			

例

ログイベントの例を以下に示します。

```
{
  "outer_key": {
    "inner_key": "inner_value"
  }
}
```

トランスフォーマーの設定は次のようになります。renameKeys と parseJSON を併用します。

```
[
  {
    "parseJSON": {}
  },
  {
    "renameKeys": {
      "entries": [
        {
          "key": "outer_key",
          "target": "new_key"
        }
      ]
    }
  }
]
```

変換されたログイベントは次のとおりです。

```
{
  "new_key": {
    "inner_key": "inner_value"
  }
}
```

```
}
```

copyValue

copyValue プロセッサを使用して、ログイベント内の値をコピーします。このプロセッサを使用して、以下のメタデータキーの値をログイベントにコピーすることでメタデータをログイベントに追加することもできます。@logGroupName、@logGroupStream、@accountId、@regionName。これについて、次の例で説明します。

フィールド	説明	必須?	デフォルト	制限
エントリ	エントリの配列。配列内の各項目には、source、target、overwrite IfExists フィールドを含めることができます。	はい		最大エントリ: 5
ソース	コピーするキー	はい		最大長: 128 ネストされたキーの最大深度: 3
target	値をコピーするキー	はい	無制限	最大長: 128 ネストされたキーの最大深度: 3
overwrite IfExists	これを true に設定すると、key が既にイベントに存在する場合、既存の値は上書きされます。デフォルト値は false です。	いいえ	false	無制限

例

ログイベントの例を以下に示します。

```
{  
  "outer_key": {
```

```
    "inner_key": "inner_value"
  }
}
```

トランスフォーマーの設定は次のようになります。copyValue と parseJSON を併用します。

```
[
  {
    "parseJSON": {}
  },
  {
    "copyValue": {
      "entries": [
        {
          "key": "outer_key.new_key",
          "target": "new_key"
        },
        {
          "source": "@logGroupName",
          "target": "log_group_name"
        },
        {
          "source": "@logGroupStream",
          "target": "log_group_stream"
        },
        {
          "source": "@accountId",
          "target": "account_id"
        },
        {
          "source": "@regionName",
          "target": "region_name"
        }
      ]
    }
  }
]
```

変換されたログイベントは次のとおりです。

```
{
  "outer_key": {
    "inner_key": "inner_value"
  }
}
```

```
},  
"new_key": "inner_value",  
"log_group_name": "myLogGroupName",  
"log_group_stream": "myLogStreamName",  
"account_id": "012345678912",  
"region_name": "us-east-1"  
}
```

listToMap

listToMap プロセッサは、キーフィールドを含むオブジェクトのリストを取得し、ターゲットキーのマップに変換します。

フィールド	説明	必須?	デフォルト	制限
ソース	マップに変換されるオブジェクトのリストを含む ProcessingEvent のキー	はい		最大長: 128 ネストされたキーの最大深度: 3
key	生成されたマップでキーとして抽出されるフィールドのキー	はい		最大長: 128
valueKey	これを指定すると、このパラメータで指定した値が source オブジェクトから抽出され、生成されたマップの値に配置されます。それ以外の場合、ソースリスト内の元のオブジェクトは、生成されたマップの値に配置されます。	いいえ		最大長: 128
target	生成されたマップを保持するフィールドのキー	いいえ	ルートノード	最大長: 128 ネストされたキーの最大深度: 3
flatten	リストを単一の項目にフラット化するか、生成されたマップの値をリストにするかを示すブール値。	いいえ	false	

フィールド	説明	必須?	デフォルト	制限
	デフォルトでは、一致するキーの値は配列で表されます。flatten を true に設定すると、配列は flattenedElement の値に基づいて 1 つの値に変換されます。			
flattened Element	flatten を true に設定する場合は、flattenedElement を使用して、first または last のどちらの要素を保持するかを指定します。	flatten が true に設定されている場合は必須です。		値は first または last です。

例

ログイベントの例を以下に示します。

```
{
  "outer_key": [
    {
      "inner_key": "a",
      "inner_value": "val-a"
    },
    {
      "inner_key": "b",
      "inner_value": "val-b1"
    },
    {
      "inner_key": "b",
      "inner_value": "val-b2"
    },
    {
      "inner_key": "c",
      "inner_value": "val-c"
    }
  ]
}
```

ユースケース 1 のトランスフォーマー: flatten は false です

```
[
  {
    "parseJSON": {}
  },
  {
    "listToMap": {
      "source": "outer_key"
      "key": "inner_key",
      "valueKey": "inner_value",
      "flatten": false
    }
  }
]
```

変換されたログイベントは次のとおりです。

```
{
  "outer_key": [
    {
      "inner_key": "a",
      "inner_value": "val-a"
    },
    {
      "inner_key": "b",
      "inner_value": "val-b1"
    },
    {
      "inner_key": "b",
      "inner_value": "val-b2"
    },
    {
      "inner_key": "c",
      "inner_value": "val-c"
    }
  ],
  "a": [
    "val-a"
  ],
  "b": [
    "val-b1",
    "val-b2"
  ]
}
```



```
    ],
    "c": [
      "val-c"
    ]
  }
}
```

ユースケース 2 のトランスフォーマー: `flatten` は `true` で、`flattenedElement` は `first` です

```
[
  {
    "parseJSON": {}
  },
  {
    "listToMap": {
      "source": "outer_key",
      "key": "inner_key",
      "valueKey": "inner_value",
      "flatten": true,
      "flattenedElement": "first"
    }
  }
]
```

変換されたログイベントは次のとおりです。

```
{
  "outer_key": [
    {
      "inner_key": "a",
      "inner_value": "val-a"
    },
    {
      "inner_key": "b",
      "inner_value": "val-b1"
    },
    {
      "inner_key": "b",
      "inner_value": "val-b2"
    },
    {
      "inner_key": "c",
      "inner_value": "val-c"
    }
  ]
}
```

```
    }
  ],
  "a": "val-a",
  "b": "val-b1",
  "c": "val-c"
}
```

ユースケース 3 のトランスフォーマー: `flatten` は `true` で、`flattenedElement` は `last` です

```
[
  {
    "parseJSON": {}
  },
  {
    "listToMap": {
      "source": "outer_key"
      "key": "inner_key",
      "valueKey": "inner_value",
      "flatten": true,
      "flattenedElement": "last"
    }
  }
]
```

変換されたログイベントは次のとおりです。

```
{
  "outer_key": [
    {
      "inner_key": "a",
      "inner_value": "val-a"
    },
    {
      "inner_key": "b",
      "inner_value": "val-b1"
    },
    {
      "inner_key": "b",
      "inner_value": "val-b2"
    },
    {
      "inner_key": "c",
      "inner_value": "val-c"
    }
  ]
}
```

```
    }  
  ],  
  "a": "val-a",  
  "b": "val-b2",  
  "c": "val-c"  
}
```

データ型コンバータプロセッサ

このセクションでは、ログイベントトランスフォーマーで使えるデータ型コンバータプロセッサについて説明します。

目次

- [typeConverter](#)
- [datetimeConverter](#)

typeConverter

typeConverter プロセッサを使用して、指定されたキーに関連付けられた値タイプを指定されたタイプに変換します。これは、指定されたフィールドのタイプを変更するキャストプロセッサです。値は、integer、double、string、boolean のいずれかのデータ型に変換できます。

フィールド	説明	必須?	デフォルト	制限
エントリ	エントリの配列。配列内の各項目には、key フィールドと type フィールドが含まれている必要があります。	はい		最大エントリ: 10
key	別のタイプに変換される値を持つキー	はい		最大長: 128 ネストされたキーの最大深度: 3
型	変換するタイプ。有効な値は、integer、double、string、です。	はい		

例

ログイベントの例を以下に示します。

```
{
  "name": "value",
  "status": "200"
}
```

トランスフォーマーの設定は次のようになります。typeConverter と parseJSON を併用します。

```
[
  {
    "parseJSON": {}
  },
  {
    "typeConverter": {
      "entries": [
        {
          "key": "status",
          "type": "integer"
        }
      ]
    }
  }
]
```

変換されたログイベントは次のとおりです。

```
{
  "name": "value",
  "status": 200
}
```

datetimeConverter

datetimeConverter プロセッサを使用して、日時文字列を指定した形式に変換します。

フィールド	説明	必須?	デフォルト	制限
ソース	日付変換を適用するキー。	はい		最大エントリ: 10
MatchPattern	source フィールドに一致するパターンのリスト	はい		最大エントリ: 5
target	結果を保存する JSON フィールド。	はい		最大長: 128 ネストされたキーの最大深度: 3
targetFormat	ターゲットフィールドで変換されたデータに使用する日時形式。	いいえ	yyyy-MM-dd'T'HH:mm:ss.SSS'Z'	最大長: 64
sourceTimezone	ソースフィールドのタイムゾーン。 可能な値のリストについては、 「Java でサポートされているゾーン ID とオフセット」 を参照してください。	いいえ	UTC	最小長: 1
targetTimezone	ターゲットフィールドのタイムゾーン。 可能な値のリストについては、 「Java でサポートされているゾーン ID とオフセット」 を参照してください。	いいえ	UTC	最小長: 1
サイト	ソースフィールドのロケール。 可能な値のリストについては、 「例を含む Java のロケール」	はい		最小長: 1

フィールド	説明	必須?	デフォルト	制限
	getAvailableLocales() メソッド を参照してください。			

例

ログイベントの例を以下に示します。

```
{"german_datetime": "Samstag 05. Dezember 1998 11:00:00"}
```

トランスフォーマーの設定は次のようになります。dateTimeConverter と parseJSON を併用します。

```
[
  {
    "parseJSON": {}
  },
  {
    "dateTimeConverter": {
      "source": "german_datetime",
      "target": "target_1",
      "locale": "de",
      "matchPatterns": ["EEEE dd. MMMM yyyy HH:mm:ss"],
      "sourceTimezone": "Europe/Berlin",
      "targetTimezone": "America/New_York",
      "targetFormat": "yyyy-MM-dd'T'HH:mm:ss z"
    }
  }
]
```

変換されたログイベントは次のとおりです。

```
{
  "german_datetime": "Samstag 05. Dezember 1998 11:00:00",
  "target_1": "1998-12-05T17:00:00 MEZ"
}
```

変換メトリクスとエラー

CloudWatch Logs は、変換メトリクスを CloudWatch に発行します。メトリクスには TransformedLogEvents、TransformedBytes、TransformationErrors が含まれます。詳細については、「[トランスフォーマーのメトリクスとディメンションをログに記録する](#)」を参照してください。

CloudWatch Logs がロギイベントの変換を試行して変換に失敗するたびに、そのロギイベントに @transformationError システムフィールドが追加されます。CloudWatch Logs Insights クエリを実行すると、変換に失敗したすべてのロギイベントにこのフィールドが表示されます。このフィールドを filter ispresent(@transformationError) などのクエリでクエリして、失敗した変換イベントをすべて検索できます。

Amazon OpenSearch Service で分析する

CloudWatch Logs は と統合 Amazon OpenSearch Service されているため、OpenSearch Service がサービスから AWS 提供されたログから派生する主要なメトリクスを表示する自動選別ダッシュボードを作成できます。以下のダッシュボードを使用できます。

- Amazon VPC フローログダッシュボードは、Amazon VPC のネットワークフローデータをキャプチャします。ネットワークトラフィックの分析、異常なパターンの検出、リソースの使用状況のモニタリングに役立ちます。表示される主要なメトリクスは次のとおりです。
 - フローの合計と、これらのフローの承諾と拒否
 - 経時的なトラフィックパターン
 - 送信元 IP と送信先 IP (トップトーカー) 間のデータフローを示す Sankey 図
 - 転送されたバイト数とパケット数の上位 IP

Note

現在、VPC バージョン 2 フィールド形式のみがサポートされています。

- AWS WAF ログダッシュボードは、モニタリング対象のウェブトラフィックに関するインサイトを提供します AWS WAF。このダッシュボードは、特定のリージョンまたは IP からのトラフィックパターン、ブロックされたリクエスト、潜在的な脅威を特定するのに役立ちます。表示される主要なメトリクスは次のとおりです。
 - 「ALLOW」カウントと「BLOCK」カウントを含むリクエストの合計。
 - 経時的なリクエスト履歴。許可されたリクエストとブロックされたリクエストが表示されます。
 - ウェブ ACL 名、終了ルールによるブロックされたリクエスト、ソース IP によるリクエストの内訳。
 - リクエストオリジンの地理的分布。
 - リクエスト数別の上位クライアント IP と終了ルール。
- CloudTrail ログダッシュボードには、CloudTrail ログを使用した AWS 環境内の API アクティビティの概要が表示されます。API アクティビティのモニタリング、アクションの監査、潜在的なセキュリティまたはコンプライアンスの問題の特定に役立ちます。表示される主要なメトリクスは次のとおりです。
 - 経時的なイベント数とイベント履歴の合計
 - アカウント ID、カテゴリ、リージョン別のイベントの内訳。

- イベントの生成に関連する上位 API、サービス、ソース IP。
- イベントを生成している上位ユーザーのテーブル。ユーザーアカウント情報とイベント数の詳細を示します。
- AWS Network Firewall ダッシュボードは、ネットワークトラフィックの可視性を強化し、セキュリティのモニタリングと分析に役立つインサイトを提供します。このダッシュボードは、潜在的なセキュリティ問題をすばやく特定し、ネットワーク設定を最適化するために、さまざまなネットワークのメトリクスとパターンを包括的に表示します。表示される主要なメトリクスは次のとおりです。
 - 最上位のトーカーとプロトコル
 - PrivateLink エンドポイントに関するインサイト
 - 許可およびブロックされた TLS サーバー名表示トラフィック

これらのキュレートされたダッシュボードに表示されるメトリクスは Amazon OpenSearch Service 分析から導出されます。

これらのダッシュボードを表示する前に、IAM ロールを作成し、CloudWatch Logs と Amazon OpenSearch Service の 1 回限りの統合を実行する必要があります。この 1 回限りの統合により、ダッシュボードの作成とレンダリングに必要な Amazon OpenSearch Service リソースが設定されます。使用した OpenSearch サービスには料金が発生します。詳細については、「[Amazon CloudWatch 料金表](#)」をご覧ください。

これらのキュレートされたダッシュボードは、標準ログクラスのロググループに対してのみ作成できます。

Important

提供されるログダッシュボードを作成するロググループには、[ログトランスフォーマー](#)を使用しないでください。ログイベントを変換すると、ダッシュボードのデータが空になります。

トピック

- [ステップ 1: OpenSearch Service との統合を作成する](#)
- [ステップ 2: 提供されるログダッシュボードを作成する](#)
- [提供されるログダッシュボードの表示、編集、削除](#)
- [ユーザー向けの IAM ポリシー](#)

- [統合に必要なアクセス許可](#)

ステップ 1: OpenSearch Service との統合を作成する

最初のステップでは、OpenSearch Service との統合を作成します。これは 1 回だけ実行する必要があります。この統合を作成すると、お客様のアカウントに以下のリソースが作成されます。

- 高可用性のない [OpenSearch Service 時系列コレクション](#)。

コレクションとは、連携してワークロードをサポートする一連の OpenSearch Service インデックスです。

- コレクションの 2 つのセキュリティポリシー。1 つは暗号化タイプを定義します。これは、カスタマーマネージド AWS KMS キーまたはサービス所有キーのいずれかです。もう 1 つのポリシーはネットワークアクセスを定義し、OpenSearch Service アプリケーションがコレクションにアクセスできるようにします。詳細については、「[Amazon OpenSearch Service に保管中のデータの暗号化](#)」を参照してください。
- コレクション内のデータにアクセスできるユーザーを定義する [OpenSearch Service データアクセスポリシー](#)。
- ソースとして CloudWatch Logs が定義された [OpenSearch Service ダイレクトクエリデータソース](#)。
- aws-analytics という名前の [OpenSearch Service アプリケーション](#)。アプリケーションは、ワークスペースの作成を許可するように設定されます。aws-analytics という名前のアプリケーションが既に存在する場合、このコレクションをデータソースとして追加するように更新されます。
- ダッシュボードをホストし、アクセス権限が付与されたすべてのユーザーにワークスペースからの読み取りを許可する [OpenSearch Service ワークスペース](#)。

トピック

- [必要なアクセス許可](#)
- [統合を作成する](#)

必要なアクセス許可

統合を作成するには、ここに表示される CloudWatchOpenSearchDashboardsFullAccess マネージド IAM ポリシーまたは同等のアクセス許可を持つアカウントでサインインする必要があります。統合

の削除、ダッシュボードの作成、編集、削除、ダッシュボードの手動更新を行うには、これらのアクセス許可も必要です。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Sid": "CloudWatchOpenSearchDashboardsIntegration",
    "Effect": "Allow",
    "Action": [
      "logs:ListIntegrations",
      "logs:GetIntegration",
      "logs>DeleteIntegration",
      "logs:PutIntegration",
      "logs:DescribeLogGroups",
      "opensearch:ApplicationAccessAll",
      "iam:ListRoles",
      "iam:ListUsers"
    ],
    "Resource": "*"
  },
  {
    "Sid": "CloudWatchLogsOpenSearchReadAPIs",
    "Effect": "Allow",
    "Action": [
      "aoss:BatchGetCollection",
      "aoss:BatchGetLifecyclePolicy",
      "es:ListApplications"
    ],
    "Resource": "*",
    "Condition": {
      "StringEquals": {
        "aws:CalledViaFirst": "logs.amazonaws.com"
      }
    }
  },
  {
    "Sid": "CloudWatchLogsOpenSearchCreateServiceLinkedAccess",
    "Effect": "Allow",
    "Action": [
      "iam:CreateServiceLinkedRole"
    ],
```

```

        "Resource": "arn:aws:iam::*:role/aws-service-role/
opensearchservice.amazonaws.com/AWSServiceRoleForAmazonOpenSearchService",
        "Condition": {
            "StringEquals": {
                "iam:AWSServiceName": "opensearchservice.amazonaws.com",
                "aws:CalledViaFirst": "logs.amazonaws.com"
            }
        },
    },
    {
        "Sid": "CloudWatchLogsObservabilityCreateServiceLinkedAccess",
        "Effect": "Allow",
        "Action": [
            "iam:CreateServiceLinkedRole"
        ],
        "Resource": "arn:aws:iam::*:role/aws-service-role/
observability.aoss.amazonaws.com/AWSServiceRoleForAmazonOpenSearchServerless",
        "Condition": {
            "StringEquals": {
                "iam:AWSServiceName": "observability.aoss.amazonaws.com",
                "aws:CalledViaFirst": "logs.amazonaws.com"
            }
        }
    },
    {
        "Sid": "CloudWatchLogsCollectionRequestAccess",
        "Effect": "Allow",
        "Action": [
            "aoss:CreateCollection"
        ],
        "Resource": "*",
        "Condition": {
            "StringEquals": {
                "aws:CalledViaFirst": "logs.amazonaws.com",
                "aws:RequestTag/CloudWatchOpenSearchIntegration": [
                    "Dashboards"
                ]
            },
            "ForAllValues:StringEquals": {
                "aws:TagKeys": "CloudWatchOpenSearchIntegration"
            }
        }
    },
    {

```

```

        "Sid": "CloudWatchLogsApplicationRequestAccess",
        "Effect": "Allow",
        "Action": [
            "es:CreateApplication"
        ],
        "Resource": "*",
        "Condition": {
            "StringEquals": {
                "aws:CalledViaFirst": "logs.amazonaws.com",
                "aws:RequestTag/OpenSearchIntegration": [
                    "Dashboards"
                ]
            },
            "ForAllValues:StringEquals": {
                "aws:TagKeys": "OpenSearchIntegration"
            }
        }
    },
    {
        "Sid": "CloudWatchLogsCollectionResourceAccess",
        "Effect": "Allow",
        "Action": [
            "aoss:DeleteCollection"
        ],
        "Resource": "*",
        "Condition": {
            "StringEquals": {
                "aws:CalledViaFirst": "logs.amazonaws.com",
                "aws:ResourceTag/CloudWatchOpenSearchIntegration": [
                    "Dashboards"
                ]
            }
        }
    },
    {
        "Sid": "CloudWatchLogsApplicationResourceAccess",
        "Effect": "Allow",
        "Action": [
            "es:UpdateApplication",
            "es:GetApplication"
        ],
        "Resource": "*",
        "Condition": {
            "StringEquals": {

```

```

        "aws:CalledViaFirst": "logs.amazonaws.com",
        "aws:ResourceTag/OpenSearchIntegration": [
            "Dashboards"
        ]
    }
},
{
    "Sid": "CloudWatchLogsCollectionPolicyAccess",
    "Effect": "Allow",
    "Action": [
        "aoss:CreateSecurityPolicy",
        "aoss:CreateAccessPolicy",
        "aoss:DeleteAccessPolicy",
        "aoss:DeleteSecurityPolicy",
        "aoss:GetAccessPolicy",
        "aoss:GetSecurityPolicy"
    ],
    "Resource": "*",
    "Condition": {
        "StringLike": {
            "aoss:collection": "cloudwatch-logs-*",
            "aws:CalledViaFirst": "logs.amazonaws.com"
        }
    }
},
{
    "Sid": "CloudWatchLogsAPIAccessAll",
    "Effect": "Allow",
    "Action": [
        "aoss:APIAccessAll"
    ],
    "Resource": "*",
    "Condition": {
        "StringLike": {
            "aoss:collection": "cloudwatch-logs-*"
        }
    }
},
{
    "Sid": "CloudWatchLogsIndexPolicyAccess",
    "Effect": "Allow",
    "Action": [
        "aoss:CreateAccessPolicy",

```

```

        "aoss:DeleteAccessPolicy",
        "aoss:GetAccessPolicy",
        "aoss:CreateLifecyclePolicy",
        "aoss:DeleteLifecyclePolicy"
    ],
    "Resource": "*",
    "Condition": {
        "StringLike": {
            "aoss:index": "cloudwatch-logs-*",
            "aws:CalledViaFirst": "logs.amazonaws.com"
        }
    }
},
{
    "Sid": "CloudWatchLogsDQSRequestQueryAccess",
    "Effect": "Allow",
    "Action": [
        "es:AddDirectQueryDataSource"
    ],
    "Resource": "arn:aws:opensearch:*:*:datasource/cloudwatch_logs_*",
    "Condition": {
        "StringEquals": {
            "aws:CalledViaFirst": "logs.amazonaws.com",
            "aws:RequestTag/CloudWatchOpenSearchIntegration": [
                "Dashboards"
            ]
        },
        "ForAllValues:StringEquals": {
            "aws:TagKeys": "CloudWatchOpenSearchIntegration"
        }
    }
},
{
    "Sid": "CloudWatchLogsStartDirectQueryAccess",
    "Effect": "Allow",
    "Action": [
        "opensearch:StartDirectQuery",
        "opensearch:GetDirectQuery"
    ],
    "Resource": "arn:aws:opensearch:*:*:datasource/cloudwatch_logs_*"
},
{
    "Sid": "CloudWatchLogsDQSResourceQueryAccess",
    "Effect": "Allow",

```

```

        "Action": [
            "es:GetDirectQueryDataSource",
            "es:DeleteDirectQueryDataSource"
        ],
        "Resource": "arn:aws:opensearch:*:*:datasource/cloudwatch_logs_*",
        "Condition": {
            "StringEquals": {
                "aws:CalledViaFirst": "logs.amazonaws.com",
                "aws:ResourceTag/CloudWatchOpenSearchIntegration": [
                    "Dashboards"
                ]
            }
        }
    },
    {
        "Sid": "CloudWatchLogsPassRoleAccess",
        "Effect": "Allow",
        "Action": [
            "iam:PassRole"
        ],
        "Resource": "*",
        "Condition": {
            "StringLike": {
                "iam:PassedToService":
"directquery.opensearchservice.amazonaws.com",
                "aws:CalledViaFirst": "logs.amazonaws.com"
            }
        }
    },
    {
        "Sid": "CloudWatchLogsAossTagsAccess",
        "Effect": "Allow",
        "Action": [
            "aoss:TagResource"
        ],
        "Resource": "arn:aws:aoss:*:*:collection/*",
        "Condition": {
            "StringEquals": {
                "aws:CalledViaFirst": "logs.amazonaws.com",
                "aws:ResourceTag/CloudWatchOpenSearchIntegration": [
                    "Dashboards"
                ]
            },
            "ForAllValues:StringEquals": {

```



```

        "aws:TagKeys": "CloudWatchOpenSearchIntegration"
    }
}
},
{
    "Sid": "CloudWatchLogsEsApplicationTagsAccess",
    "Effect": "Allow",
    "Action": [
        "es:AddTags"
    ],
    "Resource": "arn:aws:opensearch:*:*:application/*",
    "Condition": {
        "StringEquals": {
            "aws:ResourceTag/OpenSearchIntegration": [
                "Dashboards"
            ],
            "aws:CalledViaFirst": "logs.amazonaws.com"
        },
        "ForAllValues:StringEquals": {
            "aws:TagKeys": "OpenSearchIntegration"
        }
    }
},
{
    "Sid": "CloudWatchLogsEsDataSourceTagsAccess",
    "Effect": "Allow",
    "Action": [
        "es:AddTags"
    ],
    "Resource": "arn:aws:opensearch:*:*:datasource/*",
    "Condition": {
        "StringEquals": {
            "aws:ResourceTag/CloudWatchOpenSearchIntegration": [
                "Dashboards"
            ],
            "aws:CalledViaFirst": "logs.amazonaws.com"
        },
        "ForAllValues:StringEquals": {
            "aws:TagKeys": "CloudWatchOpenSearchIntegration"
        }
    }
}
]

```

```
}
```

統合を作成する

以下のステップを使用して統合を作成します。

CloudWatch Logs を と統合するには Amazon OpenSearch Service

1. CloudWatch コンソールの <https://console.aws.amazon.com/cloudwatch/> を開いてください。
2. 左側のナビゲーションペインで [ログのインサイト] を選択した後、[OpenSearch で分析] タブを選択します。
3. [統合を作成する] を選択します。
4. [統合名] に、統合の名前を入力します。
5. (オプション) OpenSearch Service Serverless に書き込まれたデータを暗号化するには、KMS AWS KMS キー ARN で使用するキーの ARN を入力します。詳細については、「Amazon OpenSearch Service 開発者ガイド」の「[保管中の暗号化](#)」を参照してください。
6. データ保持の場合、OpenSearch Service データインデックスを保持する時間を入力します。これにより、ダッシュボードでデータを表示できる最大期間も定義されます。データ保持期間を長くすると、検索とインデックス作成に伴う追加コストが発生します。詳細については、「[OpenSearch Service Serverless の料金](#)」を参照してください。

最大保持期間は 30 日間です。

データ保持期間の長さは、OpenSearch Service コレクションライフサイクルポリシーの作成にも使用されます。

7. OpenSearch コレクションに書き込む IAM ロールの場合は、新しい IAM ロールを作成するか、OpenSearch Service コレクションへの書き込みに使用する既存の IAM ロールを選択します。

新しいロールの作成が最も簡単な方法です。ロールは必要なアクセス許可を使用して作成されます。

Note

ロールを作成すると、アカウント内のすべてのロググループから読み取るアクセス許可が付与されます。

既存のロールを選択する場合は、[統合に必要なアクセス許可](#)に記載されているアクセス許可が必要です。または、[既存のロールを使用]を選択し、[選択したロールのアクセス許可の検証]セクションで[ロールの作成]を選択します。これにより、[統合に必要なアクセス許可](#)にリストされているアクセス許可をテンプレートとして使用、変更できます。例えば、ロググループのよりきめ細かな制御を指定する場合です。

8. ダッシュボードを表示できる IAM ロールとユーザーの場合、提供されるログダッシュボードアクセス向けの IAM ロールと IAM ユーザーへのアクセスを許可する方法を選択します。
 - ダッシュボードへのアクセスを一部のユーザーのみに制限するには、[ダッシュボードを表示できる IAM ロールとユーザーの選択]を選択し、テキストボックスでアクセスを許可する IAM ロールと IAM ユーザーを検索して選択します。
 - ダッシュボードへのアクセス権をすべてのユーザーに付与するには、[このアカウントのすべてのロールとユーザーにダッシュボードの表示を許可]を選択します。

 Important

ロールまたはユーザーを選択するか、すべてのユーザーを選択すると、ダッシュボードデータを保存する OpenSearch Service コレクションへのアクセスに必要な[データアクセスポリシー](#)にのみそれらが追加されます。提供されるログダッシュボードをそれらに対して表示できるようにするには、それらのロールとユーザーに[CloudWatchOpenSearchDashboardAccess](#) マネージド IAM ポリシーも付与する必要があります。

9. [統合を作成]を選択します。

統合の作成には数分かかります。

ステップ 2: 提供されるログダッシュボードを作成する

統合を作成したら、ダッシュボードを作成できます。ダッシュボードは、Amazon VPC フローログ、CloudTrail ログ、および AWS WAF ログで使用できます。

OpenSearch Service によって導出されたメトリクスを使用して提供されるログダッシュボードを作成するには

1. CloudWatch コンソールの <https://console.aws.amazon.com/cloudwatch/> を開いてください。

2. 左側のナビゲーションペインで [ログのインサイト] を選択した後、[OpenSearch で分析] タブを選択します。
3. [ダッシュボードを作成] を選択します。
4. ダッシュボードを作成するログのタイプ、Amazon VPC フローログ AWS WAF、CloudTrail、またはを選択します AWS Network Firewall。
5. ダッシュボードの名前を入力し、必要に応じて説明を入力します。
6. データ同期頻度には、OpenSearch Service で作成されたメトリクスとインデックスを新しいデータと同期させて更新できるように、OpenSearch Service が CloudWatch をクエリする頻度を入力します。OpenSearch Service は、ダッシュボードをレンダリングするためのメトリクスとインデックスをログに作成します。

短い時間を選択すると、データはより最新に保たれ、コストが高くなります。

7. このダッシュボードからデータを収集するロググループを選択します。作成するダッシュボードのタイプに一致するロググループを必ず選択してください。

[ロググループを参照] ボタンと [選択したロググループからログサンプルを表示] オプションを使用し、選択するたびに必要なロググループを取得できているか確認することができます。

8. [ダッシュボードを作成] を選択します。

最初は、ダッシュボードはデータなしで表示されます。数分後、データがダッシュボードに表示されます。データが最初に表示されると、最新の 15 分間のログエントリになります。

提供されるログダッシュボードの表示、編集、削除

CloudWatch Logs または OpenSearch Service で提供されるログダッシュボードを表示する

ダッシュボードを表示するには、CloudWatchOpenSearchDashboardAccess IAM ポリシーを持つ IAM プリンシパルにサインインする必要があります。

提供されるログダッシュボードを表示するには

1. CloudWatch コンソールの <https://console.aws.amazon.com/cloudwatch/> を開いてください。
2. 左側のナビゲーションペインで [ログのインサイト] を選択した後、[OpenSearch で分析] タブを選択します。
3. OpenSearch ダッシュボード ボックスでダッシュボードを選択します。

4. (オプション) 右上で、[OpenSearch で表示] を選択します。

OpenSearch Service コンソールが開き、そこに同じダッシュボードが表示されます。OpenSearch Service コンソールでは、ダッシュボードとそのウィジェットを変更でき、これらの変更は CloudWatch Logs でダッシュボードを表示するときにも表示されます。

追加の IAM ロールまたは IAM ユーザーにダッシュボード表示アクセス権を付与する

統合の作成後に追加の IAM プリンシパルへのアクセスを許可するには、次の手順を実行します。

追加の IAM ロールまたはユーザーに提供されるログダッシュボードへのアクセス権を付与するには

1. コレクションのデータアクセスポリシーを編集して、これらのロールまたはユーザーを追加します。詳細については、「OpenSearch Service デベロッパーガイド」の「[Amazon OpenSearch Service Serverless のデータアクセスコントロール](#)」を参照してください。
2. これらのユーザーに CloudWatchOpenSearchDashboardAccess を付与します。このポリシーの内容の詳細については、「[CloudWatchOpenSearchDashboardAccess](#)」を参照してください。

ダッシュボード設定を編集する

提供された既存のログダッシュボードの名前、説明、同期頻度を編集できます。

提供されたログダッシュボードを編集するには

1. CloudWatch コンソールの <https://console.aws.amazon.com/cloudwatch/> を開いてください。
2. 左側のナビゲーションペインで [ログのインサイト] を選択した後、[OpenSearch で分析] タブを選択します。
3. OpenSearch ダッシュボード ボックスでダッシュボードを選択します。
4. [アクション]、[ダッシュボードの詳細の変更] を選択します。
5. 変更後、[変更の確認] を選択します。

提供されたログダッシュボードを削除する

提供されたログダッシュボードを削除できます。これを行うと、OpenSearch Service コレクションで作成されたダッシュボード、メトリクス、インデックスがすべて削除されます。

 Note

提供されたログダッシュボードを削除したら、6 時間以上経過後に同じダッシュボードを再作成できます。6 時間待機しない場合、再作成されたダッシュボードは正しく動作しません。

提供されたログダッシュボードを削除するには

1. CloudWatch コンソールの <https://console.aws.amazon.com/cloudwatch/> を開いてください。
2. 左側のナビゲーションペインで [ログのインサイト] を選択した後、[OpenSearch で分析] タブを選択します。
3. OpenSearch ダッシュボード ボックスでダッシュボードを選択します。
4. [アクション]、[削除] の順に選択します。
5. **delete** を入力して決定を確認し、[削除] を選択します。

OpenSearch Service とのすべての提供されたログダッシュボード統合を削除する

OpenSearch 統合全体を削除できます。これを行うと、すべての提供されたログダッシュボードとダッシュボードに表示されたデータは削除されます。

 Important

継続的なコストを避けるため、統合を削除する前に、次のリソースを手動で削除することを強くお勧めします。統合を削除しても、これらのリソースは自動的に削除されません。統合を削除した後は、これらのリソースにアクセスしてそれらのリソースを削除することはできません。削除するリソースの名前を確認するには、次の手順を参照してください。

- [データソース](#)
- [コレクション](#)
- [データアクセスポリシー](#)
- [暗号化ポリシー](#)
- [ネットワークポリシー](#)
- [ライフサイクルポリシー](#)

OpenSearch Service と提供されたログダッシュボードの統合全体を削除するには

1. CloudWatch コンソールの <https://console.aws.amazon.com/cloudwatch/> を開いてください。
2. 左側のナビゲーションペインで [設定] を選択します。
3. [ログ] タブを選択します。
4. [OpenSearch 統合] セクションで、[統合の削除] を選択します。

次の画面には、統合を削除する前に削除する必要がある OpenSearch Service リソースの名前が表示されます。

5. **delete** を入力して決定を確認し、[統合の削除] を選択します。

ユーザー向けの IAM ポリシー

CloudWatch Logs は、CloudWatchOpenSearchDashboardsFullAccess と CloudWatchOpenSearchDashboardAccess の 2 つの IAM ポリシーを作成しました。次の表に、これらの各ポリシーが有効にするアクションを示します。

Action	IAM ポリシー	必要となる追加のアクセス許可
統合の作成	CloudWatchOpenSearchDashboardsFullAccess	
統合の削除	CloudWatchOpenSearchDashboardsFullAccess	
ダッシュボードの作成	CloudWatchOpenSearchDashboardsFullAccess	
ダッシュボードの編集	CloudWatchOpenSearchDashboardsFullAccess	
ダッシュボードの削除	CloudWatchOpenSearchDashboardsFullAccess	
[今すぐ同期]を使用したダッシュボード更新	CloudWatchOpenSearchDashboardsFullAccess	

Action	IAM ポリシー	必要となる追加のアクセス許可
[設定]で統合を表示	CloudWatchOpenSearchDashboardAccess または CloudWatchOpenSearchDashboardsFullAccess	
ダッシュボードの表示	CloudWatchOpenSearchDashboardAccess または CloudWatchOpenSearchDashboardsFullAccess	統合を作成するときにロールまたはユーザーを指定するか、コレクションのデータアクセスポリシーを編集してこれらのロールまたはユーザーを追加します。詳細については、「OpenSearch Service デベロッパーガイド」の「 Amazon OpenSearch Service Serverless のデータアクセスコントロール 」を参照してください。
OpenSearch Service コンソールでダッシュボードを表示	CloudWatchOpenSearchDashboardAccess または CloudWatchOpenSearchDashboardsFullAccess	統合を作成するときにロールまたはユーザーを指定するか、コレクションのデータアクセスポリシーを編集してこれらのロールまたはユーザーを追加します。詳細については、「OpenSearch Service デベロッパーガイド」の「 Amazon OpenSearch Service Serverless のデータアクセスコントロール 」を参照してください。

統合に必要なアクセス許可

統合で使用する IAM ロールを作成する場合、CloudWatch Logs にロールの作成を許可する代わりに、次のアクセス許可と信頼ポリシーを含める必要があります。IAM ロールを作成する方法の詳細については、[「AWS サービスにアクセス許可を委任するロールを作成する」](#)を参照してください。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "CloudWatchLogsAccess",
      "Effect": "Allow",
      "Action": [
        "logs:StartQuery",
        "logs:GetLogGroupFields",
        "logs:GetQueryResults"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "CloudWatchLogsDescribeLogGroupsAccess",
      "Effect": "Allow",
      "Action": [
        "logs:DescribeLogGroups"
      ],
      "Resource": "*"
    },
    {
      "Sid": "AmazonOpenSearchCollectionAccess",
      "Effect": "Allow",
      "Action": [
        "aoss:APIAccessAll"
      ],
      "Resource": "*",
      "Condition": {
        "StringLike": {
          "aoss:collection": "cloudwatch-logs-*"
        }
      }
    }
  ]
}
```

```
    }  
  }  
]  
}
```

Note

前のロールは、アカウント内のすべてのロググループからの読み取りアクセスを許可し、クロスアカウントロググループを含む任意のログアカウントのダッシュボードを作成できるようにします。特定のロググループへのアクセスを制限し、それらのロググループのみのダッシュボードを作成する場合は、そのポリシーの最初のステートメントを次のように更新できます。

```
{  
  "Sid": "CloudWatchLogsAccess",  
  "Effect": "Allow",  
  "Action": [  
    "logs:StartQuery",  
    "logs:GetLogGroupFields",  
    "logs:GetQueryResults"  
  ],  
  "Resource": [  
    "arn:aws:logs:us-east-1:123456789012:log-group:myLogGroup:*",  
    "arn:aws:logs:us-east-1:123456789012:log-group:myLogGroup"  
  ]  
}
```

S3 Tables 統合によるログへのアクセス

S3 Tables Integration with CloudWatch を使用すると、Amazon Athena、Amazon Redshift、Apache Iceberg 互換ストアへの接続をサポートするサードパーティーツールなどの分析エンジンを使用して、CloudWatch に取り込まれたログデータにアクセスできます。この統合により、任意のツールを使用して包括的なログ分析を実行し、CloudWatch Logs のデータを CloudWatch 以外のデータと関連付けることができます。

S3 Tables の統合について

Amazon S3 Tables Integration は、CloudWatch Logs のログをマネージド Amazon S3 テーブルとして利用できるようにするフルマネージドソリューションです。この統合により、CloudWatch Logs の機能に加えて、ログの分析方法の柔軟性が向上します。

統合は、マネージド Amazon S3 テーブルバケット (aws-cloudwatch) を作成し、データソース名とタイプに基づいて特定のログソースを Amazon S3 テーブルに関連付けることで機能します (CloudWatch Logs コンソールのログ管理 > データソースタブから管理できます)。関連付けると、CloudWatch Logs データは Apache Iceberg 形式を使用して Amazon S3 Tables からアクセスできるようになります。この形式は、さまざまな分析エンジンがデータを効率的にクエリするための標準化された方法を提供します。

コアコンポーネント

データソースの関連付け

データソースとタイプ基準に基づいて、特定の CloudWatch Logs ソースを S3 Tables 統合にリンクするプロセス。

Apache Iceberg テーブル

構造化データストレージを提供し、複数の分析エンジンとの互換性を可能にする S3 Tables で使用される基盤となるテーブル形式。

S3 テーブルへのデータフロー

CloudWatch Logs と S3 Tables 間のデータフローを理解すると、統合を計画し、ログデータを効果的に管理できます。

関連付けを作成すると、CloudWatch Logs は、関連付けられたデータソース名とタイプに一致する新しいログイベントを CloudWatch 管理の S3 テーブルバケットに自動的に送信します。これらのイベントは、そのデータソースの対応するテーブルのログ名前空間にあります。統合は、関連付けの作成後に追加されたイベントのみをログに記録し、関連付けの作成前からログをバックフィルしません。

S3 テーブルバケットのデータ保持は、ロググループに設定されている保持ポリシーと一致します。たとえば、ロググループを 1 日間の保持に設定すると、CloudWatch Logs は 1 日後に CloudWatch Logs と S3 Table の両方からデータを削除します。ロググループまたはログストリームを削除すると、CloudWatch Logs は S3 テーブルバケットからデータも削除します。

S3 テーブル統合を使用するタイミング

S3 Tables 統合を使用して、ログデータを他の外部データまたは CloudWatch 以外のデータと関連付けたり、Amazon Athena などの他の分析ツールを使用して CloudWatch Logs データに対して分析を実行したりすることを検討してください。CloudWatch Logs で利用できる機能を超える機能が必要な場合は、この統合を使用します。この統合は、次の場合に特に役立ちます。

- 大量のログデータに対して複雑な SQL のようなクエリを実行する必要があります
- ログ分析を既存の分析ワークフローやツールと統合したい
- 複数のデータソースにまたがる包括的なログ分析機能が必要です

この統合によって作成された S3 テーブルには、既存の CloudWatch の取り込みとストレージ料金以外に、追加のストレージやテーブルのメンテナンス料金はかかりません。

前提条件

統合を実装する前に、以下があることを確認してください。

- 既存の CloudWatch Logs データ
- 次のセクションで説明するように、CloudWatch Logs と S3 Tables 間のクロスサービスアクセスのための適切な IAM アクセス許可

IAM アクセス許可

CloudWatch Logs を S3 Tables と統合するには、統合を設定するユーザーまたはロールと、CloudWatch Logs が S3 Tables にデータを書き込むために引き受けるサービスロールの 2 つのエンティティに IAM アクセス許可を設定する必要があります。

統合を作成するロールまたはユーザーの場合

統合を設定するユーザーまたはロールには、次のアクセス許可が必要です。

- `observabilityadmin:CreateS3TableIntegration` 統合を作成し `logs:AssociateSourceToS3TableIntegration`、ソースを追加するには
- `s3tables:CreateTableBucket`、`s3tables:PutTableBucketEncryption`、および `s3tables:PutTableBucketPolicy` を使用して S3 テーブルバケットを設定する

サービスロールの場合

CloudWatch Logs がテーブルバケットへのデータの書き込みに使用する IAM サービスロールに、次の IAM ポリシーをアタッチします。このポリシーは、テーブルに書き込むアクセス許可を付与します。*aws-region*、*123456789012*、*log-group-name* を AWS リージョン、アカウント ID、ロググループ名に置き換えます。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:integrateWithS3Table"
      ],
      "Resource": ["arn:aws:logs:aws-region:123456789012:log-group:log-group-name"],
      "Condition": {
        "StringEquals": {
          "aws:ResourceAccount": "123456789012"
        }
      }
    }
  ]
}
```

CloudWatch Logs が S3 Tables にログデータを書き込むために引き受ける IAM サービスロールに、次の信頼ポリシーをアタッチします。このロールは、統合セットアップ中に作成または選択します。条件によってロールが制限されるため、CloudWatch Logs は指定されたアカウントとロググループに対してのみロールを引き受けることができます。*aws-region*、*123456789012*、*log-group-name* を AWS リージョン、アカウント ID、ロググループ名に置き換えます。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "logs.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "123456789012"
        },
        "ArnLike": {
          "aws:SourceArn": ["arn:aws:logs:aws-region:123456789012:log-group:log-group-name"]
        }
      }
    }
  ]
}
```

KMS キーポリシー (暗号化されたデータ用)

カスタマーマネージドキーを使用してログデータを暗号化する場合は、CloudWatch サービスプリンシパルと S3 Tables メンテナンスサービスプリンシパルにキーへのアクセスを許可する必要があります。KMS キーポリシーに次のステートメントを追加します。プレースホルダー値を AWS アカウント ID、リージョン、KMS キー ID、S3 テーブルまたはテーブルバケット ARN に置き換えます。

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

```

    "Principal": {
      "Service": "systemtables.cloudwatch.amazonaws.com"
    },
    "Action": [
      "kms:DescribeKey",
      "kms:GenerateDataKey",
      "kms:Decrypt"
    ],
    "Resource": "arn:aws:kms:aws-region:123456789012:key/key-id",
    "Condition": {
      "StringEquals": {
        "aws:SourceAccount": "123456789012"
      }
    }
  },
  {
    "Sid": "EnableKeyUsage",
    "Effect": "Allow",
    "Principal": {
      "Service": "maintenance.s3tables.amazonaws.com"
    },
    "Action": [
      "kms:GenerateDataKey",
      "kms:Decrypt"
    ],
    "Resource": "arn:aws:kms:aws-region:123456789012:key/key-id",
    "Condition": {
      "StringLike": {
        "kms:EncryptionContext:aws:s3:arn": "<table-or-table-bucket-arn>/*"
      }
    }
  }
]
}

```

開始方法

S3 Tables 統合を開始するには、CloudWatch Logs と S3 Tables の統合を設定する必要があります。このプロセスでは、データソースの関連付けを設定し、適切な IAM アクセス許可を設定します。

S3 テーブル統合を作成するには

1. <https://console.aws.amazon.com/cloudwatch/> で CloudWatch Logs コンソールを開きます。

2. 設定、グローバル、S3 テーブル統合の作成を選択します。
3. S3 Tables でログを暗号化する方法と、CloudWatch Logs がログを S3 Tables に書き込むために使用するロールをカスタマイズします。
4. S3 テーブル統合の作成 を選択します。

ソースを S3 テーブル統合に関連付けるには

1. <https://console.aws.amazon.com/cloudwatch/> で CloudWatch Logs コンソールを開きます。
2. 設定、グローバル、S3 テーブル統合の管理を選択します。
3. データソースの関連付けを選択します。
4. 統合を有効にするデータソース名とデータソースタイプを選択します。
5. データソースの関連付けを選択します。

ログ管理ページからソースを S3 テーブル統合に関連付けるには

1. <https://console.aws.amazon.com/cloudwatch/> で CloudWatch Logs コンソールを開きます。
2. ナビゲーションペインでログ管理を選択します。
3. データソースタブを選択します。
4. 統合するデータソース名とデータソースタイプを選択します。
5. データソースアクションを選択します。
6. S3 テーブル統合に関連付けるを選択します。
7. データソースを確認し、データソースの関連付けを選択します。

データを使用する前に、次の 3 つのステップを実行する必要があります。

1. Amazon S3 Tables と AWS 分析サービスの統合 - Amazon S3 コンソールの使用
2. Lake Formation アクセス許可を設定する
3. 分析ツールで接続する

Amazon S3 Tables と AWS 分析サービスの統合 - Amazon S3 コンソールの使用 ([リンク](#))

S3 コンソールを使用して S3 Tables 統合を有効にするには

1. Amazon S3 コンソール (<https://console.aws.amazon.com/s3/>) を開きます。
2. 左側のナビゲーションペインで、[テーブルバケット] を選択します。
3. 上部の統合を有効にするをクリックします。
4. 任意のリージョンでテーブルバケットを初めて統合すると、Amazon S3 はユーザーに代わって新しい IAM サービスロールを作成します。このロールにより、Lake Formation はアカウント内のすべてのテーブルバケットにアクセスし、Glue Data Catalog AWS のテーブルへのフェデレーションアクセスが可能になります。

Lake Formation アクセス許可を設定する

CloudWatch Logs にはテーブルに書き込むアクセス許可がありますが (前のステップで設定)、ユーザーと分析ロールにはデータを読み取るアクセス許可が自動的にありません。AWS Lake Formation を使用してアクセスを明示的に許可する必要があります。これは、テーブルへのアクセスを許可するすべての IAM プリンシパルに対して実行する必要があります。

ユーザーまたはロールにクエリアクセスを許可するには

Athena または Redshift でクエリを実行する IAM プリンシパル (ユーザーまたはロール) に SELECT および DESCRIBE アクセス許可を付与する必要があります。

1. AWS Lake Formation コンソールを開きます。
2. ナビゲーションペインの [Permissions] (許可) で [Data lake permissions] (データレイクの許可) を選択します。
3. [付与] を選択します。
4. プリンシパル: アクセスを必要とする IAM ユーザーまたはロール (データアナリストや現在使用している管理者ロールなど) を選択します。
5. LF タグまたはカタログリソース: 名前付きデータカタログリソースを選択します。
6. データベースとテーブル:
 - CloudWatch 統合 () によって作成された S3 Table バケットを選択しますaws-cloudwatch。
 - データソースに関連付けられている特定のテーブルを選択します (オプション)。

7. テーブルのアクセス許可: 選択と説明を選択します。
8. [付与] を選択します。

Note

Athena でログをクエリするときに「アクセス拒否」エラーが発生した場合は、クエリを実行しているユーザーに Athena の適切な IAM アクセス許可と上記で定義された Lake Formation アクセス許可の両方があることを確認してください。

Lake Formation のアクセス許可の詳細については、<https://docs.aws.amazon.com/lake-formation/latest/dg/granting-catalog-permissions.html> を参照してください。

分析ツールで接続する

アクセス許可が付与されたら、S3 テーブルをクエリするように任意の分析サービスを設定できます。S3 Tables は、Amazon Athena、Amazon Redshift、Amazon EMR でネイティブにサポートされている Apache Iceberg 形式を使用します。

Amazon Athena でログデータをクエリするには

Amazon Athena は、Amazon S3 Tables カタログを介して Amazon S3 Tables とやり取りします。

ログデータをクエリするように Athena を設定するには

1. <https://console.aws.amazon.com/athena/> で Amazon Athena コンソールを開きます。
2. クエリエディタで、データソースドロップダウンから Amazon S3 Tables カタログを選択します。
3. カタログが表示されない場合は、特定のユーザーロールに対して上記の Lake Formation アクセス許可ステップを完了していることを確認してください。
4. カタログを選択すると、ログテーブルがデータベースリストに表示されます。ログデータに対して標準 SQL クエリを実行できるようになりました。

クエリの例: `SELECT * FROM "amazon_vpc__flow" LIMIT 100;`

Analytics サービスと S3 Tables の接続の詳細については、<https://docs.aws.amazon.com/AmazonS3/latest/userguide/s3-tables-integrating-aws.html> を参照してください。

フィルターを使用したログイベントからのメトリクスの作成

1 つまたは複数のメトリクスフィルターを作成することで、CloudWatch Logs で受信するログデータを検索およびフィルタリングできます。メトリクスフィルタは CloudWatch Logs に送信されたログデータを検索するための語句とパターンを定義します。CloudWatch Logs は、これらのメトリクスフィルタを使用して、ログデータを数値の CloudWatch メトリクスに変換し、グラフを作成したり、アラームを設定したりできます。

ログフィルターからメトリクスを作成するときに、ディメンションと単位をメトリクスに割り当てることもできます。単位を指定する場合は、フィルターの作成時に必ず正しい単位を指定してください。フィルターの単位を後で変更しても何も起こりません。

メンバーアカウントを設定 AWS Organizations して操作している場合は、ログの一元化を使用して、ソースアカウントから中央モニタリングアカウントにログデータを収集できます。

一元化されたロググループを使用する場合は、メトリクスフィルターを作成する際にこれらのシステムフィールドのディメンションを使用できます。

- `@aws.account` - このディメンションは、ログイベントの発生元の AWS アカウント ID を表します。
- `@aws.region` - このディメンションは、ログイベントが生成された AWS リージョンを表します。

これらのディメンションはログデータのソースを特定するのに役立ちます。これにより、一元化されたログから派生したメトリクスのフィルタリングと分析をさらに細かく行えるようになります。詳細については、「[クロスアカウントクロスリージョンログの一元化](#)」を参照してください。

サブスクリプションを持つロググループがログ変換を使用する場合、フィルターパターンは変換されたバージョンのログイベントに適用されます。詳細については、「[取り込み中のログの変換](#)」を参照してください。

Note

メトリクスフィルターは、標準ログクラスのロググループでのみサポートされます。ログクラスの詳細については、「[ログクラス](#)」を参照してください。

これらのメトリクスを表示する、またはアラームを設定するときは、パーセンタイル統計など、任意のタイプの CloudWatch 統計を使用できます。

Note

パーセンタイル統計は、メトリクスの値がいずれも負でない場合にのみメトリクスでサポートされます。負の数を報告できるようにメトリクスフィルタを設定した場合、値に負の数があると、パーセンタイル統計はそのメトリクスで使用できません。詳細については、[パーセンタイル](#)を参照してください。

フィルターは、遡及的にデータをフィルターしません。フィルターは、フィルターが作成された後に発生したイベントのメトリクスのデータポイントをパブリッシュするだけです。フィルターパターンをテストすると、検証を目的として一致するログ行が [フィルター結果] のプレビューに最初の 50 行まで表示されます。フィルターされた結果のタイムスタンプがメトリクスの作成時刻より前の場合、ログは表示されません。

内容

- [概念](#)
- [メトリクスフィルターのフィルターパターン構文](#)
- [メトリクスフィルターの作成](#)
- [メトリクスフィルターの一覧表示](#)
- [メトリクスフィルターの削除](#)

概念

各メトリクスフィルターは以下のキー要素で構成されています。

デフォルト値

ログが取り込まれたものの一致するログが見つからなかった期間中にメトリクスフィルターに報告された値。この値を 0 に設定することで、データはこのような各期間の間にも報告されるため、一致するデータがない期間がある「むらがある」メトリクスを回避できます。1 分間の期間内に取り込まれたログがない場合は、値は報告されません。

メトリクスフィルターによって作成されたメトリクスにディメンションを割り当てると、そのメトリクスにデフォルト値を割り当てることはできません。

ディメンション

ディメンションは、メトリクスをさらに定義するキーと値のペアです。メトリクスフィルターから作成されたメトリクスにディメンションを割り当てることができます。ディメンションはメトリクスの一意の識別子の一部であるため、ログから一意の名前/値のペアが抽出されるたびに、そのメトリクスの新しいバリエーションが作成されます。

フィルタパターン

各ログイベントのデータを CloudWatch Logs がどのように解釈するかについての記号による説明です。例えば、ログエントリにはタイムスタンプ、IP アドレス、文字列などが含まれる可能性があります。パターンを使用して、ログファイルの検索対象を指定します。

メトリクス名

モニタリングされたログ情報が発行される CloudWatch メトリクスの名前です。例えば、ErrorCount というメトリクスに発行できます。

メトリクス名前空間

新しい CloudWatch メトリクスの送信先名前空間です。

メトリクス値

一致するログが見つかるたびにメトリクスに発行する数値。例えば、「Error」など特定の語句の発生回数をカウントする場合、その値は発生するごとに「1」になります。転送されたバイト数をカウントする場合は、ログイベントに見つかった実際のバイト数で増分できます。

メトリクスフィルターのフィルターパターン構文

Note

メトリクスフィルターと CloudWatch Logs Insights クエリの違い

一致するログが見つかるたびに、指定された数値がメトリクスフィルターに追加されるという点において、メトリクスフィルターは CloudWatch Logs Insights クエリと異なります。詳細については、「[メトリクスフィルターのメトリクス値を設定する](#)」を参照してください。Amazon CloudWatch Logs Insights のクエリ言語でロググループをクエリする方法については、[CloudWatch Logs Insights 言語のクエリ構文](#) を参照してください。

[一般的なフィルターパターンの例]

メトリクスフィルターの他に、[サブスクリプションフィルター](#)と[フィルターログイベント](#)に適用される汎用フィルターパターン構文の詳細については、次の例を含む[メトリクスフィ](#)

[ルター、サブスクリプションフィルター、フィルターログイベントのフィルターパターン構文](#)を参照してください。

- サポートされている正規表現 (regex) 構文
- 非構造化ログイベントでの語句の一致
- JSON ログイベントの語句の一致
- スペース区切りのログイベントの語句一致

メトリクスフィルターを使用すると、CloudWatch ログに入力されるログデータを検索およびフィルタリングし、フィルタリングされたログデータからメトリクス観測データを抽出し、データポイントを CloudWatch ログメトリクスに変換できます。ユーザーは、CloudWatch ログに送信されるログデータを検索するための語句とパターンを定義します。メトリクスフィルターはロググループに割り当てられ、ロググループに割り当てられたすべてのフィルターはそのログストリームに適用されます。

メトリクスフィルターが語句と一致するとき、メトリクスの数を特定の数値で増えます。例えば、ログイベントの中で ERROR という単語の出現回数を数えるメトリクスフィルターを作成します。

メトリクスに測定単位と寸法を割り当てることができます。例えば、ログイベント内で [ERROR] という単語の出現回数を数えるメトリクスフィルターを作成する場合、[ERROR] という単語が含まれるログイベントの合計数を示す `ErrorCount` というディメンションを指定し、レポートされたエラーコードでデータをフィルタリングすることができます。

Tip

測定の単位をメトリクスに割り当てるとき、必ず正しい単位を指定してください。後で単位を変更すると、変更が有効にならない場合があります。CloudWatch がサポートする単位の詳細なリストについては、「Amazon CloudWatch API リファレンス」の「[MetricDatum](#)」を参照してください。

トピック

- [メトリクスフィルターのメトリクス値を設定する](#)
- [JSON の値またはスペース区切りログイベントからメトリクスとともにディメンションを発行する](#)
- [ログイベントの値を使用してメトリクスの値を増分する](#)

メトリクスフィルターのメトリクス値を設定する

メトリクスフィルターを作成する際は、フィルターパターンを定義し、メトリクス値とデフォルト値を指定します。メトリクス値は、数値、名前付き識別子、または数値識別子に設定できます。デフォルト値を指定しないと、メトリクスフィルターで一致するものが見つからない場合、CloudWatch はデータをレポートしません。値が 0 であっても、デフォルト値を指定することをお勧めします。デフォルト値を設定すると、CloudWatch がデータをより正確にレポートし、CloudWatch がむらがあるメトリクスを集計するのを防ぐことができます。CloudWatch は、1 分ごとにメトリクス値を集計してレポートします。

メトリクスフィルターがログイベント内で一致するものを見つけたら、メトリクスの数がメトリクスの値だけ増加します。メトリクスフィルターで一致が見つからない場合、CloudWatch はメトリクスのデフォルト値をレポートします。例えば、ロググループが毎分 2 つのレコードを公開し、メトリクス値は 1 で、デフォルト値は 0 であるとします。最初の 1 分で両方のログレコードに一致が見つかった場合、その分のメトリクス値は 2 になります。次の 1 分間にどちらのレコードでも一致が見つからなかった場合、その分のデフォルト値は 0 となります。メトリクスフィルターが生成するメトリクスにディメンションを割り当てると、それらのメトリクスのデフォルト値を指定することはできません。

また、静的値ではなく、ログイベントから抽出された値でメトリクスを増分するようにメトリクスフィルターを設定することもできます。詳細については、「[ログイベントの値を使用してメトリクスの値を増分する](#)」を参照してください。

JSON の値またはスペース区切りログイベントからメトリクスとともにディメンションを発行する

CloudWatch コンソールまたは AWS CLI を使用して、JSON およびスペース区切りのログイベントが生成するメトリクスを使用してディメンションを発行するメトリクスフィルターを作成できます。ディメンションとは名前と値のペアであり、JSON およびスペース区切りフィルターパターンでのみ使用できます。最大 3 つのディメンションを持つ JSON およびスペース区切りメトリクスフィルターを作成できます。ディメンションの詳細とディメンションをメトリクスに割り当てる方法の詳細については、以下のセクションを参照してください。

- 「Amazon CloudWatch ユーザーガイド」の「[Dimensions](#)」
- 「Amazon CloudWatch Logs ユーザーガイド<https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/ExtractBytesExample.html>」の「例: Apache ログからフィールドを抽出してディメンションを割り当てる」

⚠ Important

ディメンションには、カスタムメトリクスと同じ請求を収集する値が含まれています。不測の請求を防ぐため、IPAddress または requestId などディメンションとするなど、高カーディナリティフィールドを指定しないでください。

メトリクスをログイベントから抽出すると、カスタムメトリクスとして料金が発生します。意図しない高額請求の徴収を防ぐため、Amazon は、特定の時間内に指定したディメンションのために 1000 の異なる名前と値のペアを生成した場合、メトリクスフィルターを無効化することがあります。

見積費用を通知する請求アラームを作成できます。詳細については、「[AWS 予想請求額をモニタリングする請求アラームの作成](#)」を参照してください。

JSON ログイベントからメトリクスとともにディメンションを発行する

次の例には、JSON メトリクスフィルターでディメンションを指定する方法を説明するコードスニペットが含まれています。

Example: JSON log event

```
{
  "eventType": "UpdateTrail",
  "sourceIPAddress": "111.111.111.111",
  "arrayKey": [
    "value",
    "another value"
  ],
  "objectList": [
    {
      "name": "a",
      "id": 1
    },
    {
      "name": "b",
      "id": 2
    }
  ]
}
```


 Note

サンプルの JSON ログイベントを使用してサンプルメトリクスフィルターをテストする場合は、サンプル JSON ログを 1 行で入力する必要があります。

Example: Metric filter

JSON ログイベントにプロパティ `eventType` および `"sourceIPAddress"` が含まれるたびに、メトリクスフィルターによってメトリクスが増分されます。

```
{ $.eventType = "*" && $.sourceIPAddress != 123.123.* }
```

JSON メトリクスフィルターを作成するときに、メトリクスフィルター内の任意のプロパティをディメンションとして指定できます。例えば、`eventType` をディメンションとして設定するには、以下を使用します。

```
"eventType" : $.eventType
```

サンプルメトリクスには、`"eventType"` という名前のディメンションが含まれており、サンプルログイベント内のディメンションの値は `"UpdateTrail"` です。

スペース区切りのログイベントからメトリクスとともにディメンションを発行する

次の例には、スペース区切りのメトリクスフィルターでディメンションを指定する方法を説明するコードスニペットが含まれています。

Example: Space-delimited log event

```
127.0.0.1 Prod frank [10/Oct/2000:13:25:15 -0700] "GET /index.html HTTP/1.0" 404  
1534
```

Example: Metric filter

```
[ip, server, username, timestamp, request, status_code, bytes > 1000]
```

メトリクスフィルターは、スペース区切りのログイベントにフィルターで指定されているいずれかのフィールドが含まれる場合に、メトリクスを増分します。例えば、メトリクスフィルターは、サンプルのスペース区切りのログイベントで次のフィールドと値を検索します。

```
{
  "$bytes": "1534",
  "$status_code": "404",

  "$request": "GET /index.html HTTP/1.0",
  "$timestamp": "10/Oct/2000:13:25:15 -0700",
  "$username": "frank",
  "$server": "Prod",
  "$ip": "127.0.0.1"
}
```

スペース区切りのメトリクスフィルターを作成するときに、メトリクスフィルター内の任意のフィールドをディメンションとして指定できます。例えば、`server` をディメンションとして設定するには、以下を使用します。

```
"server" : $server
```

サンプルメトリクスフィルターには、`server` という名前のディメンションがあり、サンプルログイベント内のディメンションの値は `"Prod"` です。

Example: Match terms with AND (&&) and OR (||)

論理演算子 AND (「&&」) および OR (「||」) を使用して、条件を含むスペース区切りメトリクスフィルターを作成できます。次のメトリクスフィルターでは、イベントの最初の単語が `ERROR` (エラー) または `WARN` (警告) の超文字列としてログイベントが返されます。

```
[w1=ERROR || w1=%WARN%, w2]
```

ログイベントの値を使用してメトリクスの値を増分する

ログイベントで見つかった数値を公開するメトリクスフィルターを作成できます。このセクションの手順では、次のサンプルメトリクスフィルターを使用して、JSON ログイベントの数値をメトリクスに公開する方法を示します。

```
{ $.latency = * } metricValue: $.latency
```

ログイベントの値を発行するメトリクスフィルターを作成するには

1. CloudWatch コンソールの <https://console.aws.amazon.com/cloudwatch/> を開いてください。
2. ナビゲーションペインで、[ログ]、[ロググループ] の順に選択します。
3. ロググループを選択または作成します。

ロググループの作成手順については、「Amazon CloudWatch Logs ユーザーガイド」の「[CloudWatch Logs にロググループを作成する](#)」を参照してください。

4. [アクション]、[メトリクスフィルターの作成] の順に選択します。
5. [Filter Pattern] (フィルターパターン) に **{ \$.latency = * }** と入力し、[Next] (次へ) を選択します。
6. [Metric Name] (メトリクス名) に、「myMetric」と入力します。
7. [メトリクス値] に「**\$.latency**」と入力します。
8. [Default Value] (デフォルト値) に 0 と入力し、[Next] (次へ) を選択します。

値が 0 であっても、デフォルト値を指定することをお勧めします。デフォルト値を設定すると、CloudWatch がデータをより正確にレポートし、CloudWatch がむらがあるメトリクスを集計するのを防ぐことができます。CloudWatch は、1 分ごとにメトリクス値を集計してレポートします。

9. [Create metric filter] (メトリクスフィルターの作成) を選択します。

サンプルメトリクスフィルターは、語句 "latency" をサンプル JSON ログイベントで照合し、数値 50 をメトリクスの [myMetric] に発行します。

```
{
```

```
"latency": 50,  
"requestType": "GET"  
}
```

メトリクスフィルターの作成

以下の手順は、メトリクスフィルターの作成方法を示しています。

例

- [ロググループのメトリクスフィルターの作成](#)
- [例: ログイベントのカウント](#)
- [例: 語句の出現回数をカウントする](#)
- [例: HTTP 404 コードをカウントする](#)
- [例: HTTP 4xx コードをカウントする](#)
- [例: Apache ログからフィールドを抽出してディメンションを割り当てる](#)

ロググループのメトリクスフィルターの作成

ロググループのメトリクスフィルターを作成するには、次の手順に従います。メトリクスは、データポイントがいくつか見つかるまで表示されません。

CloudWatch コンソールを使用してメトリクスフィルターを作成するには

1. CloudWatch コンソールの <https://console.aws.amazon.com/cloudwatch/> を開いてください。
2. ナビゲーションペインで、[ログ]、[ロググループ] の順に選択します。
3. ロググループの名前を選択します。
4. [Actions]、[メトリクスフィルターの作成] の順に選択します。
5. [フィルターパターン] に、フィルターパターンを入力します。詳細については、「[メトリクスフィルター、サブスクリプションフィルター、フィルターログイベント、およびライブテールのフィルターパターン構文](#)」を参照してください。
6. (オプション) 一元化されたロググループを使用している場合は、[フィルター選択基準] で、ソースアカウント (@aws.account)、ソースリージョン (@aws.region)、またはその両方の条件に基づいてフィルターを指定できます。
7. (オプション) フィルターパターンをテストするには、[テストパターン] に、パターンをテストする 1 つまたは複数のログイベントを入力します。各ログイベントは 1 行にフォーマットする必

必要があります。改行は、[ログイベントメッセージ] ボックスのログイベントを区切るために使用されます。

8. [次へ] を選択し、メトリクスフィルターの名前を入力します。
9. [Metric details (メトリクスの詳細)] の [Metric namespace (メトリクス名前空間)] で、メトリクスを発行する CloudWatch 名前空間の名前を入力します。名前空間がまだ存在しない場合は、[新規作成] が選択されていることを確認します。
10. [メトリクス名] に、新しいメトリクスの名前を入力します。
11. メトリクスフィルターでフィルター内のキーワードの出現回数をカウントする場合は、[メトリクス値] に「1」と入力します。これにより、キーワードの 1 つを含むログイベントごとに、メトリクスが 1 ずつ増加します。

または、**\$size** などのトークンを入力します。これにより、size フィールドを含むすべてのログイベントについて、size フィールド内の数値だけメトリクスが増加します。

12. (オプション) [Unit] (単位) で、メトリクスに割り当てる単位を選択します。単位を指定しない場合、単位は None に設定されます。
13. (オプション) メトリクスの 3 つのディメンションの名前とトークンを入力します。メトリクスフィルターが生成するメトリクスにディメンションを割り当てると、それらのメトリクスのデフォルト値を指定することはできません。

 Note

ディメンションは、JSON またはスペース区切りメトリクスフィルターでのみサポートされます。

14. [Create metric filter] (メトリクスフィルターの作成) を選択します。ナビゲーションペインから作成したメトリクスフィルターを見つけることができます。[Logs] を選択し、ロググループを選択します。メトリクスフィルターを作成したロググループの名前を選択し、[メトリクスフィルター] タブを選択します。

例: ログイベントのカウント

ログイベントのモニタリングで最もシンプルなタイプは、発生したログのイベント数のカウントです。目的はすべてのイベントのカウントや、「ハートビート」形式のモニタリングの作成、あるいは単にメトリクスフィルターの作成練習の場合もあります。

次の CLI の例では、MyAppAccessCount というメトリクスフィルタをロググループ MyApp/access.log に適用して、CloudWatch の名前空間 MyNamespace にメトリクス EventCount を生成します。フィルタは、すべてのログイベントコンテンツに一致し、メトリクスを 1 ずつ増加させるように設定されています。

CloudWatch コンソールを使用してメトリクスフィルタを作成するには

1. CloudWatch コンソール (<https://console.aws.amazon.com/cloudwatch/>) を開きます。
2. ナビゲーションペインで、[Log groups] (ロググループ) を選択します。
3. ロググループの名前を選択します。
4. Actions、[メトリクスフィルターの作成] を選択します。
5. [フィルターパターン] および [テストするログデータの選択] は空白のままにします。
6. [次へ] を選択し、[フィルター名] に **EventCount** と入力します。
7. [メトリクスの詳細] の [メトリクス名前空間] に、「**MyNameSpace**」と入力します。
8. [メトリクス名] に「**MyAppEventCount**」と入力します。
9. [メトリクス値] が 1 であることを確認します。これにより、各ログイベントのカウントは 1 ずつ増分されます。
10. [デフォルト値] に 0 と入力し、[次へ] を選択します。デフォルト値を指定すると、ログイベントが発生しない期間でもデータが報告され、データが存在しないことがある、むらのあるメトリクスを回避できます。
11. [メトリクスフィルターの作成] を選択します。

を使用してメトリクスフィルターを作成するには AWS CLI

コマンドプロンプトで、次のコマンドを実行します。

```
aws logs put-metric-filter \  
  --log-group-name MyApp/access.log \  
  --filter-name EventCount \  
  --filter-pattern " " \  
  --metric-transformations \  
    metricName=MyAppEventCount,metricNamespace=MyNamespace,metricValue=1,defaultValue=0
```

イベントデータを投稿することで、この新しいポリシーをテストできます。メトリクス MyAppAccessEventCount に発行されたデータポイントを参照する必要があります。

を使用してイベントデータを投稿するには AWS CLI

コマンドプロンプトで、次のコマンドを実行します。

```
aws logs put-log-events \  
  --log-group-name MyApp/access.log --log-stream-name TestStream1 \  
  --log-events \  
    timestamp=1394793518000,message="Test event 1" \  
    timestamp=1394793518000,message="Test event 2" \  
    timestamp=1394793528000,message="This message also contains an Error"
```

例: 語句の出現回数をカウントする

ログイベントにはよく、カウントしておきたい重要なメッセージが含まれています。操作の成功または失敗についてなどです。例えば、特定の操作に失敗すると、エラーが発生してログファイルに記録される場合があります。エラーの傾向を理解するためのこれらのエントリをモニタリングする場合があります。

次の例では、Error という語句をモニタリングするメトリクスフィルタが作成されます。ポリシーはすでに作成されてロググループ MyApp/message.log に追加されています。CloudWatch Logs は、MyApp/message.log という名前空間の CloudWatch カスタムメトリクス ErrorCount に、Error を含むイベントごとに「1」の値をともなったデータポイントを発行します。Error という単語を含むイベントがない場合、値 0 は発行されません。このデータを CloudWatch コンソールでグラフ化するときには、必ず合計の統計を使用してください。

メトリクスフィルタを作成した後、CloudWatch コンソールでメトリクスを表示できます。表示するメトリクスを選択するときに、ロググループ名と一致するメトリクス名前空間を選択します。詳細については、[使用可能なメトリクスの表示](#)を参照してください。

CloudWatch コンソールを使用してメトリクスフィルタを作成するには

1. CloudWatch コンソール (<https://console.aws.amazon.com/cloudwatch/>) を開きます。
2. ナビゲーションペインで、[Log groups] (ロググループ) を選択します。
3. ロググループの名前を選択します。
4. [アクション]、[メトリクスフィルタの作成] の順に選択します。
5. [フィルターパターン] に **Error** と入力します。

Note

[フィルターパターン] のすべての項目は大文字と小文字が区別されます。

6. (オプション) フィルターパターンをテストするには、[テストパターン] に、パターンのテストに使用する 1 つまたは複数のログイベントを入力します。[ログイベントメッセージ] ボックスのログイベントを区切るために改行が使用されるため、各ログイベントは 1 行以内である必要があります。
7. [次へ] を選択し、[メトリクスの割り当て] ページの [フィルター名] に **MyAppErrorCount** と入力します。
8. [メトリクスの詳細] の [メトリクス名前空間] に、「MyNameSpace」と入力します。
9. [メトリクス名] に「ErrorCount」と入力します。
10. [メトリクス値] が 1 であることを確認します。これにより、「Error」を含む各ログイベントのカウントは 1 ずつ増分されます。
11. [デフォルト値] に 0 と入力し、[次へ] を選択します。
12. [メトリクスフィルターの作成] を選択します。

を使用してメトリクスフィルターを作成するには AWS CLI

コマンドプロンプトで、次のコマンドを実行します。

```
aws logs put-metric-filter \  
  --log-group-name MyApp/message.log \  
  --filter-name MyAppErrorCount \  
  --filter-pattern 'Error' \  
  --metric-transformations \  
    metricName=ErrorCount,metricNamespace=MyNameSpace,metricValue=1,defaultValue=0
```

メッセージに「Error」を含むイベントを投稿することで、この新しいポリシーをテストできます。

を使用してイベントを投稿するには AWS CLI

コマンドプロンプトで、次の コマンドを実行します。パターンでは大文字と小文字が区別されます。

```
aws logs put-log-events \  
  --log-group-name MyApp/access.log --log-stream-name TestStream1 \  
  --log-events \  
    timestamp=1394793518000,message="This message contains an Error" \  
    timestamp=1394793528000,message="This message also contains an Error"
```


例: HTTP 404 コードをカウントする

CloudWatch Logs を使用して、Apache サーバーが HTTP 404 レスポンス (ページが見つからない場合のレスポンスコード) を返した数をモニタリングできます。サイトの訪問者が目的のリソースを見つけられなかった頻度を理解するためにモニタリングする場合があります。ログレコードが各ロギイベント (サイト訪問) に関する次の情報を含むように設定されている前提です。

- 要求者の IP アドレス
- RFC 1413 ID
- Username
- タイムスタンプ
- リクエスト方法およびリクエストされたリソースとプロトコル
- リクエストに対する HTTP レスポンスコード
- リクエストで転送されたバイト数

例は次のようになります。

```
127.0.0.1 - frank [10/Oct/2000:13:55:36 -0700] "GET /apache_pb.gif HTTP/1.0" 404 2326
```

以下の例に示すように、HTTP 404 エラーの構造にイベントが一致するようにルールを指定できます。

CloudWatch コンソールを使用してメトリクスフィルタを作成するには

1. CloudWatch コンソール (<https://console.aws.amazon.com/cloudwatch/>) を開きます。
2. ナビゲーションペインで、[Log groups] (ロググループ) を選択します。
3. Actions、[メトリクスフィルタの作成] を選択します。
4. [フィルターパターン] には **[IP, UserInfo, User, Timestamp, RequestInfo, StatusCode=404, Bytes]** と入力します。
5. (オプション) フィルターパターンをテストするには、[テストパターン] に、パターンのテストに使用する 1 つまたは複数のロギイベントを入力します。[ロギイベントメッセージ] ボックスのロギイベントを区切るために改行が使用されるため、各ロギイベントは 1 行以内である必要があります。
6. [次へ] を選択し、[フィルター名] に HTTP404Errors と入力します。

7. [メトリクスの詳細] の [メトリクス名前空間] に、**MyNamespace** と入力します。
8. [メトリクス名] に、**ApacheNotFoundErrorCode** を入力します。
9. [メトリクス値] が 1 であることを確認します。これにより、各 404 エラーイベントのカウントは 1 ずつ増分されます。
10. [デフォルト値] に 0 と入力し、[次へ] を選択します。
11. [メトリクスフィルターの作成] を選択します。

を使用してメトリクスフィルターを作成するには AWS CLI

コマンドプロンプトで、次のコマンドを実行します。

```
aws logs put-metric-filter \
  --log-group-name MyApp/access.log \
  --filter-name HTTP404Errors \
  --filter-pattern '[ip, id, user, timestamp, request, status_code=404, size]' \
  --metric-transformations \
    metricName=ApacheNotFoundErrorCode,metricNamespace=MyNamespace,metricValue=1
```

この例では、右角括弧や左角括弧、二重引用符、および文字列 404 のようなリテラル文字列が使用されていました。このパターンでは、ログイベントをモニタリングするにはログイベントメッセージ全体が一致する必要があります。

describe-metric-filters コマンドを使用して、メトリクスフィルターの作成を検証できます。このような出力が表示されます。

```
aws logs describe-metric-filters --log-group-name MyApp/access.log

{
  "metricFilters": [
    {
      "filterName": "HTTP404Errors",
      "metricTransformations": [
        {
          "metricValue": "1",
          "metricNamespace": "MyNamespace",
          "metricName": "ApacheNotFoundErrorCode"
        }
      ],
      "creationTime": 1399277571078,
    }
  ]
}
```

```
        "filterPattern": "[ip, id, user, timestamp, request, status_code=404,
size]"
    }
]
```

これでイベントをいくつか手動で投稿できます。

```
aws logs put-log-events \
--log-group-name MyApp/access.log --log-stream-name hostname \
--log-events \
timestamp=1394793518000,message="127.0.0.1 - bob [10/Oct/2000:13:55:36 -0700] \"GET /
apache_pb.gif HTTP/1.0\" 404 2326" \
timestamp=1394793528000,message="127.0.0.1 - bob [10/Oct/2000:13:55:36 -0700] \"GET /
apache_pb2.gif HTTP/1.0\" 200 2326"
```

サンプルログイベントを入力してすぐに、CloudWatch コンソールで `ApacheNotFoundErrorCode` という名前のメトリクスを取得できます。

例: HTTP 4xx コードをカウントする

前の例と同じように、ウェブサービスアクセスログをモニタリングしたり HTTP 応答コードレベルをモニタリングする場合があります。例えば、HTTP 400 レベルのエラーをすべてモニタリングする場合があります。ただし、それぞれのリターンコードに 1 つずつ新しいメトリクスフィルターを指定したくない場合があります。

以下の例は、「[例: HTTP 404 コードをカウントする](#)」の例の Apache アクセスログ形式を使用して、アクセスログから 400 レベルの HTTP コードレスポンスを含むメトリクスを作成する方法を示しています。

CloudWatch コンソールを使用してメトリクスフィルターを作成するには

1. CloudWatch コンソール (<https://console.aws.amazon.com/cloudwatch/>) を開きます。
2. ナビゲーションペインで、[Log groups] (ロググループ) を選択します。
3. Apache サーバーのロググループの名前を選択します。
4. Actions、[メトリクスフィルターの作成] を選択します。
5. [フィルターパターン] に「**[ip, id, user, timestamp, request, status_code=4*, size]**」と入力します。

6. (オプション) フィルターパターンをテストするには、[テストパターン] に、パターンのテストに使用する 1 つまたは複数のログイベントを入力します。[ログイベントメッセージ] ボックスのログイベントを区切るために改行が使用されるため、各ログイベントは 1 行以内である必要があります。
7. [次へ] を選択し、[フィルター名] に「**HTTP4xxErrors**」と入力します。
8. [メトリクスの詳細] の [メトリクス名前空間] に、「**MyNamespace**」と入力します。
9. [メトリクス名] に、「**HTTP4xxErrors**」と入力します。
10. [メトリクス値] に「1」と入力します。これにより、4xx エラーを含む各ログイベントのカウントは 1 ずつ増分されます。
11. [デフォルト値] に「0」と入力し、[次へ] を選択します。
12. [メトリクスフィルターの作成] を選択します。

を使用してメトリクスフィルターを作成するには AWS CLI

コマンドプロンプトで、次のコマンドを実行します。

```
aws logs put-metric-filter \  
  --log-group-name MyApp/access.log \  
  --filter-name HTTP4xxErrors \  
  --filter-pattern '[ip, id, user, timestamp, request, status_code=4*, size]' \  
  --metric-transformations \  
    metricName=HTTP4xxErrors,metricNamespace=MyNamespace,metricValue=1,defaultValue=0
```

put-event 呼び出しの次のデータを使用してこのルールをテストできます。前の例のモニタリングのルールを削除していない場合は、2 つの異なるメトリクスを生成します。

```
127.0.0.1 - - [24/Sep/2013:11:49:52 -0700] "GET /index.html HTTP/1.1" 404 287  
127.0.0.1 - - [24/Sep/2013:11:49:52 -0700] "GET /index.html HTTP/1.1" 404 287  
127.0.0.1 - - [24/Sep/2013:11:50:51 -0700] "GET /~test/ HTTP/1.1" 200 3  
127.0.0.1 - - [24/Sep/2013:11:50:51 -0700] "GET /favicon.ico HTTP/1.1" 404 308  
127.0.0.1 - - [24/Sep/2013:11:50:51 -0700] "GET /favicon.ico HTTP/1.1" 404 308  
127.0.0.1 - - [24/Sep/2013:11:51:34 -0700] "GET /~test/index.html HTTP/1.1" 200 3
```

例: Apache ログからフィールドを抽出してディメンションを割り当てる

カウントの代わりに、個別のログイベント内の値をメトリクス値に使用の方が役に立つ場合があります。この例では、Apache ウェブサーバーが転送したバイト数を計測するメトリクスを作成する抽出ルールの作成方法を示しています。

この例では、作成するメトリクスにディメンションを割り当てる方法も示します。

CloudWatch コンソールを使用してメトリクスフィルタを作成するには

1. CloudWatch コンソール (<https://console.aws.amazon.com/cloudwatch/>) を開きます。
2. ナビゲーションペインで、[Log groups] (ロググループ) を選択します。
3. Apache サーバーのロググループの名前を選択します。
4. Actions、[メトリクスフィルターの作成] を選択します。
5. [フィルターパターン] に「**[ip, id, user, timestamp, request, status_code, size]**」と入力します。
6. (オプション) フィルターパターンをテストするには、[テストパターン] に、パターンのテストに使用する 1 つまたは複数のログイベントを入力します。[ログイベントメッセージ] ボックスのログイベントを区切るために改行が使用されるため、各ログイベントは 1 行以内である必要があります。
7. [次へ] を選択し、[フィルター名] に「**size**」と入力します。
8. [メトリクスの詳細] の [メトリクス名前空間] に、「**MyNameSpace**」と入力します。これは新しい名前空間であるため、[新規作成] が選択されていることを確認してください。
9. [メトリクス名] に、「**BytesTransferred**」と入力します。
10. [メトリクス値] に「**\$size**」と入力します。
11. [Unit] (単位) で、[Bytes] (バイト) を選択します。
12. [Dimension Name] (ディメンション名) で、**IP** と入力します。
13. [Dimension Value] (ディメンションの値) に、**\$ip** と入力し、[Next] (次へ) を選択します。
14. [メトリクスフィルターの作成] を選択します。

を使用してこのメトリクスフィルタを作成するには AWS CLI

コマンドプロンプトで、次のコマンドを実行します。

```
aws logs put-metric-filter \  
--log-group-name MyApp/access.log \  
--filter-name BytesTransferred \  
--filter-pattern '[ip, id, user, timestamp, request, status_code, size]' \  
--metric-transformations \  
metricName=BytesTransferred,metricNamespace=MyNamespace,metricValue='$size'
```

```
aws logs put-metric-filter \  
--log-group-name MyApp/access.log \  
--filter-name BytesTransferred \  
--filter-pattern '[ip, id, user, timestamp, request, status_code, size]' \  
--metric-transformations \  
metricName=BytesTransferred,metricNamespace=MyNamespace,metricValue='$size',unit=Bytes,dimension1=$ip}}'
```

Note

このコマンドでは、この形式を使用して複数のディメンションを指定します。

```
aws logs put-metric-filter \  
--log-group-name my-log-group-name \  
--filter-name my-filter-name \  
--filter-pattern 'my-filter-pattern' \  
--metric-transformations \  
metricName=my-metric-name,metricNamespace=my-metric-namespace,metricValue=my-token,unit=unit,dimensions='{dimension1=$dim,dimension2=$dim2,dim3=$dim3}'
```

put-log-event 呼び出しの次のデータを使用してこのルールをテストできます。前の例のモニタリンググループを削除していない場合は、2つの異なるメトリクスを生成します。

```
127.0.0.1 - - [24/Sep/2013:11:49:52 -0700] "GET /index.html HTTP/1.1" 404 287  
127.0.0.1 - - [24/Sep/2013:11:49:52 -0700] "GET /index.html HTTP/1.1" 404 287  
127.0.0.1 - - [24/Sep/2013:11:50:51 -0700] "GET /~test/ HTTP/1.1" 200 3  
127.0.0.1 - - [24/Sep/2013:11:50:51 -0700] "GET /favicon.ico HTTP/1.1" 404 308  
127.0.0.1 - - [24/Sep/2013:11:50:51 -0700] "GET /favicon.ico HTTP/1.1" 404 308  
127.0.0.1 - - [24/Sep/2013:11:51:34 -0700] "GET /~test/index.html HTTP/1.1" 200 3
```

メトリクスフィルターの一覧表示

ロググループ内のメトリクスフィルタを一覧表示できます。

CloudWatch コンソールを使用してメトリクスフィルタを一覧表示するには

1. CloudWatch コンソール (<https://console.aws.amazon.com/cloudwatch/>) を開きます。
2. ナビゲーションペインで、[Log groups] (ロググループ) を選択します。

3. コンテンツペインのロググループのリストで、[メトリクスフィルター] 列でフィルター数を選択します。

[ロググループ > フィルター] 画面にそのロググループに関連付けられたすべてのメトリクスフィルターが一覧表示されます。

を使用してメトリクスフィルターを一覧表示するには AWS CLI

コマンドプロンプトで、次のコマンドを実行します。

```
aws logs describe-metric-filters --log-group-name MyApp/access.log
```

出力例を次に示します。

```
{
  "metricFilters": [
    {
      "filterName": "HTTP404Errors",
      "metricTransformations": [
        {
          "metricValue": "1",
          "metricNamespace": "MyNamespace",
          "metricName": "ApacheNotFoundErrorCode"
        }
      ],
      "creationTime": 1399277571078,
      "filterPattern": "[ip, id, user, timestamp, request, status_code=404,
size]"
    }
  ]
}
```

メトリクスフィルターの削除

ポリシーは、名前と所属するロググループで識別されます。

CloudWatch コンソールを使用してメトリクスフィルタを削除するには

1. CloudWatch コンソール (<https://console.aws.amazon.com/cloudwatch/>) を開きます。
2. ナビゲーションペインで、[Log groups] (ロググループ) を選択します。

3. コンテンツペインの [メトリクスフィルター] 列で、ロググループのメトリクスフィルター数を選択します。
4. [メトリクスフィルター] 画面で、削除するフィルター名の右側にあるチェックボックスをオンにします。その後、[Delete] を選択します。
5. 確認を求めるメッセージが表示されたら、[Delete] を選択します。

を使用してメトリクスフィルターを削除するには AWS CLI

コマンドプロンプトで、次のコマンドを実行します。

```
aws logs delete-metric-filter --log-group-name MyApp/access.log \  
--filter-name MyFilterName
```


サブスクリプションを使用したログデータのリアルタイム処理

サブスクリプションを使用して CloudWatch Logs からログイベントのリアルタイムフィードにアクセスし、カスタム処理、分析、他のシステムへのロードを行うために、Amazon Kinesis ストリーム、Amazon Data Firehose ストリーム、AWS Lambda などの他のサービスに配信することができます。ログイベントが宛先サービスに送信されると、base64 でエンコードされ、gzip 形式で圧縮されます。

CloudWatch Logs の一元化を使用して、複数のアカウントとリージョンのログデータを一元的にレプリケートすることもできます。詳細については、「[クロスアカウントクロスリージョンログの一元化](#)」を参照してください。

ログイベントのサブスクライブを開始するには、イベントが配信される Amazon Kinesis Data Streams ストリームなどの受信リソースを作成します。サブスクリプションフィルターは、AWS リソースに配信されるログイベントをフィルタリングするために使用するフィルターパターンと、一致するログイベントの送信先に関する情報を定義します。ログイベントは、取り込み後すぐに (通常は 3 分未満) 受信側リソースに送信されます。

Note

サブスクリプションを持つロググループがログ変換を使用する場合、フィルターパターンは変換されたバージョンのログイベントと比較されます。詳細については、「[取り込み中のログの変換](#)」を参照してください。

サブスクリプションは、アカウントレベルとロググループレベルで作成できます。各アカウントには、リージョンごとに 1 つのアカウントレベルのサブスクリプションフィルターを含めることができます。ロググループごとに、最大 2 つのサブスクリプションフィルターを関連付けることができます。

Note

送信先のサービスが、スロットリングの例外や再試行可能なサービス例外 (HTTP 5xx など) の再試行可能なエラーを返した場合、CloudWatch Logs は最大 24 時間配信を再試行し続けます。AccessDeniedException や ResourceNotFoundException などの再試行不可能なエラーの場合、CloudWatch Logs は再配信を試みません。このような場合、サブスクリプショ

ンフィルターは最大 10 分間無効になり、CloudWatch Logs は送信先へのログの送信を再試行します。この無効化期間中、ログはスキップされます。

また、CloudWatch Logs は、サブスクリプションへのログイベントの転送に関する CloudWatch メトリクスを作成します。詳細については、「[CloudWatch メトリクスによるモニタリング](#)」を参照してください。

CloudWatch Logs サブスクリプションを使用して、Amazon OpenSearch Service クラスターにほぼリアルタイムでログデータをストリーミングすることもできます。詳細については、「[Amazon OpenSearch Service への CloudWatch Logs データのストリーミング](#)」を参照してください。

サブスクリプションは、標準ログクラスのロググループでのみサポートされます。ログクラスの詳細については、「[ログクラス](#)」を参照してください。

 Note

サブスクリプションフィルターは、ログイベントをバッチ処理して送信を最適化し、送信先への呼び出しの量を減らすことができます。バッチ処理は必ず行われるわけではありませんが、可能な場合は使用されます。

スケジュールに従ってログデータをバッチ処理および分析するには、[の使用を検討してください](#) [スケジュールされたクエリによるログ分析の自動化](#)。スケジュールされたクエリは CloudWatch Logs Insights クエリを自動的に実行し、Amazon S3 バケットや Amazon EventBridge イベントバスなどの送信先に結果を配信します。

 Note

サブスクリプションフィルターは、少なくとも 1 回イベントを確実に配信しますが、重複イベントが発生することがあります。

内容

- [概念](#)
- [ロググループレベルのサブスクリプションフィルター](#)
- [アカウントレベルのサブスクリプションフィルター](#)
- [クロスアカウント、クロスリージョンのサブスクリプション](#)

- [混乱した代理の防止](#)
- [ログの再帰防止](#)

概念

各サブスクリプションフィルタは以下のキー要素で構成されています。

フィルタパターン

CloudWatch Logs が各ログイベントのデータを解釈する方法のシンボリックな説明と、宛先 AWS リソースに配信される内容を制限するフィルタリング式。フィルタパターンの構文の詳細については、「[メトリクスフィルター、サブスクリプションフィルター、フィルターログイベント、およびライブテールのフィルターパターン構文](#)」を参照してください。

destination arn

サブスクリプションフィードの送信先として使用する Amazon Kinesis Data Streams ストリーム、Firehose ストリーム、または Lambda 関数の Amazon リソースネーム (ARN)。

role arn

選択された宛先にデータを置くのに必要な権限を CloudWatch Logs に付与する IAM ロール。CloudWatch Logs は Lambda 関数自体のアクセスコントロール設定から必要なアクセス権限を取得できるため、Lambda の送信先にはこのロールは必要ありません。

ディストリビューション

送信先にログデータを配信するのに使用する方法。この場合、宛先は Amazon Kinesis Data Streams です。デフォルトでは、ログデータは、ログストリームによってグループ化されています。さらに詳細に分散する場合でも、ログデータはランダムにグループ化することができます。

ロググループレベルのサブスクリプションには、次のキー要素も含まれます。

log group name

サブスクリプションフィルタを関連付けるロググループ。このロググループにアップロードされたすべてのログイベントにはサブスクリプションフィルターが適用され、フィルターに一致するログイベントは、一致するログイベントを受信する宛先サービスに配信されます。

アカウントレベルのサブスクリプションには、次のキー要素も含まれます。

選択基準

アカウントレベルのサブスクリプションフィルターが適用されているロググループを選択するために使用される基準。これを指定しない場合、アカウントレベルのサブスクリプションフィルターは、アカウントのすべてのロググループに適用されます。このフィールドは、無限のロググループを防ぐために使用します。無限ロググループの問題の詳細については、「[ログの再帰防止](#)」を参照してください。

選択基準のサイズ制限は 25 KB です。

一元化されたロググループには、次の主要な要素も含まれます。これらの要素は、ログデータのソースを識別するのに役立つフィールド選択基準として使用でき、一元化されたログから派生したメトリクスのフィルタリングと分析をさらに細かく行えるようになります。

@aws.account

このフィールドは、ログイベントの発生元の AWS アカウント ID を識別します。

@aws.region

このフィールドは、ログイベントが生成された AWS リージョンを識別します。

ロググループレベルのサブスクリプションフィルター

Amazon Kinesis Data Streams、Amazon Data Firehose AWS Lambda、または Amazon OpenSearch Service でサブスクリプションフィルターを使用できます。サブスクリプションフィルターを介してサービスに送信されるログは、base64 でエンコードされ、gzip 形式で圧縮されます。で集中ログを使用している場合は AWS Organizations、@aws.account および @aws.region システムフィールドを出力して、組織内のどのアカウントとリージョンからどのデータを取得するかを特定できます。このセクションでは、Firehose、Lambda、Amazon Kinesis Data Streams、OpenSearch Service にログデータを送信する CloudWatch Logs サブスクリプションフィルターを作成する例を示します。

Note

ログデータを検索する場合は、「[Filter and pattern syntax](#)」を参照してください。

例

- [例 1: Amazon Kinesis Data Streams を使用したサブスクリプションフィルター](#)
- [例 2: を使用したサブスクリプションフィルター AWS Lambda](#)
- [例 3: Amazon Data Firehose を使用したサブスクリプションフィルター](#)
- [例 4: Amazon OpenSearch Service を使用したサブスクリプションフィルター](#)

例 1: Amazon Kinesis Data Streams を使用したサブスクリプションフィルター

次の例では、AWS CloudTrail イベントを含むロググループにサブスクリプションフィルターを関連付けます。サブスクリプションフィルターは、「Root AWS」認証情報によって行われたすべての記録されたアクティビティを、「RootAccess」と呼ばれる Amazon Kinesis Data Streams のストリームに配信します。RootAccess." CloudWatch Logs に AWS CloudTrail イベントを送信する方法の詳細については、AWS CloudTrail [「ユーザーガイド」のCloudTrail イベントを CloudWatch Logs に送信する](#)」を参照してください。

Note

ストリームを作成する前に、生成するログデータのボリュームを計算します。このボリュームを処理するために十分なシャードでストリームを作成するように注意してください。ストリームに十分なシャードがないと、ログストリームはスロットリングされます。ストリームボリューム制限に関する詳細は、「[クォータと制限](#)」を参照してください。

スロットリングされた成果物は、最大 24 時間再試行されます。24 時間が経過すると、失敗した成果物は破棄されます。

スロットリングのリスクを軽減するには、次のステップに従います。

- [PutSubscriptionFilter](#) または [put-subscription-filter](#) を使用してサブスクリプションフィルターを作成するときは、distribution に random を指定します。デフォルトでは、ストリームフィルターディストリビューションはログストリームによって行われ、スロットリングが発生する可能性があります。
- CloudWatch メトリクスを使用したストリームのモニタリング これにより、スロットリングを特定し、必要に応じて構成を調整できます。たとえば、DeliveryThrottling メトリクスを使用してデータをサブスクリプション送信先に転送するときに CloudWatch Logs がスロットルされたログイベントの数を追跡できます。モニタリングの詳細については、「[CloudWatch メトリクスによるモニタリング](#)」をご参照ください。
- Amazon Kinesis Data Streams のストリームにオンデマンドキャパシティモードを使用します。オンデマンドモードは、ワークロードが増加または縮小すると、即座にワークロー

ドに対応します。オンデマンドキャパシティモードの詳細については、「[オンデマンドモード](#)」を参照してください。

- Amazon Kinesis Data Streams のストリームの容量と一致するように CloudWatch サブスクリプションフィルターパターンを制限します。ストリームに送信するデータが多すぎる場合、フィルターサイズを小さくするか、フィルター条件を調整する必要があります。

Amazon Kinesis Data Streams のサブスクリプションフィルターを作成するには

1. 次のコマンドを使用して送信先ストリームを作成します。

```
$ C:\> aws kinesis create-stream --stream-name "RootAccess" --shard-count 1
```

2. ストリームが [アクティブ] になるまで待ちます (これには 1~2 分かかる可能性があります)。次の Amazon Kinesis Data Streams [describe-stream](#) コマンドを使用して、StreamDescription.StreamStatus プロパティを確認できます。さらに、後のステップで必要になるため StreamDescription.StreamARN 値を書き留めます。

```
aws kinesis describe-stream --stream-name "RootAccess"
```

出力例を次に示します。

```
{
  "StreamDescription": {
    "StreamStatus": "ACTIVE",
    "StreamName": "RootAccess",
    "StreamARN": "arn:aws:kinesis:us-east-1:123456789012:stream/RootAccess",
    "Shards": [
      {
        "ShardId": "shardId-000000000000",
        "HashKeyRange": {
          "EndingHashKey": "340282366920938463463374607431768211455",
          "StartingHashKey": "0"
        },
        "SequenceNumberRange": {
          "StartingSequenceNumber":
            "49551135218688818456679503831981458784591352702181572610"
        }
      }
    ]
  }
}
```

```
}
}
```

3. ストリームにデータを置くアクセス許可を CloudWatch Logs に付与する IAM ロールを作成します。まず、ファイル (~/*TrustPolicyForCWL-Kinesis.json* など) で信頼ポリシーを作成する必要があります。テキストエディタを使用してこのポリシーを作成します。IAM コンソールを使用してポリシーを作成しないでください。

このポリシーには、「混乱した代理」のセキュリティ上の問題を防止するための `aws:SourceArn` グローバル条件コンテキストキーが含まれています。詳細については、「[混乱した代理の防止](#)」を参照してください。

```
{
  "Statement": {
    "Effect": "Allow",
    "Principal": { "Service": "logs.amazonaws.com" },
    "Action": "sts:AssumeRole",
    "Condition": {
      "StringLike": { "aws:SourceArn": "arn:aws:logs:region:123456789012:*" }
    }
  }
}
```

4. `create-role` コマンドを使用し、信頼ポリシーファイルを指定して IAM ロールを作成します。後のステップで必要になるため、返された `Role.Arn` 値も書き留めます。

```
aws iam create-role --role-name CWLtoKinesisRole --assume-role-policy-document
file:///~/TrustPolicyForCWL-Kinesis.json
```

次は出力の例です。

```
{
  "Role": {
    "AssumeRolePolicyDocument": {
      "Statement": {
        "Action": "sts:AssumeRole",
        "Effect": "Allow",
        "Principal": {
          "Service": "logs.amazonaws.com"
        },
        "Condition": {
```

```

        "StringLike": {
            "aws:SourceArn": { "arn:aws:logs:region:123456789012:*" }
        }
    },
    "RoleId": "AA0IIAH450GAB4HC5F431",
    "CreateDate": "2015-05-29T13:46:29.431Z",
    "RoleName": "CWLtoKinesisRole",
    "Path": "/",
    "Arn": "arn:aws:iam::123456789012:role/CWLtoKinesisRole"
}

```

5. 権限ポリシーを作成し、CloudWatch Logs がアカウントで実行できるアクションを定義します。まず、ファイル (~/PermissionsForCWL-Kinesis.json など) で権限ポリシーを作成します。テキストエディタを使用してこのポリシーを作成します。IAM コンソールを使用してポリシーを作成しないでください。

```

{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "kinesis:PutRecord",
      "Resource": "arn:aws:kinesis:region:123456789012:stream/RootAccess"
    }
  ]
}

```

6. 次の [put-role-policy](#) コマンドを使用して、アクセス許可ポリシーをロールに関連付けます。

```

aws iam put-role-policy --role-name CWLtoKinesisRole --policy-name Permissions-
Policy-For-CWL --policy-document file://~/PermissionsForCWL-Kinesis.json

```

7. ストリームが [アクティブ] 状態になり、IAM ロールを作成したら、CloudWatch Logs サブスクリプションフィルタを作成できます。サブスクリプションフィルタにより、選択されたロググループから ストリームへのリアルタイムログデータの流れがすぐに開始されます。

```

aws logs put-subscription-filter \
  --log-group-name "CloudTrail/logs" \
  --filter-name "RootAccess" \
  --filter-pattern "{$.userIdentity.type = Root}" \

```



```
--destination-arn "arn:aws:kinesis:region:123456789012:stream/RootAccess" \
--role-arn "arn:aws:iam::123456789012:role/CWLtoKinesisRole"
```

8. サブスクリプションフィルタを設定したら、CloudWatch Logs によりフィルタパターンに一致するすべての受信イベントがストリームに転送されます。Amazon Kinesis Data Streams シャードイテレーターを取得し、Amazon Kinesis Data Streams get-records コマンドを使用して Amazon Kinesis Data Streams レコードを取得することで、これが起きていることを確認できます。

```
aws kinesis get-shard-iterator --stream-name RootAccess --shard-id
shardId-000000000000 --shard-iterator-type TRIM_HORIZON
```

```
{
  "ShardIterator":
  "AAAAAAAAAAAFGU/
kLvNggvndHq2UIF0w5PZc6F01s3e3afsSscRM70JSbjIefg2ub07nk1y6CDxYR1UoGHJNP4m4NFUetzfL
+wev+e2P4djJg4L9wmXKvQYoE+rMUiFq
+p4Cn3Igvq0b5dRA0yybNdRcdzvnC35KQANoHzzahKdRgB9v4scv+3vaq+f+0IK8zM5My8ID
+g6rMo7UKWeI4+IWIK20Sh0uP"
}
```

```
aws kinesis get-records --limit 10 --shard-iterator "AAAAAAAAAAAFGU/
kLvNggvndHq2UIF0w5PZc6F01s3e3afsSscRM70JSbjIefg2ub07nk1y6CDxYR1UoGHJNP4m4NFUetzfL
+wev+e2P4djJg4L9wmXKvQYoE+rMUiFq
+p4Cn3Igvq0b5dRA0yybNdRcdzvnC35KQANoHzzahKdRgB9v4scv+3vaq+f+0IK8zM5My8ID
+g6rMo7UKWeI4+IWIK20Sh0uP"
```

Amazon Kinesis Data Streams がデータを返す前に、この呼び出しを数回行う必要がある場合があります。

レコードの配列を含むレスポンスが表示されます。Amazon Kinesis Data Streams レコードのデータ属性は base64 でエンコードされ、gzip 形式で圧縮されます。raw データは、コマンドラインから次の UNIX コマンドを使用して調べることができます。

```
echo -n "<Content of Data>" | base64 -d | zcat
```

base64 でデコードおよび解凍されたデータは、次の構造を使用して JSON としてフォーマットされます。

```
{
  "owner": "111111111111",
  "logGroup": "CloudTrail/logs",
  "logStream": "111111111111_CloudTrail/logs_us-east-1",
  "subscriptionFilters": [
    "Destination"
  ],
  "messageType": "DATA_MESSAGE",
  "logEvents": [
    {
      "id": "31953106606966983378809025079804211143289615424298221568",
      "timestamp": 1432826855000,
      "message": "{\"eventVersion\":\"1.03\",\"userIdentity\":{\"type\":\n\"Root\"}}",
    },
    {
      "id": "31953106606966983378809025079804211143289615424298221569",
      "timestamp": 1432826855000,
      "message": "{\"eventVersion\":\"1.03\",\"userIdentity\":{\"type\":\n\"Root\"}}",
    },
    {
      "id": "31953106606966983378809025079804211143289615424298221570",
      "timestamp": 1432826855000,
      "message": "{\"eventVersion\":\"1.03\",\"userIdentity\":{\"type\":\n\"Root\"}}",
    }
  ]
}
```

上のデータ構造の主要要素は次のとおりです。

owner (オーナー)

元のログデータの AWS アカウント ID。

logGroup

発行元ログデータのロググループ名。

logStream

発行元ログデータのログストリーム名。

subscriptionFilters

発行元ログデータと一致したサブスクリプションフィルタ名のリスト。

messageType

データメッセージは、"DATA_MESSAGE" 型を使用します。CloudWatch Logs は、主に送信先に到達可能かどうかを確認するために、「CONTROL_MESSAGE」タイプの Amazon Kinesis Data Streams レコードを出力することがあります。

logEvents

ログイベントレコードの配列として表される実際のログデータ。"id" プロパティは、各ログイベントの一意識別子です。

例 2: を使用したサブスクリプションフィルタ AWS Lambda

この例では、ログデータを AWS Lambda 関数に送信する CloudWatch Logs サブスクリプションフィルタを作成します。

Note

Lambda 関数を作成する前に、生成するログデータのボリュームを計算します。このボリュームを処理できる関数を作成するように注意してください。関数に十分なボリュームがないと、ログストリームはスロットリングされます。Lambda の制限の詳細については、「[AWS Lambda の制限](#)」を参照してください。

Lambda のサブスクリプションフィルタを作成するには

1. AWS Lambda 関数を作成します。

Lambda 実行ロールをセットアップ済みであることを確認します。詳細については、AWS Lambda デベロッパーガイドの「[ステップ 2.2: IAM ロール \(実行ロール\) を作成する](#)」を参照してください。

2. テキストエディターを開き、以下の内容で helloWorld.js という名前のファイルを作成します。

```
var zlib = require('zlib');
exports.handler = function(input, context) {
```

```
var payload = Buffer.from(input.awslogs.data, 'base64');
zlib.gunzip(payload, function(e, result) {
  if (e) {
    context.fail(e);
  } else {
    result = JSON.parse(result.toString());
    console.log("Event Data:", JSON.stringify(result, null, 2));
    context.succeed();
  }
});
};
```

3. helloWorld.js ファイルを圧縮して helloWorld.zip という名前で保存します。
4. 次のコマンドを使用します。ロールは、最初のステップで設定した Lambda 実行ロールです。

```
aws lambda create-function \
  --function-name helloworld \
  --zip-file fileb://file-path/helloWorld.zip \
  --role lambda-execution-role-arn \
  --handler helloworld.handler \
  --runtime nodejs12.x
```

5. CloudWatch Logs に、関数を実行するためのアクセス権限を付与します。次のコマンドを使用します。プレースホルダーは自身のアカウントに置き換え、プレースホルダーロググループは処理するロググループに置き換えます。

```
aws lambda add-permission \
  --function-name "helloworld" \
  --statement-id "helloworld" \
  --principal "logs.amazonaws.com" \
  --action "lambda:InvokeFunction" \
  --source-arn "arn:aws:logs:region:123456789123:log-group:TestLambda:*" \
  --source-account "123456789012"
```

6. 次のコマンドを使用してサブスクリプションフィルタを作成します。プレースホルダーアカウントは自身のアカウントに置き換え、プレースホルダーロググループは処理するロググループに置き換えます。

```
aws logs put-subscription-filter \
  --log-group-name myLogGroup \
  --filter-name demo \
  --filter-pattern "" \
```

```
--destination-arn arn:aws:lambda:region:123456789123:function:helloworld
```

7. (オプション) サンプルのログイベントを使用してテストします。コマンドプロンプトで、次のコマンドを実行します。これにより、サブスクライブしたストリームに単純なログメッセージが送信されます。

Lambda 関数の出力を確認するには、Lambda 関数に移動して、/aws/lambda/helloworld にある出力を参照します。

```
aws logs put-log-events --log-group-name myLogGroup --log-stream-name stream1 --
log-events "[{\"timestamp\":<CURRENT TIMESTAMP MILLIS> , \"message\": \"Simple
Lambda Test\"}]"
```

Lambda の配列を含むレスポンスが表示されます。Lambda レコードの [Data] (データ) 属性は、base64 でエンコードされており、gzip 形式で圧縮されています。Lambda が受け取る実際のペイロードは、{ "awslogs": { "data": "BASE64ENCODED_GZIP_COMPRESSED_DATA" } } の形式になります。raw データは、コマンドラインから次の UNIX コマンドを使用して調べることができます。

```
echo -n "<BASE64ENCODED_GZIP_COMPRESSED_DATA>" | base64 -d | zcat
```

base64 でデコードおよび解凍されたデータは、次の構造を使用して JSON としてフォーマットされます。

```
{
  "owner": "123456789012",
  "logGroup": "CloudTrail",
  "logStream": "123456789012_CloudTrail_us-east-1",
  "subscriptionFilters": [
    "Destination"
  ],
  "messageType": "DATA_MESSAGE",
  "logEvents": [
    {
      "id": "31953106606966983378809025079804211143289615424298221568",
      "timestamp": 1432826855000,
      "message": "{\"eventVersion\":\"1.03\",\"userIdentity\":{\"type\":
\\\"Root\\\"}"}",
    },
    {
      "id": "31953106606966983378809025079804211143289615424298221569",
```

```
      "timestamp": 1432826855000,
      "message": "{\"eventVersion\":\"1.03\",\"userIdentity\":{\"type\":\n\n\"Root\"}}",
    },
    {
      "id": "31953106606966983378809025079804211143289615424298221570",
      "timestamp": 1432826855000,
      "message": "{\"eventVersion\":\"1.03\",\"userIdentity\":{\"type\":\n\n\"Root\"}}",
    }
  ]
}
```

上のデータ構造の主要要素は次のとおりです。

owner (オーナー)

元のログデータの AWS アカウント ID。

logGroup

発行元ログデータのロググループ名。

logStream

発行元ログデータのログストリーム名。

subscriptionFilters

発行元ログデータと一致したサブスクリプションフィルタ名のリスト。

messageType

データメッセージは、"DATA_MESSAGE" 型を使用します。CloudWatch Logs は、主に送信先が到達可能であるかどうかをチェックするために、"CONTROL_MESSAGE" 型の Lambda レコードを発行することがあります。

logEvents

ログイベントレコードの配列として表される実際のログデータ。"id" プロパティは、各ログイベントの一意識別子です。

例 3: Amazon Data Firehose を使用したサブスクリプションフィルター

この例では、CloudWatch Logs サブスクリプションを作成して、定義されたフィルターに一致する受信ログイベントを Amazon Data Firehose 配信ストリームに送信します。CloudWatch Logs から Amazon Data Firehose に送信されたデータは、既に gzip レベル 6 圧縮で圧縮されているため、Firehose 配信ストリーム内で圧縮を使用する必要はありません。その後、Firehose の解凍機能を使用して、ログを自動的に解凍できます。詳細については、「[CloudWatch Logs を Firehose に送信する](#)」を参照してください。

Note

Firehose ストリームを作成する前に、生成するログデータのボリュームを計算します。このボリュームを処理できる Firehose ストリームを作成するように注意してください。ストリームがこのボリュームを処理できない場合、ログストリームはスロットリングされます。Firehose ストリームボリュームの制限事項の詳細については、「[Amazon Data Firehose Data Limits](#)」をご参照ください。

Firehose のサブスクリプションフィルターを作成するには

1. Amazon Simple Storage Service (Amazon S3) バケットを作成します。CloudWatch Logs 専用に作成したバケットを使用することをお勧めします。ただし、既存のバケットを使用する場合は、ステップ 2 に進みます。

次のコマンドを実行します。プレースホルダーリージョンは、使用するリージョンに置き換えます。

```
aws s3api create-bucket --bucket amzn-s3-demo-bucket2 --create-bucket-configuration LocationConstraint=region
```

出力例を次に示します。

```
{
  "Location": "/amzn-s3-demo-bucket2"
}
```

2. Amazon S3 バケットにデータを置くためのアクセス許可を Amazon Data Firehose に付与する IAM ロールを作成します。

詳細については、「Amazon Data Firehose 開発者ガイド」の「[Controlling Access with Amazon Data Firehose](#)」を参照してください。

まず、テキストエディタを使用して、次のようにファイル `~/TrustPolicyForFirehose.json` で信頼ポリシーを作成します。

```
{
  "Statement": {
    "Effect": "Allow",
    "Principal": { "Service": "firehose.amazonaws.com" },
    "Action": "sts:AssumeRole"
  }
}
```

3. `create-role` コマンドを使用し、信頼ポリシーファイルを指定して IAM ロールを作成します。後のステップで必要になるため、返された `Role.Arn` 値を書き留めます。

```
aws iam create-role \
  --role-name FirehoseToS3Role \
  --assume-role-policy-document file://~/TrustPolicyForFirehose.json

{
  "Role": {
    "AssumeRolePolicyDocument": {
      "Statement": {
        "Action": "sts:AssumeRole",
        "Effect": "Allow",
        "Principal": {
          "Service": "firehose.amazonaws.com"
        }
      }
    },
    "RoleId": "AA0IIAH450GAB4HC5F431",
    "CreateDate": "2015-05-29T13:46:29.431Z",
    "RoleName": "FirehoseToS3Role",
    "Path": "/",
    "Arn": "arn:aws:iam::123456789012:role/FirehoseToS3Role"
  }
}
```


4. アクセス許可ポリシーを作成し、Firehose がアカウントで実行できるアクションを定義します。まず、テキストエディタを使用してファイル `~/PermissionsForFirehose.json` で権限ポリシーを作成します。

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:AbortMultipartUpload",
        "s3:GetBucketLocation",
        "s3:GetObject",
        "s3:ListBucket",
        "s3:ListBucketMultipartUploads",
        "s3:PutObject" ],
      "Resource": [
        "arn:aws:s3:::amzn-s3-demo-bucket2",
        "arn:aws:s3:::amzn-s3-demo-bucket2/*" ]
    }
  ]
}
```

5. 次の `put-role-policy` コマンドを使用して、アクセス許可ポリシーをロールに関連付けます。

```
aws iam put-role-policy --role-name FirehoseToS3Role --policy-name Permissions-Policy-For-Firehose --policy-document file:///~/PermissionsForFirehose.json
```

6. 次のように、送信先 Firehose 配信ストリームを作成します。RoleARN と BucketARN のプレースホルダー値を、作成したロールおよびバケット ARN に置き換えます。

```
aws firehose create-delivery-stream \
  --delivery-stream-name 'my-delivery-stream' \
  --s3-destination-configuration \
  '{"RoleARN": "arn:aws:iam::123456789012:role/FirehoseToS3Role", "BucketARN":  
  "arn:aws:s3:::amzn-s3-demo-bucket2"}'
```

Firehose は、配信された Amazon S3 オブジェクトに対して、YYYY/MM/DD/HH UTC 時間形式のプレフィックスを自動的に使用することに注意してください。時間形式プレフィックスの前に、追加のプレフィックスを指定できます。プレフィックスの最後がフォワードスラッシュ (/) の場合は、Amazon S3 バケット内のフォルダとして表示されます。

7. ストリームがアクティブになるまで待ちます (これには数分かかる可能性があります)。Firehose `describe-delivery-stream` コマンドを使用して、`DeliveryStreamDescription.DeliveryStreamStatus` プロパティをチェックできます。さらに、後のステップで必要になるため、`DeliveryStreamDescription.DeliveryStreamARN` 値を書き留めます。

```
aws firehose describe-delivery-stream --delivery-stream-name "my-delivery-stream"
{
  "DeliveryStreamDescription": {
    "HasMoreDestinations": false,
    "VersionId": "1",
    "CreateTimestamp": 1446075815.822,
    "DeliveryStreamARN": "arn:aws:firehose:us-east-1:123456789012:deliverystream/my-delivery-stream",
    "DeliveryStreamStatus": "ACTIVE",
    "DeliveryStreamName": "my-delivery-stream",
    "Destinations": [
      {
        "DestinationId": "destinationId-0000000000001",
        "S3DestinationDescription": {
          "CompressionFormat": "UNCOMPRESSED",
          "EncryptionConfiguration": {
            "NoEncryptionConfig": "NoEncryption"
          },
          "RoleARN": "delivery-stream-role",
          "BucketARN": "arn:aws:s3:::amzn-s3-demo-bucket2",
          "BufferingHints": {
            "IntervalInSeconds": 300,
            "SizeInMBs": 5
          }
        }
      }
    ]
  }
}
```

8. Firehose 配信ストリームにデータを置くためのアクセス許可を CloudWatch Logs に付与する IAM ロールを作成します。まず、テキストエディタを使用してファイル `~/TrustPolicyForCWL.json` で信頼ポリシーを作成します。

このポリシーには、「混乱した代理」のセキュリティ上の問題を防止するための `aws:SourceArn` グローバル条件コンテキストキーが含まれています。詳細については、「[混乱した代理の防止](#)」を参照してください。

```
{
  "Statement": {
    "Effect": "Allow",
    "Principal": { "Service": "logs.amazonaws.com" },
    "Action": "sts:AssumeRole",
    "Condition": {
      "StringLike": {
        "aws:SourceArn": "arn:aws:logs:region:123456789012:*"
      }
    }
  }
}
```

9. create-role コマンドを使用し、信頼ポリシーファイルを指定して IAM ロールを作成します。後のステップで必要になるため、返された Role.Arn 値を書き留めます。

```
aws iam create-role \
--role-name CWLtoKinesisFirehoseRole \
--assume-role-policy-document file://~/TrustPolicyForCWL.json

{
  "Role": {
    "AssumeRolePolicyDocument": {
      "Statement": {
        "Action": "sts:AssumeRole",
        "Effect": "Allow",
        "Principal": {
          "Service": "logs.amazonaws.com"
        },
        "Condition": {
          "StringLike": {
            "aws:SourceArn": "arn:aws:logs:region:123456789012:*"
          }
        }
      }
    },
    "RoleId": "AA0IIAH450GAB4HC5F431",
    "CreateDate": "2015-05-29T13:46:29.431Z",
    "RoleName": "CWLtoKinesisFirehoseRole",
    "Path": "/",
    "Arn": "arn:aws:iam::123456789012:role/CWLtoKinesisFirehoseRole"
  }
}
```

```
}
```

10. 権限ポリシーを作成し、CloudWatch Logs がアカウントで実行できるアクションを定義します。まず、テキストエディタを使用して権限ポリシーファイル (例: ~/PermissionsForCWL.json) を作成します。

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["firehose:PutRecord"],
      "Resource": [
        "arn:aws:firehose:region:account-id:deliverystream/delivery-stream-
name"]
      }
    ]
  }
}
```

11. put-role-policy コマンドを使用して、権限ポリシーをロールに関連付けます。

```
aws iam put-role-policy --role-name CWLtoKinesisFirehoseRole --policy-
name Permissions-Policy-For-CWL --policy-document file://~/PermissionsForCWL.json
```

12. Amazon Data Firehose 配信ストリームがアクティブ状態になり、IAM ロールを作成したら、CloudWatch Logs サブスクリプションフィルターを作成できます。サブスクリプションフィルターにより、選択されたロググループから Amazon Data Firehose 配信ストリームへのリアルタイムログデータのフローがすぐに開始されます。

```
aws logs put-subscription-filter \
  --log-group-name "CloudTrail" \
  --filter-name "Destination" \
  --filter-pattern "{$.userIdentity.type = Root}" \
  --destination-arn "arn:aws:firehose:region:123456789012:deliverystream/my-
delivery-stream" \
  --role-arn "arn:aws:iam::123456789012:role/CWLtoKinesisFirehoseRole"
```

13. サブスクリプションフィルターを設定したら、CloudWatch Logs からフィルタパターンに一致するすべての受信ログイベントが Amazon Data Firehose 配信ストリームに転送されます。Amazon Data Firehose 配信ストリームに設定された時間バッファ間隔に基づいて、Amazon S3 にデータが表示されるようになります。十分な時間が経過すると、Amazon S3 バケットをチェックしてデータを確認できます。

```
aws s3api list-objects --bucket 'amzn-s3-demo-bucket2' --prefix 'firehose/'
{
  "Contents": [
    {
      "LastModified": "2015-10-29T00:01:25.000Z",
      "ETag": "\"a14589f8897f4089d3264d9e2d1f1610\"",
      "StorageClass": "STANDARD",
      "Key": "firehose/2015/10/29/00/my-delivery-stream-2015-10-29-00-01-21-a188030a-62d2-49e6-b7c2-b11f1a7ba250",
      "Owner": {
        "DisplayName": "cloudwatch-logs",
        "ID": "1ec9cf700ef6be062b19584e0b7d84ecc19237f87b5"
      },
      "Size": 593
    },
    {
      "LastModified": "2015-10-29T00:35:41.000Z",
      "ETag": "\"a7035b65872bb2161388ffb63dd1aec5\"",
      "StorageClass": "STANDARD",
      "Key": "firehose/2015/10/29/00/my-delivery-stream-2015-10-29-00-35-40-7cc92023-7e66-49bc-9fd4-fc9819cc8ed3",
      "Owner": {
        "DisplayName": "cloudwatch-logs",
        "ID": "1ec9cf700ef6be062b19584e0b7d84ecc19237f87b6"
      },
      "Size": 5752
    }
  ]
}
```

```
aws s3api get-object --bucket 'amzn-s3-demo-bucket2' --key 'firehose/2015/10/29/00/my-delivery-stream-2015-10-29-00-01-21-a188030a-62d2-49e6-b7c2-b11f1a7ba250' testfile.gz
{
  "AcceptRanges": "bytes",
  "ContentType": "application/octet-stream",
  "LastModified": "Thu, 29 Oct 2015 00:07:06 GMT",
  "ContentLength": 593,
  "Metadata": {}
}
```

Simple Storage Service (Amazon S3) オブジェクトのデータは、gzip 形式で圧縮されます。raw データは、コマンドラインから次の UNIX コマンドを使用して調べることができます。

```
zcat testfile.gz
```

例 4: Amazon OpenSearch Service を使用したサブスクリプションフィルター

この例では、CloudWatch Logs サブスクリプションを作成して、定義されたフィルターに一致する受信ロギイベントを OpenSearch Service ドメインに送信します。

OpenSearch Service のサブスクリプションフィルターを作成するには

1. OpenSearch Service ドメインを作成します。詳細については、「[OpenSearch Service ドメインの作成](#)」を参照してください。
2. CloudWatch コンソールの <https://console.aws.amazon.com/cloudwatch/> を開いてください。
3. ナビゲーションペインで、[Log groups] (ロググループ) を選択します。
4. ロググループの名前を選択します。
5. [Actions] (アクション)、[Subscription filters] (サブスクリプションフィルター)、[Create Amazon OpenSearch Service subscription filter] (Amazon OpenSearch Service サブスクリプションフィルターを作成) の順に選択します。
6. このアカウントのクラスターにストリーミングするか、別のアカウントにストリーミングするかを選択します。
 - [このアカウント] を選択した場合は、ステップ 1 で作成したドメインを選択します。
 - [別のアカウント] を選択した場合は、そのドメインの ARN とエンドポイントを入力します。
7. 別のアカウントを選択した場合は、ドメイン ARN とエンドポイントを指定します。
8. Amazon OpenSearch Service クラスターの場合、ロググループデータが配信されるクラスターの名前を選択します。
9. ログ形式を選択します。
10. [サブスクリプションフィルターのパターン] に、ロギイベントで検索する用語やパターンを入力します。これにより、関心のあるデータだけが OpenSearch Service クラスターに送信されるようになります。詳細については、「[メトリクスフィルターのフィルターパターン構文](#)」を参照してください。

11. (オプション) [Select log data to test] (テストするログデータの選択) を開き、ログストリームを選択してから、[Test pattern] (パターンをテスト) を選択して、期待通りの結果が出ることを確認します。
12. [Start streaming] (ストリーミングの開始) を選択します。

アカウントレベルのサブスクリプションフィルター

Important

サブスクリプションフィルターで無限の再帰ループが発生するリスクがあり、対処しなければ取り込みにかかる料金が大幅に増加する可能性があります。このリスクを軽減するには、アカウントレベルのサブスクリプションフィルターで選択基準を使用して、サブスクリプション配信ワークフローの一部であるリソースからログデータを取り込むロググループを除外することをお勧めします。この問題および除外するロググループを判断する方法については、「[ログの再帰防止](#)」を参照してください。

アカウント内のロググループのサブセットを含むアカウントレベルのサブスクリプションポリシーを設定できます。アカウントサブスクリプションポリシーは、Amazon Kinesis Data Streams AWS Lambda、または Amazon Data Firehose と連携できます。アカウントレベルのサブスクリプションポリシーを介してサービスに送信されるログは、base64 でエンコードされ、gzip 形式で圧縮されます。このセクションでは、Amazon Kinesis Data Streams、Lambda、Firehose のアカウントレベルのサブスクリプションを作成するための例を示します。

Note

アカウント内のすべてのサブスクリプションフィルターポリシーのリストを表示するには、`--policy-type` パラメータに `SUBSCRIPTION_FILTER_POLICY` の値を持つ `describe-account-policies` コマンドを使用します。詳細については、「[describe-account-policies](#)」を参照してください。

例

- [例 1: Amazon Kinesis Data Streams を使用したサブスクリプションフィルター](#)
- [例 2: を使用したサブスクリプションフィルター AWS Lambda](#)
- [例 3: Amazon Data Firehose を使用したサブスクリプションフィルター](#)

例 1: Amazon Kinesis Data Streams を使用したサブスクリプションフィルター

アカウントレベルのサブスクリプションポリシーで使用する Amazon Kinesis Data Streams データストリームを作成する前に、生成されるログデータのボリュームを計算します。このボリュームを処理するために十分なシャードでストリームを作成するように注意してください。ストリームに十分なシャードがない場合は、スロットリングされます。ストリームボリュームの制限の詳細については、Amazon Kinesis Data Streams ドキュメントの「[クォータと制限](#)」を参照してください。

Warning

複数のロググループのログイベントは宛先に転送されるため、スロットリングのリスクがあります。スロットリングされた成果物は、最大 24 時間再試行されます。24 時間が経過すると、失敗した成果物は破棄されます。

スロットリングのリスクを軽減するには、次のステップに従います。

- CloudWatch メトリクスを使用して Amazon Kinesis Data Streams ストリームをモニタリングします。これにより、スロットリングを特定し、必要に応じて構成を調整することができます。例えば、DeliveryThrottling メトリクスはデータをサブスクリプション送信先に転送するときに CloudWatch Logs がスロットリングされたログイベントの数を追跡できます。詳細については、「[CloudWatch メトリクスによるモニタリング](#)」を参照してください。
- Amazon Kinesis Data Streams のストリームにオンデマンドキャパシティモードを使用します。オンデマンドモードは、ワークロードが増加または縮小すると、即座にワークロードに対応します。詳細については、「[On-demand mode](#)」をご参照ください。
- Amazon Kinesis Data Streams のストリームの容量と一致するように CloudWatch Logs サブスクリプションフィルターパターンを制限します。ストリームに送信するデータが多すぎる場合、フィルターサイズを小さくするか、フィルター条件を調整する必要があります。

次の例では、アカウントレベルのサブスクリプションポリシーを使用して、すべてのログイベントを Amazon Kinesis Data Streams のストリームに転送します。フィルターパターンは、すべてのログイベントをテキストと照合 Test し、Amazon Kinesis Data Streams のストリームに転送します。

Amazon Kinesis Data Streams のアカウントレベルのサブスクリプションポリシーを作成するには

1. 次のコマンドを使用して送信先ストリームを作成します。

```
$ C:\> aws kinesis create-stream --stream-name "TestStream" --shard-count 1
```

2. ストリームがアクティブになるまで数分待ちます。ストリームがアクティブ化どうか確認するには、[describe-stream](#) コマンドを使用して、StreamDescription.StreamStatus プロパティをチェックできます。

```
aws kinesis describe-stream --stream-name "TestStream"
```

出力例を次に示します。

```
{
  "StreamDescription": {
    "StreamStatus": "ACTIVE",
    "StreamName": "TestStream",
    "StreamARN": "arn:aws:kinesis:region:123456789012:stream/TestStream",
    "Shards": [
      {
        "ShardId": "shardId-000000000000",
        "HashKeyRange": {
          "EndingHashKey": "EXAMPLE8463463374607431768211455",
          "StartingHashKey": "0"
        },
        "SequenceNumberRange": {
          "StartingSequenceNumber":
            "EXAMPLE688818456679503831981458784591352702181572610"
        }
      }
    ]
  }
}
```

3. ストリームにデータを置くアクセス許可を CloudWatch Logs に付与する IAM ロールを作成します。まず、ファイル (~/.TrustPolicyForCWL-Kinesis.json など) で信頼ポリシーを作成する必要があります。テキストエディタを使用してこのポリシーを作成します。

このポリシーには、「混乱した代理」のセキュリティ上の問題を防止するための `aws:SourceArn` グローバル条件コンテキストキーが含まれています。詳細については、「[混乱した代理の防止](#)」を参照してください。

```
{
  "Statement": {
    "Effect": "Allow",
    "Principal": { "Service": "logs.amazonaws.com" },
    "Action": "sts:AssumeRole",
    "Condition": {
      "StringLike": { "aws:SourceArn": "arn:aws:logs:region:123456789012:*" }
    }
  }
}
```

4. `create-role` コマンドを使用し、信頼ポリシーファイルを指定して IAM ロールを作成します。後のステップで必要になるため、返された `Role.Arn` 値も書き留めます。

```
aws iam create-role --role-name CWLtoKinesisRole --assume-role-policy-document
file:///~/TrustPolicyForCWL-Kinesis.json
```

次は出力の例です。

```
{
  "Role": {
    "AssumeRolePolicyDocument": {
      "Statement": {
        "Action": "sts:AssumeRole",
        "Effect": "Allow",
        "Principal": {
          "Service": "logs.amazonaws.com"
        },
        "Condition": {
          "StringLike": {
            "aws:SourceArn": { "arn:aws:logs:region:123456789012:*" }
          }
        }
      }
    },
    "RoleId": "EXAMPLE450GAB4HC5F431",
    "CreateDate": "2023-05-29T13:46:29.431Z",
  }
}
```

```

    "RoleName": "CWLtoKinesisRole",
    "Path": "/",
    "Arn": "arn:aws:iam::123456789012:role/CWLtoKinesisRole"
  }
}

```

5. 権限ポリシーを作成し、CloudWatch Logs がアカウントで実行できるアクションを定義します。まず、ファイル (~/PermissionsForCWL-Kinesis.json など) で権限ポリシーを作成します。テキストエディタを使用してこのポリシーを作成します。IAM コンソールを使用してポリシーを作成しないでください。

```

{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "kinesis:PutRecord",
      "Resource": "arn:aws:kinesis:region:123456789012:stream/TestStream"
    }
  ]
}

```

6. 次の [put-role-policy](#) コマンドを使用して、アクセス許可ポリシーをロールに関連付けます。

```

aws iam put-role-policy --role-name CWLtoKinesisRole --policy-name Permissions-
Policy-For-CWL --policy-document file://~/PermissionsForCWL-Kinesis.json

```

7. ストリームが [アクティブ] 状態になり、IAM ロールを作成したら、CloudWatch Logs サブスクリプションフィルターポリシーを作成できます。ポリシーは、ストリームへのリアルタイムログデータのフローをすぐに開始します。この例では、文字列 ERROR を含むすべてのログイベントがストリーミングされます。ただし、LogGroupToExclude1 および LogGroupToExclude2 という名前のロググループ内のイベントは除外されます。

```

aws logs put-account-policy \
  --policy-name "ExamplePolicy" \
  --policy-type "SUBSCRIPTION_FILTER_POLICY" \
  --policy-document '{"RoleArn":"arn:aws:iam::123456789012:role/
CWLtoKinesisRole", "DestinationArn":"arn:aws:kinesis:region:123456789012:stream/
TestStream", "FilterPattern": "Test", "Distribution": "Random"}' \
  --selection-criteria 'LogGroupName NOT IN ["LogGroupToExclude1",
"LogGroupToExclude2"]' \
  --scope "ALL"

```

8. サブスクリプションフィルターを設定したら、CloudWatch Logs によりフィルタパターンと選択基準に一致するすべての受信ログイベントがストリームに転送されます。

selection-criteria フィールドはオプションですが、サブスクリプションフィルターから無限のログ再帰を引き起こす可能性のあるロググループを除外するための重要なフィールドです。この問題および除外するロググループを判断する方法については、「[ログの再帰防止](#)」を参照してください。現在、NOT IN は selection-criteria でサポートされている唯一の演算子です。

Amazon Kinesis Data Streams シャードイテレーターを使用し、Amazon Kinesis Data Streams get-records コマンドを使用して Amazon Kinesis Amazon Kinesis Data Streams レコードを取得することで、ログイベントのフローを確認できます。

```
aws kinesis get-shard-iterator --stream-name TestStream --shard-id
shardId-000000000000 --shard-iterator-type TRIM_HORIZON
```

```
{
  "ShardIterator":
  "AAAAAAAAAAFGU/
kLvNggvndHq2UIF0w5PZc6F01s3e3afsSscRM70JSbjIefg2ub07nk1y6CDxYR1UoGHJNP4m4NFUetzfL
+wev+e2P4djJg4L9wmXKvQYoE+rMUiFq
+p4Cn3Igvq0b5dRA0yybNdRcdzvnC35KQANoHzzahKdRGb9v4scv+3vaq+f+0IK8zM5My8ID
+g6rMo7UKWeI4+IWIK20Sh0uP"
}
```

```
aws kinesis get-records --limit 10 --shard-iterator "AAAAAAAAAAFGU/
kLvNggvndHq2UIF0w5PZc6F01s3e3afsSscRM70JSbjIefg2ub07nk1y6CDxYR1UoGHJNP4m4NFUetzfL
+wev+e2P4djJg4L9wmXKvQYoE+rMUiFq
+p4Cn3Igvq0b5dRA0yybNdRcdzvnC35KQANoHzzahKdRGb9v4scv+3vaq+f+0IK8zM5My8ID
+g6rMo7UKWeI4+IWIK20Sh0uP"
```

Amazon Kinesis Data Streams がデータを返す前に、このコマンドを数回使用する必要がある場合があります。

レコードの配列を含むレスポンスが表示されます。Amazon Kinesis Data Streams レコードのデータ属性は base64 でエンコードされ、gzip 形式で圧縮されます。raw データは、コマンドラインから次の UNIX コマンドを使用して調べることができます。

```
echo -n "<Content of Data>" | base64 -d | zcat
```

base64 でデコードおよび解凍されたデータは、次の構造を使用して JSON としてフォーマットされます。

```
{
  "messageType": "DATA_MESSAGE",
  "owner": "123456789012",
  "logGroup": "Example1",
  "logStream": "logStream1",
  "subscriptionFilters": [
    "ExamplePolicy"
  ],
  "logEvents": [
    {
      "id": "31953106606966983378809025079804211143289615424298221568",
      "timestamp": 1432826855000,
      "message": "{\"eventVersion\":\"1.03\",\"userIdentity\":{\"type\":\n\"Root\"}}",
    },
    {
      "id": "31953106606966983378809025079804211143289615424298221569",
      "timestamp": 1432826855000,
      "message": "{\"eventVersion\":\"1.03\",\"userIdentity\":{\"type\":\n\"Root\"}}",
    },
    {
      "id": "31953106606966983378809025079804211143289615424298221570",
      "timestamp": 1432826855000,
      "message": "{\"eventVersion\":\"1.03\",\"userIdentity\":{\"type\":\n\"Root\"}}",
    }
  ],
  "policyLevel": "ACCOUNT_LEVEL_POLICY"
}
```

データ構造の主な要素は次のとおりです。

messageType

データメッセージは、"DATA_MESSAGE" 型を使用します。CloudWatch Logs は、主に送信先に到達可能かどうかを確認するために、「CONTROL_MESSAGE」タイプの Amazon Kinesis Data Streams レコードを出力することがあります。

owner (オーナー)

元のログデータの AWS アカウント ID。

logGroup

発行元ログデータのロググループ名。

logStream

発行元ログデータのログストリーム名。

subscriptionFilters

発行元ログデータと一致したサブスクリプションフィルタ名のリスト。

logEvents

ログイベントレコードの配列として表される実際のログデータ。"id" プロパティは、各ログイベントの一意識別子です。

policyLevel

ポリシーが適用されたレベル。「ACCOUNT_LEVEL_POLICY」は、アカウントレベルのサブスクリプションフィルターポリシーの policyLevel です。

例 2: を使用したサブスクリプションフィルター AWS Lambda

この例では、ログデータを AWS Lambda 関数に送信する CloudWatch Logs アカウントレベルのサブスクリプションフィルターポリシーを作成します。

Warning

Lambda 関数を作成する前に、生成するログデータのボリュームを計算します。このボリュームを処理できる関数を作成するように注意してください。関数がこのボリュームを処理できない場合、ログストリームはスロットリングされます。すべてのロググループまたはアカウントのロググループのサブセットのログイベントは送信先に転送されるため、スロットリングのリスクがあります。Lambda の制限の詳細については、「[AWS Lambda の制限](#)」を参照してください。

Lambda のアカウントレベルのサブスクリプションフィルターポリシーを作成するには

1. AWS Lambda 関数を作成します。

Lambda 実行ロールをセットアップ済みであることを確認します。詳細については、AWS Lambda デベロッパーガイドの「[ステップ 2.2: IAM ロール \(実行ロール\) を作成する](#)」を参照してください。

2. テキストエディターを開き、以下の内容で helloWorld.js という名前のファイルを作成します。

```
var zlib = require('zlib');
exports.handler = function(input, context) {
  var payload = Buffer.from(input.awslogs.data, 'base64');
  zlib.gunzip(payload, function(e, result) {
    if (e) {
      context.fail(e);
    } else {
      result = JSON.parse(result.toString());
      console.log("Event Data:", JSON.stringify(result, null, 2));
      context.succeed();
    }
  });
};
```

3. helloWorld.js ファイルを圧縮して helloWorld.zip という名前で保存します。

4. 次のコマンドを使用します。ロールは、最初のステップで設定した Lambda 実行ロールです。

```
aws lambda create-function \
  --function-name helloworld \
  --zip-file fileb://file-path/helloWorld.zip \
  --role lambda-execution-role-arn \
  --handler helloworld.handler \
  --runtime nodejs18.x
```

5. CloudWatch Logs に、関数を実行するためのアクセス権限を付与します。次のコマンドを使用して、プレースホルダーアカウントを独自のアカウントに置き換えます。

```
aws lambda add-permission \
  --function-name "helloworld" \
  --statement-id "helloworld" \
  --principal "logs.amazonaws.com" \
```

```
--action "lambda:InvokeFunction" \  
--source-arn "arn:aws:logs:region:123456789012:log-group:*" \  
--source-account "123456789012"
```

6. 次のコマンドを使用してアカウントレベルのサブスクリプションフィルターポリシーを作成し、プレースホルダーアカウントを独自のアカウントに置き換えます。この例では、文字列 ERROR を含むすべてのログイベントがストリーミングされます。ただし、LogGroupToExclude1 および LogGroupToExclude2 という名前のロググループ内のイベントは除外されます。

```
aws logs put-account-policy \  
  --policy-name "ExamplePolicyLambda" \  
  --policy-type "SUBSCRIPTION_FILTER_POLICY" \  
  --policy-document '  
    {"DestinationArn":"arn:aws:lambda:region:123456789012:function:helloWorld",  
    "FilterPattern": "Test", "Distribution": "Random"}' \  
  --selection-criteria 'LogGroupName NOT IN ["LogGroupToExclude1",  
    "LogGroupToExclude2"]' \  
  --scope "ALL"
```

サブスクリプションフィルターを設定したら、CloudWatch Logs によりフィルタパターンと選択基準に一致するすべての受信ログイベントがストリームに転送されます。

selection-criteria フィールドはオプションですが、サブスクリプションフィルターから無限のログ再帰を引き起こす可能性のあるロググループを除外するための重要なフィールドです。この問題および除外するロググループを判断する方法については、「[ログの再帰防止](#)」を参照してください。現在、NOT IN は selection-criteria でサポートされている唯一の演算子です。

7. (オプション) サンプルのログイベントを使用してテストします。コマンドプロンプトで、次のコマンドを実行します。これにより、サブスクライブしたストリームに単純なログメッセージが送信されます。

Lambda 関数の出力を確認するには、Lambda 関数に移動して、/aws/lambda/helloworld にある出力を参照します。

```
aws logs put-log-events --log-group-name Example1 --log-stream-name logStream1 --  
log-events "[{\"timestamp\":\"CURRENT_TIMESTAMP_MILLIS\", \"message\": \"Simple Lambda  
Test\"}]"
```


Lambda の配列を含むレスポンスが表示されます。Lambda レコードの [Data] (データ) 属性は、base64 でエンコードされており、gzip 形式で圧縮されています。Lambda が受け取る実際のペイロードは、{ "awslogs": { "data": "BASE64ENCODED_GZIP_COMPRESSED_DATA" } } の形式になります。raw データは、コマンドラインから次の UNIX コマンドを使用して調べることができます。

```
echo -n "<BASE64ENCODED_GZIP_COMPRESSED_DATA>" | base64 -d | zcat
```

base64 でデコードおよび解凍されたデータは、次の構造を使用して JSON としてフォーマットされます。

```
{
  "messageType": "DATA_MESSAGE",
  "owner": "123456789012",
  "logGroup": "Example1",
  "logStream": "logStream1",
  "subscriptionFilters": [
    "ExamplePolicyLambda"
  ],
  "logEvents": [
    {
      "id": "31953106606966983378809025079804211143289615424298221568",
      "timestamp": 1432826855000,
      "message": "{\"eventVersion\":\"1.03\",\"userIdentity\":{\"type\":\n\"Root\"}}",
    },
    {
      "id": "31953106606966983378809025079804211143289615424298221569",
      "timestamp": 1432826855000,
      "message": "{\"eventVersion\":\"1.03\",\"userIdentity\":{\"type\":\n\"Root\"}}",
    },
    {
      "id": "31953106606966983378809025079804211143289615424298221570",
      "timestamp": 1432826855000,
      "message": "{\"eventVersion\":\"1.03\",\"userIdentity\":{\"type\":\n\"Root\"}}",
    }
  ],
  "policyLevel": "ACCOUNT_LEVEL_POLICY"
}
```

```
}
```

Note

アカウントレベルのサブスクリプションフィルターは、送信先の Lambda 関数のロググループには適用されません。これは、取り込み料金の増加につながる可能性のある無限のログ再帰を防ぐためです。この問題の詳細については、「[ログの再帰防止](#)」を参照してください。

データ構造の主な要素は次のとおりです。

messageType

データメッセージは、"DATA_MESSAGE" 型を使用します。CloudWatch Logs は、主に送信先に到達可能かどうかを確認するために、「CONTROL_MESSAGE」タイプの Amazon Kinesis Data Streams レコードを出力することがあります。

owner (オーナー)

元のログデータの AWS アカウント ID。

logGroup

発行元ログデータのロググループ名。

logStream

発行元ログデータのログストリーム名。

subscriptionFilters

発行元ログデータと一致したサブスクリプションフィルター名のリスト。

logEvents

ログイベントレコードの配列として表される実際のログデータ。"id" プロパティは、各ログイベントの一意識別子です。

policyLevel

ポリシーが適用されたレベル。「ACCOUNT_LEVEL_POLICY」は、アカウントレベルのサブスクリプションフィルターポリシーの policyLevel です。

例 3: Amazon Data Firehose を使用したサブスクリプションフィルター

この例では、CloudWatch Logs のアカウントレベルのサブスクリプションフィルターポリシーを作成して、定義されたフィルタに一致する受信ログイベントを Amazon Data Firehose 配信ストリームに送信します。CloudWatch Logs から Amazon Data Firehose に送信されたデータは、既に gzip レベル 6 圧縮で圧縮されているため、Firehose 配信ストリーム内で圧縮を使用する必要はありません。その後、Firehose の解凍機能を使用して、ログを自動的に解凍できます。詳細については、「[Writing to Kinesis Data Firehose Using CloudWatch Logs](#)」を参照してください。

Warning

Firehose ストリームを作成する前に、生成するログデータのボリュームを計算します。このボリュームを処理できる Firehose ストリームを作成するように注意してください。ストリームがこのボリュームを処理できない場合、ログストリームはスロットリングされます。Firehose ストリームボリュームの制限事項の詳細については、「[Amazon Data Firehose Data Limits](#)」をご参照ください。

Firehose のサブスクリプションフィルターを作成するには

1. Amazon Simple Storage Service (Amazon S3) バケットを作成します。CloudWatch Logs 専用に作成したバケットを使用することをお勧めします。ただし、既存のバケットを使用する場合は、ステップ 2 に進みます。

次のコマンドを実行します。プレースホルダーリージョンは、使用するリージョンに置き換えます。

```
aws s3api create-bucket --bucket amzn-s3-demo-bucket2 --create-bucket-configuration LocationConstraint=region
```

出力例を次に示します。

```
{
  "Location": "/amzn-s3-demo-bucket2"
}
```

2. Amazon S3 バケットにデータを置くためのアクセス許可を Amazon Data Firehose に付与する IAM ロールを作成します。

詳細については、「Amazon Data Firehose 開発者ガイド」の「[Controlling Access with Amazon Data Firehose](#)」を参照してください。

まず、テキストエディタを使用して、次のようにファイル `~/TrustPolicyForFirehose.json` で信頼ポリシーを作成します。

```
{
  "Statement": {
    "Effect": "Allow",
    "Principal": { "Service": "firehose.amazonaws.com" },
    "Action": "sts:AssumeRole"
  }
}
```

3. `create-role` コマンドを使用し、信頼ポリシーファイルを指定して IAM ロールを作成します。後のステップで必要になるため、返された `Role.Arn` 値を書き留めます。

```
aws iam create-role \
  --role-name FirehoseToS3Role \
  --assume-role-policy-document file://~/TrustPolicyForFirehose.json

{
  "Role": {
    "AssumeRolePolicyDocument": {
      "Statement": {
        "Action": "sts:AssumeRole",
        "Effect": "Allow",
        "Principal": {
          "Service": "firehose.amazonaws.com"
        }
      }
    },
    "RoleId": "EXAMPLE50GAB4HC5F431",
    "CreateDate": "2023-05-29T13:46:29.431Z",
    "RoleName": "FirehoseToS3Role",
    "Path": "/",
    "Arn": "arn:aws:iam::123456789012:role/FirehoseToS3Role"
  }
}
```

4. アクセス許可ポリシーを作成し、Firehose がアカウントで実行できるアクションを定義します。まず、テキストエディタを使用してファイル `~/PermissionsForFirehose.json` で権限ポリシーを作成します。

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:AbortMultipartUpload",
        "s3:GetBucketLocation",
        "s3:GetObject",
        "s3:ListBucket",
        "s3:ListBucketMultipartUploads",
        "s3:PutObject" ],
      "Resource": [
        "arn:aws:s3:::amzn-s3-demo-bucket2",
        "arn:aws:s3:::amzn-s3-demo-bucket2/*" ]
    }
  ]
}
```

5. 次の `put-role-policy` コマンドを使用して、アクセス許可ポリシーをロールに関連付けます。

```
aws iam put-role-policy --role-name FirehoseToS3Role --policy-name Permissions-
Policy-For-Firehose --policy-document file://~/PermissionsForFirehose.json
```

6. 次のように、送信先 Firehose 配信ストリームを作成します。RoleARN と BucketARN のプレースホルダー値を、作成したロールおよびバケット ARN に置き換えます。

```
aws firehose create-delivery-stream \
  --delivery-stream-name 'my-delivery-stream' \
  --s3-destination-configuration \
  '{"RoleARN": "arn:aws:iam::123456789012:role/FirehoseToS3Role", "BucketARN":
  "arn:aws:s3:::amzn-s3-demo-bucket2"}'
```

NFirehose は、配信された Amazon S3 オブジェクトに対して、YYYY/MM/DD/HH UTC 時間形式のプレフィックスを自動的に使用します。時間形式プレフィックスの前に、追加のプレフィックスを指定できます。プレフィックスの最後がフォワードスラッシュ (/) の場合は、Amazon S3 バケット内のフォルダとして表示されます。

7. ストリームがアクティブになるまで数分待ちます。Firehose describe-delivery-stream コマンドを使用して、DeliveryStreamDescription.DeliveryStreamStatus プロパティをチェックできます。さらに、後のステップで必要になるため、DeliveryStreamDescription.DeliveryStreamARN 値を書き留めます。

```
aws firehose describe-delivery-stream --delivery-stream-name "my-delivery-stream"
{
  "DeliveryStreamDescription": {
    "HasMoreDestinations": false,
    "VersionId": "1",
    "CreateTimestamp": 1446075815.822,
    "DeliveryStreamARN": "arn:aws:firehose:us-east-1:123456789012:deliverystream/my-delivery-stream",
    "DeliveryStreamStatus": "ACTIVE",
    "DeliveryStreamName": "my-delivery-stream",
    "Destinations": [
      {
        "DestinationId": "destinationId-0000000000001",
        "S3DestinationDescription": {
          "CompressionFormat": "UNCOMPRESSED",
          "EncryptionConfiguration": {
            "NoEncryptionConfig": "NoEncryption"
          },
          "RoleARN": "delivery-stream-role",
          "BucketARN": "arn:aws:s3:::amzn-s3-demo-bucket2",
          "BufferingHints": {
            "IntervalInSeconds": 300,
            "SizeInMBs": 5
          }
        }
      }
    ]
  }
}
```

8. Firehose 配信ストリームにデータを置くためのアクセス許可を CloudWatch Logs に付与する IAM ロールを作成します。まず、テキストエディタを使用してファイル ~/TrustPolicyForCWL.json で信頼ポリシーを作成します。

このポリシーには、「混乱した代理」のセキュリティ上の問題を防止するための `aws:SourceArn` グローバル条件コンテキストキーが含まれています。詳細については、「[混乱した代理の防止](#)」を参照してください。

```
{
  "Statement": {
    "Effect": "Allow",
    "Principal": { "Service": "logs.amazonaws.com" },
    "Action": "sts:AssumeRole",
    "Condition": {
      "StringLike": {
        "aws:SourceArn": "arn:aws:logs:region:123456789012:*"
      }
    }
  }
}
```

9. create-role コマンドを使用し、信頼ポリシーファイルを指定して IAM ロールを作成します。後のステップで必要になるため、返された Role.Arn 値を書き留めます。

```
aws iam create-role \
--role-name CWLtoKinesisFirehoseRole \
--assume-role-policy-document file://~/TrustPolicyForCWL.json

{
  "Role": {
    "AssumeRolePolicyDocument": {
      "Statement": {
        "Action": "sts:AssumeRole",
        "Effect": "Allow",
        "Principal": {
          "Service": "logs.amazonaws.com"
        },
        "Condition": {
          "StringLike": {
            "aws:SourceArn": "arn:aws:logs:region:123456789012:*"
          }
        }
      }
    },
    "RoleId": "AA0IIAH450GAB4HC5F431",
    "CreateDate": "2015-05-29T13:46:29.431Z",
    "RoleName": "CWLtoKinesisFirehoseRole",
    "Path": "/",
    "Arn": "arn:aws:iam::123456789012:role/CWLtoKinesisFirehoseRole"
  }
}
```

```
}
```

10. 権限ポリシーを作成し、CloudWatch Logs がアカウントで実行できるアクションを定義します。まず、テキストエディタを使用して権限ポリシーファイル (例: ~/PermissionsForCWL.json) を作成します。

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["firehose:PutRecord"],
      "Resource": [
        "arn:aws:firehose:region:account-id:deliverystream/delivery-stream-
name"]
      }
    ]
  }
}
```

11. put-role-policy コマンドを使用して、権限ポリシーをロールに関連付けます。

```
aws iam put-role-policy --role-name CWLtoKinesisFirehoseRole --policy-
name Permissions-Policy-For-CWL --policy-document file://~/PermissionsForCWL.json
```

12. Amazon Data Firehose 配信ストリームがアクティブ状態になり、IAM ロールを作成したら、CloudWatch Logs のアカウントレベルのサブスクリプションフィルタポリシーを作成できます。ポリシーにより、選択されたロググループから Amazon Data Firehose 配信ストリームへのリアルタイムログデータのフローがすぐに開始されます。

```
aws logs put-account-policy \
  --policy-name "ExamplePolicyFirehose" \
  --policy-type "SUBSCRIPTION_FILTER_POLICY" \
  --policy-document '{"RoleArn":"arn:aws:iam::123456789012:role/
CWLtoKinesisFirehoseRole", "DestinationArn":"arn:aws:firehose:us-
east-1:123456789012:deliverystream/delivery-stream-name", "FilterPattern": "Test",
"Distribution": "Random"}' \
  --selection-criteria 'LogGroupName NOT IN ["LogGroupToExclude1",
"LogGroupToExclude2"]' \
  --scope "ALL"
```

13. サブスクリプションフィルターを設定したら、CloudWatch Logs からフィルターパターンに一致するすべての受信ログイベントが Amazon Data Firehose 配信ストリームに転送されます。

selection-criteria フィールドはオプションですが、サブスクリプションフィルターから無限のログ再帰を引き起こす可能性のあるロググループを除外するための重要なフィールドです。この問題および除外するロググループを判断する方法については、「[ログの再帰防止](#)」を参照してください。現在、NOT IN は selection-criteria でサポートされている唯一の演算子です。

Amazon Data Firehose 配信ストリームに設定された時間バッファ間隔に基づいて、Amazon S3 にデータが表示されるようになります。十分な時間が経過すると、Amazon S3 バケットをチェックしてデータを確認できます。

```
aws s3api list-objects --bucket 'amzn-s3-demo-bucket2' --prefix 'firehose/'
{
  "Contents": [
    {
      "LastModified": "2023-10-29T00:01:25.000Z",
      "ETag": "\"a14589f8897f4089d3264d9e2d1f1610\"",
      "StorageClass": "STANDARD",
      "Key": "firehose/2015/10/29/00/my-delivery-stream-2015-10-29-00-01-21-a188030a-62d2-49e6-b7c2-b11f1a7ba250",
      "Owner": {
        "DisplayName": "cloudwatch-logs",
        "ID": "1ec9cf700ef6be062b19584e0b7d84ecc19237f87b5"
      },
      "Size": 593
    },
    {
      "LastModified": "2015-10-29T00:35:41.000Z",
      "ETag": "\"a7035b65872bb2161388ffb63dd1aec5\"",
      "StorageClass": "STANDARD",
      "Key": "firehose/2023/10/29/00/my-delivery-stream-2023-10-29-00-35-40-EXAMPLE-7e66-49bc-9fd4-fc9819cc8ed3",
      "Owner": {
        "DisplayName": "cloudwatch-logs",
        "ID": "EXAMPLE6be062b19584e0b7d84ecc19237f87b6"
      },
      "Size": 5752
    }
  ]
}
```

```
aws s3api get-object --bucket 'amzn-s3-demo-bucket2' --key 'firehose/2023/10/29/00/
my-delivery-stream-2023-10-29-00-01-21-a188030a-62d2-49e6-b7c2-b11f1a7ba250'
testfile.gz

{
  "AcceptRanges": "bytes",
  "ContentType": "application/octet-stream",
  "LastModified": "Thu, 29 Oct 2023 00:07:06 GMT",
  "ContentLength": 593,
  "Metadata": {}
}
```

Simple Storage Service (Amazon S3) オブジェクトのデータは、gzip 形式で圧縮されます。raw データは、コマンドラインから次の UNIX コマンドを使用して調べることができます。

```
zcat testfile.gz
```

クロスアカウント、クロスリージョンのサブスクリプション

別の AWS アカウントの所有者と協力して、Amazon Kinesis または Amazon Data Firehose ストリーム (これはクロスアカウントデータ共有と呼ばれます) などの AWS リソースでロギイベントを受信できます。たとえば、このロギイベントデータは、一元化された Amazon Kinesis Data Streams または Firehose ストリームから読み取り、カスタム処理と分析を実行できます。カスタム処理は、多数のアカウントが協力しデータを分析する場合に特に便利です。

たとえば、ある会社の情報セキュリティグループがリアルタイムで侵入または異常な挙動を検出するためにデータを分析するとします。この場合、会社の全部署のアカウントのフェデレーションされた本稼働ログを中央処理のために収集することによって、これらのアカウントの監査を行うことができます。これらのアカウント全体のイベントデータのリアルタイムストリームをアセンブルして情報セキュリティグループに配信できます。情報セキュリティグループは、Amazon Kinesis Data Streams を使用して既存のセキュリティ分析システムにデータをアタッチできます。

Note

ロググループと送信先は同じ AWS リージョンにある必要があります。ただし、送信先が指す AWS リソースは、別のリージョンに配置することができます。次のセクションの例では、リージョン固有のリソースはすべて米国東部 (バージニア北部) で作成されます。

メンバーアカウントを設定 AWS Organizations して操作している場合は、ログの一元化を使用して、ソースアカウントから中央モニタリングアカウントにログデータを収集できます。

一元化されたロググループを使用する場合は、サブスクリプションフィルターを作成する際にこれらのシステムフィールドのディメンションを使用できます。

- `@aws.account` - このディメンションは、ログイベントの発生元の AWS アカウント ID を表します。
- `@aws.region` - このディメンションは、ログイベントが生成された AWS リージョンを表します。

これらのディメンションはログデータのソースを特定するのに役立ちます。これにより、一元化されたログから派生したメトリクスのフィルタリングと分析をさらに細かく行えるようになります。

トピック

- [Amazon Kinesis Data Streams を使用したクロスアカウントクロスリージョンログデータ共有](#)
- [Firehose を使用したクロスアカウント、クロスリージョンのログデータ共有](#)
- [Amazon Kinesis Data Streams を使用したクロスアカウントクロスリージョンアカウントレベルのサブスクリプション](#)
- [Firehose を使用したクロスアカウント、クロスリージョン、アカウントレベルのサブスクリプション](#)

Amazon Kinesis Data Streams を使用したクロスアカウントクロスリージョンログデータ共有

クロスアカウントサブスクリプションを作成するときに、単一のアカウントまたは組織を送信者として指定できます。組織を指定した場合、この手順により組織内のすべてのアカウントがレシーバーアカウントにログを送信できるようになります。

複数のアカウントでログデータを共有するには、ログデータの送信者と受信者を確立する必要があります。

- [Log data sender] (ログデータの送信者) — 受信者から送信先情報を取得し、そのログイベントを特定の送信先に送信する準備が完了していることを CloudWatch Logs に通知します。このセクションの残りの手順では、ログデータ送信者は架空の AWS アカウント番号 111111111111 で表示されます。

1つの組織で複数のアカウントを1つの受信者アカウントにログを送信する場合は、組織内のすべてのアカウントに対して受信者アカウントにログを送信するアクセス許可を付与するポリシーを作成することができます。引き続き、送信者アカウントごとに個別のサブスクリプションフィルターを設定する必要があります。

- ログデータ受信者 - Amazon Kinesis Data Streams ストリームをカプセル化する送信先を設定し、受信者がログデータを受信したいことを CloudWatch Logs に知らせます。この後、受信者は自分の送信先に関する情報を送信者と共有します。このセクションの残りの手順では、ログデータ受信者は架空の AWS アカウント番号 999999999999 で表示されます。

クロスアカウントのユーザーからのログイベントの受け取りを開始するには、ログデータの受取人がまず CloudWatch Logs 送信先を作成する必要があります。各送信先は以下のキー要素で構成されています。

送信先名

作成する送信先の名前。

ターゲット ARN

サブスクリプションフィードの送信先として使用するリソースの Amazon AWS リソースネーム (ARN)。

ロールの ARN

選択したストリームにデータを配置するために必要なアクセス許可を CloudWatch Logs に付与する AWS Identity and Access Management (IAM) ロール。

アクセスポリシー

送信先に書き込むことが許可されている一連のユーザーを管理する IAM ポリシードキュメント (IAM ポリシー構文を使用して記述された JSON 形式のドキュメント)。

Note

ロググループと送信先は同じ AWS リージョンにある必要があります。ただし、送信先が指す AWS リソースは、別のリージョンに配置することができます。次のセクションの例では、リージョン固有のリソースはすべて米国東部 (バージニア北部) で作成されます。

トピック

- [新しいクロスアカウントサブスクリプションの設定](#)
- [既存のクロスアカウントサブスクリプションの更新](#)

新しいクロスアカウントサブスクリプションの設定

次のセクションの手順に従って、新しいクロスアカウントログサブスクリプションを設定します。

トピック

- [ステップ 1: 送信先を作成する](#)
- [ステップ 2: IAM ロールを作成する \(組織を使用している場合のみ\)](#)
- [ステップ 3: クロスアカウント宛先の IAM アクセス許可を追加/検証する](#)
- [ステップ 4: サブスクリプションフィルターを作成する](#)
- [ログイベントの送信を検証](#)
- [ランタイムの送信先のメンバーシップを変更](#)

ステップ 1: 送信先を作成する

Important

この手順のステップは、ログデータの受取人アカウントで行われます。

この例では、ログデータ受信者アカウントの AWS アカウント ID は 999999999999 で、ログデータ送信者 AWS アカウント ID は 111111111111 です。

この例では、RecipientStream という Amazon Kinesis Data Streams ストリームと、CloudWatch Logs によるデータの書き込みを可能にするロールを使用して送信先を作成します。

送信先が作成されると、CloudWatch Logs は受信者アカウントに代わってテストメッセージを宛先に送信します。サブスクリプションフィルターが後でアクティブになると、CloudWatch Logs はソースアカウントに代わってログイベントを宛先に送信します。

送信先を作成するには

1. 受信者アカウントで、Amazon Kinesis Data Streams で送信先ストリームを作成します。コマンドプロンプトで、次のように入力します。

```
aws kinesis create-stream --stream-name "RecipientStream" --shard-count 1
```

2. ストリームがアクティブになるまで待ちます。aws kinesis describe-stream コマンドを使用して、StreamDescription.StreamStatus プロパティをチェックできます。さらに、StreamDescription.StreamARN の値は、後で CloudWatch Logs に渡すので書き留めておいてください。

```
aws kinesis describe-stream --stream-name "RecipientStream"
{
  "StreamDescription": {
    "StreamStatus": "ACTIVE",
    "StreamName": "RecipientStream",
    "StreamARN": "arn:aws:kinesis:us-east-1:999999999999:stream/RecipientStream",
    "Shards": [
      {
        "ShardId": "shardId-000000000000",
        "HashKeyRange": {
          "EndingHashKey": "34028236692093846346337460743176EXAMPLE",
          "StartingHashKey": "0"
        },
        "SequenceNumberRange": {
          "StartingSequenceNumber":
            "4955113521868881845667950383198145878459135270218EXAMPLE"
        }
      }
    ]
  }
}
```

ストリームがアクティブ状態で表示されるまでに 1~2 分かかる場合があります。

3. ストリームにデータを置くアクセス権限を CloudWatch Logs に付与する IAM ロールを作成します。まず、ファイル ~/TrustPolicyForCWL.json で信頼ポリシーを作成する必要があります。このポリシーの作成にはテキストエディタを使用します。IAM コンソールは使用しないでください。

このポリシーには、「混乱した代理」のセキュリティ上の問題を防止するための sourceAccountId が指定された aws:SourceArn グローバル条件コンテキストキーが含まれています。最初の呼び出しでソースアカウント ID が不明な場合は、送信元 ARN フィールドに送信先 ARN を指定することをお勧めします。後続の呼び出しでは、送信元 ARN を、最初の呼

び出しで取得した実際の送信元 ARN に設定する必要があります。詳細については、「[混乱した代理の防止](#)」を参照してください。

```
{
  "Statement": {
    "Effect": "Allow",
    "Principal": {
      "Service": "logs.amazonaws.com"
    },
    "Condition": {
      "StringLike": {
        "aws:SourceArn": [
          "arn:aws:logs:region:sourceAccountId:",
          "arn:aws:logs:region:recipientAccountId:"
        ]
      }
    },
    "Action": "sts:AssumeRole"
  }
}
```

4. `aws iam create-role` コマンドを使用して、信頼ポリシーファイルを指定する IAM ロールを作成します。返された `Role.Arn` 値を書き留めておきます。これも後で CloudWatch Logs に渡されるからです。

```
aws iam create-role \
--role-name CWLtoKinesisRole \
--assume-role-policy-document file://~/TrustPolicyForCWL.json

{
  "Role": {
    "AssumeRolePolicyDocument": {
      "Statement": {
        "Action": "sts:AssumeRole",
        "Effect": "Allow",
        "Condition": {
          "StringLike": {
            "aws:SourceArn": [
              "arn:aws:logs:region:sourceAccountId:",
              "arn:aws:logs:region:recipientAccountId:"
            ]
          }
        }
      }
    }
  },
}
```

```

        "Principal": {
            "Service": "logs.amazonaws.com"
        }
    },
    "RoleId": "AA0IIAH450GAB4HC5F431",
    "CreateDate": "2015-05-29T13:46:29.431Z",
    "RoleName": "CWLtoKinesisRole",
    "Path": "/",
    "Arn": "arn:aws:iam::999999999999:role/CWLtoKinesisRole"
}

```

5. CloudWatch Logs がアカウントで実行できるアクションを定義するアクセス許可ポリシーを作成します。まず、テキストエディタを使用してファイル `~/PermissionsForCWL.json` でアクセス許可ポリシーを作成します。

```

{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "kinesis:PutRecord",
      "Resource": "arn:aws:kinesis:region:999999999999:stream/RecipientStream"
    }
  ]
}

```

6. `aws iam put-role-policy` コマンドを使用して、アクセス許可ポリシーをロールに関連付けます。

```

aws iam put-role-policy \
  --role-name CWLtoKinesisRole \
  --policy-name Permissions-Policy-For-CWL \
  --policy-document file:///~/PermissionsForCWL.json

```

7. ストリームがアクティブ状態になり、IAM ロールが作成されたら、CloudWatch Logs の送信先を作成できます。
 - a. このステップでは、アクセスポリシーと送信先は関連付けられません。送信先の作成を完了するには 2 つのステップを行う必要がありますが、このステップはその最初のステップです。ペイロードで返された `DestinationArn` を書き留めておいてください。


```
aws logs put-destination \  
  --destination-name "testDestination" \  
  --target-arn "arn:aws:kinesis:region:999999999999:stream/RecipientStream" \  
  --role-arn "arn:aws:iam::999999999999:role/CWLtoKinesisRole"  
  
{  
  "DestinationName" : "testDestination",  
  "RoleArn" : "arn:aws:iam::999999999999:role/CWLtoKinesisRole",  
  "DestinationArn" : "arn:aws:logs:us-  
east-1:999999999999:destination:testDestination",  
  "TargetArn" : "arn:aws:kinesis:us-east-1:999999999999:stream/RecipientStream"  
}
```

- b. ステップ 7a が完了したら、ログデータの受取人アカウントで、アクセスポリシーを送信先に関連付けます。このポリシーでは、送信者アカウントが送信先にアクセスするためのアクセス許可が付与され、logs:PutSubscriptionFilter アクションが指定されている必要があります。

このポリシーは、ログを送信する AWS アカウントにアクセス許可を付与します。ポリシーの中で対象のアカウントを 1 つだけ指定してもよいですが、送信者アカウントが組織のメンバーのものである場合は組織 ID を指定することもできます。このように、ポリシーを 1 つ作成するだけで、1 つの組織内の複数のアカウントが送信先アカウントにログを送信できるように設定できます。

テキストエディタを使用して ~/AccessPolicy.json という名前のファイルを作成し、以下のいずれかのポリシーステートメントを使用します。

この最初の例のポリシーでは、組織内で o-1234567890 という ID を持つすべてのアカウントが、受信者アカウントにログを送信することを許可します。

JSON

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Sid": "",  
      "Effect": "Allow",  
      "Principal": "*",  
      "Action": "logs:PutSubscriptionFilter",
```

```

        "Resource": "arn:aws:logs:us-
east-1:999999999999:destination:testDestination",
        "Condition": {
            "StringEquals": {
                "aws:PrincipalOrgID": [
                    "o-1234567890"
                ]
            }
        }
    }
]
}

```

次の例では、ログデータの送信者アカウント (111111111111) がログデータの受信者アカウントにログを送信できるようにします。

JSON

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "AWS": "111111111111"
            },
            "Action": "logs:PutSubscriptionFilter",
            "Resource": "arn:aws:logs:us-
east-1:999999999999:destination:testDestination"
        }
    ]
}

```

- c. 前のステップで作成したポリシーを送信先に添付します。

```

aws logs put-destination-policy \
  --destination-name "testDestination" \
  --access-policy file://~/AccessPolicy.json

```

このアクセスポリシーにより、ID 111111111111 の AWS アカウントのユーザーは、ARN `arn:aws:logs:region:999999999999:destination:testDestination` の送信先に対して `PutSubscriptionFilter` を呼び出すことができます。他のユーザーがこの送信先に対して `PutSubscriptionFilter` を呼び出そうとしても、それは却下されます。

アクセスポリシーに照らし合わせてユーザーの権限を検証するには、「IAM ユーザーガイド」の「[Using Policy Validator](#)」(Policy Validator の使用) を参照してください。

完了したら、クロスアカウントアクセス許可 AWS Organizations に を使用している場合は、「」の順に従います [ステップ 2: IAM ロールを作成する \(組織を使用している場合のみ\)](#)。組織を使用せずに他のアカウントに直接アクセス許可を付与する場合は、そのステップを飛ばして「[ステップ 4: サブスクリプションフィルターを作成する](#)」に進みます。

ステップ 2: IAM ロールを作成する (組織を使用している場合のみ)

前のセクションで アカウント 111111111111 に直接アクセス許可を付与するのではなく、アカウント 111111111111 が属する組織にアクセス許可を付与するアクセスポリシーを使用することにより送信先を作成した場合は、このセクションのステップを実行します。それ以外の場合は、「[ステップ 4: サブスクリプションフィルターを作成する](#)」に進みます。

このセクションのステップにより、CloudWatch が想定する IAM ロールを作成し、送信者アカウントが受信者の送信先に対してサブスクリプションフィルターを作成する権限を持っているかどうかを検証できます。

このセクションの手順は、送信者アカウントで実行してください。ロールは送信者アカウントに存在する必要があり、このロールの ARN はサブスクリプションフィルターで指定します。この例では、送信者アカウントは 111111111111 です。

を使用してクロスアカウントログサブスクリプションに必要な IAM ロールを作成するには AWS Organizations

1. 以下の信頼ポリシーを作成し、`/TrustPolicyForCWLSubscriptionFilter.json` という名前のテキストファイルに保存します。このポリシーの作成にはテキストエディタを使用します。IAM コンソールは使用しないでください。

```
{
  "Statement": {
    "Effect": "Allow",
    "Principal": { "Service": "logs.amazonaws.com" },
```

```
"Action": "sts:AssumeRole"
}
}
```

2. このポリシーを使用する IAM ロールを作成します。下記のコマンドが返す Arn の値は後ほど必要になるため、書き留めておきます。この例では、作成するロールに CWLtoSubscriptionFilterRole という名前を付けます。

```
aws iam create-role \
  --role-name CWLtoSubscriptionFilterRole \
  --assume-role-policy-document file://~/
TrustPolicyForCWLSubscriptionFilter.json
```

3. アクセス許可ポリシーを作成して、CloudWatch Logs がアカウントで実行できるアクションを定義します。

- a. まず、テキストエディタを使用して、~/PermissionsForCWLSubscriptionFilter.json という名前のファイルに以下のようなアクセス許可ポリシーを作成します。

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "logs:PutLogEvents",
      "Resource": "arn:aws:logs:region:111111111111:log-
group:LogGroupOnWhichSubscriptionFilterIsCreated:*"
    }
  ]
}
```

- b. 次のコマンドを入力して、先ほど作成したアクセス許可ポリシーを、ステップ 2 で作成したロールに関連付けます。

```
aws iam put-role-policy
  --role-name CWLtoSubscriptionFilterRole
  --policy-name Permissions-Policy-For-CWL-Subscription-filter
  --policy-document file://~/PermissionsForCWLSubscriptionFilter.json
```

終了したら、「[ステップ 4: サブスクリプションフィルターを作成する](#)」に進みます。

ステップ 3: クロスアカウント宛先の IAM アクセス許可を追加/検証する

AWS クロスアカウントポリシーの評価ロジックに従って、クロスアカウントリソース (サブスクリプションフィルターの送信先として使用される Kinesis または Firehose ストリームなど) にアクセスするには、クロスアカウントの送信先リソースへの明示的なアクセスを提供する ID ベースのポリシーを送信アカウントに含める必要があります。ポリシーの評価論理の詳細については、「[クロスアカウントポリシーの評価論理](#)」を参照してください。

ID ベースのポリシーは、サブスクリプションフィルターの作成に使用している IAM ロールまたは IAM ユーザーにアタッチできます。送信アカウントにこのポリシーが存在する必要があります。管理者ロールを使用してサブスクリプションフィルターを作成している場合は、このステップをスキップして [ステップ 4: サブスクリプションフィルターを作成する](#) に進んでください。

クロスアカウントに必要な IAM アクセス許可を追加または検証するには

1. 次のコマンドを入力して、AWS ログコマンドの実行に使用されている IAM ロールまたは IAM ユーザーを確認します。

```
aws sts get-caller-identity
```

このコマンドにより、以下のような出力が返されます。

```
{
  "UserId": "User ID",
  "Account": "sending account id",
  "Arn": "arn:aws:sending account id:role/user:RoleName/UserName"
}
```

RoleName または *UserName* で表される値を書き留めておいてください。

2. 送信アカウント AWS マネジメントコンソール でサインインし、ステップ 1 で入力したコマンドの出力で返された IAM ロールまたは IAM ユーザーを使用して、アタッチされたポリシーを検索します。
3. このロールまたはユーザーにアタッチされたポリシーが、クロスアカウントの宛先リソースで `logs:PutSubscriptionFilter` を呼び出すための明示的なアクセス許可が付与されていることを確認します。

次のポリシーは、1 つの AWS アカウント、アカウント でのみ、任意の送信先リソースにサブスクリプションフィルターを作成するアクセス許可を提供します999999999999。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowSubscriptionFiltersOnAccountResources",
      "Effect": "Allow",
      "Action": "logs:PutSubscriptionFilter",
      "Resource": [
        "arn:aws:logs:*:*:log-group:*",
        "arn:aws:logs:*:123456789012:destination:*"
      ]
    }
  ]
}
```

次のポリシーは、単一の AWS アカウント、アカウント `sampleDestination` という名前の特定の送信先リソースに対してのみサブスクリプションフィルターを作成するアクセス許可を提供します123456789012。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowSubscriptionFiltersonAccountResource",
      "Effect": "Allow",
      "Action": "logs:PutSubscriptionFilter",
      "Resource": [
        "arn:aws:logs:*:*:log-group:*",
        "arn:aws:logs:*:123456789012:destination:sampleDestination"
      ]
    }
  ]
}
```

ステップ 4: サブスクリプションフィルターを作成する

送信先を作成したら、ログデータの受信者アカウントは、送信先の ARN (arn:aws:logs:us-east-1:999999999999:destination:testDestination) を他の AWS アカウントと共有できるようになります。これにより、これらのアカウントは同じ送信先にログイベントを送信できます。この後、これらの他の送信アカウントのユーザーは、この送信先に対するサブスクリプションフィルタをそれぞれのロググループに作成します。サブスクリプションフィルタは、特定のロググループから特定の送信先へのリアルタイムログデータの送信をすぐに開始します。

Note

サブスクリプションフィルターのためのアクセス許可を組織全体に付与する際は、[ステップ 2: IAM ロールを作成する \(組織を使用している場合のみ\)](#) で作成した IAM ロールの ARN を使用する必要があります。

次の例では、サブスクリプションフィルターが送信アカウントに作成されます。フィルターはイベントを含む AWS CloudTrail ロググループに関連付けられ、「ルート AWS」認証情報によって行われたすべてのログアクティビティが、以前に作成した送信先に配信されます。この送信先は、「RecipientStream」というストリームをカプセル化します。

以降のセクションの残りのステップは、「AWS CloudTrail ユーザーガイド」の「[CloudWatch Logs への CloudTrail イベントの送信](#)」の指示に従って、CloudTrail イベントを含むロググループが作成済みであることを前提としています。そのステップでは、ロググループに CloudTrail/logs という名前を付けることになっています。

次のコマンドを入力するときは、必ず IAM ユーザーとしてサインインしているか、[ステップ 3: クロスアカウント宛先の IAM アクセス許可を追加/検証する](#) にポリシーを追加した IAM ロールを使用してサインインしていることを確認してください。

```
aws logs put-subscription-filter \  
  --log-group-name "CloudTrail/logs" \  
  --filter-name "RecipientStream" \  
  --filter-pattern "{$.userIdentity.type = Root}" \  
  --destination-arn "arn:aws:logs:region:999999999999:destination:testDestination"
```

ロググループと送信先は同じ AWS リージョンにある必要があります。ただし、送信先は、別のリージョンにある Amazon Kinesis Data Streams ストリームなどの AWS リソースを指すことができます。

ログイベントの送信を検証

サブスクリプションフィルタを作成したら、CloudWatch Logs が、フィルタパターンと一致するすべての受信ログイベントを、送信先ストリーム内でカプセル化されている「RecipientStream」という名前のストリームに転送します。送信先所有者は、aws kinesis get-shard-iterator コマンドを使用して Amazon Kinesis Data Streams シャードを取得し、aws kinesis get-records コマンドを使用して Amazon Kinesis Data Streams レコードを取得することで、これが起こっていることを確認できます。

```
aws kinesis get-shard-iterator \  
    --stream-name RecipientStream \  
    --shard-id shardId-000000000000 \  
    --shard-iterator-type TRIM_HORIZON  
  
{  
    "ShardIterator":  
    "AAAAAAAAAAFGU/  
kLvNggvndHq2UIF0w5PZc6F01s3e3afsSscRM70JSbjIefg2ub07nk1y6CDxYR1UoGHJNP4m4NFUetzfL+wev  
+e2P4djJg4L9wmXKvQYoE+rMUiFq+p4Cn3Igvq0b5dRA0yybNdRcdzvnC35KQANoHzzahKdRGb9v4scv+3vaq+f  
+0IK8zM5My8ID+g6rMo7UKWeI4+IWiKEXAMPLE"  
}  
  
aws kinesis get-records \  
    --limit 10 \  
    --shard-iterator  
    "AAAAAAAAAAFGU/  
kLvNggvndHq2UIF0w5PZc6F01s3e3afsSscRM70JSbjIefg2ub07nk1y6CDxYR1UoGHJNP4m4NFUetzfL+wev  
+e2P4djJg4L9wmXKvQYoE+rMUiFq+p4Cn3Igvq0b5dRA0yybNdRcdzvnC35KQANoHzzahKdRGb9v4scv+3vaq+f  
+0IK8zM5My8ID+g6rMo7UKWeI4+IWiKEXAMPLE"
```

Note

Amazon Kinesis Data Streams がデータを返す前に、get-records コマンドを数回再実行する必要がある場合があります。

Amazon Kinesis Data Streams レコードの配列を含むレスポンスが表示されます。Amazon Kinesis Data Streams レコードのデータ属性は gzip 形式で圧縮され、base64 でエンコードされます。raw データは、コマンドラインから次の UNIX コマンドを使用して調べることができます。


```
echo -n "<Content of Data>" | base64 -d | zcat
```

base64 でデコードおよび解凍されたデータは、次の構造を使用して JSON としてフォーマットされます。

```
{
  "owner": "111111111111",
  "logGroup": "CloudTrail/logs",
  "logStream": "111111111111_CloudTrail/logs_us-east-1",
  "subscriptionFilters": [
    "RecipientStream"
  ],
  "messageType": "DATA_MESSAGE",
  "logEvents": [
    {
      "id": "3195310660696698337880902507980421114328961542429EXAMPLE",
      "timestamp": 1432826855000,
      "message": "{\"eventVersion\":\"1.03\",\"userIdentity\":{\"type\":\"Root
    }",
    },
    {
      "id": "3195310660696698337880902507980421114328961542429EXAMPLE",
      "timestamp": 1432826855000,
      "message": "{\"eventVersion\":\"1.03\",\"userIdentity\":{\"type\":\"Root
    }",
    },
    {
      "id": "3195310660696698337880902507980421114328961542429EXAMPLE",
      "timestamp": 1432826855000,
      "message": "{\"eventVersion\":\"1.03\",\"userIdentity\":{\"type\":\"Root
    }",
  ]
}
```

このデータ構造のキー要素は以下のとおりです。

owner (オーナー)

元のログデータの AWS アカウント ID。

logGroup

発行元ログデータのロググループ名。

logStream

発行元ログデータのログストリーム名。

subscriptionFilters

発行元ログデータと一致したサブスクリプションフィルタ名のリスト。

messageType

データメッセージは、「DATA_MESSAGE」型を使用します。CloudWatch Logs は、主に送信先に到達可能かどうかを確認するために、「CONTROL_MESSAGE」タイプの Amazon Kinesis Data Streams レコードを出力することがあります。

logEvents

ログイベントレコードの配列として表される実際のログデータ。ID プロパティは、各ログイベントの一意の識別子です。

ランタイムの送信先のメンバーシップを変更

所有する送信先のユーザーのメンバーシップを追加または削除する必要がある場合があります。新しいアクセスポリシーを使用して、送信先で `put-destination-policy` コマンドを使用できます。次の例では、先ほど追加したアカウント 111111111111 がログデータの送信を停止し、アカウント 222222222222 が有効になります。

1. 現在 `testDestination` という送信先に関連付けられているポリシーをフェッチし、`AccessPolicy` を書き留めておきます。

```
aws logs describe-destinations \
  --destination-name-prefix "testDestination"

{
  "Destinations": [
    {
      "DestinationName": "testDestination",
      "RoleArn": "arn:aws:iam::999999999999:role/CWLtoKinesisRole",
      "DestinationArn":
        "arn:aws:logs:region:999999999999:destination:testDestination",
```

```

    "TargetArn": "arn:aws:kinesis:region:999999999999:stream/RecipientStream",
    "AccessPolicy": "{ \"Version\": \"2012-10-17\", \"Statement\":
[ { \"Sid\": \"\", \"Effect\": \"Allow\", \"Principal\": { \"AWS\":
  \"111111111111\" }, \"Action\": \"logs:PutSubscriptionFilter\", \"Resource\":
  \"arn:aws:logs:region:999999999999:destination:testDestination\" } ] }"
  }
]
}

```

2. アカウント 111111111111 が停止したとアカウント 222222222222 が有効になったことを反映させるためにポリシーを更新します。このポリシーを ~/NewAccessPolicy.json ファイルに配置します。

JSON

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "AWS": "222222222222"
      },
      "Action": "logs:PutSubscriptionFilter",
      "Resource": "arn:aws:logs:us-east-1:999999999999:destination:testDestination"
    }
  ]
}

```

3. PutDestinationPolicy を呼び出して、NewAccessPolicy.json ファイルで定義されているポリシーを送信先に関連付けます。

```

aws logs put-destination-policy \
--destination-name "testDestination" \
--access-policy file://~/NewAccessPolicy.json

```

これにより、最終的には、アカウント ID 111111111111 からのログイベントが無効になります。アカウント ID 222222222222 の所有者がサブスクリプションフィルターを作成すると、すぐに 222222222222 からログイベントが送信先に送信されるようになります。

既存のクロスアカウントサブスクリプションの更新

送信先アカウントが特定の送信者アカウントにのみアクセス許可を付与しているクロスアカウントのログサブスクリプションがあり、このサブスクリプションを更新して送信先アカウントが組織内のすべてのアカウントにアクセスできるようにする場合は、このセクションのステップを実施します。

トピック

- [ステップ 1: サブスクリプションフィルターを更新する](#)
- [ステップ 2: 既存の送信先アクセスポリシーを更新する](#)

ステップ 1: サブスクリプションフィルターを更新する

Note

この手順は、[AWS サービスからのログ記録を有効にする](#)に記載されているサービスによって作成されたログのクロスアカウントのサブスクリプションにのみ必要です。これらのロググループのいずれかで作成されたログを操作していない場合は、[ステップ 2: 既存の送信先アクセスポリシーを更新する](#)にスキップできます。

場合によっては、送信先アカウントにログを送信する、すべての送信者アカウントのサブスクリプションフィルターを更新する必要があります。この更新により IAM ロールが追加されます。IAM ロールは CloudWatch が引き受けることができ、送信者アカウントが受信者アカウントにログを送信する権限を持っていることを検証できます。

すべての送信者アカウントについてクロスアカウントサブスクリプションのアクセス許可に組織 ID を使用するよう更新するには、このセクションのステップを実施します。

このセクションの例では、2 つのアカウント 111111111111 と 222222222222 は、アカウント 999999999999 にログを送信するために作成されたサブスクリプションフィルターをすでに持っています。既存のサブスクリプションフィルター値は次のとおりです。

```
## Existing Subscription Filter parameter values
\ --log-group-name "my-log-group-name"
\ --filter-name "RecipientStream"
\ --filter-pattern "${$.userIdentity.type = Root}"
\ --destination-arn "arn:aws:logs:region:999999999999:destination:testDestination"
```

現在のサブスクリプションフィルターパラメータ値を見つける必要がある場合は、次のコマンドを入力します。

```
aws logs describe-subscription-filters
\ --log-group-name "my-log-group-name"
```

サブスクリプションフィルターを更新して、クロスアカウントログの権限で組織 ID の使用をスタートする方法

1. 以下の信頼ポリシーを作成し、~/TrustPolicyForCWL.json という名前のテキストファイルに保存します。このポリシーの作成にはテキストエディタを使用します。IAM コンソールは使用しないでください。

```
{
  "Statement": {
    "Effect": "Allow",
    "Principal": { "Service": "logs.amazonaws.com" },
    "Action": "sts:AssumeRole"
  }
}
```

2. このポリシーを使用する IAM ロールを作成します。下記のコマンドが返す Arn 値の Arn の値は後ほど必要になるため、書き留めておきます。この例では、作成するロールに CWLtoSubscriptionFilterRole という名前を付けます。

```
aws iam create-role
\ --role-name CWLtoSubscriptionFilterRole
\ --assume-role-policy-document file:///~/TrustPolicyForCWL.json
```

3. アクセス許可ポリシーを作成して、CloudWatch Logs がアカウントで実行できるアクションを定義します。
 - a. まず、テキストエディタを使用して、/PermissionsForCWLSubscriptionFilter.json という名前のファイルに以下のようなアクセス許可ポリシーを作成します。

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "logs:PutLogEvents",
```

```

        "Resource": "arn:aws:logs:region:111111111111:log-
        group:LogGroupOnWhichSubscriptionFilterIsCreated:*"
      }
    ]
  }
}

```

- b. 次のコマンドを入力して、先ほど作成したアクセス許可ポリシーを、ステップ 2 で作成したロールに関連付けます。

```

aws iam put-role-policy
  --role-name CWLtoSubscriptionFilterRole
  --policy-name Permissions-Policy-For-CWL-Subscription-filter
  --policy-document file://~/PermissionsForCWLSubscriptionFilter.json

```

4. 次のコマンドを入力して、サブスクリプションフィルターを更新します。

```

aws logs put-subscription-filter
  \ --log-group-name "my-log-group-name"
  \ --filter-name "RecipientStream"
  \ --filter-pattern "${$.userIdentity.type = Root}"
  \ --destination-arn
  "arn:aws:logs:region:999999999999:destination:testDestination"
  \ --role-arn "arn:aws:iam::111111111111:role/CWLtoSubscriptionFilterRole"

```

ステップ 2: 既存の送信先アクセスポリシーを更新する

すべての送信者アカウントのサブスクリプションフィルターを更新した後、受信者アカウントの送信先アクセスポリシーを更新できます。

以下の例では、受信者アカウントは 999999999999、送信先は testDestination となっています。

この更新により、ID o-1234567890 を持つ組織に属するすべてのアカウントが、受信者アカウントにログを送信できるようになりました。サブスクリプションフィルターが作成されたアカウントのみが、実際に受信者アカウントにログを送信します。

受信者アカウントの送信先アクセスポリシーを更新して、権限の組織 ID の使用をスタートする方法

1. 受信者アカウントで、テキストエディタを使用して、以下の内容の ~/AccessPolicy.json ファイルを作成します。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "logs:PutSubscriptionFilter",
      "Resource": "arn:aws:logs:us-east-1:999999999999:destination:testDestination",
      "Condition": {
        "StringEquals": {
          "aws:PrincipalOrgID": [
            "o-1234567890"
          ]
        }
      }
    }
  ]
}
```

2. 次のコマンドを入力して、先ほど作成したポリシーを既存の送信先にアタッチします。特定の AWS アカウント ID を一覧表示するアクセスポリシーではなく、組織 ID を持つアクセスポリシーを使用するように送信先を更新するには、forceパラメータを含めIDs。

 Warning

にリストされている AWS サービスによって送信されたログを使用している場合は[AWS サービスからのログ記録を有効にする](#)、このステップを実行する前に、「」で説明されているように、まずすべての送信者アカウントのサブスクリプションフィルターを更新しておく必要があります[ステップ 1: サブスクリプションフィルターを更新する](#)。

```
aws logs put-destination-policy
\ --destination-name "testDestination"
\ --access-policy file://~/AccessPolicy.json
\ --force
```

Firehose を使用したクロスアカウント、クロスリージョンのログデータ共有

複数のアカウントでログデータを共有するには、ログデータの送信者と受信者を確立する必要があります。

- [Log data sender (ログデータの送信者)] — 受取人から送信先情報を取得し、そのログイベントを特定の送信先に送信する準備が完了していることを CloudWatch Logs に通知します。このセクションの残りの手順では、ログデータ送信者は架空の AWS アカウント番号 111111111111 で表示されます。
- ログデータ受信者 — Amazon Kinesis Data Streams ストリームをカプセル化する送信先を設定し、受信者がログデータを受信したいことを CloudWatch Logs に知らせます。この後、受信者は自分の送信先に関する情報を送信者と共有します。このセクションの残りの手順では、ログデータ受信者は架空の AWS アカウント番号 222222222222 で表示されます。

このセクションの例では、Amazon S3 ストレージで Firehose 配信ストリームを使用しています。異なる設定で Firehose 配信ストリームを設定することもできます。詳細については、「[Creating a Firehose Delivery Stream](#)」を参照してください。

Note

ロググループと送信先は同じ AWS リージョンにある必要があります。ただし、送信先が指す AWS リソースは、別のリージョンに配置することができます。

Note

[同じアカウント] と [クロスリージョン] 配信ストリームに対して Firehose サブスクリプションフィルターがサポートされています。

トピック

- [ステップ 1: Firehose 配信ストリームを作成する](#)
- [ステップ 2: 送信先を作成する](#)
- [ステップ 3: クロスアカウント宛先の IAM アクセス許可を追加/検証する](#)
- [ステップ 4: サブスクリプションフィルターを作成する](#)

- [ログイベントの送信の検証](#)
- [実行時の送信先のメンバーシップの変更](#)

ステップ 1: Firehose 配信ストリームを作成する

Important

以下の手順を実行する前に、Firehose が Amazon S3 バケットにアクセスできるように、アクセスポリシーを使用する必要があります。詳細については、「[Amazon Data Firehose 開発者ガイド](#)」の「Controlling Access」を参照してください。

このセクションのすべての手順 (ステップ 1) は、ログデータの受取人アカウントで行われます。

次のサンプルコマンドでは、米国東部 (バージニア北部) が使用されています。このリージョンを、デプロイに適したリージョンに置き換えます。

送信先として使用する Firehose 配信ストリームを作成するには

1. Amazon S3 バケットの作成

```
aws s3api create-bucket --bucket amzn-s3-demo-bucket --create-bucket-configuration
LocationConstraint=us-east-1
```

2. バケットにデータを配置するためのアクセス許可を Firehose に付与する IAM ロールを作成します。

- まず、テキストエディタを使用して、ファイル `~/TrustPolicyForFirehose.json` で信頼ポリシーを作成します。

```
{ "Statement": { "Effect": "Allow", "Principal": { "Service":
"firehose.amazonaws.com" }, "Action": "sts:AssumeRole", "Condition":
{ "StringEquals": { "sts:ExternalId":"222222222222" } } } }
```

- 作成したばかりの信頼ポリシーファイルを指定して、IAM ロールを作成します。

```
aws iam create-role \
--role-name FirehoseToS3Role \
--assume-role-policy-document file:///~/TrustPolicyForFirehose.json
```

- c. このコマンドの出力は、次のようになります。ロール名とロール ARN を書き留めます。

```
{
  "Role": {
    "Path": "/",
    "RoleName": "FirehoseToS3Role",
    "RoleId": "AROAR3BXASEKW7K635M53",
    "Arn": "arn:aws:iam::222222222222:role/FirehoseToS3Role",
    "CreateDate": "2021-02-02T07:53:10+00:00",
    "AssumeRolePolicyDocument": {
      "Statement": {
        "Effect": "Allow",
        "Principal": {
          "Service": "firehose.amazonaws.com"
        },
        "Action": "sts:AssumeRole",
        "Condition": {
          "StringEquals": {
            "sts:ExternalId": "222222222222"
          }
        }
      }
    }
  }
}
```

3. アクセス許可ポリシーを作成し、Firehose がアカウントで実行できるアクションを定義します。
- a. まず、テキストエディタを使用して、~/PermissionsForFirehose.json という名前のファイルに以下のようなアクセス許可ポリシーを作成します。ユースケースによっては、このファイルにさらにアクセス権限を追加する必要がある場合があります。

```
{
  "Statement": [{
    "Effect": "Allow",
    "Action": [
      "s3:PutObject",
      "s3:PutObjectAcl",
      "s3:ListBucket"
    ],
    "Resource": [
```

```
        "arn:aws:s3:::amzn-s3-demo-bucket",
        "arn:aws:s3:::amzn-s3-demo-bucket/*"
    ]
  }]
}
```

- b. 次のコマンドを入力して、先ほど作成したアクセス権限ポリシーを IAM ロールに関連付けます。

```
aws iam put-role-policy --role-name FirehoseToS3Role --policy-name
Permissions-Policy-For-Firehose-To-S3 --policy-document file:///~/
PermissionsForFirehose.json
```

4. 次のコマンドを入力して、Firehose 配信ストリームを作成します。*my-role-arn* および *amzn-s3-demo-bucket2-arn* をデプロイに適した値に置き換えます。

```
aws firehose create-delivery-stream \
  --delivery-stream-name 'my-delivery-stream' \
  --s3-destination-configuration \
  '{"RoleARN": "arn:aws:iam::222222222222:role/FirehoseToS3Role", "BucketARN":
  "arn:aws:s3:::amzn-s3-demo-bucket"}'
```

出力は次の例に類似したものになります:

```
{
  "DeliveryStreamARN": "arn:aws:firehose:us-east-1:222222222222:deliverystream/
my-delivery-stream"
}
```

ステップ 2: 送信先を作成する

Important

この手順のステップは、ログデータの受取人アカウントで行われます。

送信先が作成されると、CloudWatch Logs は受信者アカウントに代わってテストメッセージを宛先に送信します。サブスクリプションフィルターが後でアクティブになると、CloudWatch Logs はソースアカウントに代わってログイベントを宛先に送信します。

送信先を作成するには

1. [ステップ 1: Firehose 配信ストリームを作成する](#) で作成した Firehose ストリームがアクティブになるまで待ちます。次のコマンドを使用して、StreamDescription.StreamStatus プロパティを確認できます。

```
aws firehose describe-delivery-stream --delivery-stream-name "my-delivery-stream"
```

さらに、後の手順で使用する必要があるため、DeliveryStreamDescription.DeliveryStreamARN 値を書き留めます。このコマンドの出力例:

```
{
  "DeliveryStreamDescription": {
    "DeliveryStreamName": "my-delivery-stream",
    "DeliveryStreamARN": "arn:aws:firehose:us-east-1:222222222222:deliverystream/my-delivery-stream",
    "DeliveryStreamStatus": "ACTIVE",
    "DeliveryStreamEncryptionConfiguration": {
      "Status": "DISABLED"
    },
    "DeliveryStreamType": "DirectPut",
    "VersionId": "1",
    "CreateTimestamp": "2021-02-01T23:59:15.567000-08:00",
    "Destinations": [
      {
        "DestinationId": "destinationId-000000000001",
        "S3DestinationDescription": {
          "RoleARN": "arn:aws:iam::222222222222:role/FirehoseToS3Role",
          "BucketARN": "arn:aws:s3:::amzn-s3-demo-bucket",
          "BufferingHints": {
            "SizeInMBs": 5,
            "IntervalInSeconds": 300
          },
          "CompressionFormat": "UNCOMPRESSED",
          "EncryptionConfiguration": {
            "NoEncryptionConfig": "NoEncryption"
          },
          "CloudWatchLoggingOptions": {
            "Enabled": false
          }
        },
        "ExtendedS3DestinationDescription": {
```

```

        "RoleARN": "arn:aws:iam::222222222222:role/FirehoseToS3Role",
        "BucketARN": "arn:aws:s3:::amzn-s3-demo-bucket",
        "BufferingHints": {
            "SizeInMBs": 5,
            "IntervalInSeconds": 300
        },
        "CompressionFormat": "UNCOMPRESSED",
        "EncryptionConfiguration": {
            "NoEncryptionConfig": "NoEncryption"
        },
        "CloudWatchLoggingOptions": {
            "Enabled": false
        },
        "S3BackupMode": "Disabled"
    }
},
    "HasMoreDestinations": false
}
}

```

配信ストリームがアクティブ状態が表示されるまでに 1~2 分かかる場合があります。

2. 配信ストリームがアクティブになったら、Firehose ストリームにデータを置くためのアクセス許可を CloudWatch Logs に付与する IAM ロールを作成します。まず、ファイル `~/TrustPolicyForCWL.json` で信頼ポリシーを作成する必要があります。テキストエディタを使用してこのポリシーを作成します。CloudWatch Logs エンドポイントの詳細については、[Amazon CloudWatch Logs エンドポイントおよびクォータ](#)を参照してください。

このポリシーには、「混乱した代理」のセキュリティ上の問題を防止するための `sourceAccountId` が指定された `aws:SourceArn` グローバル条件コンテキストキーが含まれています。最初の呼び出しでソースアカウント ID が不明な場合は、送信元 ARN フィールドに送信先 ARN を指定することをお勧めします。後続の呼び出しでは、送信元 ARN を、最初の呼び出しで取得した実際の送信元 ARN に設定する必要があります。詳細については、「[混乱した代理の防止](#)」を参照してください。

```

{
    "Statement": {
        "Effect": "Allow",
        "Principal": {
            "Service": "logs.region.amazonaws.com"
        },
    },
}

```

```

    "Action": "sts:AssumeRole",
    "Condition": {
      "StringLike": {
        "aws:SourceArn": [
          "arn:aws:logs:region:sourceAccountId:*",
          "arn:aws:logs:region:recipientAccountId:"
        ]
      }
    }
  }
}

```

3. `aws iam create-role` コマンドを使用して、作成した信頼ポリシーファイルを指定して IAM ロールを作成します。

```

aws iam create-role \
  --role-name CWLtoKinesisFirehoseRole \
  --assume-role-policy-document file://~/TrustPolicyForCWL.json

```

以下は出力例です。後のステップで使用する必要があるため、`Role.Arn` の戻り値を書き留めます。

```

{
  "Role": {
    "Path": "/",
    "RoleName": "CWLtoKinesisFirehoseRole",
    "RoleId": "AROAR3BXASEKYJYWF243H",
    "Arn": "arn:aws:iam::222222222222:role/CWLtoKinesisFirehoseRole",
    "CreateDate": "2021-02-02T08:10:43+00:00",
    "AssumeRolePolicyDocument": {
      "Statement": {
        "Effect": "Allow",
        "Principal": {
          "Service": "logs.region.amazonaws.com"
        },
        "Action": "sts:AssumeRole",
        "Condition": {
          "StringLike": {
            "aws:SourceArn": [
              "arn:aws:logs:region:sourceAccountId:*",
              "arn:aws:logs:region:recipientAccountId:"
            ]
          }
        }
      }
    }
  }
}

```

```

    }
  }
}

```

4. CloudWatch Logs がアカウントで実行できるアクションを定義するアクセス許可ポリシーを作成します。まず、テキストエディタを使用してファイル `~/PermissionsForCWL.json` でアクセス許可ポリシーを作成します。

```

{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["firehose:*"],
      "Resource": ["arn:aws:firehose:region:222222222222:*"]
    }
  ]
}

```

5. 次のコマンドを入力して、アクセス権限ポリシーをロールに関連付けます。

```

aws iam put-role-policy --role-name CWLtoKinesisFirehoseRole --policy-name
Permissions-Policy-For-CWL --policy-document file://~/PermissionsForCWL.json

```

6. Firehose 配信ストリームがアクティブ状態になり、IAM ロールが作成されたら、CloudWatch Logs の送信先を作成できます。
 - a. このステップでは、アクセスポリシーと送信先は関連付けられません。送信先の作成を完了するには 2 つのステップを行う必要がありますが、このステップはその最初のステップです。後のステップでこれを `destination.arn` として使用するため、ペイロードで返される新しい宛先の ARN を書き留めます。

```

aws logs put-destination \

    --destination-name "testFirehoseDestination" \
    --target-arn "arn:aws:firehose:us-east-1:222222222222:deliverystream/my-
delivery-stream" \
    --role-arn "arn:aws:iam::222222222222:role/CWLtoKinesisFirehoseRole"

{

```

```

    "destination": {
      "destinationName": "testFirehoseDestination",
      "targetArn": "arn:aws:firehose:us-east-1:222222222222:deliverystream/
my-delivery-stream",
      "roleArn": "arn:aws:iam::222222222222:role/CWLtoKinesisFirehoseRole",
      "arn": "arn:aws:logs:us-
east-1:222222222222:destination:testFirehoseDestination"}
  }

```

- b. 前のステップが完了したら、ログデータ受取人アカウント (222222222222) で、アクセスポリシーを送信先に関連付けます。

このポリシーにより、ログデータの送信者アカウント (111111111111) に対し、ログデータの受信者アカウント (222222222222) にある送信先にアクセスすることを許可します。テキストエディタを使用して、このポリシーを `~/AccessPolicy.json` ファイルに配置できます。

JSON

```

{
  "Version": "2012-10-17",
  "Statement" : [
    {
      "Sid" : "",
      "Effect" : "Allow",
      "Principal" : {
        "AWS" : "111111111111"
      },
      "Action" : "logs:PutSubscriptionFilter",
      "Resource" : "arn:aws:logs:us-
east-1:222222222222:destination:testFirehoseDestination"
    }
  ]
}

```

- c. これにより、誰が送信先に書き込むことができるかを定義するポリシーが作成されます。このポリシーでは、送信先にアクセスするための `logs:PutSubscriptionFilter` アクションが指定されている必要があります。クロスアカウントのユーザーは、`PutSubscriptionFilter` アクションを使用して送信先にログイベントを送信します。

```

aws logs put-destination-policy \
  --destination-name "testFirehoseDestination" \

```



```
--access-policy file://~/AccessPolicy.json
```

ステップ 3: クロスアカウント宛先の IAM アクセス許可を追加/検証する

AWS クロスアカウントポリシーの評価ロジックに従って、クロスアカウントリソース (サブスクリプションフィルターの送信先として使用される Kinesis または Firehose ストリームなど) にアクセスするには、クロスアカウントの送信先リソースへの明示的なアクセスを提供する ID ベースのポリシーを送信アカウントに含める必要があります。ポリシーの評価論理の詳細については、「[クロスアカウントポリシーの評価論理](#)」を参照してください。

ID ベースのポリシーは、サブスクリプションフィルターの作成に使用している IAM ロールまたは IAM ユーザーにアタッチできます。送信アカウントにこのポリシーが存在する必要があります。管理者ロールを使用してサブスクリプションフィルターを作成している場合は、このステップをスキップして [ステップ 4: サブスクリプションフィルターを作成する](#) に進んでください。

クロスアカウントに必要な IAM アクセス許可を追加または検証するには

1. 次のコマンドを入力して、AWS ログコマンドの実行に使用されている IAM ロールまたは IAM ユーザーを確認します。

```
aws sts get-caller-identity
```

このコマンドにより、以下のような出力が返されます。

```
{
  "UserId": "User ID",
  "Account": "sending account id",
  "Arn": "arn:aws:sending account id:role/user:RoleName/UserName"
}
```

RoleName または *UserName* で表される値を書き留めておいてください。

2. 送信アカウント AWS マネジメントコンソール でサインインし、ステップ 1 で入力したコマンドの出力で返された IAM ロールまたは IAM ユーザーを使用して、アタッチされたポリシーを検索します。
3. このロールまたはユーザーにアタッチされたポリシーが、クロスアカウントの宛先リソースで `logs:PutSubscriptionFilter` を呼び出すための明示的なアクセス許可が付与されていることを確認します。

次のポリシーは、1 つの AWS アカウント、アカウント でのみ、任意の送信先リソースにサブスクリプションフィルターを作成するアクセス許可を提供します999999999999。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid":
"AllowSubscriptionFiltersOnAnyResourceInOneSpecificAccount",
      "Effect": "Allow",
      "Action": "logs:PutSubscriptionFilter",
      "Resource": [
        "arn:aws:logs:*:*:log-group:*",
        "arn:aws:logs*:123456789012:destination:*"
      ]
    }
  ]
}
```

次のポリシーは、単一の AWS アカウント、アカウント sampleDestinationで という名前の特定の送信先リソースに対してのみサブスクリプションフィルターを作成するアクセス許可を提供します123456789012。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowSubscriptionFiltersOnSpecificResource",
      "Effect": "Allow",
      "Action": "logs:PutSubscriptionFilter",
      "Resource": [
        "arn:aws:logs:*:*:log-group:*",
        "arn:aws:logs*:123456789012:destination:amzn-s3-demo-bucket"
      ]
    }
  ]
}
```

```
]
}
```

ステップ 4: サブスクリプションフィルターを作成する

送信側のアカウント (この例では 111111111111) に切り替えます。次に、送信側のアカウントにサブスクリプションフィルターを作成します。この例では、フィルターは AWS CloudTrail イベントを含むロググループに関連付けられているため、「ルート AWS」認証情報によって行われたログに記録されたすべてのアクティビティが、以前に作成した送信先に配信されます。CloudWatch Logs に AWS CloudTrail イベントを送信する方法の詳細については、AWS CloudTrail「[ユーザーガイド](#)」の [CloudTrail イベントを CloudWatch Logs に送信する](#) を参照してください。

次のコマンドを入力するときは、必ず IAM ユーザーとしてサインインしているか、[ステップ 3: クロスアカウント宛先の IAM アクセス許可を追加/検証する](#) にポリシーを追加した IAM ロールを使用してサインインしていることを確認してください。

```
aws logs put-subscription-filter \
  --log-group-name "aws-cloudtrail-logs-111111111111-300a971e" \
  --filter-name "firehose_test" \
  --filter-pattern "${.userIdentity.type = AssumedRole}" \
  --destination-arn "arn:aws:logs:us-east-1:222222222222:destination:testFirehoseDestination"
```

ロググループと送信先は同じ AWS リージョンにある必要があります。ただし、送信先は、別のリージョンにある Firehose ストリームなどの AWS リソースを指すことができます。

ログイベントの送信の検証

サブスクリプションフィルターを作成すると、CloudWatch Logs は、フィルタパターンに一致するすべての受信ログイベントを Firehose 配信ストリームに転送します。データは、Firehose 配信ストリームに設定されている時間バッファ間隔に基づいて、Amazon S3 バケットに順次表示されます。十分な時間が経過すると、Amazon S3 バケットをチェックしてデータを確認できます。バケットを確認するには、次のコマンドを入力します。

```
aws s3api list-objects --bucket 'amzn-s3-demo-bucket'
```

そのコマンドの出力は、次のようになります。

```
{
```

```
"Contents": [
  {
    "Key": "2021/02/02/08/my-delivery-
stream-1-2021-02-02-08-55-24-5e6dc317-071b-45ba-a9d3-4805ba39c2ba",
    "LastModified": "2021-02-02T09:00:26+00:00",
    "ETag": "\"EXAMPLEa817fb88fc770b81c8f990d\"",
    "Size": 198,
    "StorageClass": "STANDARD",
    "Owner": {
      "DisplayName": "firehose+2test",
      "ID": "EXAMPLE27fd05889c665d2636218451970ef79400e3d2aecca3adb1930042e0"
    }
  }
]
```

その後、次のコマンドを入力して、バケットから特定のオブジェクトを取得できます。key の値を、前のコマンドで検索した値に置き換えます。

```
aws s3api get-object --bucket 'amzn-s3-demo-bucket' --key '2021/02/02/08/my-delivery-
stream-1-2021-02-02-08-55-24-5e6dc317-071b-45ba-a9d3-4805ba39c2ba' testfile.gz
```

Simple Storage Service (Amazon S3) オブジェクトのデータは、gzip 形式で圧縮されます。raw データは、コマンドラインから次のコマンドを使用して調べることができます。

Linux:

```
zcat testfile.gz
```

macOS:

```
zcat <testfile.gz
```

実行時の送信先のメンバーシップの変更

所有している送信先からログ送信者を追加または削除しなければならない状況が発生することがあります。新しいアクセスポリシーが関連付けられている送信先に対して PutDestinationPolicy アクションを使用できます。次の例では、先ほど追加したアカウント 111111111111 がログデータの送信を停止し、アカウント 333333333333 が有効になります。

1. 現在 testDestination という送信先に関連付けられているポリシーをフェッチし、AccessPolicy を書き留めておきます。

```
aws logs describe-destinations \
  --destination-name-prefix "testFirehoseDestination"

{
  "destinations": [
    {
      "destinationName": "testFirehoseDestination",
      "targetArn": "arn:aws:firehose:us-east-1:222222222222:deliverystream/
my-delivery-stream",
      "roleArn": "arn:aws:iam:: 222222222222:role/CWLtoKinesisFirehoseRole",
      "accessPolicy": "{\n  \"Version\" : \"2012-10-17\",\n  \"Statement\"
\n : [\n    {\n      \"Sid\" : \"\",\n      \"Effect\" : \"Allow\",\n
      \"Principal\" : {\n        \"AWS\" : \"111111111111 \"\n      },\n      \"Action\"
\n : \"logs:PutSubscriptionFilter\",\n      \"Resource\" : \"arn:aws:logs:us-
east-1:222222222222:destination:testFirehoseDestination\"\n    }\n  ]\n}\n\n",
      "arn": "arn:aws:logs:us-east-1:
222222222222:destination:testFirehoseDestination",
      "creationTime": 1612256124430
    }
  ]
}
```

2. アカウント 111111111111 が停止したこととアカウント 333333333333 が有効になったことを反映させるためにポリシーを更新します。このポリシーを ~/NewAccessPolicy.json ファイルに配置します。

JSON

```
{
  "Version": "2012-10-17",
  "Statement" : [
    {
      "Sid" : "",
      "Effect" : "Allow",
      "Principal" : {
        "AWS" : "333333333333 "
      },
      "Action" : "logs:PutSubscriptionFilter",
```

```
"Resource" : "arn:aws:logs:us-east-1:222222222222:destination:testFirehoseDestination"
    }
  ]
}
```

3. 次のコマンドを使用して、NewAccessPolicy.json ファイルで定義されたポリシーを送信先に関連付けます。

```
aws logs put-destination-policy \
  --destination-name "testFirehoseDestination" \
  --access-policy file:///~/NewAccessPolicy.json
```

これにより、最終的には、アカウント ID 111111111111 からのログイベントが無効になります。アカウント ID 333333333333 の所有者がサブスクリプションフィルターを作成すると、すぐに 333333333333 からのログイベントが送信先に送信されるようになります。

Amazon Kinesis Data Streams を使用したクロスアカウントクロスリージョンアカウントレベルのサブスクリプション

クロスアカウントサブスクリプションを作成するときに、単一のアカウントまたは組織を送信者として指定できます。組織を指定した場合、この手順により組織内のすべてのアカウントがレシーバーアカウントにログを送信できるようになります。

複数のアカウントでログデータを共有するには、ログデータの送信者と受信者を確立する必要があります。

- [Log data sender] (ログデータの送信者) — 受信者から送信先情報を取得し、そのログイベントを特定の送信先に送信する準備が完了していることを CloudWatch Logs に通知します。このセクションの残りの手順では、ログデータ送信者は架空の AWS アカウント番号 111111111111 で表示されます。

1 つの組織で複数のアカウントを 1 つの受信者アカウントにログを送信する場合は、組織内のすべてのアカウントに対して受信者アカウントにログを送信するアクセス許可を付与するポリシーを作成することができます。引き続き、送信者アカウントごとに個別のサブスクリプションフィルターを設定する必要があります。

- ログデータ受信者 - Amazon Kinesis Data Streams ストリームをカプセル化する送信先を設定し、受信者がログデータを受信したいことを CloudWatch Logs に知らせます。この後、受信者は自分

の送信先に関する情報を送信者と共有します。このセクションの残りの手順では、ログデータ受信者は架空の AWS アカウント番号 999999999999 で表示されます。

クロスアカウントのユーザーからのログイベントの受け取りを開始するには、ログデータの受取人がまず CloudWatch Logs 送信先を作成する必要があります。各送信先は以下のキー要素で構成されています。

送信先名

作成する送信先の名前。

ターゲット ARN

サブスクリプションフィードの送信先として使用するリソースの Amazon AWS リソースネーム (ARN)。

ロールの ARN

選択したストリームにデータを配置するために必要なアクセス許可を CloudWatch Logs に付与する AWS Identity and Access Management (IAM) ロール。

アクセスポリシー

送信先に書き込むことが許可されている一連のユーザーを管理する IAM ポリシードキュメント (IAM ポリシー構文を使用して記述された JSON 形式のドキュメント)。

Note

ロググループと送信先は同じ AWS リージョンにある必要があります。ただし、送信先が指す AWS リソースは、別のリージョンに配置することができます。次のセクションの例では、リージョン固有のリソースはすべて米国東部 (バージニア北部) で作成されます。

トピック

- [新しいクロスアカウントサブスクリプションの設定](#)
- [既存のクロスアカウントサブスクリプションの更新](#)

新しいクロスアカウントサブスクリプションの設定

次のセクションの手順に従って、新しいクロスアカウントログサブスクリプションを設定します。

トピック

- [ステップ 1: 送信先を作成する](#)
- [ステップ 2: IAM ロールを作成する \(組織を使用している場合のみ\)](#)
- [ステップ 3: アカウントレベルのサブスクリプションフィルタポリシーを作成する](#)
- [ログイベントの送信を検証](#)
- [ランタイムの送信先のメンバーシップを変更](#)

ステップ 1: 送信先を作成する

Important

この手順のステップは、ログデータの受取人アカウントで行われます。

この例では、ログデータ受信者アカウントの AWS アカウント ID は 999999999999 で、ログデータ送信者 AWS アカウント ID は 111111111111 です。

この例では、RecipientStream という Amazon Kinesis Data Streams ストリームと、CloudWatch Logs によるデータの書き込みを可能にするロールを使用して送信先を作成します。

送信先が作成されると、CloudWatch Logs は受信者アカウントに代わってテストメッセージを宛先に送信します。サブスクリプションフィルタが後でアクティブになると、CloudWatch Logs はソースアカウントに代わってログイベントを宛先に送信します。

送信先を作成するには

1. 受信者アカウントで、Amazon Kinesis Data Streams で送信先ストリームを作成します。コマンドプロンプトで、次のように入力します。

```
aws kinesis create-stream --stream-name "RecipientStream" --shard-count 1
```

2. ストリームがアクティブになるまで待ちます。aws kinesis describe-stream コマンドを使用して、StreamDescription.StreamStatus プロパティをチェックできます。さらに、StreamDescription.StreamARN の値は、後で CloudWatch Logs に渡すので書き留めておいてください。

```
aws kinesis describe-stream --stream-name "RecipientStream"
{
```



```

"StreamDescription": {
  "StreamStatus": "ACTIVE",
  "StreamName": "RecipientStream",
  "StreamARN": "arn:aws:kinesis:us-east-1:999999999999:stream/RecipientStream",
  "Shards": [
    {
      "ShardId": "shardId-000000000000",
      "HashKeyRange": {
        "EndingHashKey": "34028236692093846346337460743176EXAMPLE",
        "StartingHashKey": "0"
      },
      "SequenceNumberRange": {
        "StartingSequenceNumber":
"4955113521868881845667950383198145878459135270218EXAMPLE"
      }
    }
  ]
}
}

```

ストリームがアクティブ状態で表示されるまでに 1~2 分かかる場合があります。

3. ストリームにデータを置くアクセス権限を CloudWatch Logs に付与する IAM ロールを作成します。まず、ファイル `~/TrustPolicyForCWL.json` で信頼ポリシーを作成する必要があります。このポリシーの作成にはテキストエディタを使用します。IAM コンソールは使用しないでください。

このポリシーには、「混乱した代理」のセキュリティ上の問題を防止するための `sourceAccountId` が指定された `aws:SourceArn` グローバル条件コンテキストキーが含まれています。最初の呼び出しでソースアカウント ID が不明な場合は、送信元 ARN フィールドに送信先 ARN を指定することをお勧めします。後続の呼び出しでは、送信元 ARN を、最初の呼び出しで取得した実際の送信元 ARN に設定する必要があります。詳細については、「[混乱した代理の防止](#)」を参照してください。

```

{
  "Statement": {
    "Effect": "Allow",
    "Principal": {
      "Service": "logs.amazonaws.com"
    },
    "Condition": {
      "StringLike": {

```

```

        "aws:SourceArn": [
            "arn:aws:logs:region:sourceAccountId:*",
            "arn:aws:logs:region:recipientAccountId:*"
        ]
    },
    "Action": "sts:AssumeRole"
}

```

4. `aws iam create-role` コマンドを使用して、信頼ポリシーファイルを指定する IAM ロールを作成します。返された `Role.Arn` 値を書き留めておきます。これも後で CloudWatch Logs に渡されるからです。

```

aws iam create-role \
--role-name CWLtoKinesisRole \
--assume-role-policy-document file://~/TrustPolicyForCWL.json

{
  "Role": {
    "AssumeRolePolicyDocument": {
      "Statement": {
        "Action": "sts:AssumeRole",
        "Effect": "Allow",
        "Condition": {
          "StringLike": {
            "aws:SourceArn": [
              "arn:aws:logs:region:sourceAccountId:*",
              "arn:aws:logs:region:recipientAccountId:*"
            ]
          }
        },
        "Principal": {
          "Service": "logs.amazonaws.com"
        }
      }
    },
    "RoleId": "AA0IIAH450GAB4HC5F431",
    "CreateDate": "2023-05-29T13:46:29.431Z",
    "RoleName": "CWLtoKinesisRole",
    "Path": "/",
    "Arn": "arn:aws:iam::999999999999:role/CWLtoKinesisRole"
  }
}

```

```
}

```

5. CloudWatch Logs がアカウントで実行できるアクションを定義するアクセス許可ポリシーを作成します。まず、テキストエディタを使用してファイル `~/PermissionsForCWL.json` でアクセス許可ポリシーを作成します。

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "kinesis:PutRecord",
      "Resource": "arn:aws:kinesis:region:999999999999:stream/RecipientStream"
    }
  ]
}
```

6. `aws iam put-role-policy` コマンドを使用して、アクセス許可ポリシーをロールに関連付けます。

```
aws iam put-role-policy \
  --role-name CWLtoKinesisRole \
  --policy-name Permissions-Policy-For-CWL \
  --policy-document file://~/PermissionsForCWL.json
```

7. ストリームがアクティブ状態になり、IAM ロールが作成されたら、CloudWatch Logs の送信先を作成できます。
 - a. このステップでは、アクセスポリシーと送信先は関連付けられません。送信先の作成を完了するには 2 つのステップを行う必要がありますが、このステップはその最初のステップです。ペイロードで返された `DestinationArn` を書き留めておいてください。

```
aws logs put-destination \
  --destination-name "testDestination" \
  --target-arn "arn:aws:kinesis:region:999999999999:stream/RecipientStream" \
  --role-arn "arn:aws:iam::999999999999:role/CWLtoKinesisRole"

{
  "DestinationName" : "testDestination",
  "RoleArn" : "arn:aws:iam::999999999999:role/CWLtoKinesisRole",
  "DestinationArn" : "arn:aws:logs:us-east-1:999999999999:destination:testDestination",
  "TargetArn" : "arn:aws:kinesis:us-east-1:999999999999:stream/RecipientStream"
```

```
}
```

- b. ステップ 7a が完了したら、ログデータの受取人アカウントで、アクセスポリシーを送信先に関連付けます。このポリシーでは、送信者アカウントが送信先にアクセスするためのアクセス許可が付与され、logs:PutSubscriptionFilter アクションが指定されている必要があります。

このポリシーは、ログを送信する AWS アカウントにアクセス許可を付与します。ポリシーの中で対象のアカウントを 1 つだけ指定してもよいですが、送信者アカウントが組織のメンバーのものである場合は組織 ID を指定することもできます。このように、ポリシーを 1 つ作成するだけで、1 つの組織内の複数のアカウントが送信先アカウントにログを送信できるように設定できます。

テキストエディタを使用して ~/AccessPolicy.json という名前のファイルを作成し、以下のいずれかのポリシーステートメントを使用します。

この最初の例のポリシーでは、組織内で o-1234567890 という ID を持つすべてのアカウントが、受信者アカウントにログを送信することを許可します。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": "*",
      "Action": [
        "logs:PutSubscriptionFilter",
        "logs:PutAccountPolicy"
      ],
      "Resource": "arn:aws:logs:us-east-1:999999999999:destination:testDestination",
      "Condition": {
        "StringEquals": {
          "aws:PrincipalOrgID": [
            "o-1234567890"
          ]
        }
      }
    }
  ]
}
```

```
    }
  ]
}
```

次の例では、ログデータの送信者アカウント (111111111111) がログデータの受信者アカウントにログを送信できるようにします。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "AWS": "111111111111"
      },
      "Action": [
        "logs:PutSubscriptionFilter",
        "logs:PutAccountPolicy"
      ],
      "Resource": "arn:aws:logs:us-east-1:999999999999:destination:testDestination"
    }
  ]
}
```

- c. 前のステップで作成したポリシーを送信先に添付します。

```
aws logs put-destination-policy \
  --destination-name "testDestination" \
  --access-policy file://~/AccessPolicy.json
```

このアクセスポリシーにより、ID 111111111111 の AWS アカウントのユーザーは、ARN `arn:aws:logs:region:999999999999:destination:testDestination` の送信先に対して `PutSubscriptionFilter` を呼び出すことができます。他のユーザーがこの送信先に対して `PutSubscriptionFilter` を呼び出そうとしても、それは却下されます。

アクセスポリシーに照らし合わせてユーザーの権限を検証するには、「IAM ユーザーガイド」の「[Using Policy Validator](#)」(Policy Validator の使用) を参照してください。

完了したら、クロスアカウントのアクセス許可 AWS Organizations に を使用している場合は、「」の順に従います[ステップ 2: IAM ロールを作成する \(組織を使用している場合のみ\)](#)。組織を使用せずに他のアカウントに直接アクセス許可を付与する場合は、そのステップを飛ばして「[ステップ 3: アカウントレベルのサブスクリプションフィルタポリシーを作成する](#)」に進みます。

ステップ 2: IAM ロールを作成する (組織を使用している場合のみ)

前のセクションで アカウント 111111111111 に直接アクセス許可を付与するのではなく、アカウント 111111111111 が属する組織にアクセス許可を付与するアクセスポリシーを使用することにより送信先を作成した場合は、このセクションのステップを実行します。それ以外の場合は、「[ステップ 3: アカウントレベルのサブスクリプションフィルタポリシーを作成する](#)」に進みます。

このセクションのステップにより、CloudWatch が想定する IAM ロールを作成し、送信者アカウントが受信者の送信先に対してサブスクリプションフィルターを作成する権限を持っているかどうかを検証できます。

このセクションの手順は、送信者アカウントで実行してください。ロールは送信者アカウントに存在する必要があり、このロールの ARN はサブスクリプションフィルターで指定します。この例では、送信者アカウントは 111111111111 です。

を使用してクロスアカウントログサブスクリプションに必要な IAM ロールを作成するには AWS Organizations

1. 以下の信頼ポリシーを作成し、/TrustPolicyForCWLSubscriptionFilter.json という名前のテキストファイルに保存します。このポリシーの作成にはテキストエディタを使用します。IAM コンソールは使用しないでください。

```
{
  "Statement": {
    "Effect": "Allow",
    "Principal": { "Service": "logs.amazonaws.com" },
    "Action": "sts:AssumeRole"
  }
}
```

2. このポリシーを使用する IAM ロールを作成します。下記のコマンドが返す Arn の値は後ほど必要になるため、書き留めておきます。この例では、作成するロールに `CWLtoSubscriptionFilterRole` という名前を付けます。

```
aws iam create-role \  
  --role-name CWLtoSubscriptionFilterRole \  
  --assume-role-policy-document file://~/TrustPolicyForCWLSubscriptionFilter.json
```

3. アクセス許可ポリシーを作成して、CloudWatch Logs がアカウントで実行できるアクションを定義します。
 - a. まず、テキストエディタを使用して、~/PermissionsForCWLSubscriptionFilter.json という名前のファイルに以下のようなアクセス許可ポリシーを作成します。

```
{  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "Action": "logs:PutLogEvents",  
      "Resource": "arn:aws:logs:region:111111111111:log-group:LogGroupOnWhichSubscriptionFilterIsCreated:*"  
    }  
  ]  
}
```

- b. 次のコマンドを入力して、先ほど作成したアクセス許可ポリシーを、ステップ 2 で作成したロールに関連付けます。

```
aws iam put-role-policy  
  --role-name CWLtoSubscriptionFilterRole  
  --policy-name Permissions-Policy-For-CWL-Subscription-filter  
  --policy-document file://~/PermissionsForCWLSubscriptionFilter.json
```

終了したら、「[ステップ 3: アカウントレベルのサブスクリプションフィルタポリシーを作成する](#)」に進みます。

ステップ 3: アカウントレベルのサブスクリプションフィルタポリシーを作成する

送信先を作成したら、ログデータの受信者アカウントは、送信先の ARN (arn:aws:logs:us-east-1:999999999999:destination:testDestination) を他の AWS アカウントと共有できるようになります。これにより、これらのアカウントは同じ送信先にログイベントを送信できます。この後、これらの他の送信アカウントのユーザーは、この送信先に対するサブスクリプションフィルタをそれぞれのロググループに作成します。サブスクリプションフィルタは、特定のロググループから特定の送信先へのリアルタイムログデータの送信をすぐに開始します。

Note

サブスクリプションフィルタのためのアクセス許可を組織全体に付与する際は、[ステップ 2: IAM ロールを作成する \(組織を使用している場合のみ\)](#) で作成した IAM ロールの ARN を使用する必要があります。

次の例では、アカウントレベルのサブスクリプションフィルタポリシーが送信アカウントで作成されます。フィルタは送信者アカウント 111111111111 に関連付けられ、フィルタと選択基準に一致するすべてのログイベントが以前に作成した送信先に配信されます。この送信先は、「RecipientStream」というストリームをカプセル化します。

selection-criteria フィールドはオプションですが、サブスクリプションフィルタから無限のログ再帰を引き起こす可能性のあるロググループを除外するための重要なフィールドです。この問題および除外するロググループを判断する方法については、「[ログの再帰防止](#)」を参照してください。現在、NOT IN は selection-criteria でサポートされている唯一の演算子です。

```
aws logs put-account-policy \
  --policy-name "CrossAccountStreamsExamplePolicy" \
  --policy-type "SUBSCRIPTION_FILTER_POLICY" \
  --policy-document
'{"DestinationArn":"arn:aws:logs:region:999999999999:destination:testDestination",
"FilterPattern": "", "Distribution": "Random"}' \
  --selection-criteria 'LogGroupName NOT IN ["LogGroupToExclude1",
"LogGroupToExclude2"]' \
  --scope "ALL"
```

送信者アカウントのロググループと送信先は同じ AWS リージョンに存在している必要があります。ただし、送信先は、別のリージョンにある Amazon Kinesis Data Streams ストリームなどの AWS リソースを指すことができます。

ログイベントの送信を検証

アカウントレベルのサブスクリプションフィルターポリシーを作成したら、CloudWatch Logs が、フィルタパターンおよび選択基準と一致するすべての受信ログイベントを、送信先ストリーム内でカプセル化されている「RecipientStream」という名前のストリームに転送します。送信先所有者は、aws kinesis get-shard-iterator コマンドを使用して Amazon Kinesis Data Streams シャードを取得し、aws kinesis get-records コマンドを使用して Amazon Kinesis Data Streams レコードを取得することで、これが起こっていることを確認できます。

```
aws kinesis get-shard-iterator \  
    --stream-name RecipientStream \  
    --shard-id shardId-000000000000 \  
    --shard-iterator-type TRIM_HORIZON  
  
{  
    "ShardIterator":  
    "AAAAAAAAAAFGU/  
kLvNggvndHq2UIF0w5PZc6F01s3e3afsSscRM70JSbjIefg2ub07nk1y6CDxYR1UoGHJNP4m4NFUetzfL+wev  
+e2P4djJg4L9wmXKvQYoE+rMUiFq+p4Cn3Igvq0b5dRA0yybNdRcdzvnC35KQANoHzzahKdRGb9v4scv+3vaq+f  
+0IK8zM5My8ID+g6rMo7UKWeI4+IWiKEXAMPLE"  
}  
  
aws kinesis get-records \  
    --limit 10 \  
    --shard-iterator  
    "AAAAAAAAAAFGU/  
kLvNggvndHq2UIF0w5PZc6F01s3e3afsSscRM70JSbjIefg2ub07nk1y6CDxYR1UoGHJNP4m4NFUetzfL+wev  
+e2P4djJg4L9wmXKvQYoE+rMUiFq+p4Cn3Igvq0b5dRA0yybNdRcdzvnC35KQANoHzzahKdRGb9v4scv+3vaq+f  
+0IK8zM5My8ID+g6rMo7UKWeI4+IWiKEXAMPLE"
```

Note

Amazon Kinesis Data Streams がデータを返す前に、get-records コマンドを数回再実行する必要があります。

Amazon Kinesis Data Streams レコードの配列を含むレスポンスが表示されます。Amazon Kinesis Data Streams レコードのデータ属性は gzip 形式で圧縮され、base64 でエンコードされます。raw データは、コマンドラインから次の UNIX コマンドを使用して調べることができます。

```
echo -n "<Content of Data>" | base64 -d | zcat
```

base64 でデコードおよび解凍されたデータは、次の構造を使用して JSON としてフォーマットされます。

```
{
  "owner": "111111111111",
  "logGroup": "CloudTrail/logs",
  "logStream": "111111111111_CloudTrail/logs_us-east-1",
  "subscriptionFilters": [
    "RecipientStream"
  ],
  "messageType": "DATA_MESSAGE",
  "logEvents": [
    {
      "id": "3195310660696698337880902507980421114328961542429EXAMPLE",
      "timestamp": 1432826855000,
      "message": "{\"eventVersion\":\"1.03\",\"userIdentity\":{\"type\":\"Root\",
    },
    {
      "id": "3195310660696698337880902507980421114328961542429EXAMPLE",
      "timestamp": 1432826855000,
      "message": "{\"eventVersion\":\"1.03\",\"userIdentity\":{\"type\":\"Root\",
    },
    {
      "id": "3195310660696698337880902507980421114328961542429EXAMPLE",
      "timestamp": 1432826855000,
      "message": "{\"eventVersion\":\"1.03\",\"userIdentity\":{\"type\":\"Root\",
    }
  ]
}
```

データ構造の主要要素は次のとおりです。

messageType

データメッセージは、"DATA_MESSAGE" 型を使用します。CloudWatch Logs は、主に送信先に到達可能かどうかを確認するために、「CONTROL_MESSAGE」タイプの Amazon Kinesis Data Streams レコードを出力することがあります。

owner (オーナー)

元のログデータの AWS アカウント ID。

logGroup

発行元ログデータのロググループ名。

logStream

発行元ログデータのログストリーム名。

subscriptionFilters

発行元ログデータと一致したサブスクリプションフィルタ名のリスト。

logEvents

ログイベントレコードの配列として表される実際のログデータ。"id" プロパティは、各ログイベントの一意識別子です。

policyLevel

ポリシーが適用されたレベル。「ACCOUNT_LEVEL_POLICY」は、アカウントレベルのサブスクリプションフィルターポリシーの policyLevel です。

ランタイムの送信先のメンバーシップを変更

所有する送信先のユーザーのメンバーシップを追加または削除する必要がある場合があります。新しいアクセスポリシーを使用して、送信先で put-destination-policy コマンドを使用できます。次の例では、先ほど追加したアカウント 111111111111 がログデータの送信を停止し、アカウント 222222222222 が有効になります。

1. 現在 testDestination という送信先に関連付けられているポリシーをフェッチし、AccessPolicy を書き留めておきます。

```
aws logs describe-destinations \
  --destination-name-prefix "testDestination"

{
  "Destinations": [
    {
      "DestinationName": "testDestination",
      "RoleArn": "arn:aws:iam::999999999999:role/CWLtoKinesisRole",
```

```

    "DestinationArn":
    "arn:aws:logs:region:999999999999:destination:testDestination",
    "TargetArn": "arn:aws:kinesis:region:999999999999:stream/RecipientStream",
    "AccessPolicy": "{ \"Version\": \"2012-10-17\", \"Statement\":
    [{ \"Sid\": \"\", \"Effect\": \"Allow\", \"Principal\": { \"AWS\":
    \"111111111111\" }, \"Action\": \"logs:PutSubscriptionFilter\", \"Resource\":
    \"arn:aws:logs:region:999999999999:destination:testDestination\" } ] }"
  }
]
}

```

2. アカウント 111111111111 が停止したとアカウント 222222222222 が有効になったことを反映させるためにポリシーを更新します。このポリシーを ~/NewAccessPolicy.json ファイルに配置します。

JSON

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "AWS": "222222222222"
      },
      "Action": [
        "logs:PutSubscriptionFilter",
        "logs:PutAccountPolicy"
      ],
      "Resource": "arn:aws:logs:us-
east-1:999999999999:destination:testDestination"
    }
  ]
}

```

3. PutDestinationPolicy を呼び出して、NewAccessPolicy.json ファイルで定義されているポリシーを送信先に関連付けます。

```

aws logs put-destination-policy \
--destination-name "testDestination" \

```

```
--access-policy file://~/NewAccessPolicy.json
```

これにより、最終的には、アカウント ID 111111111111 からのログイベントが無効になります。アカウント ID 222222222222 の所有者がサブスクリプションフィルターを作成すると、すぐに 222222222222 からのログイベントが送信先に送信されるようになります。

既存のクロスアカウントサブスクリプションの更新

送信先アカウントが特定の送信者アカウントにのみアクセス許可を付与しているクロスアカウントのログサブスクリプションがあり、このサブスクリプションを更新して送信先アカウントが組織内のすべてのアカウントにアクセスできるようにする場合は、このセクションのステップを実施します。

トピック

- [ステップ 1: サブスクリプションフィルターを更新する](#)
- [ステップ 2: 既存の送信先アクセスポリシーを更新する](#)

ステップ 1: サブスクリプションフィルターを更新する

Note

この手順は、[AWS サービスからのログ記録を有効にする](#) に記載されているサービスによって作成されたログのクロスアカウントのサブスクリプションにのみ必要です。これらのロググループのいずれかで作成されたログを操作していない場合は、[ステップ 2: 既存の送信先アクセスポリシーを更新する](#) にスキップできます。

場合によっては、送信先アカウントにログを送信する、すべての送信者アカウントのサブスクリプションフィルターを更新する必要があります。この更新により IAM ロールが追加されます。IAM ロールは CloudWatch が引き受けることができ、送信者アカウントが受信者アカウントにログを送信する権限を持っていることを検証できます。

すべての送信者アカウントについてクロスアカウントサブスクリプションのアクセス許可に組織 ID を使用するようには、このセクションのステップを実施します。

このセクションの例では、2 つのアカウント 111111111111 と 222222222222 は、アカウント 999999999999 にログを送信するために作成されたサブスクリプションフィルターをすでに持っています。既存のサブスクリプションフィルター値は次のとおりです。

```
## Existing Subscription Filter parameter values
{
  "DestinationArn": "arn:aws:logs:region:999999999999:destination:testDestination",
  "FilterPattern": "{$.userIdentity.type = Root}",
  "Distribution": "Random"
}
```

現在のサブスクリプションフィルターパラメータ値を見つける必要がある場合は、次のコマンドを入力します。

```
aws logs describe-account-policies \
--policy-type "SUBSCRIPTION_FILTER_POLICY" \
--policy-name "CrossAccountStreamsExamplePolicy"
```

サブスクリプションフィルターを更新して、クロスアカウントログの権限で組織 ID の使用をスタートする方法

1. 以下の信頼ポリシーを作成し、~/TrustPolicyForCWL.json という名前のテキストファイルに保存します。このポリシーの作成にはテキストエディタを使用します。IAM コンソールは使用しないでください。

```
{
  "Statement": {
    "Effect": "Allow",
    "Principal": { "Service": "logs.amazonaws.com" },
    "Action": "sts:AssumeRole"
  }
}
```

2. このポリシーを使用する IAM ロールを作成します。下記のコマンドが返す Arn 値の Arn の値は後ほど必要になるため、書き留めておきます。この例では、作成するロールに CWLtoSubscriptionFilterRole という名前を付けます。

```
aws iam create-role
\ --role-name CWLtoSubscriptionFilterRole
\ --assume-role-policy-document file://~/TrustPolicyForCWL.json
```

3. アクセス許可ポリシーを作成して、CloudWatch Logs がアカウントで実行できるアクションを定義します。

- a. まず、テキストエディタを使用して、/
PermissionsForCWLSubscriptionFilter.json という名前のファイルに以下のようなアクセス許可ポリシーを作成します。

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "logs:PutLogEvents",
      "Resource": "arn:aws:logs:region:111111111111:log-
group:LogGroupOnWhichSubscriptionFilterIsCreated:*"
    }
  ]
}
```

- b. 次のコマンドを入力して、先ほど作成したアクセス許可ポリシーを、ステップ 2 で作成したロールに関連付けます。

```
aws iam put-role-policy
  --role-name CWLtoSubscriptionFilterRole
  --policy-name Permissions-Policy-For-CWL-Subscription-filter
  --policy-document file://~/PermissionsForCWLSubscriptionFilter.json
```

4. 次のコマンドを入力して、サブスクリプションフィルターポリシーを更新します。

```
aws logs put-account-policy \
  --policy-name "CrossAccountStreamsExamplePolicy" \
  --policy-type "SUBSCRIPTION_FILTER_POLICY" \
  --policy-document
'{"DestinationArn":"arn:aws:logs:region:999999999999:destination:testDestination",
"FilterPattern": "${$.userIdentity.type = Root}", "Distribution": "Random"}' \
  --selection-criteria 'LogGroupName NOT IN ["LogGroupToExclude1",
"LogGroupToExclude2"]' \
  --scope "ALL"
```

ステップ 2: 既存の送信先アクセスポリシーを更新する

すべての送信者アカウントのサブスクリプションフィルターを更新した後、受信者アカウントの送信先アクセスポリシーを更新できます。

以下の例では、受信者アカウントは 999999999999、送信先は testDestination となっています。

この更新により、ID o-1234567890 を持つ組織に属するすべてのアカウントが、受信者アカウントにログを送信できるようになりました。サブスクリプションフィルターが作成されたアカウントのみが、実際に受信者アカウントにログを送信します。

受信者アカウントの送信先アクセスポリシーを更新して、権限の組織 ID の使用をスタートする方法

1. 受信者アカウントで、テキストエディタを使用して、以下の内容の ~/AccessPolicy.json ファイルを作成します。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": "*",
      "Action": [
        "logs:PutSubscriptionFilter",
        "logs:PutAccountPolicy"
      ],
      "Resource": "arn:aws:logs:us-east-1:999999999999:destination:testDestination",
      "Condition": {
        "StringEquals": {
          "aws:PrincipalOrgID": [
            "o-1234567890"
          ]
        }
      }
    }
  ]
}
```

2. 次のコマンドを入力して、先ほど作成したポリシーを既存の送信先にアタッチします。特定の AWS アカウント ID を一覧表示するアクセスポリシーではなく、組織 ID を持つアクセスポリシーを使用するように送信先を更新するには、forceパラメータを含めIDs。

⚠ Warning

にリストされている AWS サービスによって送信されたログを使用している場合は[AWS サービスからのログ記録を有効にする](#)、このステップを実行する前に、「」で説明されているように、まずすべての送信者アカウントのサブスクリプションフィルターを更新しておく必要があります[ステップ 1: サブスクリプションフィルターを更新する](#)。

```
aws logs put-destination-policy
\ --destination-name "testDestination"
\ --access-policy file://~/AccessPolicy.json
\ --force
```

Firehose を使用したクロスアカウント、クロスリージョン、アカウントレベルのサブスクリプション

複数のアカウントでログデータを共有するには、ログデータの送信者と受信者を確立する必要があります。

- [Log data sender (ログデータの送信者)] — 受取人から送信先情報を取得し、そのログイベントを特定の送信先に送信する準備が完了していることを CloudWatch Logs に通知します。このセクションの残りの手順では、ログデータ送信者は架空の AWS アカウント番号 111111111111 で表示されます。
- ログデータ受信者 — Amazon Kinesis Data Streams ストリームをカプセル化する送信先を設定し、受信者がログデータを受信したいことを CloudWatch Logs に知らせます。この後、受信者は自分の送信先に関する情報を送信者と共有します。このセクションの残りの手順では、ログデータ受信者は架空の AWS アカウント番号 222222222222 で表示されます。

このセクションの例では、Amazon S3 ストレージで Firehose 配信ストリームを使用しています。異なる設定で Firehose 配信ストリームを設定することもできます。詳細については、「[Creating a Firehose Delivery Stream](#)」を参照してください。

Note

ロググループと送信先は同じ AWS リージョンにある必要があります。ただし、送信先が指す AWS リソースは、別のリージョンに配置することができます。

Note

[同じアカウント] と [クロスリージョン] 配信ストリームに対して Firehose サブスクリプションフィルターがサポートされています。

トピック

- [ステップ 1: Firehose 配信ストリームを作成する](#)
- [ステップ 2: 送信先を作成する](#)
- [ステップ 3: アカウントレベルのサブスクリプションフィルタポリシーを作成する](#)
- [ログイベントの送信の検証](#)
- [実行時の送信先のメンバーシップの変更](#)

ステップ 1: Firehose 配信ストリームを作成する

Important

以下の手順を実行する前に、Firehose が Amazon S3 バケットにアクセスできるように、アクセスポリシーを使用する必要があります。詳細については、「[Amazon Data Firehose 開発者ガイド](#)」の「Controlling Access」を参照してください。

このセクションのすべての手順 (ステップ 1) は、ログデータの受取人アカウントで行われます。

次のサンプルコマンドでは、米国東部 (バージニア北部) が使用されています。このリージョンを、デプロイに適したリージョンに置き換えます。

送信先として使用する Firehose 配信ストリームを作成するには

1. Amazon S3 バケットの作成

```
aws s3api create-bucket --bucket amzn-s3-demo-bucket --create-bucket-configuration
LocationConstraint=us-east-1
```

2. バケットにデータを配置するためのアクセス許可を Firehose に付与する IAM ロールを作成します。

- a. まず、テキストエディタを使用して、ファイル `~/TrustPolicyForFirehose.json` で信頼ポリシーを作成します。

```
{ "Statement": { "Effect": "Allow", "Principal": { "Service":
  "firehose.amazonaws.com" }, "Action": "sts:AssumeRole", "Condition":
  { "StringEquals": { "sts:ExternalId": "222222222222" } } } }
```

- b. 作成したばかりの信頼ポリシーファイルを指定して、IAM ロールを作成します。

```
aws iam create-role \
  --role-name FirehoseToS3Role \
  --assume-role-policy-document file://~/TrustPolicyForFirehose.json
```

- c. このコマンドの出力は、次のようになります。ロール名とロール ARN を書き留めます。

```
{
  "Role": {
    "Path": "/",
    "RoleName": "FirehoseToS3Role",
    "RoleId": "AROAR3BXASEKW7K635M53",
    "Arn": "arn:aws:iam::222222222222:role/FirehoseToS3Role",
    "CreateDate": "2021-02-02T07:53:10+00:00",
    "AssumeRolePolicyDocument": {
      "Statement": {
        "Effect": "Allow",
        "Principal": {
          "Service": "firehose.amazonaws.com"
        },
        "Action": "sts:AssumeRole",
        "Condition": {
          "StringEquals": {
            "sts:ExternalId": "222222222222"
          }
        }
      }
    }
  }
}
```

```
}
}
```

3. アクセス許可ポリシーを作成し、Firehose がアカウントで実行できるアクションを定義します。
 - a. まず、テキストエディタを使用して、~/PermissionsForFirehose.json という名前のファイルに以下のようなアクセス許可ポリシーを作成します。ユースケースによっては、このファイルにさらにアクセス権限を追加する必要がある場合があります。

```
{
  "Statement": [{
    "Effect": "Allow",
    "Action": [
      "s3:PutObject",
      "s3:PutObjectAcl",
      "s3:ListBucket"
    ],
    "Resource": [
      "arn:aws:s3:::amzn-s3-demo-bucket",
      "arn:aws:s3:::amzn-s3-demo-bucket/*"
    ]
  }]
}
```

- b. 次のコマンドを入力して、先ほど作成したアクセス権限ポリシーを IAM ロールに関連付けます。

```
aws iam put-role-policy --role-name FirehoseToS3Role --policy-name
Permissions-Policy-For-Firehose-To-S3 --policy-document file:///~/
PermissionsForFirehose.json
```

4. 次のコマンドを入力して、Firehose 配信ストリームを作成します。*my-role-arn* および *amzn-s3-demo-bucket2-arn* をデプロイに適した値に置き換えます。

```
aws firehose create-delivery-stream \
  --delivery-stream-name 'my-delivery-stream' \
  --s3-destination-configuration \
  '{"RoleARN": "arn:aws:iam::222222222222:role/FirehoseToS3Role", "BucketARN":
  "arn:aws:s3:::amzn-s3-demo-bucket"}'
```

出力は次の例に類似したものになります:

```
{
  "DeliveryStreamARN": "arn:aws:firehose:us-east-1:222222222222:deliverystream/
my-delivery-stream"
}
```

ステップ 2: 送信先を作成する

Important

この手順のステップは、ログデータの受取人アカウントで行われます。

送信先が作成されると、CloudWatch Logs は受信者アカウントに代わってテストメッセージを宛先に送信します。サブスクリプションフィルターが後でアクティブになると、CloudWatch Logs はソースアカウントに代わってログイベントを宛先に送信します。

送信先を作成するには

1. [ステップ 1: Firehose 配信ストリームを作成する](#) で作成した Firehose ストリームがアクティブになるまで待ちます。次のコマンドを使用して、StreamDescription.StreamStatus プロパティを確認できます。

```
aws firehose describe-delivery-stream --delivery-stream-name "my-delivery-stream"
```

さらに、後の手順で使用する必要があるため、DeliveryStreamDescription.DeliveryStreamARN 値を書き留めます。このコマンドの出力例:

```
{
  "DeliveryStreamDescription": {
    "DeliveryStreamName": "my-delivery-stream",
    "DeliveryStreamARN": "arn:aws:firehose:us-
east-1:222222222222:deliverystream/my-delivery-stream",
    "DeliveryStreamStatus": "ACTIVE",
    "DeliveryStreamEncryptionConfiguration": {
      "Status": "DISABLED"
    },
    "DeliveryStreamType": "DirectPut",
    "VersionId": "1",
```

```

    "CreateTimestamp": "2021-02-01T23:59:15.567000-08:00",
    "Destinations": [
      {
        "DestinationId": "destinationId-000000000001",
        "S3DestinationDescription": {
          "RoleARN": "arn:aws:iam::222222222222:role/FirehoseToS3Role",
          "BucketARN": "arn:aws:s3:::amzn-s3-demo-bucket",
          "BufferingHints": {
            "SizeInMBs": 5,
            "IntervalInSeconds": 300
          },
          "CompressionFormat": "UNCOMPRESSED",
          "EncryptionConfiguration": {
            "NoEncryptionConfig": "NoEncryption"
          },
          "CloudWatchLoggingOptions": {
            "Enabled": false
          }
        },
        "ExtendedS3DestinationDescription": {
          "RoleARN": "arn:aws:iam::222222222222:role/FirehoseToS3Role",
          "BucketARN": "arn:aws:s3:::amzn-s3-demo-bucket",
          "BufferingHints": {
            "SizeInMBs": 5,
            "IntervalInSeconds": 300
          },
          "CompressionFormat": "UNCOMPRESSED",
          "EncryptionConfiguration": {
            "NoEncryptionConfig": "NoEncryption"
          },
          "CloudWatchLoggingOptions": {
            "Enabled": false
          },
          "S3BackupMode": "Disabled"
        }
      }
    ],
    "HasMoreDestinations": false
  }
}

```

配信ストリームがアクティブ状態が表示されるまでに 1~2 分かかる場合があります。

2. 配信ストリームがアクティブになったら、Firehose ストリームにデータを置くためのアクセス許可を CloudWatch Logs に付与する IAM ロールを作成します。まず、ファイル `~/TrustPolicyForCWL.json` で信頼ポリシーを作成する必要があります。テキストエディタを使用してこのポリシーを作成します。CloudWatch Logs エンドポイントの詳細については、[Amazon CloudWatch Logs エンドポイントおよびクォータ](#)を参照してください。

このポリシーには、「混乱した代理」のセキュリティ上の問題を防止するための `sourceAccountId` が指定された `aws:SourceArn` グローバル条件コンテキストキーが含まれています。最初の呼び出しでソースアカウント ID が不明な場合は、送信元 ARN フィールドに送信先 ARN を指定することをお勧めします。後続の呼び出しでは、送信元 ARN を、最初の呼び出しで取得した実際の送信元 ARN に設定する必要があります。詳細については、「[混乱した代理の防止](#)」を参照してください。

```
{
  "Statement": {
    "Effect": "Allow",
    "Principal": {
      "Service": "logs.amazonaws.com"
    },
    "Action": "sts:AssumeRole",
    "Condition": {
      "StringLike": {
        "aws:SourceArn": [
          "arn:aws:logs:region:sourceAccountId:*",
          "arn:aws:logs:region:recipientAccountId:*"
        ]
      }
    }
  }
}
```

3. `aws iam create-role` コマンドを使用して、作成した信頼ポリシーファイルを指定して IAM ロールを作成します。

```
aws iam create-role \
  --role-name CWLtoKinesisFirehoseRole \
  --assume-role-policy-document file://~/TrustPolicyForCWL.json
```

以下は出力例です。後のステップで使用する必要があるため、`Role.Arn` の戻り値を書き留めます。

```
{
  "Role": {
    "Path": "/",
    "RoleName": "CWLtoKinesisFirehoseRole",
    "RoleId": "AROAR3BXASEKYJYWF243H",
    "Arn": "arn:aws:iam::222222222222:role/CWLtoKinesisFirehoseRole",
    "CreateDate": "2023-02-02T08:10:43+00:00",
    "AssumeRolePolicyDocument": {
      "Statement": {
        "Effect": "Allow",
        "Principal": {
          "Service": "logs.amazonaws.com"
        },
        "Action": "sts:AssumeRole",
        "Condition": {
          "StringLike": {
            "aws:SourceArn": [
              "arn:aws:logs:region:sourceAccountId:*",
              "arn:aws:logs:region:recipientAccountId:"
            ]
          }
        }
      }
    }
  }
}
```

- CloudWatch Logs がアカウントで実行できるアクションを定義するアクセス許可ポリシーを作成します。まず、テキストエディタを使用してファイル `~/PermissionsForCWL.json` でアクセス許可ポリシーを作成します。

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["firehose:*"],
      "Resource": ["arn:aws:firehose:region:222222222222:*"]
    }
  ]
}
```

- 次のコマンドを入力して、アクセス権限ポリシーをロールに関連付けます。


```
aws iam put-role-policy --role-name CWLtoKinesisFirehoseRole --policy-name
Permissions-Policy-For-CWL --policy-document file://~/PermissionsForCWL.json
```

6. Firehose 配信ストリームがアクティブ状態になり、IAM ロールが作成されたら、CloudWatch Logs の送信先を作成できます。
 - a. このステップでは、アクセスポリシーと送信先は関連付けられません。送信先の作成を完了するには 2 つのステップを行う必要がありますが、このステップはその最初のステップです。後のステップでこれを `destination.arn` として使用するため、ペイロードで返される新しい宛先の ARN を書き留めます。

```
aws logs put-destination \  
  
    --destination-name "testFirehoseDestination" \  
    --target-arn "arn:aws:firehose:us-east-1:222222222222:deliverystream/my-  
delivery-stream" \  
    --role-arn "arn:aws:iam::222222222222:role/CWLtoKinesisFirehoseRole"  
  
{  
    "destination": {  
        "destinationName": "testFirehoseDestination",  
        "targetArn": "arn:aws:firehose:us-east-1:222222222222:deliverystream/  
my-delivery-stream",  
        "roleArn": "arn:aws:iam::222222222222:role/CWLtoKinesisFirehoseRole",  
        "arn": "arn:aws:logs:us-  
east-1:222222222222:destination:testFirehoseDestination"}  
}
```

- b. 前のステップが完了したら、ログデータ受取人アカウント (222222222222) で、アクセスポリシーを送信先に関連付けます。このポリシーにより、ログデータの送信者アカウント (111111111111) に対し、ログデータの受信者アカウント (222222222222) にある送信先にアクセスすることを許可します。テキストエディタを使用して、このポリシーを ~/AccessPolicy.json ファイルに配置できます。

JSON

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

```

    "Sid" : "",
    "Effect" : "Allow",
    "Principal" : {
        "AWS" : "111111111111"
    },
    "Action" : ["logs:PutSubscriptionFilter","logs:PutAccountPolicy"],
    "Resource" : "arn:aws:logs:us-
east-1:222222222222:destination:testFirehoseDestination"
  }
]
}

```

- c. これにより、誰が送信先に書き込むことができるかを定義するポリシーが作成されます。このポリシーでは、送信先にアクセスするための `logs:PutSubscriptionFilter` および `logs:PutAccountPolicy` アクションを指定する必要があります。クロスアカウントのユーザーは、`PutSubscriptionFilter` および `PutAccountPolicy` アクションを使用して送信先にログイベントを送信します。

```

aws logs put-destination-policy \
  --destination-name "testFirehoseDestination" \
  --access-policy file://~/AccessPolicy.json

```

ステップ 3: アカウントレベルのサブスクリプションフィルタポリシーを作成する

送信側のアカウント (この例では 111111111111) に切り替えます。次に、送信側のアカウントにアカウントレベルのサブスクリプションフィルタポリシーを作成します。この例では、フィルターにより、2 つを除くすべてのロググループに含まれる文字列 `ERROR` を含むすべてのログイベントが、以前に作成した送信先に配信されます。

```

aws logs put-account-policy \
  --policy-name "CrossAccountFirehoseExamplePolicy" \
  --policy-type "SUBSCRIPTION_FILTER_POLICY" \
  --policy-document '{"DestinationArn":"arn:aws:logs:us-
east-1:222222222222:destination:testFirehoseDestination", "FilterPattern":
"{$.userIdentity.type = AssumedRole}", "Distribution": "Random"}' \
  --selection-criteria 'LogGroupName NOT IN ["LogGroupToExclude1",
"LogGroupToExclude2"]' \
  --scope "ALL"

```

送信アカウントのロググループと送信先は同じ AWS リージョンに存在する必要があります。ただし、送信先は、別のリージョンにある Firehose ストリームなどの AWS リソースを指すことができます。

ログイベントの送信の検証

サブスクリプションフィルターを作成すると、CloudWatch Logs から、フィルタパターンと選択基準に一致するすべての受信ログイベントが Kinesis Data Firehose 配信ストリームに転送されます。データは、Firehose 配信ストリームに設定されている時間バッファ間隔に基づいて、Amazon S3 バケットに順次表示されます。十分な時間が経過すると、Amazon S3 バケットをチェックしてデータを確認できます。バケットを確認するには、次のコマンドを入力します。

```
aws s3api list-objects --bucket 'amzn-s3-demo-bucket'
```

そのコマンドの出力は、次のようになります。

```
{
  "Contents": [
    {
      "Key": "2021/02/02/08/my-delivery-
stream-1-2021-02-02-08-55-24-5e6dc317-071b-45ba-a9d3-4805ba39c2ba",
      "LastModified": "2023-02-02T09:00:26+00:00",
      "ETag": "\"EXAMPLEa817fb88fc770b81c8f990d\"",
      "Size": 198,
      "StorageClass": "STANDARD",
      "Owner": {
        "DisplayName": "firehose+2test",
        "ID": "EXAMPLE27fd05889c665d2636218451970ef79400e3d2aecca3adb1930042e0"
      }
    }
  ]
}
```

その後、次のコマンドを入力して、バケットから特定のオブジェクトを取得できます。key の値を、前のコマンドで検索した値に置き換えます。

```
aws s3api get-object --bucket 'amzn-s3-demo-bucket' --key '2021/02/02/08/my-delivery-
stream-1-2021-02-02-08-55-24-5e6dc317-071b-45ba-a9d3-4805ba39c2ba' testfile.gz
```

Simple Storage Service (Amazon S3) オブジェクトのデータは、gzip 形式で圧縮されます。raw データは、コマンドラインから次のコマンドを使用して調べることができます。

Linux:

```
zcat testfile.gz
```

macOS:

```
zcat <testfile.gz
```

実行時の送信先のメンバーシップの変更

所有している送信先からログ送信者を追加または削除しなければならない状況が発生することがあります。新しいアクセスポリシーが関連付けられている送信先に対して PutDestinationPolicy および PutAccountPolicy アクションを使用できます。次の例では、先ほど追加したアカウント 111111111111 がログデータの送信を停止し、アカウント 333333333333 が有効になります。

1. 現在 testDestination という送信先に関連付けられているポリシーをフェッチし、AccessPolicy を書き留めておきます。

```
aws logs describe-destinations \
  --destination-name-prefix "testFirehoseDestination"
```

返されたデータは、次のように表示される場合があります。

```
{
  "destinations": [
    {
      "destinationName": "testFirehoseDestination",
      "targetArn": "arn:aws:firehose:us-east-1:222222222222:deliverystream/
my-delivery-stream",
      "roleArn": "arn:aws:iam:: 222222222222:role/CWLtoKinesisFirehoseRole",
      "accessPolicy": "{\n  \"Version\" : \"2012-10-17\",\n  \"Statement\n\" : [\n    {\n      \"Sid\" : \"\",\n      \"Effect\" : \"Allow\",\n      \"Principal\" : {\n        \"AWS\" : \"111111111111\"\n      },\n      \"Action\n\" : \"logs:PutSubscriptionFilter\",\n      \"Resource\" : \"arn:aws:logs:us-
east-1:222222222222:destination:testFirehoseDestination\"\n    }\n  ]\n}\n",
      "arn": "arn:aws:logs:us-east-1:
222222222222:destination:testFirehoseDestination",
      "creationTime": 1612256124430
    }
  ]
}
```

```
}
```

2. アカウント 111111111111 が停止したとアカウント 333333333333 が有効になったことを反映させるためにポリシーを更新します。このポリシーを `~/NewAccessPolicy.json` ファイルに配置します。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "AWS": "333333333333"
      },
      "Action": ["logs:PutSubscriptionFilter", "logs:PutAccountPolicy"],
      "Resource": "arn:aws:logs:us-east-1:222222222222:destination:testFirehoseDestination"
    }
  ]
}
```

3. 次のコマンドを使用して、`NewAccessPolicy.json` ファイルで定義されたポリシーを送信先に関連付けます。

```
aws logs put-destination-policy \
  --destination-name "testFirehoseDestination" \
  --access-policy file:///~/NewAccessPolicy.json
```

これにより、最終的には、アカウント ID 111111111111 からのログイベントが無効になります。アカウント ID 333333333333 の所有者がサブスクリプションフィルターを作成すると、すぐに 333333333333 からのログイベントが送信先に送信されるようになります。

混乱した代理の防止

混乱した代理問題は、アクションを実行する許可を持たないエンティティが、より特権のあるエンティティにアクションを実行するように強制できるセキュリティの問題です。では AWS、サービス

間のなりすましにより、混乱した代理問題が発生する可能性があります。サービス間でのなりすましは、1つのサービス (呼び出し元サービス) が、別のサービス (呼び出し対象サービス) を呼び出すときに発生する可能性があります。呼び出し元サービスは、本来ならアクセスすることが許可されるべきではない方法でその許可を使用して、別のお客様のリソースに対する処理を実行するように操作される場合があります。これを防ぐために、は、アカウント内のリソースへのアクセス権が付与されたサービスプリンシパルを持つすべてのサービスのデータを保護するのに役立つツール AWS を提供します。

リソースポリシーで

[aws:SourceArn](#)、[aws:SourceAccount](#)、[aws:SourceOrgID](#)、[aws:SourceOrgPaths](#) のグローバル条件コンテキストキーを使用し、別のサービスに付与するアクセス許可をそのリソースに制限することをお勧めします。1つのリソースだけをクロスサービスのアクセスに関連付ける場合は、[aws:SourceArn](#) を使用します。アカウント内の任意のリソースをクロスサービスの使用に関連付ける場合は、[aws:SourceAccount](#) を使用します。組織内の任意のアカウントの任意のリソースをクロスサービスの使用に関連付ける場合は、[aws:SourceOrgID](#) を使用します。AWS Organizations パス内の任意のアカウントのリソースをクロスサービスの使用に関連付ける場合は、[aws:SourceOrgPaths](#) を使用します。パスの使用と理解の詳細については、[AWS Organizations 「エンティティパスを理解する」](#) を参照してください。

混乱した代理問題から保護するための最も効果的な方法は、リソースの完全な ARN を指定して、[aws:SourceArn](#) グローバル条件コンテキストキーを使用することです。リソースの完全な ARN が不明な場合や、複数のリソースを指定する場合には、グローバルコンテキスト条件キー [aws:SourceArn](#) で、ARN の未知部分を示すためにワイルドカード文字 (*) を使用します。例えば、`arn:aws:servicename:*:123456789012:*`。

[aws:SourceArn](#) の値に Amazon S3 バケット ARN などのアカウント ID が含まれていない場合は、両方の [aws:SourceAccount](#) と [aws:SourceArn](#) を使用して、アクセス許可を制限する必要があります。

混乱した代理問題から保護するために、リソースベースポリシー内のリソースの組織 ID または組織パスを指定しながら、[aws:SourceOrgID](#) または [aws:SourceOrgPaths](#) のグローバル条件コンテキストキーを使用してください。[aws:SourceOrgID](#) または [aws:SourceOrgPaths](#) キーを含むポリシーには正しいアカウントが自動的に組み込まれるため、組織のアカウントを追加、削除、移動する際には手動で更新する必要はありません。

で Amazon Kinesis Data Streams と Firehose にデータを書き込むための CloudWatch Logs へのアクセスを許可[ステップ 1: 送信先を作成する](#)し、グローバル[aws:SourceArn](#)条件コンテキストキーを使用して混乱した代理問題を防ぐ方法[ステップ 2: 送信先を作成する](#)を示すポリシー。

ログの再帰防止

サブスクリプションフィルターの使用によって無限のログ再帰が引き起こされるリスクがあります。これを防止しなければ CloudWatch Logs と送信先の両方で取り込み料金が大幅に増加する可能性があります。これは、サブスクリプション配信ワークフローの結果としてログイベントを受信するロググループにサブスクリプションフィルターが関連付けられている場合に発生する可能性があります。ロググループに取り込まれたログは宛先に配信され、ロググループがより多くのログを取り込むようになり、その後再び宛先に転送され、再帰ループが作成されます。

例えば、Amazon S3 にログイベントを配信する送信先を Firehose とするサブスクリプションフィルターがあるとします。さらに、Amazon S3 に配信された新しいイベントを処理し、一部のログ自体を生成する Lambda 関数もあります。サブスクリプションフィルターが Lambda 関数のロググループに適用されると、関数によって生成されたログイベントは送信先で Firehose と Amazon S3 に転送され、その後再び関数が呼び出されます。これにより、より多くのログが生成されて Firehose と Amazon S3 に転送され、関数が次々と呼び出されます。その結果、無限ループが発生し、ログの取り込み、Firehose、Amazon S3 で利用料金の不測の増加につながります。

Lambda 関数が CloudWatch Logs でフローログが有効になっている VPC にアタッチされている場合、VPC のロググループもログの再帰を引き起こす可能性があります。

サブスクリプション配信ワークフローの一部であるロググループには、サブスクリプションフィルターを適用しないことをお勧めします。アカウントレベルのサブスクリプションフィルターでは、PutAccountPolicy API の `selectionCriteria` パラメータを使用して、これらのロググループをポリシーから除外します。

ロググループを除外する場合は、ログを生成し、サブスクリプション配信ワークフローの一部である可能性がある以下の AWS サービスを検討してください。

- Amazon EC2 と Fargate
- Lambda
- AWS Step Functions
- CloudWatch Logs で有効になっている Amazon VPC フローログ

Note

Lambda 送信先のロググループによって生成されたログイベントは、アカウントレベルのサブスクリプションフィルターポリシーの Lambda 関数には転送されません。この場合、アカ

ウントサブスクリプションポリシーでは、`selectionCriteria` を使用する送信先 Lambda 関数のロググループを除外する必要はありません。

メトリクスフィルター、サブスクリプションフィルター、フィルターロギイベント、およびライブテールのフィルターパターン構文

Note

Amazon CloudWatch Logs Insights のクエリ言語でロググループをクエリする方法については、[CloudWatch Logs Insights 言語のクエリ構文](#) を参照してください。

CloudWatch Logs では、[メトリクスフィルター](#)を使用してログデータを実用的なメトリクスに変換し、[サブスクリプションフィルター](#)を使用してロギイベントを他の AWS サービスにルーティングし、[ロギイベントをフィルタリング](#)してロギイベントを検索し、[Live Tail](#) を使用してログを取り込み時にインタラクティブにリアルタイムで表示できます。

フィルターパターンは、メトリクスフィルター、サブスクリプションフィルター、ロギイベント、Live Tail がロギイベントの語句を照合するために使用する構文を構成します。語句には、単語、正確なフレーズ、または数値を指定できます。正規表現 (regex) は、スタンドアロンのフィルターパターンの作成に使用するか、JSON やスペース区切りのフィルターパターンに組み込むことができます。

照合する語句を使用してフィルターパターンを作成します。フィルターパターンは、定義する語句を含むロギイベントのみを返します。CloudWatch コンソールでフィルターパターンをテストできます。

トピック

- [サポートされている正規表現 \(regex\) 構文](#)
- [フィルターパターンを使用した正規表現 \(regex\) の語句の一致](#)
- [フィルターパターンを使用した非構造化ロギイベントの語句の一致](#)
- [フィルターパターンを使用した JSON ロギイベントの語句の一致](#)
- [フィルターパターンを使用したスペース区切りのロギイベントでの語句の一致](#)

サポートされている正規表現 (regex) 構文

サポートされている regex 構文

regex を使用してログデータを検索とフィルタリングする際は、その式を % で囲む必要があります。

regex を使ったフィルターパターンには、次のものしか含めることができません

- 英数字 - 英数字とは、文字 (A～Z または a～z) または数字 (0～9) を指します。
- サポートされている記号文字 - これには、「_」、「#」、「=」、「@」、「/」、「;」、「,」、「:」、「-」が含まれます。たとえば、「!」はサポートされていないため、%something!% は拒否されます。
- サポートされている演算子 - これには、「^」、「\$」、「?」、「[」、「]」、「{」、「}」、「|」、「\」、「*」、「+」、「.」が含まれます。

(と) 演算子はサポートされていません。括弧を使用してサブパターンを定義することはできません。

マルチバイト文字はサポートされていません。

Note

クォータ

メトリクスフィルターまたはサブスクリプションフィルターを作成するとき、ロググループごとに regex を含むフィルターパターンが最大 5 つあります。

メトリクスフィルターとサブスクリプションフィルターの区切りまたは JSON フィルターパターンを作成するとき、またはログイベントまたはライブテールをフィルタリングするとき、フィルターパターンごとに 2 つの regex の制限があります。

[サポートされている演算子の使い方]

- ^: 文字列の先頭の一致。たとえば、%^[hc]at% は「hat」と「cat」を一致とみなしますが、文字列の先頭でのみ適用されます。
- \$: 文字列の末尾の一致。たとえば、 %[hc]at\$% は「hat」と「cat」を一致とみなしますが、文字列の末尾でのみ適用されます。
- ?: 前述の用語の出現を 0 回または 1 回一致させます。たとえば、%colou?r% は「color」と「colour」を一致とみなします。

- `[]`: 文字クラスを定義します。括弧内の文字リストまたは文字範囲との一致。たとえば、`%[abc]%` は「a」、「b」、「c」を一致とみなします。`%[a-z]%` は「a」から「z」までのすべての小文字を一致とみなします。`%[abcx-z]%` は「a」、「b」、「c」、「x」、「y」、「z」を一致とみなします。
- `{m, n}`: `m` 以上の前の語句と一致し、`n` 回を超えることはありません。たとえば、`%a{3,5}%` は「aaa」、「aaaa」、「aaaaa」のみを一致とみなします。

Note

最小値または最大値を定義しない場合、`m` と `n` のいずれかを省略できます。

- `|`: 垂直バーのどちら側の語句と一致するブール値「Or」。例えば、次のようになります。
- `%gray|ey%` は「gray」または「grey」と一致できます
- `%^starting|^initializing|^shutting down%` は「開始中...」、「初期化中...」、または「シャットダウン中」と一致できますが、「スキッピング初期化中...」とは一致しません。
- `%abcc|ab[^c]$` は「abcc ...」と「aba ...」に一致できますが、「aac ...」には一致しません
- `\`: 演算子の特殊な意味ではなく、文字通りの意味を使用できるようにするエスケープ文字。たとえば、「`[a]`」、「`[b]`」、「`[7]`」、「`[@]`」、「`[]`」、「`[]`」などのように、括弧がエスケープされるため、`%\[.\]%` は「`[`」と「`]`」で囲まれたすべての 1 文字を一致とみなします。

Note

`%10\.10\.0\.1%` は、IP アドレス 10.10.0.1 を一致とみなす regex を作成する正しい方法です。

- `*`: 前の語句のインスタンスが 0 以上一致。たとえば、`%ab*c%` は「ac」、「abc」、「abbbc」と一致できます。`%ab[0-9]*%` は「ab」、「ab0」、「ab129」を一致とみなします。
- `+`: 前の語句のインスタンスが 1 以上一致。たとえば、`%ab+c%` は「abc」、「abbc」、「abbbc」を一致とみなしますが、「ac」を一致とみなしません。
- `.`: すべての 1 文字と一致します。たとえば、「hat」、「cat」、「bat」、「4at」、「#at」、「at」（先頭にスペース）を含め、`%.at%` は「at」で終わるすべての 3 文字の文字列を一致とみなします。

Note

IP アドレスと一致させる regex を作成するとき、. 演算子からエスケープすることが重要です。たとえば、%10.10.0.1% は「10010,051」を一致とみなしますが、これは表現の本来の用途とは異なる場合があります。

- \d、\D: 数字または数字以外の文字を一致とみなします。たとえば、%\d% は %[0-9]% と同等であり、%\D% は %[^0-9]% と同等です。

Note

大文字の演算子は、対応する小文字の逆を表します。

- \s、\S: 空白文字または非空白文字を一致とみなします。

Note

大文字の演算子は、対応する小文字の逆を表します。空白文字にはタブ (\t)、スペース ()、改行 (\n)文字が含まれます。

- \w、\W: 英数字または非英数字と一致します。たとえば、%\w% は %[a-zA-Z_0-9]% と同等であり、%\W% は %[^a-zA-Z_0-9]% と同等です。

Note

大文字の演算子は、対応する小文字の逆を表します。

- \xhh: 2 桁の 16 進文字の ASCII マッピングと一致します。 \x は、次の文字が ASCII の 16 進値を表すことを示すエスケープシーケンスです。hh は、ASCII 表の文字を指す 2 つの 16 進数字(0 ~ 9 と A ~ F)を指定します。

Note

\xhh を使用してフィルターパターンでサポートされていない記号文字を一致とみなすことができます。たとえば、%\x3A% は : を一致とみなし、%\x28% は (を一致とみなします。

フィルターパターンを使用した正規表現 (regex) の語句の一致

regex を使用した語句の一致

% (regex パターン前後のパーセント記号) で囲まれた regex パターンを使用し、ログイベントの語句を一致とみなすことができます。次のコードスニペットでは、[許可された]キーワードで構成されているすべてのログイベントを返すフィルターパターンの例が示されています。

サポートされている正規表現のリストについては、[「サポートされている正規表現」](#)を参照してください。

```
%AUTHORIZED%
```

このフィルターパターンは、次のようなログイベントメッセージを返します。

- [ERROR 401] UNAUTHORIZED REQUEST
- [SUCCESS 200] AUTHORIZED REQUEST

フィルターパターンを使用した非構造化ログイベントの語句の一致

非構造化ログイベントの語句の一致

次の例には、フィルターパターンを使用して非構造化ログイベントで語句をマッチさせる方法について示すコードスニペットが含まれています。

Note

フィルターパターンでは大文字と小文字が区別されます。英数字以外の文字を含む正確なフレーズと語句を、二重引用符 ([""]) で囲みます。

Example: Match a single term

次のコードスニペットは、メッセージに [ERROR] という単語が含まれるすべてのログイベントを返す単一の語句のフィルターパターンの例を示しています。

```
ERROR
```

このフィルターパターンは、次のようなログイベントメッセージを一致とみなします。

- [ERROR 400] BAD REQUEST
- [ERROR 401] UNAUTHORIZED REQUEST
- [ERROR 419] MISSING ARGUMENTS
- [ERROR 420] INVALID ARGUMENTS

Example: Match multiple terms

次のコードスニペットは、メッセージに ERROR と ARGUMENTS という単語が含まれるすべてのログイベントを返す複数の語句のフィルターパターンの例を示しています。

```
ERROR ARGUMENTS
```

フィルターは、次のようなログイベントメッセージを返します。

- [ERROR 419] MISSING ARGUMENTS
- [ERROR 420] INVALID ARGUMENTS

次のログイベントメッセージにはフィルターパターンで指定された語句が両方とも含まれないため、このフィルターパターンでは返されません。

- [ERROR 400] BAD REQUEST
- [ERROR 401] UNAUTHORIZED REQUEST

Example: Match optional terms

パターン一致を使用し、オプション語句を含むログイベントを返すフィルターパターンを作成できます。照合する語句の前に疑問符 (「?」) を配置します。次のコードスニペットは、メッセージに ERROR または ARGUMENTS という単語が含まれるすべてのログイベントを返すフィルターパターンの例を示しています。

```
?ERROR ?ARGUMENTS
```

このフィルターパターンは、次のようなログイベントメッセージを一致とみなします。

- [ERROR 400] BAD REQUEST
- [ERROR 401] UNAUTHORIZED REQUEST
- [ERROR 419] MISSING ARGUMENTS
- [ERROR 420] INVALID ARGUMENTS

 Note

疑問符 (「?」) を他のフィルターパターン (「含む」や「除外」の条件など) と組み合わせることはできません。「?」を他のフィルターパターンと組み合わせると、すべての疑問符は無視されます。

例えば、次のフィルターパターンは REQUEST という単語を含むすべてのイベントにマッチしますが、疑問符 (「?」) は無視され、何ら影響力を持ちません。

```
?ERROR ?ARGUMENTS REQUEST
```

ログイベントのマッチ

- [INFO] REQUEST FAILED
- [WARN] UNAUTHORIZED REQUEST
- [ERROR] 400 BAD REQUEST

Example: Match exact phrases

次のコードスニペットは、メッセージに INTERNAL SERVER ERROR という正確なフレーズが含まれるログイベントを返すフィルターパターンの例を示しています。

```
"INTERNAL SERVER ERROR"
```

このフィルターパターンは、次のログイベントメッセージを返します

- [ERROR 500] INTERNAL SERVER ERROR

Example: Include and exclude terms

メッセージにいくつかの語句が含まれるログイベントを返し、他の語句が除外されるフィルターパターンを作成できます。除外する語句の前にマイナス記号 ([「-」]) を配置します。次のコードスニペットは、メッセージに ERROR が含まれるログイベントを返し、ARGUMENTS という語句が除外されるフィルターパターンの例を示しています。

```
ERROR -ARGUMENTS
```

このフィルターパターンは、次のようなログイベントメッセージを返します。

- [ERROR 400] BAD REQUEST
- [ERROR 401] UNAUTHORIZED REQUEST

次のログイベントメッセージには [引数] という単語が含まれているため、このフィルターパターンでは返されません。

- [ERROR 419] MISSING ARGUMENTS
- [ERROR 420] INVALID ARGUMENTS

Example: Match everything

二重引用符で囲むことで、ログイベント内の完全一致を照合することができます。次のコードスニペットは、すべてのログイベントを返すフィルターパターンの例を示しています。

```
" "
```


フィルターパターンを使用した JSON ログイベントの語句の一致

JSON ログイベントのフィルターパターンの式

次の内容では、文字列と数値を含む JSON 語句と一致するフィルターパターンの構文を式する方法について説明します。

Writing filter patterns that match strings

JSON ログイベントで文字列とマッチさせるフィルターパターンを作成できます。次のコードスニペットには、文字列ベースのフィルターパターンの構文例が示されています。

```
{ PropertySelector EqualityOperator String }
```

フィルターパターンを中括弧(「{」)で囲みます。文字列ベースのフィルターパターンには、次の部分が含まれている必要があります。

- [Property selector] (プロパティセレクタ)

ドル記号の後にピリオド (「\$.」) が付いたプロパティセレクタをオフに設定します。プロパティセレクタは英数字の文字列であり、ハイフン (「-」) およびアンダースコア (「_」) をサポートします。文字列は科学表記をサポートしていません。プロパティセレクタは、JSON ログイベントの値ノードを指します。値ノードには、文字列または数値を指定できます。プロパティセレクタの後に配列を配置します。配列内の要素は 0 から始まる番号付けシステムに従います。つまり、配列の最初の要素は要素 0、2 番目の要素は要素 1 というようになります。要素を角カッコ (「[]」) で囲みます。プロパティセレクターが配列またはオブジェクトを指定している場合、フィルターパターンはログ形式を一致とみなしません。JSON プロパティにピリオド (".") が含まれている場合は、そのプロパティを選択するためにブラケット表記を使用できます。

Note

[ワイルドカードセレクター]

JSON ワイルドカードを使用し、任意の配列要素または JSON オブジェクトフィールドを選択できます。

クォータ

プロパティセクターでは 1 つのワイルドカードセクターしか使用できません。

- 等値演算子

等しい (「=」) または等しくない (「!=」) の記号のいずれかを使用して、等価演算子を区切ります。等値演算子は、ブール値 (true または false) を返します。

- 文字列

文字列は、二重引用符 ("") で囲むことができます。英数字とアンダースコア記号以外の種類を含む文字列は、二重引用符で囲む必要があります。アスタリスク (「*」) をワイルドカードとして使用して、テキストを照合します。

 Note

JSON ログイベントの語句と一致するフィルターパターンを作成するとき、任意の条件付き正規表現を使用できます。サポートされている正規表現のリストについては、[「サポートされている正規表現」](#)を参照してください。

次のコードスニペットには、文字列を持った JSON 語句とマッチさせるためにフィルターパターンをフォーマットする方法を示すフィルターパターンの例が含まれています。

```
{ $.eventType = "UpdateTrail" }
```

Writing filter patterns that match numeric values

JSON ログイベントの数値と一致するフィルターパターンを作成できます。次のコードスニペットには、数値とマッチさせるフィルターパターンの構文例が示されています。

```
{ PropertySelector NumericOperator Number }
```

フィルターパターンを中括弧 (「{ }」) で囲みます。数値と一致するフィルターパターンには、次の部分が含まれている必要があります。

- [Property selector] (プロパティセクター)

ドル記号の後にピリオド (「\$.」) が付いたプロパティセレクトをオフに設定します。プロパティセレクトは英数字の文字列であり、ハイフン (「-」) およびアンダースコア (「_」) をサポートします。文字列は科学表記をサポートしていません。プロパティセレクトは、JSON ログイベントの値ノードを指します。値ノードには、文字列または数値を指定できます。プロパティセレクトの後に配列を配置します。配列内の要素は 0 から始まる番号付けシステムに従います。つまり、配列の最初の要素は要素 0、2 番目の要素は要素 1 というようになります。要素を角カッコ (「[]」) で囲みます。プロパティセクターが配列またはオブジェクトを指定している場合、フィルターパターンはログ形式を一致とみなしません。JSON プロパティにピリオド (".") が含まれている場合は、そのプロパティを選択するためにブラケット表記を使用できます。

 Note

[ワイルドカードセクター]

JSON ワイルドカードを使用し、任意の配列要素または JSON オブジェクトフィールドを選択できます。

クォータ

プロパティセクターでは 1 つのワイルドカードセクターしか使用できません。

- 数値演算子

より大きい (「>」)、より小さい (「<」)、等しい (「=」)、等しくない (「!=」)、以上 (「>=」)、または以下 (「<=」) のいずれかの記号を使用して、数値演算子を区切ります。

- 数値

プラス (「+」) またはマイナス (「-」) 記号を含む整数を使用し、科学表記に従うことができます。アスタリスク (「*」) をワイルドカードとして使用して、数値を照合します。

次のコードスニペットには、JSON 語句を数値をマッチさせるためにフィルターパターンをフォーマットする方法を示す例が含まれています。

```
// Filter pattern with greater than symbol
{ $.bandwidth > 75 }
// Filter pattern with less than symbol
{ $.latency < 50 }
// Filter pattern with greater than or equal to symbol
{ $.refreshRate >= 60 }
// Filter pattern with less than or equal to symbol
```

```
{ $.responseTime <= 5 }  
// Filter pattern with equal sign  
{ $.errorCode = 400}  
// Filter pattern with not equal sign  
{ $.errorCode != 500 }  
// Filter pattern with scientific notation and plus symbol  
{ $.number[0] = 1e+3 }  
// Filter pattern with scientific notation and minus symbol  
{ $.number[0] != 1e-3 }
```

簡単な表現を使用して JSON ログイベントで語句の一致

次の例には、フィルターパターンが JSON ログイベントの語句をマッチさせる方法について示すコードスニペットが含まれています。

Note

例の JSON ログイベントを使用して例のフィルターパターンをテストする場合、例の JSON ログを 1 行で入力する必要があります。

[JSON ログイベント]

```
{  
  "eventType": "UpdateTrail",  
  "sourceIPAddress": "111.111.111.111",  
  "arrayKey": [  
    "value",  
    "another value"  
  ],  
  "objectList": [  
    {  
      "name": "a",  
      "id": 1  
    },  
    {  
      "name": "b",  
      "id": 2  
    }  
  ],  
  "SomeObject": null,  
}
```

```
"cluster.name": "c"
}
```

Example: Filter pattern that matches string values

このフィルターパターンは、プロパティ "eventType" の文字列 "UpdateTrail" と一致します。

```
{ $.eventType = "UpdateTrail" }
```

Example: Filter pattern that matches string values (IP address)

このフィルターパターンは、プレフィックス "123.123." が付いた数字が含まれていないため、ワイルドカードが含んでおり、プロパティ "sourceIPAddress" を一致とみなします。

```
{ $.sourceIPAddress != 123.123.* }
```

Example: Filter pattern that matches a specific array element with a string value

このフィルターパターンは、配列 "arrayKey" の要素 "value" と一致します。

```
{ $.arrayKey[0] = "value" }
```

Example: Filter pattern that matches a string using regex

このフィルターパターンは、プロパティ "eventType" の文字列 "Trail" と一致します。

```
{ $.eventType = %Trail% }
```

Example: Filter pattern that uses a wildcard to match values of any element in the array using regex

フィルターパターンには、配列 "arrayKey" の要素 "value" と一致する regex が含まれていません。

```
{ $.arrayKey[*] = %val.{2}% }
```

Example: Filter pattern that uses a wildcard to match values of any element with a specific prefix and subnet using regex (IP address)

このフィルターパターンには、プロパティ "sourceIPAddress" の要素 "111.111.111.111" と一致する regex が含まれています。

```
{ $.* = %111\.111\.111\.1[0-9]{1,2}% }
```

 Note

クォータ

プロパティセクターでは 1 つのワイルドカードセクターしか使用できません。

Example: Filter pattern that matches a JSON property with a period (.) in the key

```
{ $.['cluster.name'] = "c" }
```

Example: Filter pattern that matches JSON logs using IS

IS 変数で JSON ログのフィールドと一致するフィルターパターンを作成できます。IS 変数は、値 NULL、TRUE または FALSE を含むフィールドと一致させることができます。次のフィルターパターンでは、SomeObject の値が NULL の場合、JSON ログが返されます。

```
{ $.SomeObject IS NULL }
```

Example: Filter pattern that matches JSON logs using NOT EXISTS

NOT EXISTS 変数を使用してフィルターパターンを作成し、ログデータに特定フィールドを含まない JSON ログを返すことができます。次のフィルターパターンでは、NOT EXISTS を使用してフィールド SomeOtherObject を含まない JSON ログを返します。

```
{ $.SomeOtherObject NOT EXISTS }
```

Note

変数 IS NOT および EXISTS は現在サポートされていません。

複合式を使用した JSON オブジェクトの語句の一致

論理演算子 AND (「&&」) と OR (「||」) をフィルターパターンで使用し、2 つ以上の条件が真であるログイベントをマッチさせる複合式を作成できます。複合式では、カッコ (「()」) の使用と、次の標準的な演算順序 (() > && > ||) がサポートされます。次の例には、JSON オブジェクトの語句とマッチさせるための複合式を持ったフィルターパターンを使用する方法について示すコードスニペットが含まれています。

[JSON オブジェクト]

```
{
  "user": {
    "id": 1,
    "email": "John.Stiles@example.com"
  },
  "users": [
    {
      "id": 2,
      "email": "John.Doe@example.com"
    },
    {
      "id": 3,
      "email": "Jane.Doe@example.com"
    }
  ],
  "actions": [
```

```
    "GET",
    "PUT",
    "DELETE"
  ],
  "coordinates": [
    [0, 1, 2],
    [4, 5, 6],
    [7, 8, 9]
  ]
}
```

Example: Expression that matches using AND (&&)

このフィルターパターンには、"user" の "id" を 1 の数値とマッチし、文字列 "John.Doe@example.com" を使用して "users" 配列の最初の要素にある "email" とマッチする複合式が含まれています。

```
{ ($.user.id = 1) && ($.users[0].email = "John.Doe@example.com") }
```

Example: Expression that matches using OR (||)

このフィルターパターンには、"user" の "email" を文字列 "John.Stiles@example.com" とマッチさせる複合式が含まれています。

```
{ $.user.email = "John.Stiles@example.com" || $.coordinates[0][1] = "nonmatch" &&
  $.actions[2] = "nonmatch" }
```

Example: Expression that doesn't match using AND (&&)

このフィルターパターンには、式が "actions" の第 3 アクションとマッチしないため、一致が見つからない複合式が含まれています。


```
{ ($.user.email = "John.Stiles@example.com" || $.coordinates[0][1] = "nonmatch") &&
$.actions[2] = "nonmatch" }
```

Note

クォータ

プロパティセレクトではワイルドカードセレクトを 1 つしか使用できません。また、複合式を含むフィルターパターンでは最大 3 つのワイルドカードセレクトを使用することができます。

Example: Expression that doesn't match using OR (||)

このフィルターパターンには、式が "users" の最初のプロパティまたは "actions" の第 3 アクションと一致しないため、一致が見つからない複合式が含まれています。

```
{ ($.user.id = 2 && $.users[0].email = "nonmatch") || $.actions[2] = "GET" }
```

フィルターパターンを使用したスペース区切りのログイベントでの語句の一致

スペース区切りのログイベントのフィルターパターン式

フィルターパターンを作成し、スペース区切りのログイベントで語句を一致させることができます。次の内容では、スペース区切りのログイベントの例を示し、スペース区切りのログイベントで語句を一致するフィルターパターンの構文を式する方法について説明します。

Note

スペース区切りのログイベントで語句を一致するフィルターパターンを作成するとき、任意の条件付正規表現を使用できます。サポートされている正規表現のリストについては、[「サポートされている正規表現」](#)を参照してください。

Example: Space-delimited log event

次のコードスニペットは、7つのフィールド

(ip、user、username、timestamp、request、status_code、および bytes) を含むスペース区切りログイベントを示しています。

```
127.0.0.1 Prod frank [10/Oct/2000:13:25:15 -0700] "GET /index.html HTTP/1.0" 404 1534
```

Note

角かっこ (「[]」) と二重引用符 ("") の間の文字は、単一フィールドと見なされます。

Writing filter patterns that match terms in a space-delimited log event

スペース区切りのログイベントで語句を一致するフィルターパターンを作成するには、フィルターパターンを括弧(「[]」)で囲み、カンマ(「,」)で区切られた名前フィールドを指定します。次のフィルターパターンは7つのフィールドを解析します。

```
[ip=%127\.\0\.\0\.[1-9]%, user, username, timestamp, request =*.html*, status_code = 4*, bytes]
```

数値演算子 (>、<、=、!=、>=、<=) とアスタリスク (*) をワイルドカードまたは regex として使用し、フィルターパターン条件を指定できます。フィルターパターンの例では、ip は 127.0.0.1 ~ 127.0.0.9 の IP アドレス範囲に一致する regex を使用し、request は .html の値を抽出する必要があることを示すワイルドカードが含まれ、status_code は 4 で始まる値を抽出する必要があることを示すワイルドカードが含まれています。

スペース区切りログイベントで解析するフィールドの数がわからない場合は、省略記号 (...) を使用して名前のないフィールドを参照できます。省略記号を使用すると、必要な数のフィールドを

参照できます。次の例には、前のフィルターパターンの例で示された最初の 4 つの無名フィールドを表す省略記号を使用するフィルターパターンが示されています。

```
[..., request =*.html*, status_code = 4*, bytes]
```

論理演算子 AND (&&) と OR (||) を使用して複合式を作成することもできます。次のフィルターパターンには、status_code の値が 404 または 410 である必要があることを示す複合式が含まれています。

```
[ip, user, username, timestamp, request =*.html*, status_code = 404 || status_code = 410, bytes]
```

パターン マッチングを使用したスペース区切りのログイベントの語句との一致

パターンマッチングを使用し、特定の順序で語句を一致するスペース区切りのフィルターパターンを作成できます。インジケーターを使用して語句の順序を指定します。[w1] を使用して最初の語句を表し、次に [w2] などを使用して、その後の語句の順序を表します。語句の間にカンマ (「,」) を入力します。次の例には、スペース区切りのフィルターパターンでパターンマッチングを使用する方法について示すコードスニペットが含まれています。

Note

スペース区切りのログイベントで語句を一致するフィルターパターンを作成するとき、任意の条件付正規表現を使用できます。サポートされている正規表現のリストについては、[「サポートされている正規表現」](#)を参照してください。

[スペース区切りのログイベント]

```
INFO 09/25/2014 12:00:00 GET /service/resource/67 1200
INFO 09/25/2014 12:00:01 POST /service/resource/67/part/111 1310
WARNING 09/25/2014 12:00:02 Invalid user request
ERROR 09/25/2014 12:00:02 Failed to process request
```

Example: Match terms in order

次のスペース区切りのフィルターパターンは、ログイベントの最初の単語が[エラー]であるログイベントを返します。

```
[w1=ERROR, w2]
```

Note

パターンマッチングを使用するスペース区切りのフィルターパターンを作成するとき、語句の順序を指定した後に空白のインジケーターを含める必要があります。例えば、最初の単語が [エラー] であるログイベントを返すフィルターパターンを作成する場合、[w1] 語句の後に空白の [w2] インジケーターを含めます。

Example: Match terms with AND (&&) and OR (||)

論理演算子 AND (「&&」) と OR (「||」) を使用し、条件を含むスペース区切りのフィルターパターンを作成できます。次のフィルターパターンは、イベントの最初の単語が[エラー] または [警告] であるログイベントを返します。

```
[w1=ERROR || w1=WARNING, w2]
```

Example: Exclude terms from matches

1 つ以上の語句を除外するログイベントを返すスペース区切りのフィルターパターンを作成できます。除外する語句の前に等しくないの記号 (「!=」) を配置します。次のコードスニペットは、最初の単語が [エラー] と [警告] ではないログイベントを返すフィルターパターンの例を示しています。

```
[w1!=ERROR && w1!=WARNING, w2]
```

Example: Match the top level item in a resource URI

次のコードスニペットは、regex を使用してリソース URI の最上位の項目を一致するフィルターパターンの例を示しています。

```
[logLevel, date, time, method, url=%/service/resource/[0-9]+$, response_time]
```

Example: Match the child level item in a resource URI

次のコードスニペットは、regex を使用してリソース URI の子レベルの項目を一致するフィルターパターンの例を示しています。

```
[logLevel, date, time, method, url=%/service/resource/[0-9]+/part/[0-9]+$,  
response_time]
```

AWS サービスからのログ記録を有効にする

多くのサービスは CloudWatch Logs にのみログを発行しますが、一部の AWS サービスは Amazon Simple Storage Service または Amazon Data Firehose に直接ログを発行できます。ログの主な要件がストレージまたはこのいずれかのサービスでの処理である場合、追加の設定を行わずに、サービスで簡単にログを作成し、直接 Amazon S3 または Firehose に配信することができます。

Amazon S3 または Firehose に直接ログを発行しても、CloudWatch 配信料金が適用されます。Amazon S3 にログを送信すると、**AWS_REGION**-S3-Egress-BytesCost Explorer または請求書に料金が表示されます。Firehose にログを送信すると、**AWS_REGION**-FH-Egress-Bytes料金が表示されます。提供されたログの料金の詳細については、[Amazon CloudWatch 料金表](#)の「ログ」タブを参照してください。

一部の AWS サービスは、共通のインフラストラクチャを使用してログを送信します。これらのサービスからのロギングを有効にするには、特定の権限を持つユーザーとしてログインする必要があります。さらに、ログの送信を有効にする AWS には、 にアクセス許可を付与する必要があります。

これらのアクセス許可を必要とするサービスの場合、必要なアクセス許可には 2 つのバージョンがあります。これらの追加のアクセス許可を必要とするサービスは、表に [サポートあり [V1 アクセス許可]] および [サポートあり [V2 アクセス許可]] と表示されます。これらの必要な権限については、表の後のセクションを参照してください。

<integration-catalog>

<integration></integration>

<integration></integration>

<integration></integration>

<integration></integration>

<integration></integration>

<integration></integration>

<integration></integration>

<integration></integration>

<integration></integration>

<integration></integration>

<integration></integration>

<integration></integration>

<integration></integration>

<integration></integration>

<integration></integration>

[illegible]

[illegible]

追加のアクセス許可が必要なロギング [V1]

一部の AWS サービスでは、共通のインフラストラクチャを使用して CloudWatch Logs、Amazon S3、または Firehose にログを送信します。上記の表に示す AWS サービスがこれらの送信先にログを送信できるようにするには、特定のアクセス許可を持つユーザーとしてログインする必要があります。

さらに、ログの送信を有効にする AWS には、アクセス許可を に付与する必要があります。AWS は、ログの設定時にそれらのアクセス許可を自動的に作成できます。または、ログを設定する前に自分で作成することもできます。クロスアカウント配信では、アクセス許可ポリシーを手動で作成する必要があります。

自分または組織内の誰かがログの送信を最初にセットアップするときに、 で必要なアクセス許可とリソースポリシー AWS を自動的に設定することを選択した場合、ログの送信をセットアップするユーザーには、このセクションで後述する特定のアクセス許可が必要です。または、リソースポリシーをユーザーが独自に作成することもできます。そうすると、ログの送信を設定するユーザーがそれほど多くのアクセス許可を持つ必要がなくなります。

以下のトピックでは、これらの各送信先について詳しく説明します。

トピック

- [CloudWatch Logs に送信されたログ](#)
- [Amazon S3 に送信されたログ](#)
- [Firehose に送信されるログ](#)

CloudWatch Logs に送信されたログ

Important

以下のリストにあるログタイプを CloudWatch Logs に設定するようにセットアップすると、AWS がそのログを受け取るロググループに関連付けられたリソースポリシーを必要に応じて作成または変更します。詳細については、このセクションを続けてお読みください。

このセクションは、前のセクションの表に掲載されているタイプのログが CloudWatch Logs に送信される場合に適用されます。

ユーザーアクセス許可

これらのタイプのログの CloudWatch Logs への送信を初めてセットアップするには、以下のアクセス許可でアカウントにログインする必要があります。

- logs:CreateLogDelivery
- logs:PutResourcePolicy
- logs:DescribeResourcePolicies

- logs:DescribeLogGroups

Note

logs:DescribeLogGroups、logs:DescribeResourcePolicies、または logs:PutResourcePolicy アクセス許可を指定する場合は、1 つのロググループ名のみを指定するのではなく、その Resource 行の ARN で * ワイルドカードを使用するように設定してください。例: "Resource": "arn:aws:logs:us-east-1:111122223333:log-group:*"

これらのタイプのログのいずれかが CloudWatch Logs のロググループにすでに送信されている場合、これらの中の別のログを同じロググループに送信するためのセットアップに必要なのは logs:CreateLogDelivery アクセス許可のみです。

ロググループのリソースポリシー

ログが送信されているロググループには、特定のアクセス許可が含まれるリソースポリシーが必要です。現在ロググループにリソースポリシーがなく、ロギングをセットアップしているユーザーがロググループに対する logs:PutResourcePolicy、logs:DescribeResourcePolicies、および logs:DescribeLogGroups アクセス許可を持っているという場合は、CloudWatch Logs へのログの送信を開始するときに AWS が以下のポリシーを自動的に作成します。新しく作成されたサブスクリプションの場合、リソースポリシーはロググループレベルで設定され、最大サイズは 51,200 バイトです。既存のアカウントレベルのリソースポリシーがワイルドカードを介してアクセス許可を既に付与している場合、別のロググループレベルのポリシーは作成されません。特定のロググループの logGroup レベルのリソースポリシーを確認するには、--resource-arn パラメータをロググループ ARN に設定し、--policy-scope パラメータを に設定して describe-resource-policies コマンドを使用します RESOURCE。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AWSLogDeliveryWrite20150319",
      "Effect": "Allow",
      "Principal": {
        "Service": [
```

```

        "delivery.logs.amazonaws.com"
    ],
    },
    "Action": [
        "logs:CreateLogStream",
        "logs:PutLogEvents"
    ],
    "Resource": [
        "arn:aws:logs:us-east-1:111122223333:log-group:my-log-group:log-
stream:*"
    ],
    "Condition": {
        "StringEquals": {
            "aws:SourceAccount": [
                "0123456789"
            ]
        },
        "ArnLike": {
            "aws:SourceArn": [
                "arn:aws:logs:us-east-1:111122223333:*"
            ]
        }
    }
}
]
}

```

ロググループのリソースポリシーの制限は 51,200 バイトです。この制限に達すると、AWS は新しいアクセス許可を追加できません。そのためには、ポリシーを手動で変更して、logs:CreateLogStream および logs:PutLogEvents アクションに対する delivery.logs.amazonaws.com サービスプリンシパルのアクセス許可を付与する必要があります。お客様は、などのワイルドカードを含むロググループ名プレフィックスを使用し /aws/vendedlogs/*、このロググループ名を今後の配信作成に使用する必要があります。

JSON

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AWSLogDeliveryWrite20150319",

```

```

        "Effect": "Allow",
        "Principal": {
            "Service": [
                "delivery.logs.amazonaws.com"
            ]
        },
        "Action": [
            "logs:CreateLogStream",
            "logs:PutLogEvents"
        ],
        "Resource": [
            "arn:aws:logs:us-east-1:111122223333:log-group:my-log-group/aws/  

vendedlogs/*"
        ],
        "Condition": {
            "StringEquals": {
                "aws:SourceAccount": [
                    "0123456789"
                ]
            },
            "ArnLike": {
                "aws:SourceArn": [
                    "arn:aws:logs:us-east-1:111122223333:*"
                ]
            }
        }
    }
}
]
}

```

Amazon S3 に送信されたログ

Amazon S3 にログを送信するように設定すると、は、必要に応じてログを受信している S3 バケットに関連付けられたリソースポリシー AWS を作成または変更します。

Amazon S3 に直接発行されたログは、指定する既存のバケットに発行されます。指定したバケットで、5 分おきに 1 つ以上のログが作成されます。

ログを Amazon S3 バケットに初めて配信する場合、ログを配信するサービスはバケットの所有者を記録し、ログがこのアカウントに属するバケットにのみ配信されるようにします。その結果、Amazon S3 バケット所有者を変更するには、元のサービスでログサブスクリプションを再作成または更新する必要があります。

 Note

CloudFront は、S3 に vended log を送信する他のサービスとは異なるアクセス許可モデルを使用します。詳細については、「[標準ログ記録の設定およびログファイルへのアクセスに必要なアクセス許可](#)」を参照してください。

さらに、CloudFront アクセスログと別のログソースに同じ S3 バケットを使用する場合、CloudFront のバケットで ACL を有効にすると、このバケットを使用する他のすべてのログソースにもアクセス許可が付与されます。

 Important

Amazon S3 バケットにログを送信していて、バケットポリシーに NotAction または NotPrincipal 要素が含まれている場合、バケットにログ配信アクセス許可を自動的に追加し、ログサブスクリプションを作成すると失敗します。ログサブスクリプションを正常に作成するには、バケットポリシーにログ配信アクセス許可を手動で追加してから、ログサブスクリプションを作成する必要があります。詳細については、このセクションの指示を参照してください。

バケットにカスタマーマネージド AWS KMS キーを使用したサーバー側の暗号化がある場合は、カスタマーマネージドキーのキーポリシーも追加する必要があります。詳細については、「[Amazon S3](#)」を参照してください。

レプリケート先バケットで SSE-KMS とバケットキーが有効になっている場合、アタッチされたカスタマーマネージド KMS キーポリシーはすべてのリクエストで想定どおりに動作しなくなります。詳細については、「[Amazon S3 バケットキーを使用した SSE-KMS のコストの削減](#)」を参照してください。

カスタマーマネージド AWS KMS キーで販売ログと S3 暗号化を使用している場合は、バケットを設定するときに AWS KMS キー ID の代わりに完全修飾キー ARN を使用する必要があります。詳細については「[PUT バケットの暗号化](#)」を参照してください。

ユーザーアクセス許可

これらのタイプのログの Amazon S3 への送信を初めてセットアップするには、以下のアクセス許可でアカウントにログインする必要があります。

- logs:CreateLogDelivery
- S3:GetBucketPolicy

- S3:PutBucketPolicy

これらのタイプのログのいずれかが Amazon S3 バケットにすでに送信されている場合、これらの中の別のログを同じバケットに送信するためのセットアップに必要なのは logs:CreateLogDelivery アクセス許可のみです。

S3 バケットのリソースポリシー

ログが送信されている S3 バケットには、特定のアクセス許可が含まれるリソースポリシーが必要です。現在バケットにリソースポリシーがなく、ロギングをセットアップしているユーザーがバケットに対する S3:GetBucketPolicy および S3:PutBucketPolicy 許可を持っているという場合は、Amazon S3 へのログの送信を開始するときに AWS が以下のポリシーを自動的に作成します。

JSON

```
{
  "Version": "2012-10-17",
  "Id": "AWSLogDeliveryWrite20150319",
  "Statement": [
    {
      "Sid": "AWSLogDeliveryAclCheck",
      "Effect": "Allow",
      "Principal": {
        "Service": "delivery.logs.amazonaws.com"
      },
      "Action": "s3:GetBucketAcl",
      "Resource": "arn:aws:s3:::amzn-s3-demo-bucket",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": [
            "0123456789"
          ]
        },
        "ArnLike": {
          "aws:SourceArn": [
            "arn:aws:logs:us-east-1:111122223333:*"
          ]
        }
      }
    }
  ],
  {
```

```

        "Sid": "AWSLogDeliveryWrite",
        "Effect": "Allow",
        "Principal": {
            "Service": "delivery.logs.amazonaws.com"
        },
        "Action": "s3:PutObject",
        "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/AWSLogs/account-ID/*",
        "Condition": {
            "StringEquals": {
                "s3:x-amz-acl": "bucket-owner-full-control",
                "aws:SourceAccount": [
                    "0123456789"
                ]
            },
            "ArnLike": {
                "aws:SourceArn": [
                    "arn:aws:logs:us-east-1:111122223333:"
                ]
            }
        }
    }
}

```

前のポリシーでは、aws:SourceAccount にはこのバケットにログが配信されるアカウント ID のリストを指定します。aws:SourceArn には、ログを生成するリソースの ARN のリストを arn:aws:logs:*source-region*:*source-account-id*:* の形式で指定します。

バケットにリソースポリシーがあるが、上記のポリシーにあるステートメントがそのポリシーに含まれておらず、ロギングをセットアップしているユーザーがバケットに対する S3:GetBucketPolicy および S3:PutBucketPolicy アクセス許可を持っているという場合は、そのステートメントがバケットのリソースポリシーに追加されます。

Note

アクセスs3:ListBucket許可 AWS CloudTrail が 付与されて
いない場合、にAccessDeniedエラーが表示されることがありま
すdelivery.logs.amazonaws.com。CloudTrail ログにこのようなエラーが表示されな
いようにするには、s3:ListBucket が delivery.logs.amazonaws.com にアクセスア
クセス許可を付与し、前述のバケットポリシーで設定された s3:GetBucketAcl アクセス

許可で示されている Condition パラメータを含める必要があります。これを簡単にするには、新しい Statement を作成する代わりに、AWSLogDeliveryAclCheck を "Action": ["s3:GetBucketAcl", "s3:ListBucket"] であるように直接更新することができます

Amazon S3 バケットのサーバー側の暗号化

Amazon S3 バケット内のデータを保護するには、Amazon S3-managedキーによるサーバー側の暗号化 (SSE-S3) または に保存されている AWS KMS キーによるサーバー側の暗号化 AWS Key Management Service (SSE-KMS) を有効にします。詳細については、「[サーバー側の暗号化を使用したデータの保護](#)」を参照してください。

SSE-S3 を選択した場合、追加の設定は必要ありません。Amazon S3 が暗号化キーを処理します。

Warning

SSE-KMS を選択した場合、このシナリオでは マネージドキーの使用はサポートされていないため、カスタマー AWS マネージドキーを使用する必要があります。AWS マネージドキーを使用して暗号化を設定すると、ログは読み取り不可能な形式で配信されます。

カスタマーマネージド AWS KMS キーを使用する場合、バケット暗号化を有効にするときに、カスタマーマネージドキーの Amazon リソースネーム (ARN) を指定できます。ログデリバリーアカウントが S3 バケットに書き込めるように、カスタマーマネージドキーのキーポリシー (S3 バケットのバケットポリシーではありません) に次を追加する必要があります。

このシナリオでは AWS マネージドキーの使用がサポートされていないため、SSE-KMS を選択した場合は、カスタマーマネージドキーを使用する必要があります。カスタマーマネージド AWS KMS キーを使用する場合、バケット暗号化を有効にするときに、カスタマーマネージドキーの Amazon リソースネーム (ARN) を指定できます。ログデリバリーアカウントが S3 バケットに書き込めるように、カスタマーマネージドキーのキーポリシー (S3 バケットのバケットポリシーではありません) に次を追加する必要があります。

```
{
  "Sid": "Allow Logs Delivery to use the key",
  "Effect": "Allow",
  "Principal": {
    "Service": [ "delivery.logs.amazonaws.com" ]
  },
}
```



```
"Action": [
  "kms:Encrypt",
  "kms:Decrypt",
  "kms:ReEncrypt*",
  "kms:GenerateDataKey*",
  "kms:DescribeKey"
],
"Resource": "*",
"Condition": {
  "StringEquals": {
    "aws:SourceAccount": ["0123456789"]
  },
  "ArnLike": {
    "aws:SourceArn": ["arn:aws:logs:us-east-1:0123456789:*"]
  }
}
```

aws:SourceAccount には、このバケットにログが配信されるアカウント ID のリストを指定します。aws:SourceArn には、ログを生成するリソースの ARN のリストを arn:aws:logs:*source-region*:*source-account-id*:* の形式で指定します。

Firehose に送信されるログ

このセクションは、前のセクションの表に掲載されているタイプのログが Firehose に送信される場合に適用されます。

ユーザーアクセス許可

これらのタイプのログの Firehose への送信を初めて設定するには、以下のアクセス許可を使用してアカウントにログインする必要があります。

- logs:CreateLogDelivery
- firehose:TagDeliveryStream
- iam:CreateServiceLinkedRole

これらのタイプのログのいずれかが Firehose に既に送信されている場合、これらのタイプの別のログを Firehose に送信するように設定するには、logs:CreateLogDelivery および firehose:TagDeliveryStream 許可のみが必要です。

アクセス許可のために使用される IAM ロール

Firehose はリソースポリシーを使用しないため、これらのログを Firehose に送信するように設定するときに IAM ロール AWS を使用します。は、という名前のサービスにリンクされたロール AWS を作成しますAWSRoleForLogDelivery。このサービスリンクロールには、以下のアクセス許可が含まれます。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "firehose:PutRecord",
        "firehose:PutRecordBatch",
        "firehose:ListTagsForDeliveryStream"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/LogDeliveryEnabled": "true"
        }
      },
      "Effect": "Allow"
    }
  ]
}
```

このサービスにリンクされたロールは、LogDeliveryEnabled タグが に設定されているすべての Firehose 配信ストリームにアクセス許可を付与しますtrue。は、ログ記録を設定するときに、このタグを送信先配信ストリームに AWS 付与します。

このサービスリンクロールには、delivery.logs.amazonaws.com サービスプリンシパルが必要なサービスリンクロールを引き受けることを可能にする信頼ポリシーもあります。以下がその信頼ポリシーです。

JSON

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

```
"Statement": [  
  {  
    "Effect": "Allow",  
    "Principal": {  
      "Service": "delivery.logs.amazonaws.com"  
    },  
    "Action": "sts:AssumeRole"  
  }  
]  
}
```

追加のアクセス許可が必要なロギング [V2]

一部の AWS サービスでは、新しい メソッドを使用してログを送信します。これは、これらのサービスから CloudWatch Logs、Amazon S3、Firehose、X-Ray のうち 1 つ以上の宛先へのログ配信を設定できる柔軟な方法です。

動作しているログ配信は、次の 3 つの要素で構成されます。

- 実際にログを送信するリソースを表す論理オブジェクトである `DeliverySource`。
- 実際の配信先を表す論理オブジェクトである `DeliveryDestination`。
- 配信ソースを配信先に接続する `Delivery`。

サポートされている AWS サービスと送信先間のログ配信を設定するには、以下を実行する必要があります。

- [PutDeliverySource](#) を使用して配信ソースを作成します。
- [PutDeliveryDestination](#) を使用して配信先を作成します。
- ログをクロスアカウントで配信する場合は、送信先アカウントで [PutDeliveryDestinationPolicy](#) を使用して、送信先に IAM ポリシーを割り当てる必要があります。このポリシーは、アカウント A の配信ソースからアカウント B の配信先への配信の作成を許可します。クロスアカウント配信の場合は、アクセス許可ポリシーを手動で作成する必要があります。
- [CreateDelivery](#) を使用して、1 つの配信元と 1 つの配信先だけをペアリングして配信を作成します。

以下のセクションでは、V2 プロセスを使用して各タイプの宛先へのログ配信を設定するためにサインインしたときに必要なアクセス許可の詳細について説明します。これらのアクセス許可は、サインインに使用する IAM ロールに付与できます。

Important

ログ生成リソースを削除した後、ログ配信リソースを削除することはお客様の責任です。そのためには、以下の手順を実行します。

1. [DeleteDelivery](#) オペレーションを使用して Delivery を削除します。
2. [DeleteDeliverySource](#) オペレーションを使用して DeliverySource を削除します。
3. 削除した DeliverySource に関連付けられた DeliveryDestination がこの特定の DeliverySource にのみ使用されている場合は、[DeleteDeliveryDestinations](#) オペレーションを使用して削除できます。

目次

- [CloudWatch Logs に送信されたログ](#)
- [Amazon S3 に送信されたログ](#)
 - [Amazon S3 バケットのサーバー側の暗号化](#)
- [Firehose に送信されるログ](#)
- [X-Ray に送信されたトレース](#)

CloudWatch Logs に送信されたログ

ユーザーアクセス許可

CloudWatch Logs へのログ送信を有効にするには、次のアクセス許可でサインインする必要があります。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ReadWriteAccessForLogDeliveryActions",
```

```

      "Effect": "Allow",
      "Action": [
        "logs:GetDelivery",
        "logs:GetDeliverySource",
        "logs:PutDeliveryDestination",
        "logs:GetDeliveryDestinationPolicy",
        "logs>DeleteDeliverySource",
        "logs:PutDeliveryDestinationPolicy",
        "logs>CreateDelivery",
        "logs:GetDeliveryDestination",
        "logs:PutDeliverySource",
        "logs>DeleteDeliveryDestination",
        "logs>DeleteDeliveryDestinationPolicy",
        "logs>DeleteDelivery",
        "logs:UpdateDeliveryConfiguration"
      ],
      "Resource": [
        "arn:aws:logs:us-east-1:111122223333:delivery:*",
        "arn:aws:logs:us-east-1:444455556666:delivery-source:*",
        "arn:aws:logs:us-east-1:777788889999:delivery-destination:*"
      ]
    },
    {
      "Sid": "ListAccessForLogDeliveryActions",
      "Effect": "Allow",
      "Action": [
        "logs:DescribeDeliveryDestinations",
        "logs:DescribeDeliverySources",
        "logs:DescribeDeliveries",
        "logs:DescribeConfigurationTemplates"
      ],
      "Resource": "*"
    },
    {
      "Sid": "AllowUpdatesToResourcePolicyCWL",
      "Effect": "Allow",
      "Action": [
        "logs:PutResourcePolicy",
        "logs:DescribeResourcePolicies",
        "logs:DescribeLogGroups"
      ],
      "Resource": [
        "arn:aws:logs:us-east-1:123456789012:"
      ]
    }
  ]
}

```

```

    }
  ]
}

```

ロググループのリソースポリシー

ログが送信されているロググループには、特定のアクセス許可が含まれるリソースポリシーが必要です。現在ロググループにリソースポリシーがなく、ロギングをセットアップしているユーザーがロググループに対する `logs:PutResourcePolicy`、`logs:DescribeResourcePolicies`、および `logs:DescribeLogGroups` アクセス許可を持っているという場合は、CloudWatch Logs へのログの送信を開始するときに AWS が以下のポリシーを自動的に作成します。新しく作成されたサブスクリプションの場合、リソースポリシーはロググループレベルで設定され、最大サイズは 51,200 バイトです。既存のアカウントレベルのリソースポリシーがワイルドカードを介してアクセス許可を既に付与している場合、別のロググループレベルのポリシーは作成されません。特定のロググループの `logGroup` レベルのリソースポリシーを確認するには、`--resource-arn` パラメータをロググループ ARN に設定し、`--policy-scope` パラメータを `logGroup` に設定して `describe-resource-policies` コマンドを使用します RESOURCE。

JSON

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AWSLogDeliveryWrite20150319",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "delivery.logs.amazonaws.com"
        ]
      },
      "Action": [
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Resource": [
        "arn:aws:logs:us-east-1:111122223333:log-group:my-log-group:log-stream:*"
      ],
      "Condition": {

```

```

        "StringEquals": {
            "aws:SourceAccount": [
                "0123456789"
            ]
        },
        "ArnLike": {
            "aws:SourceArn": [
                "arn:aws:logs:us-east-1:111122223333:*"
            ]
        }
    }
}
]
}

```

ロググループのリソースポリシーの制限は 51,200 バイトです。この制限に達すると、AWS は新しいアクセス許可を追加できません。そのためには、ポリシーを手動で変更して、logs:CreateLogStream および logs:PutLogEvents アクションに対する delivery.logs.amazonaws.com サービスプリンシパルのアクセス許可を付与する必要があります。お客様は、などのワイルドカードを含むロググループ名プレフィックスを使用し /aws/vendedlogs/*、このロググループ名を今後の配信作成に使用する必要があります。

JSON

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AWSLogDeliveryWrite20150319",
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "delivery.logs.amazonaws.com"
                ]
            },
            "Action": [
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": [

```

```

        "arn:aws:logs:us-east-1:111122223333:log-group:my-log-group/aws/
vendedLogs/*"
    ],
    "Condition": {
        "StringEquals": {
            "aws:SourceAccount": [
                "0123456789"
            ]
        },
        "ArnLike": {
            "aws:SourceArn": [
                "arn:aws:logs:us-east-1:111122223333:*"
            ]
        }
    }
}
]
}

```

Amazon S3 に送信されたログ

ユーザーアクセス許可

Amazon S3 へのログ送信を有効にするには、次のアクセス許可でサインインする必要があります。

JSON

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ReadWriteAccessForLogDeliveryActions",
      "Effect": "Allow",
      "Action": [
        "logs:GetDelivery",
        "logs:GetDeliverySource",
        "logs:PutDeliveryDestination",
        "logs:GetDeliveryDestinationPolicy",
        "logs>DeleteDeliverySource",
        "logs:PutDeliveryDestinationPolicy",
        "logs>CreateDelivery",
        "logs:GetDeliveryDestination",

```



```

        "logs:PutDeliverySource",
        "logs>DeleteDeliveryDestination",
        "logs>DeleteDeliveryDestinationPolicy",
        "logs>DeleteDelivery",
        "logs:UpdateDeliveryConfiguration"
    ],
    "Resource": [
        "arn:aws:logs:us-east-1:111122223333:delivery:*",
        "arn:aws:logs:us-east-1:111122223333:delivery-source:*",
        "arn:aws:logs:us-east-1:111122223333:delivery-destination:*"
    ],
    },
    {
        "Sid": "ListAccessForLogDeliveryActions",
        "Effect": "Allow",
        "Action": [
            "logs:DescribeDeliveryDestinations",
            "logs:DescribeDeliverySources",
            "logs:DescribeDeliveries",
            "logs:DescribeConfigurationTemplates"
        ],
        "Resource": "*"
    },
    {
        "Sid": "AllowUpdatesToResourcePolicyS3",
        "Effect": "Allow",
        "Action": [
            "s3:PutBucketPolicy",
            "s3:GetBucketPolicy"
        ],
        "Resource": "arn:aws:s3:::bucket-name"
    }
]
}

```

ログが送信されている S3 バケットには、特定のアクセス許可が含まれるリソースポリシーが必要です。現在バケットにリソースポリシーがなく、ロギングをセットアップしているユーザーがバケットに対する S3:GetBucketPolicy および S3:PutBucketPolicy 許可を持っているという場合は、Amazon S3 へのログの送信を開始するときに AWS が以下のポリシーを自動的に作成します。

JSON

```
{
  "Version": "2012-10-17",
  "Id": "AWSLogDeliveryWrite20150319",
  "Statement": [
    {
      "Sid": "AWSLogDeliveryWrite",
      "Effect": "Allow",
      "Principal": {
        "Service": "delivery.logs.amazonaws.com"
      },
      "Action": "s3:PutObject",
      "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/AWSLogs/account-ID/*",
      "Condition": {
        "StringEquals": {
          "s3:x-amz-acl": "bucket-owner-full-control",
          "aws:SourceAccount": [
            "0123456789"
          ]
        },
        "ArnLike": {
          "aws:SourceArn": [
            "arn:aws:logs:us-east-1:111122223333:delivery-source:*"
          ]
        }
      }
    }
  ]
}
```

前のポリシーでは、aws:SourceAccount にはこのバケットにログが配信されるアカウント ID のリストを指定します。aws:SourceArn には、ログを生成するリソースの ARN のリストを arn:aws:logs:*source-region*:*source-account-id*:* の形式で指定します。

バケットにリソースポリシーがあるが、上記のポリシーにあるステートメントがそのポリシーに含まれておらず、ロギングをセットアップしているユーザーがバケットに対する S3:GetBucketPolicy および S3:PutBucketPolicy アクセス許可を持っているという場合は、そのステートメントがバケットのリソースポリシーに追加されます。

 Note

アクセスs3:ListBucket許可 AWS CloudTrail が 付与されていない場合、にAccessDeniedエラーが表示されることがあります。delivery.logs.amazonaws.com。CloudTrail ログにこのようなエラーが表示されないようにするには、s3:ListBucket が delivery.logs.amazonaws.com にアクセス許可を付与し、前述のバケットポリシーで設定された s3:GetBucketAcl アクセス許可で示されている Condition パラメータを含める必要があります。これを簡単にするには、新しい Statement を作成する代わりに、AWSLogDeliveryAclCheck を “Action”: [“s3:GetBucketAcl”, “s3:ListBucket”] であるように直接更新することができます。

Amazon S3 バケットのサーバー側の暗号化

Amazon S3 バケット内のデータを保護するには、Amazon S3-managedキーによるサーバー側の暗号化 (SSE-S3) または に保存されている AWS KMS キーによるサーバー側の暗号化 AWS Key Management Service (SSE-KMS) を有効にします。詳細については、「[サーバー側の暗号化を使用したデータの保護](#)」を参照してください。

SSE-S3 を選択した場合、追加の設定は必要ありません。Amazon S3 が暗号化キーを処理します。

 Warning

SSE-KMS を選択した場合、このシナリオでは マネージドキーの使用はサポートされていないため、カスタマー AWS マネージドキーを使用する必要があります。AWS マネージドキーを使用して暗号化を設定すると、ログは読み取り不可能な形式で配信されます。

カスタマーマネージド AWS KMS キーを使用する場合、バケット暗号化を有効にするときに、カスタマーマネージドキーの Amazon リソースネーム (ARN) を指定できます。ログデリバリーアカウントが S3 バケットに書き込めるように、カスタマーマネージドキーのキーポリシー (S3 バケットのバケットポリシーではありません) に次を追加する必要があります。

このシナリオでは AWS マネージドキーの使用がサポートされていないため、SSE-KMS を選択した場合は、カスタマーマネージドキーを使用する必要があります。カスタマーマネージド AWS KMS キーを使用する場合、バケット暗号化を有効にするときに、カスタマーマネージドキーの Amazon リソースネーム (ARN) を指定できます。ログデリバリーアカウントが S3 バケットに書き込めるよ

うに、カスターマネージドキーのキーポリシー (S3 バケットのバケットポリシーではありません) に次を追加する必要があります。

```
{
  "Sid": "Allow Logs Delivery to use the key",
  "Effect": "Allow",
  "Principal": {
    "Service": [ "delivery.logs.amazonaws.com" ]
  },
  "Action": [
    "kms:Encrypt",
    "kms:Decrypt",
    "kms:ReEncrypt*",
    "kms:GenerateDataKey*",
    "kms:DescribeKey"
  ],
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "aws:SourceAccount": ["0123456789"]
    },
    "ArnLike": {
      "aws:SourceArn": ["arn:aws:logs:us-east-1:0123456789:delivery-source:*"]
    }
  }
}
```

aws:SourceAccount には、このバケットにログが配信されるアカウント ID のリストを指定します。aws:SourceArn には、ログを生成するリソースの ARN のリストを arn:aws:logs:*source-region*:*source-account-id*:* の形式で指定します。

Firehose に送信されるログ

ユーザーアクセス許可

Firehose へのログ送信を有効にするには、次のアクセス許可を使用してサインインする必要があります。

JSON

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

```

"Statement": [
  {
    "Sid": "ReadWriteAccessForLogDeliveryActions",
    "Effect": "Allow",
    "Action": [
      "logs:GetDelivery",
      "logs:GetDeliverySource",
      "logs:PutDeliveryDestination",
      "logs:GetDeliveryDestinationPolicy",
      "logs:DeleteDeliverySource",
      "logs:PutDeliveryDestinationPolicy",
      "logs:CreateDelivery",
      "logs:GetDeliveryDestination",
      "logs:PutDeliverySource",
      "logs:DeleteDeliveryDestination",
      "logs:DeleteDeliveryDestinationPolicy",
      "logs:DeleteDelivery",
      "logs:UpdateDeliveryConfiguration"
    ],
    "Resource": [
      "arn:aws:logs:us-east-1:111122223333:delivery:*",
      "arn:aws:logs:us-east-1:111122223333:delivery-source:*",
      "arn:aws:logs:us-east-1:111122223333:delivery-destination:*"
    ]
  },
  {
    "Sid": "ListAccessForLogDeliveryActions",
    "Effect": "Allow",
    "Action": [
      "logs:DescribeDeliveryDestinations",
      "logs:DescribeDeliverySources",
      "logs:DescribeDeliveries",
      "logs:DescribeConfigurationTemplates"
    ],
    "Resource": "*"
  },
  {
    "Sid": "AllowUpdatesToResourcePolicyFH",
    "Effect": "Allow",
    "Action": [
      "firehose:TagDeliveryStream"
    ],
    "Resource": [
      "arn:aws:firehose:us-east-1:111122223333:deliverystream/*"
    ]
  }
]

```

```

    ]
  },
  {
    "Sid": "CreateServiceLinkedRole",
    "Effect": "Allow",
    "Action": [
      "iam:CreateServiceLinkedRole"
    ],
    "Resource": "arn:aws:iam::111122223333:role/aws-service-role/
delivery.logs.amazonaws.com/AWSServiceRoleForLogDelivery"
  }
]
}

```

リソースのアクセス許可のために使用される IAM ロール

Firehose はリソースポリシーを使用しないため、これらのログを Firehose に送信するように設定するときに IAM ロール AWS を使用します。は、という名前のサービスにリンクされたロール AWS を作成しますAWSServiceRoleForLogDelivery。このサービスリンクロールには、以下のアクセス許可が含まれます。

JSON

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "firehose:PutRecord",
        "firehose:PutRecordBatch",
        "firehose:ListTagsForDeliveryStream"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/LogDeliveryEnabled": "true"
        }
      },
      "Effect": "Allow"
    }
  ]
}

```

```
}
```

このサービスにリンクされたロールは、LogDeliveryEnabled タグが に設定されているすべての Firehose 配信ストリームにアクセス許可を付与しますtrue。は、ログ記録を設定するときに、この タグを送信先配信ストリームに AWS 付与します。

このサービスリンクロールには、delivery.logs.amazonaws.com サービスプリンシパルが必要なサービスリンクロールを引き受けることを可能にする信頼ポリシーもあります。以下がその信頼ポリシーです。

JSON

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

X-Ray に送信されたトレース

ユーザーアクセス許可

トレースの送信を有効にするには AWS X-Ray、次のアクセス許可でサインインする必要があります。

JSON

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

```

        "Sid": "ReadWriteAccessForLogDeliveryActions",
        "Effect": "Allow",
        "Action": [
            "logs:GetDelivery",
            "logs:GetDeliverySource",
            "logs:PutDeliveryDestination",
            "logs:GetDeliveryDestinationPolicy",
            "logs:DeleteDeliverySource",
            "logs:PutDeliveryDestinationPolicy",
            "logs:CreateDelivery",
            "logs:GetDeliveryDestination",
            "logs:PutDeliverySource",
            "logs:DeleteDeliveryDestination",
            "logs:DeleteDeliveryDestinationPolicy",
            "logs:DeleteDelivery",
            "logs:UpdateDeliveryConfiguration"
        ],
        "Resource": [
            "arn:aws:logs:us-east-1:111122223333:delivery:*",
            "arn:aws:logs:us-east-1:111122223333:delivery-source:*",
            "arn:aws:logs:us-east-1:111122223333:delivery-destination:*"
        ]
    },
    {
        "Sid": "ListAccessForLogDeliveryActions",
        "Effect": "Allow",
        "Action": [
            "logs:DescribeDeliveryDestinations",
            "logs:DescribeDeliverySources",
            "logs:DescribeDeliveries",
            "logs:DescribeConfigurationTemplates"
        ],
        "Resource": "*"
    },
    {
        "Sid": "AllowUpdatesToResourcePolicyXRay",
        "Effect": "Allow",
        "Action": [
            "xray:PutResourcePolicy",
            "xray:ListResourcePolicies",
            "xray:GetTraceSegmentDestination"
        ],
        "Resource": "*"
    }
}

```



```
]
}
```

X-Ray リソースポリシー

トレースが送信されている送信先アカウントには、特定のアクセス許可が含まれるリソースポリシーが必要です。トレースを設定するユーザーがアカウントに `xray:PutResourcePolicy` および アクセス `xray:ListResourcePolicies` 許可を持っている場合、X-Ray へのトレースの送信を開始すると、 によってリソースポリシー AWS が自動的に作成されます。作成されるポリシーは、ソース サービス によって異なります。

Amazon Bedrock AgentCore リソース

AWS は、リソースタイプごとに 1 つのリソースポリシーを作成します。このポリシーは、アカウント境界にスコープされたワイルドカードパターンを使用し、アカウント内の同じ Amazon Bedrock AgentCore リソースタイプのすべてのリソースをカバーします。例えば、Amazon Bedrock AgentCore Memory リソースがトレース配信に対して有効になっている場合、ポリシーは、将来作成されるメモリリソースを含む、そのアカウントのすべてのメモリリソースをカバーします。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AWSLogDeliveryWrite",
      "Effect": "Allow",
      "Principal": {
        "Service": "delivery.logs.amazonaws.com"
      },
      "Action": "xray:PutTraceSegments",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "123456789012"
        },
        "ForAllValues:ArnLike": {
          "logs:LogGeneratingResourceArns": "arn:aws:bedrock-agentcore:us-east-1:123456789012:memory/*"
        }
      }
    }
  ]
}
```

```
      "ArnLike": {
        "aws:SourceArn": "arn:aws:logs:us-east-1:123456789012:delivery-
source:*"
      }
    }
  ]
}
```

その他の AWS サービス

トレース配信をサポートする他のサービスの場合、AWS は特定のソースリソースを対象とするリソースポリシーを作成します。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AWSLogDeliveryWrite",
      "Effect": "Allow",
      "Principal": {
        "Service": "delivery.logs.amazonaws.com"
      },
      "Action": "xray:PutTraceSegments",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "123456789012"
        },
        "ForAllValues:ArnLike": {
          "logs:LogGeneratingResourceArns": "arn:aws:bedrock:us-
east-1:123456789012:knowledge-base/KnowledgeBaseId"
        },
        "ArnLike": {
          "aws:SourceArn": "arn:aws:logs:us-east-1:123456789012:delivery-
source:xray-test"
        }
      }
    }
  ]
}
```

トランザクション検索を有効にする

X-Ray へのトレースの送信を有効にするには、[トランザクション検索](#)を有効にする必要があります。

サービス固有のアクセス許可

前のセクションに記載されている送信先固有のアクセス許可に加えて、一部のサービスでは、セキュリティの追加のレイヤーとして、お客様がリソースからログを送信することを許可する明示的な承認が必要です。これにより、そのサービス内でログを記録したリソースの AllowVendedLogDeliveryForResource アクションが承認されます。これらのサービスでは、次のポリシーを使用し、####と#####を適切な値に置き換えます。これらのフィールドのサービス固有の値については、これらのサービスの vended log に関するドキュメントページを参照してください。次の例では、Amazon SES から提供されたログを有効にするようにポリシーが更新されています。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ServiceLevelAccessForLogDelivery",
      "Effect": "Allow",
      "Action": [
        "ses:AllowVendedLogDeliveryForResource"
      ],
      "Resource": "arn:aws:ses:us-east-1:123456789012:resource-type/*"
    }
  ]
}
```

コンソール固有のアクセス許可

API ではなくコンソールを使用してログ配信を設定する場合は、前のセクションに記載されているアクセス許可に加えて、以下の追加アクセス許可も必要です。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowLogDeliveryActionsConsoleCWL",
      "Effect": "Allow",
      "Action": [
        "logs:DescribeLogGroups",
        "logs:CreateLogGroup"
      ],
      "Resource": [
        "arn:aws:logs:us-east-1:111122223333:log-group:*"
      ]
    },
    {
      "Sid": "AllowLogDeliveryActionsConsoleS3",
      "Effect": "Allow",
      "Action": [
        "s3:ListAllMyBuckets",
        "s3:ListBucket",
        "s3:GetBucketLocation"
      ],
      "Resource": [
        "arn:aws:s3:::*"
      ]
    },
    {
      "Sid": "AllowLogDeliveryActionsConsoleFH",
      "Effect": "Allow",
      "Action": [
        "firehose:ListDeliveryStreams",
        "firehose:DescribeDeliveryStream"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

アカウント間の配信の例

この例では、2つのアカウントが関係しています。ログ生成リソースを持つアカウントはアカウント A、ID: **123456789012**、ログ消費リソースを持つアカウントはアカウント B、ID: **111122223333** です。

アカウント A は、ARN `arn:aws:bedrock:us-east-1:123456789012:knowledge-base/kb-12345678` を使用して、アカウントの Amazon Bedrock ナレッジベースからログを配信したいと考えています。

この例では、アカウント A には次のアクセス許可が必要です。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowVendedLogDeliveryForKnowledgeBase",
      "Effect": "Allow",
      "Action": [
        "bedrock:AllowVendedLogDeliveryForResource"
      ],
      "Resource": "arn:aws:bedrock:us-east-1:123456789012:knowledge-
base/XXXXXXXXXX"
    },
    {
      "Sid": "CreateLogDeliveryPermissions",
      "Effect": "Allow",
      "Action": [
        "logs:PutDeliverySource",
        "logs:CreateDelivery"
      ],
      "Resource": [
        "arn:aws:logs:us-east-1:123456789012:delivery-source:*",
        "arn:aws:logs:us-east-1:123456789012:delivery:*",
        "arn:aws:logs:us-east-1:444455556666:delivery-destination:*"
      ]
    }
  ]
}
```

配信ソースを作成する

まず、アカウント A は、Bedrock ナレッジベースを使用して配信ソースを作成します。

```
aws logs put-delivery-source --name my-delivery-source --log-type APPLICATION_LOGS --  
resource-arn arn:aws:bedrock:region:AAAAAAAAAAAAA:knowledge-base/XXXXXXXXXX
```

次に、アカウント B は、次のいずれかのフローを使用して配信先を作成する必要があります。

- [Amazon S3 バケットへの配信を設定する](#)
- [Firehose ストリームへの配信を設定する](#)

Amazon S3 バケットへの配信を設定する

アカウント B は ARN `arn:aws:s3:::amzn-s3-demo-bucket` を使用して S3 バケットでログを受信したいと考えています。この例の場合、アカウント B には次のアクセス許可が必要です。

JSON

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Sid": "PutLogDestinationPermissions",  
      "Effect": "Allow",  
      "Action": [  
        "logs:PutDeliveryDestination",  
        "logs:PutDeliveryDestinationPolicy"  
      ],  
      "Resource": "arn:aws:logs:us-east-1:111122223333:delivery-  
destination:*"  
    }  
  ]  
}
```

バケットポリシーには、次のアクセス許可が必要です。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AWSLogsDeliveryWrite",
      "Effect": "Allow",
      "Principal": {
        "Service": "delivery.logs.amazonaws.com"
      },
      "Action": [
        "s3:PutObject"
      ],
      "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/AWSLogs/123456789012/",
      "Condition": {
        "StringEquals": {
          "s3:x-amz-acl": "bucket-owner-full-control",
          "aws:SourceAccount": [
            "123456789012"
          ]
        },
        "ArnLike": {
          "aws:SourceArn": [
            "arn:aws:logs:us-east-1:123456789012:delivery-source:my-delivery-source"
          ]
        }
      }
    }
  ]
}
```

バケットが SSE-KMS で暗号化されている場合は、AWS KMS キーポリシーに適切なアクセス許可があることを確認してください。例えば、KMS キーが `arn:aws:kms:us-east-1:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab` の場合、以下を使用します。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowLogsGenerateDataKey",
      "Effect": "Allow",
      "Principal": {
        "Service": "delivery.logs.amazonaws.com"
      },
      "Action": [
        "kms:GenerateDataKey"
      ],
      "Resource": "arn:aws:kms:us-east-1:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": [
            "123456789012"
          ]
        },
        "ArnLike": {
          "aws:SourceArn": [
            "arn:aws:logs:us-east-1:123456789012:delivery-source:my-delivery-source"
          ]
        }
      }
    }
  ]
}
```

アカウント B は、S3 バケットを宛先リソースとして配信先を作成できます。

```
aws logs put-delivery-destination --name my-s3-delivery-destination --delivery-destination-configuration "destinationResourceArn=arn:aws:s3:::amzn-s3-demo-bucket"
```

次に、アカウント B は新しく作成された配信先に配信先ポリシーを作成し、アカウント A がログ配信を作成するアクセス許可を付与します。新しく作成された配信先に追加されるポリシーは次のとおりです。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowCreateDelivery",
      "Effect": "Allow",
      "Principal": {
        "AWS": "123456789012"
      },
      "Action": [
        "logs:CreateDelivery"
      ],
      "Resource": "arn:aws:logs:us-east-1:111122223333:delivery-destination:amzn-s3-demo-bucket"
    }
  ]
}
```

このポリシーは、アカウント B のコンピュータに destination-policy-s3.json として保存されます。このリソースをアタッチするために、アカウント B は次のコマンドを実行します。

```
aws logs put-delivery-destination-policy --delivery-destination-name my-s3-delivery-destination --delivery-destination-policy file://destination-policy-s3.json
```

最後に、アカウント A は配信を作成し、アカウント A の配信ソースをアカウント B の配信先にリンクします。

```
aws logs create-delivery --delivery-source-name my-delivery-source --delivery-destination-arn arn:aws:logs:region:BBBBBBBBBBBB:delivery-destination:my-s3-delivery-destination
```

Firehose ストリームへの配信を設定する

この例では、アカウント B は Firehose ストリームでログを受信したいと考えています。Firehose ストリームには次の ARN があり、DirectPut 配信ストリームタイプを使用するように設定されています。

arn:aws:firehose:*us-east-1:111122223333*:deliverystream/*log-delivery-stream*

この例では、アカウント B には次のアクセス許可が必要です。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowFirehoseCreateSLR",
      "Effect": "Allow",
      "Action": [
        "iam:CreateServiceLinkedRole"
      ],
      "Resource": "arn:aws:iam::111122223333:role/aws-service-role/delivery.logs.amazonaws.com/AWSServiceRoleForLogDelivery"
    },
    {
      "Sid": "AllowFirehoseTagging",
      "Effect": "Allow",
      "Action": [
        "firehose:TagDeliveryStream"
      ],
      "Resource": "arn:aws:firehose:us-east-1:111122223333:deliverystream/X"
    },
    {
      "Sid": "AllowFirehoseDeliveryDestination",
      "Effect": "Allow",
      "Action": [
        "logs:PutDeliveryDestination",
        "logs:PutDeliveryDestinationPolicy"
      ],
      "Resource": "arn:aws:logs:us-east-1:111122223333:delivery-destination:*"
    }
  ]
}
```

Firehose ストリームのタグ LogDeliveryEnabled は true に設定する必要があります。

その後、アカウント B は、Firehose ストリームを宛先リソースとして配信先を作成します。

```
aws logs put-delivery-destination --name my-fh-delivery-destination --delivery-destination-configuration "destinationResourceArn=arn:aws:firehose:region:BBBBBBBBBBBB:deliverystream/X"
```

次に、アカウント B は新しく作成された配信先に配信先ポリシーを作成し、アカウント A がログ配信を作成するアクセス許可を付与します。新しく作成された配信先に追加されるポリシーは次のとおりです。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowCreateDelivery",
      "Effect": "Allow",
      "Principal": {
        "AWS": "123456789012"
      },
      "Action": [
        "logs:CreateDelivery"
      ],
      "Resource": "arn:aws:logs:us-east-1:111122223333:delivery-destination:amzn-s3-demo-bucket"
    }
  ]
}
```

このポリシーは、アカウント B のコンピュータに destination-policy-fh.json として保存されます。このリソースをアタッチするために、アカウント B は次のコマンドを実行します。

```
aws logs put-delivery-destination-policy --delivery-destination-name my-fh-delivery-destination --delivery-destination-policy file://destination-policy-fh.json
```

最後に、アカウント A は配信を作成し、アカウント A の配信ソースをアカウント B の配信先にリンクします。

```
aws logs create-delivery --delivery-source-name my-delivery-source --delivery-destination-arn arn:aws:logs:region:BBBBBBBBBBBBBB:delivery-destination:my-fh-delivery-destination
```

サービス間での不分別な代理処理の防止

混乱した代理問題は、アクションを実行するためのアクセス許可を持たないエンティティが、より特権のあるエンティティにアクションの実行を強制できてしまう場合に生じる、セキュリティ上の問題です。では AWS、サービス間のなりすましにより、混乱した代理問題が発生する可能性があります。サービス間でのなりすましは、1 つのサービス (呼び出し元サービス) が、別のサービス (呼び出し対象サービス) を呼び出すときに発生する可能性があります。呼び出し元サービスは、本来ならアクセスすることが許可されるべきではない方法でその許可を使用して、別のお客様のリソースに対する処理を実行するように操作される場合があります。これを防ぐために、は、アカウント内のリソースへのアクセス権が付与されたサービスプリンシパルを持つすべてのサービスのデータを保護するのに役立つツール AWS を提供します。

リソースポリシーで

[aws:SourceArn](#)、[aws:SourceAccount](#)、[aws:SourceOrgID](#)、[aws:SourceOrgPaths](#) のグローバル条件コンテキストキーを使用し、CloudWatch Logs がそのリソースに対して別のサービスを付与する許可を制限することをお勧めします。1 つのリソースだけをクロスサービスのアクセスに関連付ける場合は、[aws:SourceArn](#) を使用します。アカウント内の任意のリソースをクロスサービスの使用に関連付ける場合は、[aws:SourceAccount](#) を使用します。組織内の任意のアカウントの任意のリソースをクロスサービスの使用に関連付ける場合は、[aws:SourceOrgID](#) を使用します。を使用して [aws:SourceOrgPaths](#)、AWS Organizations パス内のアカウントのリソースをクロスサービスの使用に関連付けます。パスの使用と理解の詳細については、[AWS Organizations「エンティティパスを理解する」](#)を参照してください。

混乱した代理問題から保護するための最も効果的な方法は、リソースの完全な ARN を指定して、[aws:SourceArn](#) グローバル条件コンテキストキーを使用することです。リソースの完全な ARN が不明な場合や、複数のリソースを指定する場合には、グローバルコンテキスト条件キー [aws:SourceArn](#) で、ARN の未知部分を示すためにワイルドカード文字 (*) を使用します。例えば、`arn:aws:servicename:*:123456789012:*`。

[aws:SourceArn](#) の値に Amazon S3 バケット ARN などのアカウント ID が含まれていない場合は、両方の [aws:SourceAccount](#) と [aws:SourceArn](#) を使用して、アクセス許可を制限する必要があります。

混乱した代理問題から保護するために、リソースベースポリシー内のリソースの組織 ID または組織パスを指定しながら、aws:SourceOrgID または aws:SourceOrgPaths のグローバル条件コンテキストキーを使用してください。aws:SourceOrgID または aws:SourceOrgPaths キーを含むポリシーには正しいアカウントが自動的に組み込まれるため、組織のアカウントを追加、削除、移動する際には手動で更新する必要はありません。

[ステップ 1: 送信先を作成する](#) と [ステップ 2: 送信先を作成する](#) に記載されている、Kinesis データストリームと Firehose にデータを書き込むためのアクセス許可を CloudWatch Logs に付与するポリシーでは、aws:SourceArn グローバル条件コンテキストキーを使用して、混乱した代理の問題を防止する方法を示しています。

テレメトリ有効化ルールを使用してログ有効化を自動化する

テレメトリ有効化ルールを使用して、AWS リソースのログ収集を自動的に設定できます。ルールは、組織またはアカウント全体でログ収集を標準化し、一貫したモニタリングカバレッジを確保するのに役立ちます。

有効化ルールを使用すると、次のことが可能になります。

- ルールスコープに一致する新規および既存のリソースのログ記録を自動的に有効にする
- 組織、組織単位 (OU)、または個々のアカウントレベルでルールを適用する
- タグを使用してルールが影響を与えるリソースをフィルタリングする

有効化ルールは現在、Amazon VPC フローログ、AWS WAF ログ、Route 53 Resolver クエリログ、NLB アクセスログ、Amazon EKS コントロールプレーンログ、CloudTrail データおよび管理イベント、Amazon Bedrock AgentCore ログ、Amazon EC2 詳細メトリクス、AWS Security Hub、Amazon Bedrock AgentCore Gateway、Amazon Bedrock AgentCore Memory、CloudFront Distribution の AWS テレメトリソースをサポートしています。

ルールの動作、ルールの管理、トラブルシューティング、サービス固有の考慮事項などの有効化ルールの詳細については、Amazon CloudWatch ユーザーガイド」の「[テレメトリ有効化ルール](#)」を参照してください。

ルール評価階層

有効化ルールは階層的に評価されます。まず組織ルール、次に OU レベルのルール、次にアカウントレベルのルールです。上位レベルのルールは、ベースラインテレメトリを提供します。下位レベル

のルールはテレメトリを追加できますが、減らすことはできません。競合するルールが同じスコープに存在する場合、競合が解決されるまで適用されません。

有効化ルールの作成

テレメトリ有効化ルールを作成する手順

1. CloudWatch コンソールの <https://console.aws.amazon.com/cloudwatch/> を開いてください。
2. ナビゲーションペインで、[テレメトリの設定] を選択します。
3. [有効化ルール] タブを選択します。
4. [ルールを追加] を選択してください。
5. ルール名、スコープ (組織、OU、またはアカウント)、データソース、テレメトリタイプを指定します。
6. 必要に応じて、ルールが影響を与えるリソースをフィルタリングするタグを追加します。
7. [Create rule] (ルールの作成) を選択します。

サードパーティーソースからのログ記録を有効にする

CloudWatch Logs は、直接 API 統合と Amazon S3 バケット統合を通じて、サードパーティーソースからのログの取り込みをサポートします。AWS Security Hub CSPM を使用して、追加のサードパーティーのセキュリティ検出結果を受け取ることもできます。

直接のサードパーティー統合

CloudWatch Logs は、以下のサードパーティーソースとの直接統合を提供します。これらの統合では、直接 API 接続または Amazon S3 バケット統合を使用して CloudWatch Logs にログを取り込みます。

- CrowdStrike Falcon
- Microsoft Office 365
- Okta Auth0
- Microsoft Entra ID
- Palo Alto Networks NGFW
- Microsoft Windows イベント
- Wiz
- Zscaler Internet Access
- Okta SSO
- SentinelOne エンドポイントセキュリティ
- GitHub 監査ログ
- ServiceNow CMDB
- Cisco Umbrella

セットアップ手順については、Amazon CloudWatch ユーザーガイドの[サードパーティー統合セットアップガイド](#)を参照してください。Amazon CloudWatch

AWS Security Hub CSPM 経由の追加ソース

直接統合に加えて、サードパーティーのセキュリティパートナーは AWS Security Hub CSPM に検出結果を送信し、CloudWatch Logs のデータソースとして利用できます。次の表に、Security Hub CSPM パートナー統合とその統合タイプを示します。

CloudWatch Logs でデータソースとして Security Hub CSPM の検出結果を有効にするには、CloudWatch コンソールで AWS Security Hub のテレメトリ有効化ルールを作成します。有効化ルールは、Security Hub CSPM から マネージドロググループに結果を自動的に取り込むように CloudWatch を設定します。step-by-stepの手順については、[Amazon CloudWatch ユーザーガイド](#)の「[テレメトリ有効化ルール](#)」を参照してください。

Partner	統合
3CORESec – NTA	Security Hub CSPM 経由で結果を送信する
アラートロジック – SIEMless Threat Management	Security Hub CSPM 経由で結果を送信する
Aqua Security – クラウドネイティブセキュリティプラットフォーム	Security Hub CSPM 経由で結果を送信する
Aqua Security – Kube-bench	Security Hub CSPM 経由で結果を送信する
Armor – Armor Anywhere	Security Hub CSPM 経由で結果を送信する
AttackIQ	Security Hub CSPM 経由で結果を送信する
Barracuda Networks – Cloud Security Guardian	Security Hub CSPM 経由で結果を送信する
BigID – BigID Enterprise	Security Hub CSPM 経由で結果を送信する
ブルー六角形	Security Hub CSPM 経由で結果を送信する
チェックポイント — CloudGuard IaaS	Security Hub CSPM 経由で結果を送信する

Partner	統合
チェックポイント — CloudGuard 姿勢管理	Security Hub CSPM 経由で結果を送信する
Claroty – xDome	Security Hub CSPM 経由で結果を送信する
クラウドストレージセキュリティ – Amazon S3 のウイルス対策	Security Hub CSPM 経由で結果を送信する
コントラストセキュリティ – コントラスト評価	Security Hub CSPM 経由で結果を送信する
CrowdStrike – CrowdStrike Falcon	Security Hub CSPM 経由で結果を送信する
CyberArk – 特権脅威分析	Security Hub CSPM 経由で結果を送信する
データ定理	Security Hub CSPM 経由で結果を送信する
Drata	Security Hub CSPM 経由で結果を送信する
Forcepoint – CASB	Security Hub CSPM 経由で結果を送信する
Forcepoint – クラウドセキュリティゲートウェイ	Security Hub CSPM 経由で結果を送信する
Forcepoint – DLP	Security Hub CSPM 経由で結果を送信する
Forcepoint – NGFW	Security Hub CSPM 経由で結果を送信する
ファーク	Security Hub CSPM 経由で結果を送信する

Partner	統合
Guardicore – Centra	Security Hub CSPM 経由で結果を送信する
HackerOne — 脆弱性インテリジェンス	Security Hub CSPM 経由で結果を送信する
JFrog – Xray	Security Hub CSPM 経由で結果を送信する
Juniper Networks – vSRX 次世代ファイアウォール	Security Hub CSPM 経由で結果を送信する
k9 セキュリティ – Access Analyzer	Security Hub CSPM 経由で結果を送信する
レースワーク	Security Hub CSPM 経由で結果を送信する
McAfee – MVISION CNAPP	Security Hub CSPM 経由で結果を送信する
Netscout – サイバー調査担当者	Security Hub CSPM 経由で結果を送信する
Orca – クラウドセキュリティプラットフォーム	Security Hub CSPM 経由で結果を送信する
Palo Alto Networks – Prisma クラウドコンピューティング	Security Hub CSPM 経由で結果を送信する
Palo Alto Networks – Prisma Cloud Enterprise	Security Hub CSPM 経由で結果を送信する
Plerion – クラウドセキュリティプラットフォーム	Security Hub CSPM 経由で結果を送信する
プロウラー	Security Hub CSPM 経由で結果を送信する

Partner	統合
Qualys – 脆弱性管理	Security Hub CSPM 経由で結果を送信する
Rapid7 – InsightVM	Security Hub CSPM 経由で結果を送信する
SentinelOne	Security Hub CSPM 経由で結果を送信する
Snyk	Security Hub CSPM 経由で結果を送信する
Sonrai Security – Sonrai Dig	Security Hub CSPM 経由で結果を送信する
Sophos – サーバー保護	Security Hub CSPM 経由で結果を送信する
StackRox – Kubernetes セキュリティ	Security Hub CSPM 経由で結果を送信する
サモロジック – マシンデータ分析	Security Hub CSPM 経由で結果を送信する
Symantec – クラウドワークロード保護	Security Hub CSPM 経由で結果を送信する
Tenable.io	Security Hub CSPM 経由で結果を送信する
Trend Micro – Cloud One	Security Hub CSPM 経由で結果を送信する
Vectra – Cognito Detect	Security Hub CSPM 経由で結果を送信する
Wiz	Security Hub CSPM 経由で結果を送信する

Partner	統合
Caveonix – Caveonix クラウド	Security Hub CSPM を介して 検出結果を送受信する
Cloud Custodian	Security Hub CSPM を介して 検出結果を送受信する
DisruptOps	Security Hub CSPM を介して 検出結果を送受信する
キオン	Security Hub CSPM を介して 検出結果を送受信する
Turbot	Security Hub CSPM を介して 検出結果を送受信する

Note

このリストには、書き込み時に結果を送信する Security Hub パートナー統合が反映されています。AWS Security Hub は定期的に新しいパートナー統合を追加するため、利用可能なパートナーのup-to-dateリストについては、[「Security Hub ユーザーガイド」の「Security Hub CSPM とのサードパーティー製品統合」](#)を参照してください。AWS

Amazon S3 へのログデータのエクスポート

この章では、ロググループのログデータを Amazon S3 バケットにエクスポートし、このデータをカスタム処理や分析で使用したり、別のシステムの読み込んだりする方法を説明します。同じアカウントまたは別のアカウントの S3 バケットにエクスポートできます。

以下を行うことができます。

- AWS Key Management Service (AWS KMS) で SSE-KMS によって暗号化された S3 バケットにログデータをエクスポートする
- 保持期間が設定されている S3 Object Lock が有効になっている S3 バケットに、ログデータをエクスポートする

ログを継続的にアーカイブする方法として、Amazon S3 に定期的にエクスポートしないことをお勧めします。そのユースケースでは、代わりにサブスクリプションを使用することをお勧めします。サブスクリプションの詳細については「[サブスクリプションを使用したログデータのリアルタイム処理](#)」を参照してください。

エクスポート処理を開始するには、エクスポートされたログデータを保存する S3 バケットを作成する必要があります。エクスポートしたファイルを S3 バケットに保存し、Amazon S3 ライフサイクルルールを定義すると、エクスポートしたファイルを自動的にアーカイブまたは削除することができます。

AES-256 または SSE-KMS で暗号化された S3 バケットへのエクスポートが可能です。DSSE-KMS で暗号化されたバケットへのエクスポートはサポートされていません。

複数のロググループからのログや、複数の時間範囲のログを同じ S3 バケットにエクスポートできます。エクスポートされたデータを整理するには、エクスポートされたすべてのオブジェクトの Amazon S3 キープレフィックスとして使用するエクスポートタスクごとにプレフィックスを指定します。例えば、`prod/app-logs/2026-01-03/`、`log-group-name/backup/` などです。

Note

エクスポートされたファイル内のログデータのチャンクに対する時間ベースのソートは保証されません。Linux ユーティリティを使用して、エクスポートされたログフィールドデータをソートできます。例えば、次のユーティリティコマンドは、1 つのフォルダー内のすべての `.gz` ファイルのイベントをソートします。

```
find . -exec zcat {} + | sed -r 's/^[0-9]+\x0&/' | sort -z
```

次のユーティリティコマンドは、複数のサブフォルダにある.gz ファイルをソートします。

```
find ./ */ -type f -exec zcat {} + | sed -r 's/^[0-9]+\x0&/' | sort -z
```

さらに、別の stdout コマンドを使用して、ソートされた出力を別のファイルにパイプして保存することもできます。

ログデータは、エクスポートできるようになるまで最大 12 時間かかる場合があります。エクスポートタスクは 24 時間後にタイムアウトします。エクスポートタスクがタイムアウトする場合は、エクスポートタスクを作成するときの時間範囲を短くしてください。

ほぼリアルタイムのログデータ分析については、[CloudWatch Logs Insights を使用したログデータの分析](#) または [サブスクリプションを使用したログデータのリアルタイム処理](#) を参照してください。

内容

- [概念](#)
- [コンソールを使用してログデータを Amazon S3 にエクスポートする](#)
- [を使用してログデータを Amazon S3 にエクスポートする AWS CLI](#)
- [エクスポートタスクを記述する \(CLI\)](#)
- [エクスポートタスクをキャンセルする \(CLI\)](#)

概念

作業を開始する前に、エクスポートに関する以下の概念を理解してください。

log group name

エクスポートタスクに関連付けられるロググループの名前。このロググループのログデータが、指定された S3 バケットにエクスポートされます。

開始日 (タイムスタンプ)

1970 年 1 月 1 日 00:00:00 UTC からの経過ミリ秒数で表されるタイムスタンプであり、指定は必須です。この時間以降に取り込まれたロググループのすべてのログイベントがエクスポートされます。

終了日 (タイムスタンプ)

1970 年 1 月 1 日 00:00:00 UTC からの経過ミリ秒数で表されるタイムスタンプであり、指定は必須です。この時刻より前に取り込まれたロググループのすべてのログイベントがエクスポートされます。

送信先バケット

エクスポートタスクに関連付けられている S3 バケットの名前。このバケットは、指定されたロググループのログデータをエクスポートするために使用されます。

送信先プレフィックス

エクスポートされたすべてのオブジェクトの Amazon S3 キープレフィックスとして使用される、オプションの属性。バケット内でフォルダのような構成を作成するのに役立ちます。

コンソールを使用してログデータを Amazon S3 にエクスポートする

以下の例では、Amazon CloudWatch コンソールを使用して、my-log-group という名前の Amazon CloudWatch Logs ロググループからすべてのデータを amzn-s3-demo-bucket という名前の Amazon S3 バケットにエクスポートします。

SSE-KMS によって暗号化された S3 バケットへのログデータのエクスポートは、サポートされています。DSSE-KMS で暗号化されたバケットへのエクスポートはサポートされていません。

エクスポートの設定方法の詳細は、エクスポート先の Amazon S3 バケットがエクスポート対象のログと同じアカウントにあるか、別のアカウントにあるかによって異なります。

トピック

- [同一アカウントのエクスポート \(コンソール\)](#)
- [クロスアカウントエクスポート \(コンソール\)](#)

同一アカウントのエクスポート (コンソール)

Amazon S3 バケットがエクスポート対象のログと同じアカウントにある場合は、このセクションの手順を使用してください。

トピック

- [Amazon S3 バケットを作成する \(コンソール\)](#)
- [アクセス許可を設定する \(コンソール\)](#)
- [Amazon S3 バケットに対するアクセス許可の設定 \(コンソール\)](#)
- [\(オプション\) SSE-KMS で暗号化された送信先 Amazon S3 バケットへのエクスポート \(コンソール\)](#)
- [エクスポートタスクを作成する \(コンソール\)](#)

Amazon S3 バケットを作成する (コンソール)

CloudWatch Logs 専用で作成したバケットを使用することをお勧めします。ただし、既存のバケットを使用する場合は、ステップ 2 に進むことができます。

Note

Amazon S3 バケットは、エクスポートするログデータと同じリージョンに存在している必要があります。CloudWatch Logs は、別のリージョンの Amazon S3 バケットへのデータのエクスポートをサポートしていません。

Amazon S3 バケットを作成するには

1. Amazon S3 コンソール (<https://console.aws.amazon.com/s3/>) を開きます。
2. 必要に応じて、リージョンを変更します。ナビゲーションバーから、CloudWatch Logs があるリージョンを選択します。
3. バケットの作成 を選択します。
4. [バケット名] にバケットの名前を入力します。
5. [Region (リージョン)] で、CloudWatch Logs データが存在するリージョンを選択します。
6. [作成] を選択します。

アクセス許可を設定する (コンソール)

エクスポートタスクを作成するには、IAM AmazonS3ReadOnlyAccess ロールと次のアクセス許可を使用してサインオンする必要があります。

- logs:CreateExportTask
- logs:CancelExportTask
- logs:DescribeExportTasks
- logs:DescribeLogStreams
- logs:DescribeLogGroups

アクセスを提供するには、ユーザー、グループ、またはロールにアクセス許可を追加します。

- のユーザーとグループ AWS IAM アイデンティティセンター:

アクセス許可セットを作成します。「AWS IAM アイデンティティセンター ユーザーガイド」の「[アクセス許可セットを作成する](#)」の手順に従ってください。

- IAM 内で、ID プロバイダーによって管理されているユーザー:

ID フェデレーションのロールを作成します。詳細については IAM ユーザーガイドの [サードパーティー ID プロバイダー \(フェデレーション\) 用のロールを作成する](#) を参照してください。

- IAM ユーザー:

- ユーザーが担当できるロールを作成します。手順については IAM ユーザーガイドの [IAM ユーザーのロールの作成](#) を参照してください。

- (お奨めできない方法) ポリシーをユーザーに直接アタッチするか、ユーザーをユーザーグループに追加します。IAM ユーザーガイドの [ユーザー \(コンソール\) へのアクセス許可の追加](#) の指示に従います。

Amazon S3 バケットに対するアクセス許可の設定 (コンソール)

デフォルトでは、すべての Amazon S3 バケットとオブジェクトはプライベートです。バケットを AWS アカウント 作成した リソース所有者のみが、バケットとそれに含まれるオブジェクトにアクセスできます。ただし、リソース所有者は、アクセスポリシーを記述することで他のリソースおよびユーザーにアクセス権限を付与することができます。

ポリシーを設定する場合は、ランダムに生成された文字列をバケットのプレフィックスとして含めることをお勧めします。これにより、意図したログストリームのみがバケットにエクスポートされます。

Important

Amazon S3 バケットへのエクスポートをより安全にするために、ログデータを S3 バケットにエクスポートできるソースアカウントのリストの指定が必要になりました。

次の例では、aws:SourceAccount キー内のアカウント ID のリストにあるのは、ユーザーがログデータを Amazon S3 バケットにエクスポートできるアカウントです。aws:SourceArn キーは、アクションが実行される対象のリソースです。これを特定のロググループに制限することも、この例のようにワイルドカードを使用することもできます。

S3 バケットが作成されたアカウントのアカウント ID も含めることで、エクスポートを同じアカウント内で行えるようにすることをお勧めします。

Amazon S3 バケットに対する権限を設定するには

1. Amazon S3 コンソールで、作成したバケットを選択します。
2. [Permissions (アクセス許可)]、[Add bucket policy (バケットポリシーの追加)] の順に選択します。
3. [Bucket Policy Editor] (バケットポリシーエディタ) で、以下のポリシーを追加します。amzn-s3-demo-bucket を Amazon S3 バケットの名前に変更します。[プリンシパル] に us-west-1 などの正しいリージョンエンドポイントを指定してください。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowCloudWatchLogsGetBucketAcl",
      "Action": "s3:GetBucketAcl",
      "Effect": "Allow",
      "Resource": "arn:aws:s3:::amzn-s3-demo-bucket",
      "Principal": { "Service": "logs.us-east-1.amazonaws.com" },
    }
  ]
}
```

```

    "Condition": {
      "StringEquals": {
        "aws:SourceAccount": [
          "123456789012",
          "111122223333"
        ]
      },
      "ArnLike": {
        "aws:SourceArn": [
          "arn:aws:logs:us-east-1:123456789012:log-group:*",
          "arn:aws:logs:us-east-1:111122223333:log-group:*"
        ]
      }
    }
  },
  {
    "Sid": "AllowCloudWatchLogsPutObject",
    "Action": "s3:PutObject",
    "Effect": "Allow",
    "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*",
    "Principal": { "Service": "logs.us-east-1.amazonaws.com" },
    "Condition": {
      "StringEquals": {
        "s3:x-amz-acl": "bucket-owner-full-control",
        "aws:SourceAccount": [
          "123456789012",
          "111122223333"
        ]
      },
      "ArnLike": {
        "aws:SourceArn": [
          "arn:aws:logs:us-east-1:123456789012:log-group:*",
          "arn:aws:logs:us-east-1:111122223333:log-group:*"
        ]
      }
    }
  }
]
}

```

4. [Save] を選択して、バケットに対するアクセスポリシーとして追加したポリシーを設定します。このポリシーにより、CloudWatch Logs が Amazon S3 バケットにログデータをエクスポート

トできるようになります。バケット所有者には、エクスポートされたすべてのオブジェクトに対する完全なアクセス権限があります。

 Warning

バケットにアタッチされているポリシーがすでに 1 つ以上ある場合は、そのポリシーに CloudWatch Logs アクセスのステートメントを追加します。バケットにアクセスするユーザーに適したアクセス許可であることを確認するために、アクセス許可の結果セットを評価することをお勧めします。

(オプション) SSE-KMS で暗号化された送信先 Amazon S3 バケットへのエクスポート (コンソール)

このステップは、でサーバー側の暗号化を使用する Amazon S3 バケットにエクスポートする場合にのみ必要です AWS KMS keys。この暗号化は SSE-KMS と呼ばれます。

SSE-KMS で暗号化されたバケットにエクスポートするには

1. <https://console.aws.amazon.com/kms> で AWS KMS コンソールを開きます。
2. を変更するには AWS リージョン、ページの右上隅にあるリージョンセレクターを使用します。
3. 左のナビゲーションバーで、[Customer managed keys] (カスタマーマネージドキー) を選択します。
4. [Create Key] (キーを作成) を選択します。
5. [キーの種類] で、[対称] を選択します。
6. [Key usage] (キーの使用) で、[Encrypt and decrypt] (暗号化および復号化)、[Next] (次へ) の順に選択します。
7. [Add labels] (ラベルを追加) で、キーのエイリアスを入力し、オプションで説明またはタグを追加します。次いで、[次へ] を選択します。
8. [Key administrators] (キー管理者) で、このキーを管理できるユーザーを選択した後、[Next] (次へ) を選択します。
9. [Define key usage permissions] (キーの使用アクセス許可を定義) の設定は変更せずに、[Next] (次へ) を選択します。
9. 設定した内容を確認し、[Finish] (終了) を選択します。

10. [Customer managed keys] (カスタマーマネージドキー) ページに戻り、この前に作成したキーの名前を選択します。
11. [Key policy] (キーポリシー) タブを表示し、次に [Switch to policy view] (ポリシービューへの切り替え) を選択します。
12. [Key policy] (キーポリシー) セクションで、[Edit] (編集) を選択します。
13. キーポリシーステートメントのリストに、次のステートメントを追加します。これを行う際、*Region* は実際のログのリージョンに置き換え、*account-ARN* は KMS キーを所有するアカウントの ARN に置き換えます。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Allow CWL Service Principal usage",
      "Effect": "Allow",
      "Principal": {
        "Service": "logs.Region.amazonaws.com"
      },
      "Action": [
        "kms:GenerateDataKey",
        "kms:Decrypt"
      ],
      "Resource": "*"
    },
    {
      "Sid": "Enable IAM User Permissions",
      "Effect": "Allow",
      "Principal": {
        "AWS": "account-ARN"
      },
      "Action": [
        "kms:GetKeyPolicy*",
        "kms:PutKeyPolicy*",
        "kms:DescribeKey*",
        "kms:CreateAlias*",
        "kms:ScheduleKeyDeletion*",
        "kms:Decrypt"
      ],
      "Resource": "*"
    }
  ]
}
```

```
}  
]  
}
```

14. [Save changes] (変更の保存) をクリックします。
15. Amazon S3 コンソール (<https://console.aws.amazon.com/s3/>) を開きます。
16. [S3 バケットを作成する \(CLI\)](#) で作成したバケットを検索し、その名前を選択します。
17. [プロパティ] タブを選択します。[Default Encryption] (デフォルトの暗号化) で、[Edit] (編集) を選択します。
18. [Server-side Encryption] (サーバー側の暗号化) で、[Enable] (有効化) を選択します。
19. 暗号キーで、AWS Key Management Service キー (SSE-KMS) を選択します。
20. AWS KMS キーから選択を選択し、作成したキーを見つけます。
21. [Bucket key] (バケットキー) で、[Enable] (有効化) を選択します。
22. [Save changes] (変更の保存) をクリックします。

エクスポートタスクを作成する (コンソール)

この手順では、ロググループからログをエクスポートするためのエクスポートタスクを作成します。

CloudWatch コンソールを使用して Amazon S3 にデータをエクスポートするには

1. [アクセス許可を設定する \(コンソール\)](#) に記載されているように、十分なアクセス許可を使用してサインインします。
2. CloudWatch コンソールの <https://console.aws.amazon.com/cloudwatch/> を開いてください。
3. ナビゲーションペインで、[Log groups] (ロググループ) を選択します。
4. [ロググループ] 画面で、ロググループの名前を選択します。
5. [Actions (アクション)]、[Export data to Amazon S3 (データを Amazon S3 にエクスポート)] の順に選択します。
6. [Export data to Amazon S3 (データを Amazon S3 にエクスポート)] 画面の [Define data export (データエクスポートを定義)] で、[From (開始)] と [To (終了)] を使用してデータをエクスポートする時間の範囲を設定します。
7. ロググループに複数のログストリームがある場合は、特定のストリームのロググループデータを制限するログストリームプレフィックスを指定できます。[Advanced (詳細設定)] を選択して、[ストリームプレフィックス] にログストリームプレフィックスを入力します。

8. [Choose S3 bucket] (S3 バケットの選択) で、S3 バケットに関連付けられたアカウントを選択します。
9. [S3 バケット名] で、S3 バケットを選択します。
10. [S3 バケットプレフィックス] にバケットポリシーで指定した、ランダムに生成された文字列を入力します。
11. [Export (エクスポート)] を選択して、ログデータを Amazon S3 にエクスポートします。
12. Amazon S3 にエクスポートしたログデータのステータスを表示するには、[Actions (アクション)]、[View all exports to Amazon S3 (Amazon S3 へのすべてのエクスポートを表示)] を選択します。

クロスアカウントエクスポート (コンソール)

Amazon S3 バケットがエクスポート対象のログとは別のアカウントにある場合は、このセクションの手順を使用してください。

トピック

- [クロスアカウントエクスポート用の Amazon S3 バケットを作成する \(コンソール\)](#)
- [クロスアカウントエクスポートのアクセス許可を設定する \(コンソール\)](#)
- [クロスアカウントエクスポート用に S3 バケットにアクセス許可を設定する \(コンソール\)](#)
- [\(オプション\) クロスアカウントエクスポート用に SSE-KMS で暗号化された送信先 Amazon S3 バケットへのエクスポート \(コンソール\)](#)
- [クロスアカウントエクスポートのエクスポートタスクを作成する \(コンソール\)](#)

クロスアカウントエクスポート用の Amazon S3 バケットを作成する (コンソール)

CloudWatch Logs 専用に作成したバケットを使用することをお勧めします。ただし、既存のバケットを使用する場合は、この手順をスキップできます。

Note

Amazon S3 バケットは、エクスポートするログデータと同じリージョンに存在している必要があります。CloudWatch Logs は、別のリージョンの Amazon S3 バケットへのデータのエクスポートをサポートしていません。

Amazon S3 バケットを作成するには

1. Amazon S3 コンソール (<https://console.aws.amazon.com/s3/>) を開きます。
2. 必要に応じて、リージョンを変更します。ナビゲーションバーから、CloudWatch Logs があるリージョンを選択します。
3. バケットの作成 を選択します。
4. [バケット名] にバケットの名前を入力します。
5. [Region (リージョン)] で、CloudWatch Logs データが存在するリージョンを選択します。
6. [作成] を選択します。

クロスアカウントエクスポートのアクセス許可を設定する (コンソール)

まず、新しい IAM ポリシーを作成して、CloudWatch Logs が送信先アカウントの宛先 Amazon S3 バケットへの `s3:PutObject` アクションを有効にする必要があります。

`s3:PutObject` アクションに加えて、ポリシーに含まれる追加のアクションは、送信先バケットが AWS KMS 暗号化を使用しているか、[S3 オブジェクト所有権](#)設定を使用して ACLs が有効になっているかによって異なります。

- KMS 暗号化を使用している場合は、キーリソースの `kms:GenerateDataKey` および `kms:Decrypt` アクションを追加します。
- バケットで ACL が有効化されている場合、バケットリソースに `s3:PutObjectAcl` アクションを追加します。

次のポリシーで、を送信先 S3 バケットの名前 `amzn-s3-demo-bucket` に変更します。

Amazon S3 バケットにログをエクスポートする IAM ポリシーを作成するには

1. IAM コンソール (<https://console.aws.amazon.com/iam/>) を開きます。
2. 左側のナビゲーションペインで、[ポリシー] を選択します。
3. [Create policy] (ポリシーの作成) を選択します。
4. [ポリシーエディター] セクションで、[JSON] を選択します。
5. 送信先バケットが AWS KMS 暗号化を使用しない場合は、次のポリシーをエディタに貼り付けます。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "s3:PutObject",
      "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*"
    }
  ]
}
```

送信先バケットが AWS KMS 暗号化を使用している場合は、次のポリシーをエディタに貼り付けます。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "s3:PutObject",
      "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "kms:GenerateDataKey",
        "kms:Decrypt"
      ],
      "Resource": "arn:aws:kms:us-east-1:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab"
    }
  ]
}
```

送信先バケットで ACLsは、上記のポリシーで s3:PutObjectAcl を s3:PutObject Action ブロックに追加します。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:PutObjectAcl"
      ],
      "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*"
    }
  ]
}
```

6. [次へ] を選択します。
7. ポリシー名を入力します。この名前を使用して、ポリシーを IAM ロールにアタッチします。
8. 次に、[ポリシーの作成] を選択してポリシーを保存します。

エクスポートタスクを作成するには、AmazonS3ReadOnlyAccess管理ポリシーがアタッチされた IAM ロール、上記で作成された IAM ポリシー、および次のアクセス許可を使用してサインインする必要があります。

- logs:CreateExportTask
- logs:CancelExportTask
- logs:DescribeExportTasks
- logs:DescribeLogStreams
- logs:DescribeLogGroups

アクセスを提供するには、ユーザー、グループ、またはロールにアクセス許可を追加します。

- のユーザーとグループ AWS IAM アイデンティティセンター:

アクセス許可セットを作成します。「AWS IAM アイデンティティセンター ユーザーガイド」の「[アクセス許可セットを作成する](#)」の手順に従ってください。

- IAM 内で、ID プロバイダーによって管理されているユーザー:

ID フェデレーションのロールを作成します。詳細については IAM ユーザーガイド の [サードパーティー ID プロバイダー \(フェデレーション\) 用のロールを作成する](#) を参照してください。

- IAM ユーザー:
 - ユーザーが担当できるロールを作成します。手順については IAM ユーザーガイド の [IAM ユーザーのロールの作成](#) を参照してください。
 - (お奨めできない方法) ポリシーをユーザーに直接アタッチするか、ユーザーをユーザーグループに追加します。IAM ユーザーガイド の [ユーザー \(コンソール\) へのアクセス許可の追加](#) の指示に従います。

クロスアカウントエクスポート用に S3 バケットにアクセス許可を設定する (コンソール)

すべての S3 バケットとオブジェクトは、デフォルト状態でプライベートに設定されます。バケットを AWS アカウント 作成した リソース所有者のみが、バケットとそれに含まれるオブジェクトにアクセスできます。ただし、リソース所有者は、アクセスポリシーを記述することで他のリソースおよびユーザーにアクセス権限を付与することができます。

ポリシーを設定する場合は、ランダムに生成された文字列をバケットのプレフィックスとして含めることをお勧めします。これにより、意図したログストリームのみがバケットにエクスポートされます。

Important

S3 バケットへのエクスポートをより安全にするために、ログデータを S3 バケットにエクスポートできるソースアカウントのリストの指定が必要になりました。

次の例では、aws:SourceAccount キー内のアカウント ID のリストにあるのは、ユーザーがログデータを S3 バケットにエクスポートできるアカウントです。aws:SourceArn キーは、アクションが実行される対象のリソースです。これを特定のロググループに制限することも、この例のようにワイルドカードを使用することもできます。

S3 バケットが作成されたアカウントのアカウント ID も含めることで、エクスポートを同じアカウント内で行えるようにすることをお勧めします。

Amazon S3 バケットに対する権限を設定するには

1. Amazon S3 コンソールで、作成したバケットを選択します。
2. [Permissions (アクセス許可)]、[Add bucket policy (バケットポリシーの追加)] の順に選択します。
3. [Bucket Policy Editor] (バケットポリシーエディタ) で、以下のポリシーを追加します。amzn-s3-demo-bucket を Amazon S3 バケットの名前に変更します。[プリンシパル] に us-east-1 などの正しいリージョンエンドポイントを指定してください。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": "s3:GetBucketAcl",
      "Effect": "Allow",
      "Resource": "arn:aws:s3:::amzn-s3-demo-bucket",
      "Principal": { "Service": "logs.us-east-1.amazonaws.com" },
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": [
            "123456789012",
            "111122223333"
          ]
        },
        "ArnLike": {
          "aws:SourceArn": [
            "arn:aws:logs:us-east-1:123456789012:log-group:*",
            "arn:aws:logs:us-east-1:111122223333:log-group:*"
          ]
        }
      }
    },
    {
      "Action": "s3:PutObject",
      "Effect": "Allow",
      "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*",
      "Principal": { "Service": "logs.us-east-1.amazonaws.com" },
      "Condition": {
```

```

    "StringEquals": {
      "s3:x-amz-acl": "bucket-owner-full-control",
      "aws:SourceAccount": [
        "123456789012",
        "111122223333"
      ]
    },
    "ArnLike": {
      "aws:SourceArn": [
        "arn:aws:logs:us-east-1:123456789012:log-group:*",
        "arn:aws:logs:us-east-1:111122223333:log-group:*"
      ]
    }
  },
  {
    "Effect": "Allow",
    "Principal": {
      "AWS": "arn:aws:iam::111122223333:role/role_name"
    },
    "Action": "s3:PutObject",
    "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*",
    "Condition": {
      "StringEquals": {
        "s3:x-amz-acl": "bucket-owner-full-control"
      }
    }
  }
]
}

```

4. [Save] を選択して、バケットに対するアクセスポリシーとして追加したポリシーを設定します。このポリシーにより、CloudWatch Logs は S3 バケットにログデータをエクスポートできるようになります。バケット所有者には、エクスポートされたすべてのオブジェクトに対する完全なアクセス権限があります。

Warning

バケットにアタッチされているポリシーがすでに 1 つ以上ある場合は、そのポリシーに CloudWatch Logs アクセスのステートメントを追加します。バケットにアクセスする

ユーザーに適したアクセス許可であることを確認するために、アクセス許可の結果セットを評価することをお勧めします。

(オプション) クロスアカウントエクスポート用に SSE-KMS で暗号化された送信先 Amazon S3 バケットへのエクスポート (コンソール)

この手順は、でサーバー側の暗号化を使用する S3 バケットにエクスポートする場合にのみ必要です AWS KMS keys。この暗号化は SSE-KMS と呼ばれます。

SSE-KMS で暗号化されたバケットにエクスポートするには

1. <https://console.aws.amazon.com/kms> で AWS KMS コンソールを開きます。
2. を変更するには AWS リージョン、ページの右上隅にあるリージョンセクターを使用します。
3. 左のナビゲーションバーで、[Customer managed keys] (カスタマーマネージドキー) を選択します。

[Create Key] (キーを作成) を選択します。
4. [キーの種類] で、[対称] を選択します。
5. [Key usage] (キーの使用) で、[Encrypt and decrypt] (暗号化および復号化)、[Next] (次へ) の順に選択します。
6. [Add labels] (ラベルを追加) で、キーのエイリアスを入力し、オプションで説明またはタグを追加します。次いで、[次へ] を選択します。
7. [Key administrators] (キー管理者) で、このキーを管理できるユーザーを選択した後、[Next] (次へ) を選択します。
8. [Define key usage permissions] (キーの使用アクセス許可を定義) の設定は変更せずに、[Next] (次へ) を選択します。
9. 設定した内容を確認し、[Finish] (終了) を選択します。
10. [Customer managed keys] (カスタマーマネージドキー) ページに戻り、この前に作成したキーの名前を選択します。
11. [Key policy] (キーポリシー) タブを表示し、次に [Switch to policy view] (ポリシービューへの切り替え) を選択します。
12. [Key policy] (キーポリシー) セクションで、[Edit] (編集) を選択します。
13. キーポリシーステートメントのリストに、次のステートメントを追加します。これを行うときは、*us-east-1* をログのリージョンに置き換え、*account-ARN* を KMS キーを所有するアカ

ウントの ARN に置き換え、**123456789012** を KMS キーを所有するアカウント番号に置き換え、**key_id** を kms-key Id に置き換え、**role_name** をエクスポートタスクの作成に使用されるロールに置き換えます。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Allow CWL Service Principal usage",
      "Effect": "Allow",
      "Principal": {
        "Service": "logs.us-east-1.amazonaws.com"
      },
      "Action": [
        "kms:GenerateDataKey",
        "kms:Decrypt"
      ],
      "Resource": "*"
    },
    {
      "Sid": "Enable IAM User Permissions",
      "Effect": "Allow",
      "Principal": {
        "AWS": "account-ARN"
      },
      "Action": [
        "kms:GetKeyPolicy*",
        "kms:PutKeyPolicy*",
        "kms:DescribeKey*",
        "kms:CreateAlias*",
        "kms:ScheduleKeyDeletion*",
        "kms:Decrypt"
      ],
      "Resource": "*"
    },
    {
      "Sid": "Enable IAM Role Permissions",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:role/role_name"
      }
    }
  ]
}
```

```
    },
    "Action": [
        "kms:GenerateDataKey",
        "kms:Decrypt"
    ],
    "Resource": "arn:aws:kms:us-east-1:123456789012:key/key-id"
}
]
```

14. [Save changes] (変更の保存) をクリックします。
15. Amazon S3 コンソール (<https://console.aws.amazon.com/s3/>) を開きます。
16. [S3 バケットを作成する \(CLI\)](#) で作成したバケットを検索し、その名前を選択します。
17. [プロパティ] タブを選択します。[Default Encryption] (デフォルトの暗号化) で、[Edit] (編集) を選択します。
18. [Server-side Encryption] (サーバー側の暗号化) で、[Enable] (有効化) を選択します。
19. 暗号キーで、AWS Key Management Service キー (SSE-KMS) を選択します。
20. AWS KMS キーから選択を選択し、作成したキーを見つけます。
21. [Bucket key] (バケットキー) で、[Enable] (有効化) を選択します。
22. [Save changes] (変更の保存) をクリックします。

クロスアカウントエクスポートのエクスポートタスクを作成する (コンソール)

この手順では、ロググループからログをエクスポートするためのエクスポートタスクを作成します。

CloudWatch コンソールを使用して Amazon S3 にデータをエクスポートするには

1. [アクセス許可を設定する \(コンソール\)](#) に記載されているように、十分なアクセス許可を使用してサインインします。
2. CloudWatch コンソールの <https://console.aws.amazon.com/cloudwatch/> を開いてください。
3. ナビゲーションペインで、[Log groups] (ロググループ) を選択します。
4. [ロググループ] 画面で、ロググループの名前を選択します。
5. [Actions (アクション)], [Export data to Amazon S3 (データを Amazon S3 にエクスポート)] の順に選択します。

6. [Export data to Amazon S3 (データを Amazon S3 にエクスポート)] 画面の [Define data export (データエクスポートを定義)] で、[From (開始)] と [To (終了)] を使用してデータをエクスポートする時間の範囲を設定します。
7. ロググループに複数のログストリームがある場合は、特定のストリームのロググループデータを制限するログストリームプレフィックスを指定できます。[Advanced (詳細設定)] を選択して、[ストリームプレフィックス] にログストリームプレフィックスを入力します。
8. [Choose S3 bucket] (S3 バケットの選択) で、S3 バケットに関連付けられたアカウントを選択します。
9. [S3 bucket name] (S3 バケット名) で、バケットを選択します。
10. [S3 バケットプレフィックス] にバケットポリシーで指定した、ランダムに生成された文字列を入力します。
11. [Export (エクスポート)] を選択して、ログデータを Amazon S3 にエクスポートします。
12. Amazon S3 にエクスポートしたログデータのステータスを表示するには、[Actions (アクション)]、[View all exports to Amazon S3 (Amazon S3 へのすべてのエクスポートを表示)] を選択します。

を使用してログデータを Amazon S3 にエクスポートする AWS CLI

次の例では、エクスポートタスクを使用して、すべてのデータを my-log-group という名前の CloudWatch Logs ロググループから amzn-s3-demo-bucket という名前の Amazon S3 バケットにエクスポートします。この例では、「my-log-group」というロググループを作成済みであることを前提としています。

で暗号化された S3 バケットへのログデータのエクスポート AWS KMS がサポートされています。DSSE-KMS で暗号化されたバケットへのエクスポートはサポートされていません。

エクスポートの設定方法の詳細は、エクスポート先の Amazon S3 バケットがエクスポート対象のログと同じアカウントにあるか、別のアカウントにあるかによって異なります。

トピック

- [同一アカウントのエクスポート \(CLI\)](#)
- [クロスアカウントエクスポート \(CLI\)](#)

同一アカウントのエクスポート (CLI)

Amazon S3 バケットがエクスポート対象のログと同じアカウントにある場合は、このセクションの手順を使用してください。

トピック

- [S3 バケットを作成する \(CLI\)](#)
- [アクセス許可の設定 \(CLI\)](#)
- [S3 バケットに対するアクセス許可の設定 \(CLI\)](#)
- [\(オプション\) SSE-KMS で暗号化された送信先 Amazon S3 バケットへのエクスポート \(CLI\)](#)
- [エクスポートタスクを作成する \(CLI\)](#)

S3 バケットを作成する (CLI)

CloudWatch Logs 専用で作成したバケットを使用することをお勧めします。ただし、既存のバケットを使用する場合は、この手順をスキップできます。

Note

S3 バケットは、エクスポートするログデータと同じリージョンに存在している必要があります。CloudWatch Logs は、別のリージョンの S3 バケットへのデータのエクスポートをサポートしていません。

を使用して S3 バケットを作成するには AWS CLI

コマンドプロンプトで、次の [create-bucket](#) コマンドを実行します。ここで、LocationConstraint はログデータをエクスポートするリージョンです。

```
aws s3api create-bucket --bucket amzn-s3-demo-bucket --create-bucket-configuration  
LocationConstraint=us-east-2
```

以下は出力の例です。

```
{  
  "Location": "/amzn-s3-demo-bucket"
```

```
}
```

アクセス許可の設定 (CLI)

後でエクスポートタスクを作成するには、IAM AmazonS3ReadOnlyAccess ロールと次のアクセス許可を使用してサインオンする必要があります。

- logs:CreateExportTask
- logs:CancelExportTask
- logs:DescribeExportTasks
- logs:DescribeLogStreams
- logs:DescribeLogGroups

アクセスを提供するには、ユーザー、グループ、またはロールにアクセス許可を追加します。

- のユーザーとグループ AWS IAM アイデンティティセンター:

アクセス許可セットを作成します。「AWS IAM アイデンティティセンター ユーザーガイド」の「[アクセス許可セットを作成する](#)」の手順に従ってください。

- IAM 内で、ID プロバイダーによって管理されているユーザー:

ID フェデレーションのロールを作成します。詳細については IAM ユーザーガイドの [サードパーティー ID プロバイダー \(フェデレーション\) 用のロールを作成する](#) を参照してください。

- IAM ユーザー:

- ユーザーが担当できるロールを作成します。手順については IAM ユーザーガイドの [IAM ユーザーのロールの作成](#) を参照してください。

- (お奨めできない方法) ポリシーをユーザーに直接アタッチするか、ユーザーをユーザーグループに追加します。IAM ユーザーガイドの [ユーザー \(コンソール\) へのアクセス許可の追加](#) の指示に従います。

S3 バケットに対するアクセス許可の設定 (CLI)

すべての S3 バケットとオブジェクトは、デフォルト状態でプライベートに設定されます。バケットを作成したアカウント (リソース所有者) のみが、バケットとそれに含まれるオブジェクトにアクセスできます。ただし、リソース所有者は、アクセスポリシーを記述することで他のリソースおよびユーザーにアクセス権限を付与することができます。

⚠ Important

S3 バケットへのエクスポートをより安全にするために、ログデータを S3 バケットにエクスポートできるソースアカウントのリストの指定が必要になりました。

次の例では、aws:SourceAccount キー内のアカウント ID のリストにあるのは、ユーザーがログデータを S3 バケットにエクスポートできるアカウントです。aws:SourceArn キーは、アクションが実行される対象のリソースです。これを特定のロググループに制限することも、この例のようにワイルドカードを使用することもできます。

S3 バケットが作成されたアカウントのアカウント ID も含めることで、エクスポートを同じアカウント内で行えるようにすることをお勧めします。

S3 バケットでアクセス許可を設定するには

1. policy.json という名前のファイルを作成し、次のアクセスポリシーを追加します。このとき、amzn-s3-demo-bucket を S3 バケットの名前に変更し、Principal をログデータのエクスポート先のリージョンのエンドポイント (us-east-1 など) に変更します。テキストエディタを使用してこのポリシーファイルを作成します。IAM コンソールを使用しないでください。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowGetBucketAcl",
      "Action": "s3:GetBucketAcl",
      "Effect": "Allow",
      "Resource": "arn:aws:s3:::amzn-s3-demo-bucket",
      "Principal": { "Service": "logs.us-east-1.amazonaws.com" },
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": [
            "123456789012",
            "111122223333"
          ]
        },
        "ArnLike": {
          "aws:SourceArn": [
```

```

        "arn:aws:logs:us-east-1:123456789012:log-group:*",
        "arn:aws:logs:us-east-1:111122223333:log-group:*"
    ]
}
},
{
    "Sid": "AllowPutObject",
    "Action": "s3:PutObject",
    "Effect": "Allow",
    "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*",
    "Principal": { "Service": "logs.us-east-1.amazonaws.com" },
    "Condition": {
        "StringEquals": {
            "s3:x-amz-acl": "bucket-owner-full-control",
            "aws:SourceAccount": [
                "123456789012",
                "111122223333"
            ]
        },
        "ArnLike": {
            "aws:SourceArn": [
                "arn:aws:logs:us-east-1:123456789012:log-group:*",
                "arn:aws:logs:us-east-1:111122223333:log-group:*"
            ]
        }
    }
}
]
}
}

```

2. [put-bucket-policy](#) コマンドを使用して、バケットでアクセスポリシーとして先ほど追加したポリシーを設定します。このポリシーにより、CloudWatch Logs は S3 バケットにログデータをエクスポートできるようになります。バケット所有者には、エクスポートされたすべてのオブジェクトに対する完全なアクセス権限があります。

```
aws s3api put-bucket-policy --bucket amzn-s3-demo-bucket --policy file://policy.json
```

⚠ Warning

バケットにアタッチされているポリシーがすでに 1 つ以上ある場合は、そのポリシーに CloudWatch Logs アクセスのステートメントを追加します。バケットにアクセスするユーザーに適したアクセス許可であることを確認するために、アクセス許可の結果セットを評価することをお勧めします。

(オプション) SSE-KMS で暗号化された送信先 Amazon S3 バケットへのエクスポート (CLI)

この手順は、でサーバー側の暗号化を使用する S3 バケットにエクスポートする場合にのみ必要です AWS KMS keys。この暗号化は SSE-KMS と呼ばれます。

SSE-KMS で暗号化されたバケットにエクスポートするには

1. テキストエディタを使用して `key_policy.json` という名前のファイルを作成し、以下のアクセスポリシーを追加します。ポリシーを追加する際、以下の点を変更します。
 - **Region** を、実際のログのリージョンに置き換えます。
 - **account-ARN** を、KMS キーを所有するアカウントの ARN に置き換えます。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Allow CWL Service Principal usage",
      "Effect": "Allow",
      "Principal": {
        "Service": "logs.Region.amazonaws.com"
      },
      "Action": [
        "kms:GenerateDataKey",
        "kms:Decrypt"
      ],
      "Resource": "*"
    }
  ]
}
```

```
    },
    {
      "Sid": "Enable IAM User Permissions",
      "Effect": "Allow",
      "Principal": {
        "AWS": "account-ARN"
      },
      "Action": [
        "kms:GetKeyPolicy*",
        "kms:PutKeyPolicy*",
        "kms:DescribeKey*",
        "kms:CreateAlias*",
        "kms:ScheduleKeyDeletion*",
        "kms:Decrypt"
      ],
      "Resource": "*"
    }
  ]
}
```

2. 次のコマンドを入力します。

```
aws kms create-key --policy file://key_policy.json
```

以下は、このコマンドに対する出力例です。

```
{
  "KeyMetadata": {
    "AWSAccountId": "account_id",
    "KeyId": "key_id",
    "Arn": "arn:aws:kms:us-east-2:account-ARN:key/key_id",
    "CreationDate": "time",
    "Enabled": true,
    "Description": "",
    "KeyUsage": "ENCRYPT_DECRYPT",
    "KeyState": "Enabled",
    "Origin": "AWS_KMS",
    "KeyManager": "CUSTOMER",
    "CustomerMasterKeySpec": "SYMMETRIC_DEFAULT",
    "KeySpec": "SYMMETRIC_DEFAULT",
    "EncryptionAlgorithms": [
      "SYMMETRIC_DEFAULT"
    ],
  },
}
```

```
"MultiRegion": false
}
```

3. テキストエディタを使用して、`bucketencryption.json` という名前のファイルを作成し、次の内容を記述します。

```
{
  "Rules": [
    {
      "ApplyServerSideEncryptionByDefault": {
        "SSEAlgorithm": "aws:kms",
        "KMSMasterKeyID": "{KMS Key ARN}"
      },
      "BucketKeyEnabled": true
    }
  ]
}
```

4. `amzn-s3-demo-bucket` を、ログをエクスポートするバケットの名前に置き換えて、次のコマンドを入力します。

```
aws s3api put-bucket-encryption --bucket amzn-s3-demo-bucket --server-side-encryption-configuration file://bucketencryption.json
```

コマンドがエラーを返さなければ、このプロセスは成功しています。

エクスポートタスクを作成する (CLI)

次のコマンドを使用してエクスポートタスクを作成します。作成すると、エクスポートするデータのサイズに応じて、エクスポートタスクに数秒から数時間かかる可能性があります。

を使用して Amazon S3 にデータをエクスポートするには AWS CLI

1. [アクセス許可の設定 \(CLI\)](#) に記載されているように、十分なアクセス許可を使用してサインインします。
2. コマンドプロンプトで、次の [create-export-task](#) コマンドを使用してエクスポートタスクを作成します。

```
aws logs create-export-task --profile CWLExportUser --task-name "my-log-group-09-10-2015" --log-group-name "my-log-group" --from 1441490400000 --
```



```
to 1441494000000 --destination "amzn-s3-demo-bucket" --destination-prefix "export-task-output"
```

以下は出力の例です。

```
{
  "taskId": "cda45419-90ea-4db5-9833-aade86253e66"
}
```

クロスアカウントエクスポート (CLI)

Amazon S3 バケットがエクスポート対象のログとは別のアカウントにある場合は、このセクションの手順を使用してください。

トピック

- [クロスアカウントエクスポート用の S3 バケットを作成する \(CLI\)](#)
- [クロスアカウントエクスポートのアクセス許可を設定する \(CLI\)](#)
- [クロスアカウントエクスポート用に S3 バケットにアクセス許可を設定する \(CLI\)](#)
- [\(オプション\) クロスアカウントエクスポート \(CLI\) のために SSE-KMS で暗号化された送信先 Amazon S3 バケットへのエクスポート](#)
- [クロスアカウントエクスポートのエクスポートタスクを作成する \(CLI\)](#)

クロスアカウントエクスポート用の S3 バケットを作成する (CLI)

CloudWatch Logs 専用に作成したバケットを使用することをお勧めします。ただし、既存のバケットを使用する場合は、ステップ 2 に進むことができます。

Note

S3 バケットは、エクスポートするログデータと同じリージョンに存在している必要があります。CloudWatch Logs は、別のリージョンの S3 バケットへのデータのエクスポートをサポートしていません。

を使用して S3 バケットを作成するには AWS CLI

コマンドプロンプトで、次の [create-bucket](#) コマンドを実行します。ここで、LocationConstraint はログデータをエクスポートするリージョンです。

```
aws s3api create-bucket --bucket amzn-s3-demo-bucket --create-bucket-configuration
LocationConstraint=us-east-2
```

以下は出力の例です。

```
{
  "Location": "/amzn-s3-demo-bucket"
}
```

クロスアカウントエクスポートのアクセス許可を設定する (CLI)

まず、新しい IAM ポリシーを作成して、CloudWatch Logs が送信先アカウントの宛先 Amazon S3 バケットへの s3:PutObject アクションを有効にする必要があります。

s3:PutObject アクションに加えて、ポリシーに含まれる追加のアクションは、送信先バケットが AWS KMS 暗号化を使用しているか、[S3 オブジェクト所有権](#)設定を使用して ACLs が有効になっているかによって異なります。

- KMS 暗号化を使用している場合は、キーリソースの kms:GenerateDataKey および kms:Decrypt アクションを追加します。
- バケットで ACL が有効化されている場合、バケットリソースに s3:PutObjectAcl アクションを追加します。

次のポリシーで、を送信先 S3 バケットの名前 **amzn-s3-demo-bucket** に変更します。

作成するポリシーは、宛先バケットが AWS KMS 暗号化を使用するかどうかによって異なります。AWS KMS 暗号化を使用しない場合は、次の内容のポリシーを作成します。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "s3:PutObject",
```

```

        "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*"
    }
]
}

```

送信先バケットが AWS KMS 暗号化を使用している場合は、次の内容のポリシーを作成します。

JSON

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "s3:PutObject",
      "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "kms:GenerateDataKey",
        "kms:Decrypt"
      ],
      "Resource": "arn:aws:kms:us-east-1:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab"
    }
  ]
}

```

送信先バケットで ACLs は、上記のポリシーで s3:PutObjectAcl を s3:PutObject Action ブロックに追加します。

JSON

```

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

```

```
        "s3:PutObject",
        "s3:PutObjectAcl"
    ],
    "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*"
}
]
```

エクスポートタスクを作成するには、AmazonS3ReadOnlyAccess管理ポリシーがアタッチされた IAM ロール、上記で作成された IAM ポリシー、および次のアクセス許可を使用してサインインする必要があります。

- logs:CreateExportTask
- logs:CancelExportTask
- logs:DescribeExportTasks
- logs:DescribeLogStreams
- logs:DescribeLogGroups

アクセスを提供するには、ユーザー、グループ、またはロールにアクセス許可を追加します。

- のユーザーとグループ AWS IAM アイデンティティセンター:

アクセス許可セットを作成します。「AWS IAM アイデンティティセンター ユーザーガイド」の「[アクセス許可セットを作成する](#)」の手順に従ってください。

- IAM 内で、ID プロバイダーによって管理されているユーザー:

ID フェデレーションのロールを作成します。詳細については IAM ユーザーガイドの [サードパーティー ID プロバイダー \(フェデレーション\) 用のロールを作成する](#) を参照してください。

- IAM ユーザー:

- ユーザーが担当できるロールを作成します。手順については IAM ユーザーガイドの [IAM ユーザーのロールの作成](#) を参照してください。

- (お奨めできない方法) ポリシーをユーザーに直接アタッチするか、ユーザーをユーザーグループに追加します。IAM ユーザーガイドの [ユーザー \(コンソール\) へのアクセス許可の追加](#) の指示に従います。

クロスアカウントエクスポート用に S3 バケットにアクセス許可を設定する (CLI)

すべての S3 バケットとオブジェクトは、デフォルト状態でプライベートに設定されます。バケットを作成したアカウント (リソース所有者) のみが、バケットとそれに含まれるオブジェクトにアクセスできます。ただし、リソース所有者は、アクセスポリシーを記述することで他のリソースおよびユーザーにアクセス権限を付与することができます。

⚠ Important

S3 バケットへのエクスポートをより安全にするために、ログデータを S3 バケットにエクスポートできるソースアカウントのリストの指定が必要になりました。

次の例では、aws:SourceAccount キー内のアカウント ID のリストにあるのは、ユーザーがログデータを S3 バケットにエクスポートできるアカウントです。aws:SourceArn キーは、アクションが実行される対象のリソースです。これを特定のロググループに制限することも、この例のようにワイルドカードを使用することもできます。

S3 バケットが作成されたアカウントのアカウント ID も含めることで、エクスポートを同じアカウント内で行えるようにすることをお勧めします。

S3 バケットでアクセス許可を設定するには

1. という名前のファイルを作成し、送信先 S3 バケットの名前 `amzn-s3-demo-bucket` をなどのログデータをエクスポートするリージョンのエンドポイントに変更 `Principal` して、次のアクセスポリシー `policy.json` を追加します `us-west-1`。テキストエディタを使用してこのポリシーファイルを作成します。IAM コンソールを使用しないでください。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": "s3:GetBucketAcl",
      "Effect": "Allow",
      "Resource": "arn:aws:s3:::amzn-s3-demo-bucket",
      "Principal": { "Service": "logs.us-east-1.amazonaws.com" },
      "Condition": {
        "StringEquals": {
```

```

        "aws:SourceAccount": [
            "123456789012",
            "111122223333"
        ],
    },
    "ArnLike": {
        "aws:SourceArn": [
            "arn:aws:logs:us-east-1:123456789012:log-group:*",
            "arn:aws:logs:us-east-1:111122223333:log-group:*"
        ]
    }
},
{
    "Action": "s3:PutObject" ,
    "Effect": "Allow",
    "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*",
    "Principal": { "Service": "logs.us-east-1.amazonaws.com" },
    "Condition": {
        "StringEquals": {
            "s3:x-amz-acl": "bucket-owner-full-control",
            "aws:SourceAccount": [
                "123456789012",
                "111122223333"
            ]
        }
    },
    "ArnLike": {
        "aws:SourceArn": [
            "arn:aws:logs:us-east-1:123456789012:log-group:*",
            "arn:aws:logs:us-east-1:111122223333:log-group:*"
        ]
    }
},
{
    "Effect": "Allow",
    "Principal": {
        "AWS": "arn:aws:iam::111122223333:role/role_name"
    },
    "Action": "s3:PutObject",
    "Resource": "arn:aws:s3:::>amzn-s3-demo-bucket/*",
    "Condition": {
        "StringEquals": {
            "s3:x-amz-acl": "bucket-owner-full-control"
        }
    }
}

```

```
}  
  }  
    }  
  ]  
}
```

2. [put-bucket-policy](#) コマンドを使用して、バケットでアクセスポリシーとして先ほど追加したポリシーを設定します。このポリシーにより、CloudWatch Logs は S3 バケットにログデータをエクスポートできるようになります。バケット所有者には、エクスポートされたすべてのオブジェクトに対する完全なアクセス権限があります。

```
aws s3api put-bucket-policy --bucket amzn-s3-demo-bucket --policy file://  
policy.json
```

 Warning

バケットにアタッチされているポリシーがすでに 1 つ以上ある場合は、そのポリシーに CloudWatch Logs アクセスのステートメントを追加します。バケットにアクセスするユーザーに適したアクセス許可であることを確認するために、アクセス許可の結果セットを評価することをお勧めします。

(オプション) クロスアカウントエクスポート (CLI) のために SSE-KMS で暗号化された送信先 Amazon S3 バケットへのエクスポート

この手順は、でサーバー側の暗号化を使用する S3 バケットにエクスポートする場合にのみ必要です AWS KMS keys。この暗号化は SSE-KMS と呼ばれます。

SSE-KMS で暗号化されたバケットにエクスポートするには

1. テキストエディタを使用して `key_policy.json` という名前のファイルを作成し、以下のアクセスポリシーを追加します。ポリシーを追加する際、以下の点を変更します。
 - `us-east-1` をログのリージョンに置き換えます。
 - `account-ARN` を、KMS キーを所有するアカウントの ARN に置き換えます。
 - `123456789012` を KMS キーを所有するアカウント番号に置き換えます。
 - `key_id` と kms-key ID。
 - `role_name` は、エクスポートタスクの作成に使用されるロールです。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowCWLSERVICEPrincipalUsage",
      "Effect": "Allow",
      "Principal": {
        "Service": "logs.us-east-1.amazonaws.com"
      },
      "Action": [
        "kms:GenerateDataKey",
        "kms:Decrypt"
      ],
      "Resource": "*"
    },
    {
      "Sid": "EnableIAMUserPermissions",
      "Effect": "Allow",
      "Principal": {
        "AWS": "account-ARN"
      },
      "Action": [
        "kms:GetKeyPolicy*",
        "kms:PutKeyPolicy*",
        "kms:DescribeKey*",
        "kms:CreateAlias*",
        "kms:ScheduleKeyDeletion*",
        "kms:Decrypt"
      ],
      "Resource": "*"
    },
    {
      "Sid": "EnableIAMRolePermissions",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:role/role_name"
      },
      "Action": [
```



```

        "kms:GenerateDataKey",
        "kms:Decrypt"
    ],
    "Resource": "arn:aws:kms:us-  
east-1:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab"
}
]
}

```

2. 次のコマンドを入力します。

```
aws kms create-key --policy file://key_policy.json
```

以下は、このコマンドに対する出力例です。

```

{
  "KeyMetadata": {
    "AWSAccountId": "account_id",
    "KeyId": "key_id",
    "Arn": "arn:aws:kms:us-east-1:123456789012:key/key_id",
    "CreationDate": "time",
    "Enabled": true,
    "Description": "",
    "KeyUsage": "ENCRYPT_DECRYPT",
    "KeyState": "Enabled",
    "Origin": "AWS_KMS",
    "KeyManager": "CUSTOMER",
    "CustomerMasterKeySpec": "SYMMETRIC_DEFAULT",
    "KeySpec": "SYMMETRIC_DEFAULT",
    "EncryptionAlgorithms": [
      "SYMMETRIC_DEFAULT"
    ],
    "MultiRegion": false
  }
}

```

3. テキストエディタを使用して、bucketencryption.json という名前のファイルを作成し、次の内容を記述します。

```

{
  "Rules": [
    {
      "ApplyServerSideEncryptionByDefault": {

```

```
    "SSEAlgorithm": "aws:kms",
    "KMSMasterKeyID": "{KMS Key ARN}"
  },
  "BucketKeyEnabled": true
}
]
```

4. 次のコマンドを入力し、*amzn-s3-demo-bucket* を、ログをエクスポートするバケットの名前に置き換えます。

```
aws s3api put-bucket-encryption --bucket amzn-s3-demo-bucket --server-side-encryption-configuration file://bucketencryption.json
```

コマンドがエラーを返さなければ、このプロセスは成功しています。

クロスアカウントエクスポートのエクスポートタスクを作成する (CLI)

次の コマンドを使用してエクスポートタスクを作成します。作成すると、エクスポートするデータのサイズに応じて、エクスポートタスクに数秒から数時間かかる可能性があります。

を使用して Amazon S3 にデータをエクスポートするには AWS CLI

1. [アクセス許可の設定 \(CLI\)](#) に記載されているように、十分なアクセス許可を使用してサインインします。
2. コマンドプロンプトで、次の [create-export-task](#) コマンドを使用してエクスポートタスクを作成します。

```
aws logs create-export-task --profile CWLExportUser --task-name "my-log-group-09-10-2015" --log-group-name "my-log-group" --from 1441490400000 --to 1441494000000 --destination "amzn-s3-demo-bucket" --destination-prefix "export-task-output"
```

以下は出力の例です。

```
{
  "taskId": "cda45419-90ea-4db5-9833-aade86253e66"
}
```

エクスポートタスクを記述する (CLI)

エクスポートタスクを作成すると、タスクの現在のステータスを取得できます。

を使用してエクスポートタスクを記述するには AWS CLI

コマンドプロンプトで、次の [describe-export-tasks](#) コマンドを使用します。

```
aws logs --profile CWLExportUser describe-export-tasks --task-id
"cd445419-90ea-4db5-9833-aade86253e66"
```

以下は出力の例です。

```
{
  "exportTasks": [
    {
      "destination": "my-exported-logs",
      "destinationPrefix": "export-task-output",
      "executionInfo": {
        "creationTime": 1441495400000
      },
      "from": 1441490400000,
      "logGroupName": "my-log-group",
      "status": {
        "code": "RUNNING",
        "message": "Started Successfully"
      },
      "taskId": "cd445419-90ea-4db5-9833-aade86253e66",
      "taskName": "my-log-group-09-10-2015",
      "tTo": 1441494000000
    }
  ]
}
```

describe-export-tasks コマンドを使用する方法は 3 通りあります。

- フィルタなし – すべてのエクスポートタスクが、作成順とは逆の順序でリストされます。
- タスク ID でフィルタリング – 指定された ID のエクスポートタスクが存在する場合に、それらがリストされます。
- タスクステータスによるフィルタリング – 指定されたステータスのエクスポートタスクがリストされます。

例えば、FAILED ステータスでフィルタリングするには、次のコマンドを使用します。

```
aws logs --profile CWLEXPOTUser describe-export-tasks --status-code "FAILED"
```

以下は出力の例です。

```
{
  "exportTasks": [
    {
      "destination": "amzn-s3-demo-bucket",
      "destinationPrefix": "export-task-output",
      "executionInfo": {
        "completionTime": 1441498600000
        "creationTime": 1441495400000
      },
      "from": 1441490400000,
      "logGroupName": "my-log-group",
      "status": {
        "code": "FAILED",
        "message": "FAILED"
      },
      "taskId": "cda45419-90ea-4db5-9833-aade86253e66",
      "taskName": "my-log-group-09-10-2015",
      "to": 1441494000000
    }
  ]
}
```

エクスポートタスクをキャンセルする (CLI)

エクスポートタスクが PENDING または RUNNING の状態にある場合、そのタスクをキャンセルできます。

を使用してエクスポートタスクをキャンセルするには AWS CLI

コマンドプロンプトで、次の [cancel-export-task](#) コマンドを使用します。

```
aws logs --profile CWLEXPOTUser cancel-export-task --task-id "cda45419-90ea-4db5-9833-aade86253e66"
```

[describe-export-tasks](#) コマンドを使用して、タスクが正常にキャンセルされたことを確認できる点に注意してください。

CloudWatch Logs データの Amazon OpenSearch Service へのストリーミング

Amazon CloudWatch Logs でロググループを設定して、ほぼリアルタイムで Amazon OpenSearch Service クラスターにデータをストリーミングできます。詳細については、「[サブスクリプションを使用したログデータのリアルタイム処理](#)」を参照してください。

Note

OpenSearch Service へのストリーミングは、標準ログクラスのロググループでのみサポートされています。ログクラスの詳細については、「[ログクラス](#)」を参照してください。

ストリーミングされるログデータの量に応じて、関数レベルの同時実行制限を設定することを検討してください。詳細については、「[Lambda 関数のスケーリング](#)」を参照してください。

Note

大量の CloudWatch Logs データを OpenSearch Service にストリーミングすると、使用料が高くなる可能性があるため、AWS Billing and Cost Management コンソールで予算を作成することをお勧めします。詳細については、[AWS「Budgets」によるコストの管理](#)」を参照してください。

このセクションでは、ロググループを OpenSearch Service にサブスクライブする前に完了する必要がある前提条件について説明します。また、ロググループを OpenSearch Service にサブスクライブする方法も説明します。

前提条件

開始する前に、OpenSearch Service ドメインを作成します。ドメインにはパブリックアクセスまたは VPC アクセスのいずれかを設定できますが、その場合、ドメインの作成後にアクセスのタイプを変更することはできません。後で OpenSearch Service ドメイン設定を確認し、クラスターで処理されるデータの量に基づいてクラスター設定を変更することができます。ドメインの作成手順については、「[OpenSearch Service のドメインの作成](#)」を参照してください。

OpenSearch Service の詳細については、「[Amazon OpenSearch Service デベロッパーガイド](#)」を参照してください。

ロググループを OpenSearch Service にサブスクライブする

CloudWatch コンソールを使用して、ロググループを OpenSearch Service にサブスクライブできます。

ロググループを OpenSearch Service にサブスクライブするには

1. CloudWatch コンソールの <https://console.aws.amazon.com/cloudwatch/> を開いてください。
2. ナビゲーションペインで、[Log groups] (ロググループ) を選択します。
3. ロググループの名前を選択します。
4. [Actions] (アクション)、[Subscription filters] (サブスクリプションフィルター)、[Create Amazon OpenSearch Service subscription filter] (Amazon OpenSearch Service サブスクリプションフィルターを作成) の順に選択します。
5. このアカウントのクラスターにストリーミングするか、別のアカウントにストリーミングするかを選択します。
 - このアカウントを選択した場合は、前のステップで作成したドメインを選択します。
 - 別のアカウントを選択した場合は、ドメイン ARN とエンドポイントを指定します。
6. [Lambda IAM Execution Role] (Lambda IAM 実行ロール) では、OpenSearch への呼び出しを実行するときに Lambda が使用する IAM ロールを選択します。

選択した IAM ロールは、これらの要件を満たす必要があります。

- 信頼関係に `lambda.amazonaws.com` が含まれている必要があります。
- 以下のポリシーが含まれている必要があります。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowOpenSearchStreamingAccess",
      "Action": [
        "es:*"
      ],
    },
  ],
}
```

```
        "Effect": "Allow",
        "Resource": "arn:aws:es:us-  
east-1:123456789012:domain/cloudwatch-logs/*"
    }
]
```

- ターゲットの OpenSearch Service ドメインで VPC アクセスを使用する場合、ロールには AWSLambdaVPCLambdaAccessExecutionRole ポリシーがアタッチされている必要があります。Amazon が管理するこのポリシーにより、お客様の VPC へのアクセスが Lambda に許可され、Lambda は VPC の OpenSearch エンドポイントに書き込むことができます。
7. [Log format] (ログの形式) で、ログの形式を選択します。
 8. [Subscription filter pattern] (サブスクリプションフィルターのパターン) に、ログイベントで検索する用語やパターンを入力します。これにより、関心のあるデータだけが OpenSearch クラスターに送信されるようになります。詳細については、「[フィルターを使用したログイベントからのメトリクスの作成](#)」を参照してください。
 9. (オプション) [Select log data to test] (テストするログデータの選択) を開き、ログストリームを選択してから、[Test pattern] (パターンをテスト) を選択して、期待通りの結果が出ることを確認します。
 10. [Start streaming] (ストリーミングの開始) を選択します。

SDK を使用した CloudWatch Logs のコード例 AWS SDKs

次のコード例は、AWS Software Development Kit (SDK) で CloudWatch Logs を使用方法を示しています。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

シナリオは、1つのサービス内から、または他の AWS のサービスと組み合わせて複数の関数を呼び出し、特定のタスクを実行する方法を示すコード例です。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK での CloudWatch Logs の使用](#)。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

コードの例

- [SDK を使用した CloudWatch Logs の基本的な例 AWS SDKs](#)
 - [SDK を使用した CloudWatch Logs のアクション AWS SDKs](#)
 - [AWS SDK AssociateKmsKeyで を使用する](#)
 - [AWS SDK CancelExportTaskで を使用する](#)
 - [AWS SDK CreateExportTaskで を使用する](#)
 - [AWS SDK または CLI CreateLogGroupで を使用する](#)
 - [AWS SDK または CLI CreateLogStreamで を使用する](#)
 - [AWS SDK または CLI DeleteLogGroupで を使用する](#)
 - [AWS SDK DeleteSubscriptionFilterで を使用する](#)
 - [AWS SDK DescribeExportTasksで を使用する](#)
 - [AWS SDK または CLI DescribeLogGroupsで を使用する](#)
 - [AWS SDK または CLI DescribeLogStreamsで を使用する](#)
 - [AWS SDK DescribeSubscriptionFiltersで を使用する](#)
 - [AWS SDK または CLI GetLogEventsで を使用する](#)
 - [AWS SDK GetQueryResultsで を使用する](#)
 - [AWS SDK PutSubscriptionFilterで を使用する](#)
 - [AWS SDK StartLiveTailで を使用する](#)

- [AWS SDK StartQueryで を使用する](#)
- [SDK を使用した CloudWatch Logs のシナリオ AWS SDKs](#)
 - [Amazon ECS Service Connect を設定する](#)
 - [最初の Lambda 関数を作成する](#)
 - [CloudWatch Logs を使用して大規模なクエリを実行する](#)
 - [スケジュールされたイベントを使用した Lambda 関数の呼び出し](#)

SDK を使用した CloudWatch Logs の基本的な例 AWS SDKs

次のコード例は、AWS SDK で Amazon CloudWatch Logs の基本的な機能を使用する方法を示しています。

例

- [SDK を使用した CloudWatch Logs のアクション AWS SDKs](#)
 - [AWS SDK AssociateKmsKeyで を使用する](#)
 - [AWS SDK CancelExportTaskで を使用する](#)
 - [AWS SDK CreateExportTaskで を使用する](#)
 - [AWS SDK または CLI CreateLogGroupで を使用する](#)
 - [AWS SDK または CLI CreateLogStreamで を使用する](#)
 - [AWS SDK または CLI DeleteLogGroupで を使用する](#)
 - [AWS SDK DeleteSubscriptionFilterで を使用する](#)
 - [AWS SDK DescribeExportTasksで を使用する](#)
 - [AWS SDK または CLI DescribeLogGroupsで を使用する](#)
 - [AWS SDK または CLI DescribeLogStreamsで を使用する](#)
 - [AWS SDK DescribeSubscriptionFiltersで を使用する](#)
 - [AWS SDK または CLI GetLogEventsで を使用する](#)
 - [AWS SDK GetQueryResultsで を使用する](#)
 - [AWS SDK PutSubscriptionFilterで を使用する](#)
 - [AWS SDK StartLiveTailで を使用する](#)
 - [AWS SDK StartQueryで を使用する](#)

SDK を使用した CloudWatch Logs のアクション AWS SDKs

次のコード例は、AWS SDKs を使用して個々の CloudWatch Logs アクションを実行する方法を示しています。それぞれの例には、GitHub へのリンクがあり、そこにはコードの設定と実行に関する説明が記載されています。

これらは CloudWatch Logs API を呼び出すもので、コンテキスト内で実行する必要がある大規模なプログラムからのコード抜粋です。アクションは [SDK を使用した CloudWatch Logs のシナリオ AWS SDKs](#) のコンテキスト内で確認できます。

以下の例には、最も一般的に使用されるアクションのみ含まれています。詳細な一覧については、「[Amazon CloudWatch Logs API Reference](#)」(Amazon CloudWatch Logs API リファレンス) を参照してください。

例

- [AWS SDK AssociateKmsKeyで を使用する](#)
- [AWS SDK CancelExportTaskで を使用する](#)
- [AWS SDK CreateExportTaskで を使用する](#)
- [AWS SDK または CLI CreateLogGroupで を使用する](#)
- [AWS SDK または CLI CreateLogStreamで を使用する](#)
- [AWS SDK または CLI DeleteLogGroupで を使用する](#)
- [AWS SDK DeleteSubscriptionFilterで を使用する](#)
- [AWS SDK DescribeExportTasksで を使用する](#)
- [AWS SDK または CLI DescribeLogGroupsで を使用する](#)
- [AWS SDK または CLI DescribeLogStreamsで を使用する](#)
- [AWS SDK DescribeSubscriptionFiltersで を使用する](#)
- [AWS SDK または CLI GetLogEventsで を使用する](#)
- [AWS SDK GetQueryResultsで を使用する](#)
- [AWS SDK PutSubscriptionFilterで を使用する](#)
- [AWS SDK StartLiveTailで を使用する](#)
- [AWS SDK StartQueryで を使用する](#)

AWS SDK **AssociateKmsKey**で を使用する

次の例は、AssociateKmsKey を使用する方法を説明しています。

.NET

SDK for .NET

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
using System;
using System.Threading.Tasks;
using Amazon.CloudWatchLogs;
using Amazon.CloudWatchLogs.Model;

/// <summary>
/// Shows how to associate an AWS Key Management Service (AWS KMS) key with
/// an Amazon CloudWatch Logs log group.
/// </summary>
public class AssociateKmsKey
{
    public static async Task Main()
    {
        // This client object will be associated with the same AWS Region
        // as the default user on this system. If you need to use a
        // different AWS Region, pass it as a parameter to the client
        // constructor.
        var client = new AmazonCloudWatchLogsClient();

        string kmsKeyId = "arn:aws:kms:us-west-2:<account-
number>:key/7c9eccc2-38cb-4c4f-9db3-766ee8dd3ad4";
        string groupName = "cloudwatchlogs-example-loggroup";

        var request = new AssociateKmsKeyRequest
        {
            KmsKeyId = kmsKeyId,
            LogGroupName = groupName,
        };

        var response = await client.AssociateKmsKeyAsync(request);

        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
```

```
        {  
            Console.WriteLine($"Successfully associated KMS key ID:  
{kmsKeyId} with log group: {groupName}.");  
        }  
        else  
        {  
            Console.WriteLine("Could not make the association between:  
{kmsKeyId} and {groupName}.");  
        }  
    }  
}
```

- API の詳細については、「AWS SDK for .NET API リファレンス」にある「[AssociateKmsKey](#)」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK での CloudWatch Logs の使用](#)。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK `CancelExportTask`で を使用する

次の例は、`CancelExportTask` を使用する方法を説明しています。

.NET

SDK for .NET

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
using System;  
using System.Threading.Tasks;  
using Amazon.CloudWatchLogs;  
using Amazon.CloudWatchLogs.Model;
```

```
/// <summary>
/// Shows how to cancel an Amazon CloudWatch Logs export task.
/// </summary>
public class CancelExportTask
{
    public static async Task Main()
    {
        // This client object will be associated with the same AWS Region
        // as the default user on this system. If you need to use a
        // different AWS Region, pass it as a parameter to the client
        // constructor.
        var client = new AmazonCloudWatchLogsClient();
        string taskId = "exampleTaskId";

        var request = new CancelExportTaskRequest
        {
            TaskId = taskId,
        };

        var response = await client.CancelExportTaskAsync(request);

        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine($"{taskId} successfully canceled.");
        }
        else
        {
            Console.WriteLine($"{taskId} could not be canceled.");
        }
    }
}
```

- APIの詳細については、「AWS SDK for .NET API リファレンス」の「[CancelExportTask](#)」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK での CloudWatch Logs の使用](#)。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK `CreateExportTask`で を使用する

次の例は、`CreateExportTask` を使用する方法を説明しています。

.NET

SDK for .NET

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
using System;
using System.Threading.Tasks;
using Amazon.CloudWatchLogs;
using Amazon.CloudWatchLogs.Model;

/// <summary>
/// Shows how to create an Export Task to export the contents of the Amazon
/// CloudWatch Logs to the specified Amazon Simple Storage Service (Amazon
S3)
/// bucket.
/// </summary>
public class CreateExportTask
{
    public static async Task Main()
    {
        // This client object will be associated with the same AWS Region
        // as the default user on this system. If you need to use a
        // different AWS Region, pass it as a parameter to the client
        // constructor.
        var client = new AmazonCloudWatchLogsClient();
        string taskName = "export-task-example";
        string logGroupName = "cloudwatchlogs-example-loggroup";
        string destination = "amzn-s3-demo-bucket";
        var fromTime = 1437584472382;
        var toTime = 1437584472833;

        var request = new CreateExportTaskRequest
        {
```

```
        From = fromTime,  
        To = toTime,  
        TaskName = taskName,  
        LogGroupName = logGroupName,  
        Destination = destination,  
    };  
  
    var response = await client.CreateExportTaskAsync(request);  
  
    if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)  
    {  
        Console.WriteLine($"The task, {taskName} with ID: " +  
            $"{response.TaskId} has been created  
successfully.");  
    }  
}
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「[CreateExportTask](#)」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください [AWS SDK での CloudWatch Logs の使用](#)。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI **CreateLogGroup**で を使用する

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

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- [Amazon ECS Service Connect を設定する](#)

.NET

SDK for .NET

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
using System;
using System.Threading.Tasks;
using Amazon.CloudWatchLogs;
using Amazon.CloudWatchLogs.Model;

/// <summary>
/// Shows how to create an Amazon CloudWatch Logs log group.
/// </summary>
public class CreateLogGroup
{
    public static async Task Main()
    {
        // This client object will be associated with the same AWS Region
        // as the default user on this system. If you need to use a
        // different AWS Region, pass it as a parameter to the client
        // constructor.
        var client = new AmazonCloudWatchLogsClient();

        string logGroupName = "cloudwatchlogs-example-loggroup";

        var request = new CreateLogGroupRequest
        {
            LogGroupName = logGroupName,
        };

        var response = await client.CreateLogGroupAsync(request);

        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine($"Successfully create log group with ID:
{logGroupName}.");
        }
    }
}
```



```
        else
        {
            Console.WriteLine("Could not create log group.");
        }
    }
}
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「[CreateLogGroup](#)」を参照してください。

CLI

AWS CLI

次のコマンドは、my-logs という名前のロググループを作成します。

```
aws logs create-log-group --log-group-name my-logs
```

- API の詳細については、AWS CLI コマンドリファレンスの「[CreateLogGroup](#)」を参照してください。

JavaScript

SDK for JavaScript (v3)

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
import { CreateLogGroupCommand } from "@aws-sdk/client-cloudwatch-logs";
import { client } from "../libs/client.js";

const run = async () => {
    const command = new CreateLogGroupCommand({
        // The name of the log group.
```

```
    logGroupName: process.env.CLOUDWATCH_LOGS_LOG_GROUP,  
  });  
  
  try {  
    return await client.send(command);  
  } catch (err) {  
    console.error(err);  
  }  
};  
  
export default run();
```

- APIの詳細については、AWS SDK for JavaScript API リファレンスの「[CreateLogGroup](#)」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK での CloudWatch Logs の使用](#)。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI **CreateLogStream**で を使用する

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

.NET

SDK for .NET

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
using System;  
using System.Threading.Tasks;  
using Amazon.CloudWatchLogs;  
using Amazon.CloudWatchLogs.Model;  
  
///  
/// <summary>
```

```
/// Shows how to create an Amazon CloudWatch Logs stream for a CloudWatch
/// log group.
/// </summary>
public class CreateLogStream
{
    public static async Task Main()
    {
        // This client object will be associated with the same AWS Region
        // as the default user on this system. If you need to use a
        // different AWS Region, pass it as a parameter to the client
        // constructor.
        var client = new AmazonCloudWatchLogsClient();
        string logGroupName = "cloudwatchlogs-example-loggroup";
        string logStreamName = "cloudwatchlogs-example-logstream";

        var request = new CreateLogStreamRequest
        {
            LogGroupName = logGroupName,
            LogStreamName = logStreamName,
        };

        var response = await client.CreateLogStreamAsync(request);

        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine($"{logStreamName} successfully created for
{logGroupName}.");
        }
        else
        {
            Console.WriteLine("Could not create stream.");
        }
    }
}
```

- APIの詳細については、「AWS SDK for .NET API リファレンス」の「[CreateLogStream](#)」を参照してください。

CLI

AWS CLI

次のコマンドは、ロググループ `my-logs` に `20150601` という名前のログストリームを作成します。

```
aws logs create-log-stream --log-group-name my-logs --log-stream-name 20150601
```

- API の詳細については、AWS CLI コマンドリファレンスの「[CreateLogStream](#)」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください [AWS SDK での CloudWatch Logs の使用](#)。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI `DeleteLogGroup` で を使用する

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

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- [Amazon ECS Service Connect を設定する](#)
- [最初の Lambda 関数を作成する](#)

.NET

SDK for .NET

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
using System;  
using System.Threading.Tasks;
```

```
using Amazon.CloudWatchLogs;
using Amazon.CloudWatchLogs.Model;

/// <summary>
/// Uses the Amazon CloudWatch Logs Service to delete an existing
/// CloudWatch Logs log group.
/// </summary>
public class DeleteLogGroup
{
    public static async Task Main()
    {
        var client = new AmazonCloudWatchLogsClient();
        string logGroupName = "cloudwatchlogs-example-loggroup";

        var request = new DeleteLogGroupRequest
        {
            LogGroupName = logGroupName,
        };

        var response = await client.DeleteLogGroupAsync(request);

        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine($"Successfully deleted CloudWatch log group,
{logGroupName}.");
        }
    }
}
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「[DeleteLogGroup](#)」を参照してください。

CLI

AWS CLI

以下のコマンドは、my-logs という名前のロググループを削除します。

```
aws logs delete-log-group --log-group-name my-logs
```

- API の詳細については、AWS CLI コマンドリファレンスの「[DeleteLogGroup](#)」を参照してください。

JavaScript

SDK for JavaScript (v3)

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
import { DeleteLogGroupCommand } from "@aws-sdk/client-cloudwatch-logs";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new DeleteLogGroupCommand({
    // The name of the log group.
    logGroupName: process.env.CLOUDWATCH_LOGS_LOG_GROUP,
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

- API の詳細については、AWS SDK for JavaScript API リファレンスの「[DeleteLogGroup](#)」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK での CloudWatch Logs の使用](#)。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK DeleteSubscriptionFilterで 使用する

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

C++

SDK for C++

Note

GitHub には、その他のリソースもあります。完全な例を見つけて、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

必要なファイルを含めます。

```
#include <aws/core/Aws.h>
#include <aws/core/utils/Outcome.h>
#include <aws/logs/CloudWatchLogsClient.h>
#include <aws/logs/model/DeleteSubscriptionFilterRequest.h>
#include <iostream>
```

サブスクリプションフィルターを削除します。

```
Aws::CloudWatchLogs::CloudWatchLogsClient cwl;
Aws::CloudWatchLogs::Model::DeleteSubscriptionFilterRequest request;
request.SetFilterName(filter_name);
request.SetLogGroupName(log_group);

auto outcome = cwl.DeleteSubscriptionFilter(request);
if (!outcome.IsSuccess()) {
    std::cout << "Failed to delete CloudWatch log subscription filter "
                << filter_name << ": " << outcome.GetError().GetMessage() <<
                std::endl;
} else {
    std::cout << "Successfully deleted CloudWatch logs subscription " <<
                "filter " << filter_name << std::endl;
}
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「[DeleteSubscriptionFilter](#)」を参照してください。

Java

SDK for Java 2.x

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
import software.amazon.awssdk.services.cloudwatch.model.CloudWatchException;
import software.amazon.awssdk.services.cloudwatchlogs.CloudWatchLogsClient;
import
    software.amazon.awssdk.services.cloudwatchlogs.model.DeleteSubscriptionFilterRequest;

/**
 * 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
 */
public class DeleteSubscriptionFilter {
    public static void main(String[] args) {
        final String usage = ""

            Usage:
                <filter> <logGroup>

            Where:
                filter - The name of the subscription filter (for example,
MyFilter).

                logGroup - The name of the log group. (for example, testgroup).
"";
    }
}
```



```
        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String filter = args[0];
        String logGroup = args[1];
        CloudWatchLogsClient logs = CloudWatchLogsClient.builder()
            .build();

        deleteSubFilter(logs, filter, logGroup);
        logs.close();
    }

    public static void deleteSubFilter(CloudWatchLogsClient logs, String filter,
        String logGroup) {
        try {
            DeleteSubscriptionFilterRequest request =
                DeleteSubscriptionFilterRequest.builder()
                    .filterName(filter)
                    .logGroupName(logGroup)
                    .build();

            logs.deleteSubscriptionFilter(request);
            System.out.printf("Successfully deleted CloudWatch logs subscription
filter %s", filter);

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

- APIの詳細については、AWS SDK for Java 2.x API リファレンスの「[DeleteSubscriptionFilter](#)」を参照してください。

JavaScript

SDK for JavaScript (v3)

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
import { DeleteSubscriptionFilterCommand } from "@aws-sdk/client-cloudwatch-logs";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new DeleteSubscriptionFilterCommand({
    // The name of the filter.
    filterName: process.env.CLOUDWATCH_LOGS_FILTER_NAME,
    // The name of the log group.
    logGroupName: process.env.CLOUDWATCH_LOGS_LOG_GROUP,
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

- API の詳細については、AWS SDK for JavaScript API リファレンスの「[DeleteSubscriptionFilter](#)」を参照してください。

SDK for JavaScript (v2)

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the CloudWatchLogs service object
var cwl = new AWS.CloudWatchLogs({ apiVersion: "2014-03-28" });

var params = {
  filterName: "FILTER",
  logGroupName: "LOG_GROUP",
};

cwl.deleteSubscriptionFilter(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 詳細については、「[AWS SDK for JavaScript デベロッパーガイド](#)」を参照してください。
- API の詳細については、AWS SDK for JavaScript API リファレンスの「[DeleteSubscriptionFilter](#)」を参照してください。

Kotlin

SDK for Kotlin

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
suspend fun deleteSubFilter(
    filter: String?,
    logGroup: String?,
) {
```

```
val request =
    DeleteSubscriptionFilterRequest {
        filterName = filter
        logGroupName = logGroup
    }

CloudWatchLogsClient.fromEnvironment { region = "us-west-2" }.use { logs ->
    logs.deleteSubscriptionFilter(request)
    println("Successfully deleted CloudWatch logs subscription filter named
$filter")
}
}
```

- API の詳細については、「AWS SDK for Kotlin API リファレンス」の「[DeleteSubscriptionFilter](#)」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK での CloudWatch Logs の使用](#)。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK **DescribeExportTasks**で 使用する

次の例は、DescribeExportTasks を使用する方法を説明しています。

.NET

SDK for .NET

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
using System;
using System.Threading.Tasks;
using Amazon.CloudWatchLogs;
using Amazon.CloudWatchLogs.Model;
```

```
/// <summary>
/// Shows how to retrieve a list of information about Amazon CloudWatch
/// Logs export tasks.
/// </summary>
public class DescribeExportTasks
{
    public static async Task Main()
    {
        // This client object will be associated with the same AWS Region
        // as the default user on this system. If you need to use a
        // different AWS Region, pass it as a parameter to the client
        // constructor.
        var client = new AmazonCloudWatchLogsClient();

        var request = new DescribeExportTasksRequest
        {
            Limit = 5,
        };

        var response = new DescribeExportTasksResponse();

        do
        {
            response = await client.DescribeExportTasksAsync(request);
            response.ExportTasks.ForEach(t =>
            {
                Console.WriteLine($"{t.TaskName} with ID: {t.TaskId} has
status: {t.Status}");
            });
        }
        while (response.NextToken is not null);
    }
}
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「[DescribeExportTasks](#)」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK での CloudWatch Logs の使用](#)。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI **DescribeLogGroups**で を使用する

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

.NET

SDK for .NET

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
using System;
using System.Threading.Tasks;
using Amazon.CloudWatchLogs;
using Amazon.CloudWatchLogs.Model;

/// <summary>
/// Retrieves information about existing Amazon CloudWatch Logs log groups
/// and displays the information on the console.
/// </summary>
public class DescribeLogGroups
{
    public static async Task Main()
    {
        // Creates a CloudWatch Logs client using the default
        // user. If you need to work with resources in another
        // AWS Region than the one defined for the default user,
        // pass the AWS Region as a parameter to the client constructor.
        var client = new AmazonCloudWatchLogsClient();

        bool done = false;
        string newToken = null;

        var request = new DescribeLogGroupsRequest
        {
            Limit = 5,
        };

        DescribeLogGroupsResponse response;
```

```
do
{
    if (newToken is not null)
    {
        request.NextToken = newToken;
    }

    response = await client.DescribeLogGroupsAsync(request);

    response.LogGroups.ForEach(lg =>
    {
        Console.WriteLine($"{lg.LogGroupName} is associated with the
key: {lg.KmsKeyId}.");
        Console.WriteLine($"Created on:
{lg.CreationTime.Date.Date}");
        Console.WriteLine($"Date for this group will be stored for:
{lg.RetentionInDays} days.\n");
    });

    if (response.NextToken is null)
    {
        done = true;
    }
    else
    {
        newToken = response.NextToken;
    }
}
while (!done);
}
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「[DescribeLogGroups](#)」を参照してください。

CLI

AWS CLI

次のコマンドは、my-logs という名前のロググループを記述します。

```
aws logs describe-log-groups --log-group-name-prefix my-logs
```

出力:

```
{
  "logGroups": [
    {
      "storedBytes": 0,
      "metricFilterCount": 0,
      "creationTime": 1433189500783,
      "logGroupName": "my-logs",
      "retentionInDays": 5,
      "arn": "arn:aws:logs:us-west-2:0123456789012:log-group:my-logs:*"
    }
  ]
}
```

- API の詳細については、AWS CLI コマンドリファレンスの「[DescribeLogGroups](#)」を参照してください。

JavaScript

SDK for JavaScript (v3)

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
import {
  paginateDescribeLogGroups,
  CloudWatchLogsClient,
} from "@aws-sdk/client-cloudwatch-logs";

const client = new CloudWatchLogsClient({});

export const main = async () => {
  const paginatedLogGroups = paginateDescribeLogGroups({ client }, {});
  const logGroups = [];
```



```
for await (const page of paginatedLogGroups) {
  if (page.logGroups?.every((lg) => !!lg)) {
    logGroups.push(...page.logGroups);
  }
}

console.log(logGroups);
return logGroups;
};
```

- API の詳細については、AWS SDK for JavaScript API リファレンスの「[DescribeLogGroups](#)」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK での CloudWatch Logs の使用](#)。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI **DescribeLogStreams**で を使用する

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

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- [最初の Lambda 関数を作成する](#)

CLI

AWS CLI

次のコマンドは、ロググループ `my-logs` のプレフィックス `2015` で始まるすべてのログストリームを表示します。

```
aws logs describe-log-streams --log-group-name my-logs --log-stream-name-prefix 2015
```

出力:

```
{
```

```
"logStreams": [  
  {  
    "creationTime": 1433189871774,  
    "arn": "arn:aws:logs:us-west-2:0123456789012:log-group:my-logs:log-stream:20150531",  
    "logStreamName": "20150531",  
    "storedBytes": 0  
  },  
  {  
    "creationTime": 1433189873898,  
    "arn": "arn:aws:logs:us-west-2:0123456789012:log-group:my-logs:log-stream:20150601",  
    "logStreamName": "20150601",  
    "storedBytes": 0  
  }  
]
```

- API の詳細については、「AWS CLI コマンドリファレンス」の「[DescribeLogStreams](#)」を参照してください。

Java

SDK for Java 2.x

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

指定されたプレフィックスに一致する、指定されたロググループ内のログストリームを検索します。

```
/**  
 * Before running this Java V2 code example, set up your development  
 * environment, including your credentials.  
 * <p>  
 * For more information, see the following documentation topic:  
 * <p>
```

```
* https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
*/
public class CloudWatchLogsSearch {

    public static void main(String[] args) {
        final String usage = ""

            Usage:
                <logGroupName> <logStreamName>

            Where:
                logGroupName - The name of the log group (for example,
WeathertopJavaContainerLogs).
                logStreamName - The name of the log stream (for example,
weathertop-java-stream).
                pattern - the pattern to use (for example, INFO)

            """;

        if (args.length != 3) {
            System.out.print(usage);
            System.exit(1);
        }

        String logGroupName = args[0] ;
        String logStreamName = args[1] ;
        String pattern = args[2] ;

        CloudWatchLogsClient cwlClient = CloudWatchLogsClient.builder()
            .region(Region.US_EAST_1)
            .build();

        searchLogStreamsAndFilterEvents(cwlClient, logGroupName, logStreamName,
pattern);
    }

    /**
     * Searches for log streams with a specific prefix within a log group and
filters log events based on a specified pattern.
     *
     * @param cwlClient      the CloudWatchLogsClient used to interact with AWS
CloudWatch Logs
     * @param logGroupName  the name of the log group to search within
    */
}
```

```
* @param logStreamPrefix the prefix of the log streams to search for
* @param pattern          the pattern to filter log events by
*/
public static void searchLogStreamsAndFilterEvents(CloudWatchLogsClient
cwlClient, String logGroupName, String logStreamPrefix, String pattern) {
    DescribeLogStreamsRequest describeLogStreamsRequest =
DescribeLogStreamsRequest.builder()
        .logGroupName(logGroupName)
        .logStreamNamePrefix(logStreamPrefix)
        .build();

    DescribeLogStreamsResponse describeLogStreamsResponse =
cwlClient.describeLogStreams(describeLogStreamsRequest);
    List<LogStream> logStreams = describeLogStreamsResponse.logStreams();

    for (LogStream logStream : logStreams) {
        String logStreamName = logStream.logStreamName();
        System.out.println("Searching in log stream: " + logStreamName);

        FilterLogEventsRequest filterLogEventsRequest =
FilterLogEventsRequest.builder()
            .logGroupName(logGroupName)
            .logStreamNames(logStreamName)
            .filterPattern(pattern)
            .build();

        FilterLogEventsResponse filterLogEventsResponse =
cwlClient.filterLogEvents(filterLogEventsRequest);

        for (FilteredLogEvent event : filterLogEventsResponse.events()) {
            System.out.println(event.message());
        }

        System.out.println("-----"); //
        Separator for better readability
    }
}
```

指定されたロググループ内の最新のログストリームに関するメタデータを出力します。

```
/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 * <p>
 * For more information, see the following documentation topic:
 * <p>
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 */
public class CloudWatchLogQuery {
    public static void main(final String[] args) {
        final String usage = ""
            Usage:
                <logGroupName>

            Where:
                logGroupName - The name of the log group (for example, /aws/
lambda/ChatAIHandler).
            """;

        if (args.length != 1) {
            System.out.print(usage);
            System.exit(1);
        }

        String logGroupName = "/aws/lambda/ChatAIHandler" ; //args[0];
        CloudWatchLogsClient logsClient = CloudWatchLogsClient.builder()
            .region(Region.US_EAST_1)
            .build();

        describeMostRecentLogStream(logsClient, logGroupName);
    }

    /**
     * Describes and prints metadata about the most recent log stream in the
     specified log group.
     *
     * @param logsClient the CloudWatchLogsClient used to interact with AWS
     CloudWatch Logs
     * @param logGroupName the name of the log group
     */
}
```

```
public static void describeMostRecentLogStream(CloudWatchLogsClient
logsClient, String logGroupName) {
    DescribeLogStreamsRequest streamsRequest =
DescribeLogStreamsRequest.builder()
        .logGroupName(logGroupName)
        .orderBy(OrderBy.LAST_EVENT_TIME)
        .descending(true)
        .limit(1)
        .build();

    try {
        DescribeLogStreamsResponse streamsResponse =
logsClient.describeLogStreams(streamsRequest);
        List<LogStream> logStreams = streamsResponse.logStreams();

        if (logStreams.isEmpty()) {
            System.out.println("No log streams found for log group: " +
logGroupName);
            return;
        }

        LogStream stream = logStreams.get(0);
        System.out.println("Most Recent Log Stream:");
        System.out.println("  Name: " + stream.logStreamName());
        System.out.println("  ARN: " + stream.arn());
        System.out.println("  Creation Time: " + stream.creationTime());
        System.out.println("  First Event Time: " +
stream.firstEventTimestamp());
        System.out.println("  Last Event Time: " +
stream.lastEventTimestamp());
        System.out.println("  Stored Bytes: " + stream.storedBytes());
        System.out.println("  Upload Sequence Token: " +
stream.uploadSequenceToken());

    } catch (CloudWatchLogsException e) {
        System.err.println("Failed to describe log stream: " +
e.awsErrorDetails().errorMessage());
    }
}
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「[DescribeLogStreams](#)」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK での CloudWatch Logs の使用](#)。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK `DescribeSubscriptionFilters`で を使用する

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

C++

SDK for C++

Note

GitHub には、その他のリソースもあります。完全な例を見つけて、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

必要なファイルを含めます。

```
#include <aws/core/Aws.h>
#include <aws/core/utils/Outcome.h>
#include <aws/logs/CloudWatchLogsClient.h>
#include <aws/logs/model/DescribeSubscriptionFiltersRequest.h>
#include <aws/logs/model/DescribeSubscriptionFiltersResult.h>
#include <iostream>
#include <iomanip>
```

サブスクリプションフィルターを一覧表示します。

```
Aws::CloudWatchLogs::CloudWatchLogsClient cwl;
Aws::CloudWatchLogs::Model::DescribeSubscriptionFiltersRequest request;
request.SetLogGroupName(log_group);
request.SetLimit(1);

bool done = false;
```

```
bool header = false;
while (!done) {
    auto outcome = cw1.DescribeSubscriptionFilters(
        request);
    if (!outcome.IsSuccess()) {
        std::cout << "Failed to describe CloudWatch subscription filters
"
        << "for log group " << log_group << ": " <<
        outcome.GetError().GetMessage() << std::endl;
        break;
    }

    if (!header) {
        std::cout << std::left << std::setw(32) << "Name" <<
        std::setw(64) << "FilterPattern" << std::setw(64) <<
        "DestinationArn" << std::endl;
        header = true;
    }

    const auto &filters = outcome.GetResult().GetSubscriptionFilters();
    for (const auto &filter : filters) {
        std::cout << std::left << std::setw(32) <<
        filter.GetFilterName() << std::setw(64) <<
        filter.GetFilterPattern() << std::setw(64) <<
        filter.GetDestinationArn() << std::endl;
    }

    const auto &next_token = outcome.GetResult().GetNextToken();
    request.SetNextToken(next_token);
    done = next_token.empty();
}
```

- APIの詳細については、AWS SDK for C++ API リファレンスの「[DescribeSubscriptionFilters](#)」を参照してください。

Java

SDK for Java 2.x

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.services.cloudwatch.model.CloudWatchException;
import software.amazon.awssdk.services.cloudwatchlogs.CloudWatchLogsClient;
import
    software.amazon.awssdk.services.cloudwatchlogs.model.DescribeSubscriptionFiltersRequest;
import
    software.amazon.awssdk.services.cloudwatchlogs.model.DescribeSubscriptionFiltersResponse;
import software.amazon.awssdk.services.cloudwatchlogs.model.SubscriptionFilter;

/**
 * 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
 */
public class DescribeSubscriptionFilters {
    public static void main(String[] args) {

        final String usage = ""

            Usage:
            <logGroup>

            Where:
            logGroup - A log group name (for example, myloggroup).
            """;

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

```
        System.exit(1);
    }

    String logGroup = args[0];
    CloudWatchLogsClient logs = CloudWatchLogsClient.builder()
        .credentialsProvider(ProfileCredentialsProvider.create())
        .build();

    describeFilters(logs, logGroup);
    logs.close();
}

public static void describeFilters(CloudWatchLogsClient logs, String
logGroup) {
    try {
        boolean done = false;
        String newToken = null;

        while (!done) {
            DescribeSubscriptionFiltersResponse response;
            if (newToken == null) {
                DescribeSubscriptionFiltersRequest request =
DescribeSubscriptionFiltersRequest.builder()
                    .logGroupName(logGroup)
                    .limit(1).build();

                response = logs.describeSubscriptionFilters(request);
            } else {
                DescribeSubscriptionFiltersRequest request =
DescribeSubscriptionFiltersRequest.builder()
                    .nextToken(newToken)
                    .logGroupName(logGroup)
                    .limit(1).build();
                response = logs.describeSubscriptionFilters(request);
            }

            for (SubscriptionFilter filter : response.subscriptionFilters())
            {
                System.out.printf("Retrieved filter with name %s, " +
"pattern %s " + "and destination arn %s",
                    filter.filterName(),
                    filter.filterPattern(),
                    filter.destinationArn());
            }
        }
    }
}
```

```
        if (response.nextToken() == null) {
            done = true;
        } else {
            newToken = response.nextToken();
        }
    }

    } catch (CloudWatchException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
    System.out.printf("Done");
}
}
```

- API の詳細については、AWS SDK for Java 2.x API リファレンスの「[DescribeSubscriptionFilters](#)」を参照してください。

JavaScript

SDK for JavaScript (v3)

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
import { DescribeSubscriptionFiltersCommand } from "@aws-sdk/client-cloudwatch-logs";
import { client } from "../libs/client.js";

const run = async () => {
    // This will return a list of all subscription filters in your account
    // matching the log group name.
    const command = new DescribeSubscriptionFiltersCommand({
        logGroupName: process.env.CLOUDWATCH_LOGS_LOG_GROUP,
        limit: 1,
    });
```

```
try {
  return await client.send(command);
} catch (err) {
  console.error(err);
}
};

export default run();
```

- API の詳細については、AWS SDK for JavaScript API リファレンスの「[DescribeSubscriptionFilters](#)」を参照してください。

SDK for JavaScript (v2)

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the CloudWatchLogs service object
var cwl = new AWS.CloudWatchLogs({ apiVersion: "2014-03-28" });

var params = {
  logGroupName: "GROUP_NAME",
  limit: 5,
};

cwl.describeSubscriptionFilters(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.subscriptionFilters);
  }
});
```

- 詳細については、「[AWS SDK for JavaScript デベロッパーガイド](#)」を参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「[DescribeSubscriptionFilters](#)」を参照してください。

Kotlin

SDK for Kotlin

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
suspend fun describeFilters(logGroup: String) {
    val request =
        DescribeSubscriptionFiltersRequest {
            logGroupName = logGroup
            limit = 1
        }

    CloudWatchLogsClient.fromEnvironment { region = "us-west-2" }.use { cwlClient
->
        val response = cwlClient.describeSubscriptionFilters(request)
        response.subscriptionFilters?.forEach { filter ->
            println("Retrieved filter with name ${filter.filterName} pattern
${filter.filterPattern} and destination ${filter.destinationArn}")
        }
    }
}
```

- API の詳細については、「AWS SDK for Kotlin API リファレンス」の「[DescribeSubscriptionFilters](#)」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK での CloudWatch Logs の使用](#)。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI **GetLogEvents**で を使用する

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

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- [最初の Lambda 関数を作成する](#)

CLI

AWS CLI

次のコマンドは、ロググループ `my-logs` の `20150601` という名前のログストリームからログイベントを取得します。

```
aws logs get-log-events --log-group-name my-logs --log-stream-name 20150601
```

出力:

```
{
  "nextForwardToken":
  "f/31961209122447488583055879464742346735121166569214640130",
  "events": [
    {
      "ingestionTime": 1433190494190,
      "timestamp": 1433190184356,
      "message": "Example Event 1"
    },
    {
      "ingestionTime": 1433190516679,
      "timestamp": 1433190184356,
      "message": "Example Event 1"
    },
    {
      "ingestionTime": 1433190494190,
      "timestamp": 1433190184358,
      "message": "Example Event 2"
    }
  ]
}
```

```
    }  
    ],  
    "nextBackwardToken":  
    "b/31961209122358285602261756944988674324553373268216709120"  
  }  
}
```

- API の詳細については、「AWS CLI コマンドリファレンス」の「[GetLogEvents](#)」を参照してください。

Java

SDK for Java 2.x

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
import software.amazon.awssdk.regions.Region;  
import software.amazon.awssdk.services.cloudwatch.model.CloudWatchException;  
import software.amazon.awssdk.services.cloudwatchlogs.CloudWatchLogsClient;  
import  
    software.amazon.awssdk.services.cloudwatchlogs.model.DescribeLogStreamsRequest;  
import  
    software.amazon.awssdk.services.cloudwatchlogs.model.DescribeLogStreamsResponse;  
import software.amazon.awssdk.services.cloudwatchlogs.model.GetLogEventsRequest;  
import software.amazon.awssdk.services.cloudwatchlogs.model.GetLogEventsResponse;  
  
import java.time.Instant;  
import java.time.temporal.ChronoUnit;  
  
/**  
 * 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  
 */
```

```

public class GetLogEvents {

    public static void main(String[] args) {

        final String usage = ""

            Usage:
                <logGroupName> <logStreamName>

            Where:
                logGroupName - The name of the log group (for example,
myloggroup).
                logStreamName - The name of the log stream (for example,
mystream).

            "";

        // if (args.length != 2) {
        //     System.out.print(usage);
        //     System.exit(1);
//     }

        String logGroupName = "WeathertopJavaContainerLogs" ; //args[0];
        String logStreamName = "weathertop-java-stream" ; //args[1];

        Region region = Region.US_EAST_1 ;
        CloudWatchLogsClient cloudWatchLogsClient =
CloudWatchLogsClient.builder()
            .region(region)
            .build();

        getCWLogEvents(cloudWatchLogsClient, logGroupName, logStreamName);
        cloudWatchLogsClient.close();
    }

    public static void getCWLogEvents(CloudWatchLogsClient cloudWatchLogsClient,
                                     String logGroupName,
                                     String logStreamPrefix) {

        try {
            // First, find the exact log stream name
            DescribeLogStreamsRequest describeRequest =
DescribeLogStreamsRequest.builder()
                .logGroupName(logGroupName)
                .logStreamNamePrefix(logStreamPrefix)

```



```

        .limit(1) // get the first matching stream
        .build();

        DescribeLogStreamsResponse describeResponse =
cloudWatchLogsClient.describeLogStreams(describeRequest);

        if (describeResponse.logStreams().isEmpty()) {
            System.out.println("No matching log streams found for prefix: " +
logStreamPrefix);
            return;
        }

        String exactLogStreamName =
describeResponse.logStreams().get(0).logStreamName();
        System.out.println("Using exact log stream: " + exactLogStreamName);

        long startTime = Instant.now().minus(7,
ChronoUnit.DAYS).toEpochMilli();
        long endTime = Instant.now().toEpochMilli();

        GetLogEventsRequest getLogEventsRequest =
GetLogEventsRequest.builder()
            .logGroupName(logGroupName)
            .logStreamName(exactLogStreamName) // <-- exact name, not
prefix
            .startTime(startTime)
            .endTime(endTime)
            .startFromHead(true)
            .build();

        GetLogEventsResponse response =
cloudWatchLogsClient.getLogEvents(getLogEventsRequest);

        if (response.events().isEmpty()) {
            System.out.println("No log events found in the past 7 days.");
        } else {
            response.events().forEach(e -> System.out.println(e.message()));
        }

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

```

```
}
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「[GetLogEvents](#)」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK での CloudWatch Logs の使用](#)。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK **GetQueryResults**で を使用する

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

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- [大規模なクエリを実行する](#)

.NET

SDK for .NET (v4)

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
/// <summary>
/// Gets the results of a CloudWatch Logs Insights query.
/// </summary>
/// <param name="queryId">The ID of the query.</param>
/// <returns>The query results response.</returns>
public async Task<GetQueryResultsResponse?> GetQueryResultsAsync(string
queryId)
{
    try
    {
        var request = new GetQueryResultsRequest
```

```
        {
            QueryId = queryId
        };

        var response = await
        _amazonCloudWatchLogs.GetQueryResultsAsync(request);
        return response;
    }
    catch (ResourceNotFoundException ex)
    {
        _logger.LogError($"Query not found: {ex.Message}");
        return null;
    }
    catch (Exception ex)
    {
        _logger.LogError($"An error occurred while getting query results:
        {ex.Message}");
        return null;
    }
}
```

- APIの詳細については、「AWS SDK for .NET API リファレンス」の「[GetQueryResults](#)」を参照してください。

JavaScript

SDK for JavaScript (v3)

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
/**
 * Simple wrapper for the GetQueryResultsCommand.
 * @param {string} queryId
 */
_getQueryResults(queryId) {
    return this.client.send(new GetQueryResultsCommand({ queryId }));
}
```

```
}
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「[GetQueryResults](#)」を参照してください。

Python

SDK for Python (Boto3)

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
def _wait_for_query_results(self, client, query_id):
    """
    Waits for the query to complete and retrieves the results.

    :param query_id: The ID of the initiated query.
    :type query_id: str
    :return: A list containing the results of the query.
    :rtype: list
    """
    while True:
        time.sleep(1)
        results = client.get_query_results(queryId=query_id)
        if results["status"] in [
            "Complete",
            "Failed",
            "Cancelled",
            "Timeout",
            "Unknown",
        ]:
            return results.get("results", [])
```

- API の詳細については、「AWS SDK for Python (Boto3) API Reference」の「[GetQueryResults](#)」を参照してください。

SAP ABAP

SDK for SAP ABAP

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
TRY.  
    oo_result = lo_cwl->getqueryresults(  
        iv_queryid = iv_query_id ).  
  
    " Display query status and result count  
    DATA(lv_status) = oo_result->get_status( ).  
    DATA(lt_results) = oo_result->get_results( ).  
    DATA(lv_result_count) = lines( lt_results ).  
  
    MESSAGE |Query status: { lv_status }. Retrieved { lv_result_count } log  
event(s).| TYPE 'I'.  
    CATCH /aws1/cx_cwlinvalidparameterex.  
        MESSAGE 'Invalid parameter.' TYPE 'E'.  
    CATCH /aws1/cx_cwlresourcenotfoundex.  
        MESSAGE 'Resource not found.' TYPE 'E'.  
    CATCH /aws1/cx_cwlserviceunavailex.  
        MESSAGE 'Service unavailable.' TYPE 'E'.  
ENDTRY.
```

- API の詳細については、AWS SDK for SAP ABAP API リファレンスの[GetQueryResults](#)」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK での CloudWatch Logs の使用](#)。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK **PutSubscriptionFilter**で を使用する

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

C++

SDK for C++

 Note

GitHub には、その他のリソースもあります。完全な例を見つけて、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

必要なファイルを含めます。

```
#include <aws/core/Aws.h>
#include <aws/logs/CloudWatchLogsClient.h>
#include <aws/logs/model/PutSubscriptionFilterRequest.h>
#include <aws/core/utils/Outcome.h>
#include <iostream>
```

サブスクリプションフィルターを作成します。

```
Aws::CloudWatchLogs::CloudWatchLogsClient cwl;
Aws::CloudWatchLogs::Model::PutSubscriptionFilterRequest request;
request.SetFilterName(filter_name);
request.SetFilterPattern(filter_pattern);
request.SetLogGroupName(log_group);
request.SetDestinationArn(dest_arn);
auto outcome = cwl.PutSubscriptionFilter(request);
if (!outcome.IsSuccess())
{
    std::cout << "Failed to create CloudWatch logs subscription filter "
                << filter_name << ": " << outcome.GetError().GetMessage() <<
                std::endl;
}
else
{
    std::cout << "Successfully created CloudWatch logs subscription " <<
                "filter " << filter_name << std::endl;
}
```

- APIの詳細については、「AWS SDK for C++ API リファレンス」の「[PutSubscriptionFilter](#)」を参照してください。

Java

SDK for Java 2.x

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cloudwatchlogs.CloudWatchLogsClient;
import
    software.amazon.awssdk.services.cloudwatchlogs.model.CloudWatchLogsException;
import
    software.amazon.awssdk.services.cloudwatchlogs.model.PutSubscriptionFilterRequest;

/**
 * Before running this code example, you need to grant permission to CloudWatch
 * Logs the right to execute your Lambda function.
 * To perform this task, you can use this CLI command:
 *
 * aws lambda add-permission --function-name "lamda1" --statement-id "lamda1"
 * --principal "logs.us-west-2.amazonaws.com" --action "lambda:InvokeFunction"
 * --source-arn "arn:aws:logs:us-west-2:111111111111:log-group:testgroup:*"
 * --source-account "111111111111"
 *
 * Make sure you replace the function name with your function name and replace
 * '111111111111' with your account details.
 * For more information, see "Subscription Filters with AWS Lambda" in the
 * Amazon CloudWatch Logs Guide.
 *
 * Also, 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
*
*/

public class PutSubscriptionFilter {
    public static void main(String[] args) {
        final String usage = ""

            Usage:
                <filter> <pattern> <logGroup> <functionArn>\s

            Where:
                filter - A filter name (for example, myfilter).
                pattern - A filter pattern (for example, ERROR).
                logGroup - A log group name (testgroup).
                functionArn - An AWS Lambda function ARN (for example,
arn:aws:lambda:us-west-2:111111111111:function:lambda1) .
                """;

        if (args.length != 4) {
            System.out.println(usage);
            System.exit(1);
        }

        String filter = args[0];
        String pattern = args[1];
        String logGroup = args[2];
        String functionArn = args[3];
        Region region = Region.US_WEST_2;
        CloudWatchLogsClient cwl = CloudWatchLogsClient.builder()
            .region(region)
            .build();

        putSubFilters(cwl, filter, pattern, logGroup, functionArn);
        cwl.close();
    }

    public static void putSubFilters(CloudWatchLogsClient cwl,
        String filter,
        String pattern,
        String logGroup,
        String functionArn) {
```



```
try {
    PutSubscriptionFilterRequest request =
    PutSubscriptionFilterRequest.builder()
        .filterName(filter)
        .filterPattern(pattern)
        .logGroupName(logGroup)
        .destinationArn(functionArn)
        .build();

    cw1.putSubscriptionFilter(request);
    System.out.printf(
        "Successfully created CloudWatch logs subscription filter
%s",
        filter);

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

- API の詳細については、AWS SDK for Java 2.x API リファレンスの「[PutSubscriptionFilter](#)」を参照してください。

JavaScript

SDK for JavaScript (v3)

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
import { PutSubscriptionFilterCommand } from "@aws-sdk/client-cloudwatch-logs";
import { client } from "../libs/client.js";

const run = async () => {
    const command = new PutSubscriptionFilterCommand({
```

```
// An ARN of a same-account Kinesis stream, Kinesis Firehose
// delivery stream, or Lambda function.
// https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/
SubscriptionFilters.html
destinationArn: process.env.CLOUDWATCH_LOGS_DESTINATION_ARN,

// A name for the filter.
filterName: process.env.CLOUDWATCH_LOGS_FILTER_NAME,

// A filter pattern for subscribing to a filtered stream of log events.
// https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/
FilterAndPatternSyntax.html
filterPattern: process.env.CLOUDWATCH_LOGS_FILTER_PATTERN,

// The name of the log group. Messages in this group matching the filter
pattern
// will be sent to the destination ARN.
logGroupName: process.env.CLOUDWATCH_LOGS_LOG_GROUP,
});

try {
  return await client.send(command);
} catch (err) {
  console.error(err);
}
};

export default run();
```

- APIの詳細については、AWS SDK for JavaScript API リファレンスの「[PutSubscriptionFilter](#)」を参照してください。

SDK for JavaScript (v2)

 Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
```

```
// Set the region
AWS.config.update({ region: "REGION" });

// Create the CloudWatchLogs service object
var cw1 = new AWS.CloudWatchLogs({ apiVersion: "2014-03-28" });

var params = {
    destinationArn: "LAMBDA_FUNCTION_ARN",
    filterName: "FILTER_NAME",
    filterPattern: "ERROR",
    logGroupName: "LOG_GROUP",
};

cw1.putSubscriptionFilter(params, function (err, data) {
    if (err) {
        console.log("Error", err);
    } else {
        console.log("Success", data);
    }
});
```

- 詳細については、「[AWS SDK for JavaScript デベロッパーガイド](#)」を参照してください。
- API の詳細については、AWS SDK for JavaScript API リファレンスの「[PutSubscriptionFilter](#)」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK での CloudWatch Logs の使用](#)。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK **StartLiveTail**で を使用する

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

.NET

SDK for .NET

必要なファイルを含めます。

```
using Amazon;
using Amazon.CloudWatchLogs;
```

```
using Amazon.CloudWatchLogs.Model;
```

Live Tail セッションを開始します。

```
var client = new AmazonCloudWatchLogsClient();
var request = new StartLiveTailRequest
{
    LogGroupIdentifiers = logGroupIdentifiers,
    LogStreamNames = logStreamNames,
    LogEventFilterPattern = filterPattern,
};

var response = await client.StartLiveTailAsync(request);

// Catch if request fails
if (response.HttpStatusCode != System.Net.HttpStatusCode.OK)
{
    Console.WriteLine("Failed to start live tail session");
    return;
}
```

Live Tail セッションのイベントは 2 つの方法で処理できます。

```
/* Method 1
 * 1). Asynchronously loop through the event stream
 * 2). Set a timer to dispose the stream and stop the Live Tail
session at the end.
*/
var eventStream = response.ResponseStream;
var task = Task.Run(() =>
{
    foreach (var item in eventStream)
    {
        if (item is LiveTailSessionUpdate liveTailSessionUpdate)
        {
            foreach (var sessionResult in
liveTailSessionUpdate.SessionResults)
            {
                Console.WriteLine("Message : {0}",
sessionResult.Message);
            }
        }
    }
});
```

```

    }
    if (item is LiveTailSessionStart)
    {
        Console.WriteLine("Live Tail session started");
    }
    // On-stream exceptions are processed here
    if (item is CloudWatchLogsEventStreamException)
    {
        Console.WriteLine($"ERROR: {item}");
    }
}
});
// Close the stream to stop the session after a timeout
if (!task.Wait(TimeSpan.FromSeconds(10))){
    eventStream.Dispose();
    Console.WriteLine("End of line");
}

```

```

/* Method 2
 * 1). Add event handlers to each event variable
 * 2). Start processing the stream and wait for a timeout using
AutoResetEvent
*/
AutoResetEvent endEvent = new AutoResetEvent(false);
var eventStream = response.ResponseStream;
using (eventStream) // automatically disposes the stream to stop the
session after execution finishes
{
    eventStream.SessionStartReceived += (sender, e) =>
    {
        Console.WriteLine("LiveTail session started");
    };
    eventStream.SessionUpdateReceived += (sender, e) =>
    {
        foreach (LiveTailSessionLogEvent logEvent in
e.EventStreamEvent.SessionResults){
            Console.WriteLine("Message: {0}", logEvent.Message);
        }
    };
    // On-stream exceptions are captured here
    eventStream.ExceptionReceived += (sender, e) =>
    {

```

```
        Console.WriteLine($"ERROR:
{e.EventStreamException.Message}");
    };

    eventStream.StartProcessing();
    // Stream events for this amount of time.
    endEvent.WaitOne(TimeSpan.FromSeconds(10));
    Console.WriteLine("End of line");
}
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「[StartLiveTail](#)」を参照してください。

Go

SDK for Go V2

必要なファイルを含めます。

```
import (
    "context"
    "log"
    "time"

    "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/service/cloudwatchlogs"
    "github.com/aws/aws-sdk-go-v2/service/cloudwatchlogs/types"
)
```

Live Tail セッションのイベントを処理します。

```
func handleEventStreamAsync(stream *cloudwatchlogs.StartLiveTailEventStream) {
    eventsChan := stream.Events()
    for {
        event := <-eventsChan
        switch e := event.(type) {
        case *types.StartLiveTailResponseStreamMemberSessionStart:
            log.Println("Received SessionStart event")
        case *types.StartLiveTailResponseStreamMemberSessionUpdate:
            for _, logEvent := range e.Value.SessionResults {
```

```
    log.Println(*logEvent.Message)
}
default:
    // Handle on-stream exceptions
    if err := stream.Err(); err != nil {
        log.Fatalf("Error occurred during streaming: %v", err)
    } else if event == nil {
        log.Println("Stream is Closed")
        return
    } else {
        log.Fatalf("Unknown event type: %T", e)
    }
}
}
```

Live Tail セッションを開始します。

```
cfg, err := config.LoadDefaultConfig(context.TODO())
if err != nil {
    panic("configuration error, " + err.Error())
}
client := cloudwatchlogs.NewFromConfig(cfg)

request := &cloudwatchlogs.StartLiveTailInput{
    LogGroupIdentifiers: logGroupIdentifiers,
    LogStreamNames:      logStreamNames,
    LogEventFilterPattern: logEventFilterPattern,
}

response, err := client.StartLiveTail(context.TODO(), request)
// Handle pre-stream Exceptions
if err != nil {
    log.Fatalf("Failed to start streaming: %v", err)
}

// Start a Goroutine to handle events over stream
stream := response.GetStream()
go handleEventStreamAsync(stream)
```

一定時間が経過したら Live Tail セッションを停止します。

```
// Close the stream (which ends the session) after a timeout
time.Sleep(10 * time.Second)
stream.Close()
log.Println("Event stream closed")
```

- APIの詳細については、「AWS SDK for Go API リファレンス」の「[StartLiveTail](#)」を参照してください。

Java

SDK for Java 2.x

必要なファイルを含めます。

```
import io.reactivex.FlowableSubscriber;
import io.reactivex.annotations.NonNull;
import org.reactivestreams.Subscription;
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.services.cloudwatchlogs.CloudWatchLogsAsyncClient;
import
    software.amazon.awssdk.services.cloudwatchlogs.model.LiveTailSessionLogEvent;
import software.amazon.awssdk.services.cloudwatchlogs.model.LiveTailSessionStart;
import
    software.amazon.awssdk.services.cloudwatchlogs.model.LiveTailSessionUpdate;
import software.amazon.awssdk.services.cloudwatchlogs.model.StartLiveTailRequest;
import
    software.amazon.awssdk.services.cloudwatchlogs.model.StartLiveTailResponseHandler;
import
    software.amazon.awssdk.services.cloudwatchlogs.model.CloudWatchLogsException;
import
    software.amazon.awssdk.services.cloudwatchlogs.model.StartLiveTailResponseStream;

import java.util.Date;
import java.util.List;
import java.util.concurrent.atomic.AtomicReference;
```

Live Tail セッションのイベントを処理します。

```
private static StartLiveTailResponseHandler
getStartLiveTailResponseStreamHandler(
```



```

        AtomicReference<Subscription> subscriptionAtomicReference) {
    return StartLiveTailResponseHandler.builder()
        .onResponse(r -> System.out.println("Received initial response"))
        .onError(throwable -> {
            CloudWatchLogsException e = (CloudWatchLogsException)
throwable.getCause();
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        })
        .subscriber(() -> new FlowableSubscriber<>() {
            @Override
            public void onSubscribe(@NonNull Subscription s) {
                subscriptionAtomicReference.set(s);
                s.request(Long.MAX_VALUE);
            }

            @Override
            public void onNext(StartLiveTailResponseStream event) {
                if (event instanceof LiveTailSessionStart) {
                    LiveTailSessionStart sessionStart =
(LiveTailSessionStart) event;
                    System.out.println(sessionStart);
                } else if (event instanceof LiveTailSessionUpdate) {
                    LiveTailSessionUpdate sessionUpdate =
(LiveTailSessionUpdate) event;
                    List<LiveTailSessionLogEvent> logEvents =
sessionUpdate.sessionResults();
                    logEvents.forEach(e -> {
                        long timestamp = e.timestamp();
                        Date date = new Date(timestamp);
                        System.out.println "[" + date + "] " + e.message());
                    });
                } else {
                    throw CloudWatchLogsException.builder().message("Unknown
event type").build();
                }
            }

            @Override
            public void onError(Throwable throwable) {
                System.out.println(throwable.getMessage());
                System.exit(1);
            }
        })
    }
}

```

```

        @Override
        public void onComplete() {
            System.out.println("Completed Streaming Session");
        }
    })
    .build();
}

```

Live Tail セッションを開始します。

```

CloudWatchLogsAsyncClient cloudWatchLogsAsyncClient =
    CloudWatchLogsAsyncClient.builder()
        .credentialsProvider(ProfileCredentialsProvider.create())
        .build();

StartLiveTailRequest request =
    StartLiveTailRequest.builder()
        .logGroupIdentifiers(logGroupIdentifiers)
        .logStreamNames(logStreamNames)
        .logEventFilterPattern(logEventFilterPattern)
        .build();

/* Create a reference to store the subscription */
final AtomicReference<Subscription> subscriptionAtomicReference = new
AtomicReference<>(null);

cloudWatchLogsAsyncClient.startLiveTail(request,
getStartLiveTailResponseStreamHandler(subscriptionAtomicReference));

```

一定時間が経過したら Live Tail セッションを停止します。

```

/* Set a timeout for the session and cancel the subscription. This will:
 * 1). Close the stream
 * 2). Stop the Live Tail session
 */
try {
    Thread.sleep(10000);
} catch (InterruptedException e) {
    throw new RuntimeException(e);
}
if (subscriptionAtomicReference.get() != null) {

```

```
subscriptionAtomicReference.get().cancel();
System.out.println("Subscription to stream closed");
}
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「[StartLiveTail](#)」を参照してください。

JavaScript

SDK for JavaScript (v3)

必要なファイルを含めます。

```
import { CloudWatchLogsClient, StartLiveTailCommand } from "@aws-sdk/client-cloudwatch-logs";
```

Live Tail セッションのイベントを処理します。

```
async function handleResponseAsync(response) {
  try {
    for await (const event of response.responseStream) {
      if (event.sessionStart !== undefined) {
        console.log(event.sessionStart);
      } else if (event.sessionUpdate !== undefined) {
        for (const logEvent of event.sessionUpdate.sessionResults) {
          const timestamp = logEvent.timestamp;
          const date = new Date(timestamp);
          console.log "[" + date + "]" + logEvent.message);
        }
      } else {
        console.error("Unknown event type");
      }
    }
  } catch (err) {
    // On-stream exceptions are captured here
    console.error(err)
  }
}
```

Live Tail セッションを開始します。

```
const client = new CloudWatchLogsClient();

const command = new StartLiveTailCommand({
  logGroupIdentifiers: logGroupIdentifiers,
  logStreamNames: logStreamNames,
  logEventFilterPattern: filterPattern
});
try{
  const response = await client.send(command);
  handleResponseAsync(response);
} catch (err){
  // Pre-stream exceptions are captured here
  console.log(err);
}
```

一定時間が経過したら Live Tail セッションを停止します。

```
/* Set a timeout to close the client. This will stop the Live Tail session.
*/
setTimeout(function() {
  console.log("Client timeout");
  client.destroy();
}, 10000);
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「[StartLiveTail](#)」を参照してください。

Kotlin

SDK for Kotlin

必要なファイルを含めます。

```
import aws.sdk.kotlin.services.cloudwatchlogs.CloudWatchLogsClient
import aws.sdk.kotlin.services.cloudwatchlogs.model.StartLiveTailRequest
import aws.sdk.kotlin.services.cloudwatchlogs.model.StartLiveTailResponseStream
import kotlinx.coroutines.flow.takeWhile
```

Live Tail セッションを開始します。

```
val client = CloudWatchLogsClient.fromEnvironment()

val request = StartLiveTailRequest {
    logGroupIdentifiers = logGroupIdentifiersVal
    logStreamNames = logStreamNamesVal
    logEventFilterPattern = logEventFilterPatternVal
}

val startTime = System.currentTimeMillis()

try {
    client.startLiveTail(request) { response ->
        val stream = response.responseStream
        if (stream != null) {
            /* Set a timeout to unsubscribe from the flow. This will:
             * 1). Close the stream
             * 2). Stop the Live Tail session
             */
            stream.takeWhile { System.currentTimeMillis() - startTime <
10000 }.collect { value ->
                if (value is StartLiveTailResponseStream.SessionStart) {
                    println(value.asSessionStart())
                } else if (value is
StartLiveTailResponseStream.SessionUpdate) {
                    for (e in value.asSessionUpdate().sessionResults!!) {
                        println(e)
                    }
                } else {
                    throw IllegalArgumentException("Unknown event type")
                }
            }
        } else {
            throw IllegalArgumentException("No response stream")
        }
    }
} catch (e: Exception) {
    println("Exception occurred during StartLiveTail: $e")
    System.exit(1)
}
```

- API の詳細については、「AWS SDK for Kotlin API リファレンス」の「[StartLiveTail](#)」を参照してください。

Python

SDK for Python (Boto3)

必要なファイルを含めます。

```
import boto3
import time
from datetime import datetime
```

Live Tail セッションを開始します。

```
# Initialize the client
client = boto3.client('logs')

start_time = time.time()

try:
    response = client.start_live_tail(
        logGroupIdentifiers=log_group_identifiers,
        logStreamNames=log_streams,
        logEventFilterPattern=filter_pattern
    )
    event_stream = response['responseStream']
    # Handle the events streamed back in the response
    for event in event_stream:
        # Set a timeout to close the stream.
        # This will end the Live Tail session.
        if (time.time() - start_time >= 10):
            event_stream.close()
            break
        # Handle when session is started
        if 'sessionStart' in event:
            session_start_event = event['sessionStart']
            print(session_start_event)
        # Handle when log event is given in a session update
```

```
elif 'sessionUpdate' in event:
    log_events = event['sessionUpdate']['sessionResults']
    for log_event in log_events:
        print('[{date}]'
              {log}'.format(date=datetime.fromtimestamp(log_event['timestamp']/1000), log=log_event['message']))
    else:
        # On-stream exceptions are captured here
        raise RuntimeError(str(event))
except Exception as e:
    print(e)
```

- API の詳細については、「AWS SDK for Python (Boto3) API リファレンス」の「[StartLiveTail](#)」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK での CloudWatch Logs の使用](#)。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK **StartQuery**で を使用する

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

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- [大規模なクエリを実行する](#)

.NET

SDK for .NET (v4)

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
/// <summary>
/// Starts a CloudWatch Logs Insights query.
```

```
    /// </summary>
    /// <param name="logGroupName">The name of the log group to query.</param>
    /// <param name="queryString">The CloudWatch Logs Insights query string.</
param>
    /// <param name="startTime">The start time for the query (seconds since
epoch).</param>
    /// <param name="endTime">The end time for the query (seconds since epoch).</
param>
    /// <param name="limit">The maximum number of results to return.</param>
    /// <returns>The query ID if successful, null otherwise.</returns>
    public async Task<string?> StartQueryAsync(
        string logGroupName,
        string queryString,
        long startTime,
        long endTime,
        int limit = 10000)
    {
        try
        {
            var request = new StartQueryRequest
            {
                LogGroupName = logGroupName,
                QueryString = queryString,
                StartTime = startTime,
                EndTime = endTime,
                Limit = limit
            };

            var response = await _amazonCloudWatchLogs.StartQueryAsync(request);
            return response.QueryId;
        }
        catch (InvalidParameterException ex)
        {
            _logger.LogError($"Invalid parameter for query: {ex.Message}");
            return null;
        }
        catch (ResourceNotFoundException ex)
        {
            _logger.LogError($"Log group not found: {ex.Message}");
            return null;
        }
        catch (Exception ex)
        {

```



```
        _logger.LogError($"An error occurred while starting query: {ex.Message}");
        return null;
    }
}
```

- APIの詳細については、「AWS SDK for .NET API リファレンス」の「[StartQuery](#)」を参照してください。

JavaScript

SDK for JavaScript (v3)

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
/**
 * Wrapper for the StartQueryCommand. Uses a static query string
 * for consistency.
 * @param {[Date, Date]} dateRange
 * @param {number} maxLogs
 * @returns {Promise<{ queryId: string }>}
 */
async _startQuery([startDate, endDate], maxLogs = 10000) {
    try {
        return await this.client.send(
            new StartQueryCommand({
                logGroupNames: this.logGroupNames,
                queryString: "fields @timestamp, @message | sort @timestamp asc",
                startTime: startDate.valueOf(),
                endTime: endDate.valueOf(),
                limit: maxLogs,
            }),
        );
    } catch (err) {
        /** @type {string} */
        const message = err.message;
```

```
    if (message.startsWith("Query's end date and time")) {  
        // This error indicates that the query's start or end date occur  
        // before the log group was created.  
        throw new DateOutOfBoundsError(message);  
    }  
  
    throw err;  
}  
}
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「[StartQuery](#)」を参照してください。

Python

SDK for Python (Boto3)

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
def perform_query(self, date_range):  
    """  
    Performs the actual CloudWatch log query.  
  
    :param date_range: A tuple representing the start and end datetime for  
    the query.  
    :type date_range: tuple  
    :return: A list containing the query results.  
    :rtype: list  
    """  
    client = boto3.client("logs")  
    try:  
        try:  
            start_time = round(  
  
self.date_utilities.convert_iso8601_to_unix_timestamp(date_range[0])  
            )
```

```

        end_time = round(

self.date_utilities.convert_iso8601_to_unix_timestamp(date_range[1])
        )
        response = client.start_query(
            logGroupName=self.log_group,
            startTime=start_time,
            endTime=end_time,
            queryString=self.query_string,
            limit=self.limit,
        )
        query_id = response["queryId"]
    except client.exceptions.ResourceNotFoundException as e:
        raise DateOutOfBoundsError(f"Resource not found: {e}")
    while True:
        time.sleep(1)
        results = client.get_query_results(queryId=query_id)
        if results["status"] in [
            "Complete",
            "Failed",
            "Cancelled",
            "Timeout",
            "Unknown",
        ]:
            return results.get("results", [])
    except DateOutOfBoundsError:
        return []

def _initiate_query(self, client, date_range, max_logs):
    """
    Initiates the CloudWatch logs query.

    :param date_range: A tuple representing the start and end datetime for
the query.
    :type date_range: tuple
    :param max_logs: The maximum number of logs to retrieve.
    :type max_logs: int
    :return: The query ID as a string.
    :rtype: str
    """
    try:
        start_time = round(

self.date_utilities.convert_iso8601_to_unix_timestamp(date_range[0])

```

```

    )
    end_time = round(

self.date_utilities.convert_iso8601_to_unix_timestamp(date_range[1])
    )
    response = client.start_query(
        logGroupName=self.log_group,
        startTime=start_time,
        endTime=end_time,
        queryString=self.query_string,
        limit=max_logs,
    )
    return response["queryId"]
except client.exceptions.ResourceNotFoundException as e:
    raise DateOutOfBoundsError(f"Resource not found: {e}")

```

- APIの詳細については、AWS SDK for Python (Boto3) API リファレンスの「[StartQuery](#)」を参照してください。

SAP ABAP

SDK for SAP ABAP

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```

TRY.
    " iv_log_group_name = '/aws/lambda/my-function'
    " iv_query_string = 'fields @timestamp, @message | sort @timestamp desc |
limit 20'
    " iv_start_time and iv_end_time must be in Unix epoch milliseconds (ms
since Jan 1, 1970 00:00:00 UTC)
    oo_result = lo_cwl->startquery(
        iv_loggroupname = iv_log_group_name
        iv_starttime     = iv_start_time
        iv_endtime       = iv_end_time

```

```

        iv_querystring = iv_query_string
        iv_limit        = iv_limit ).

" Display the query ID for tracking
DATA(lv_query_id) = oo_result->get_queryid( ).
MESSAGE |Query started successfully with ID: { lv_query_id }| TYPE 'I'.
CATCH /aws1/cx_cwlinvalidparameterex.
MESSAGE 'Invalid parameter.' TYPE 'E'.
CATCH /aws1/cx_cwllimitexceededex.
MESSAGE 'Limit exceeded.' TYPE 'E'.
CATCH /aws1/cx_cwlmalformedqueryex.
MESSAGE 'Malformed query.' TYPE 'E'.
CATCH /aws1/cx_cwlresourcenotfoundex.
MESSAGE 'Resource not found.' TYPE 'E'.
CATCH /aws1/cx_cwlserviceunavailex.
MESSAGE 'Service unavailable.' TYPE 'E'.
ENDTRY.

```

- APIの詳細については、AWS SDK for SAP ABAP API リファレンスの[StartQuery](#)を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK での CloudWatch Logs の使用](#)。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

SDK を使用した CloudWatch Logs のシナリオ AWS SDKs

次のコード例は、AWS SDKs を使用して CloudWatch Logs に一般的なシナリオを実装する方法を示しています。これらのシナリオは、CloudWatch Logs 内または他の AWS のサービスと組み合わせた複数の関数を呼び出し、特定のタスクを実行する方法を示します。各シナリオには、完全なソースコードへのリンクが含まれており、そこからコードの設定方法と実行方法に関する手順を確認できます。

シナリオは、サービスアクションをコンテキストで理解するのに役立つ中級レベルの経験を対象としています。

例

- [Amazon ECS Service Connect を設定する](#)
- [最初の Lambda 関数を作成する](#)

- [CloudWatch Logs を使用して大規模なクエリを実行する](#)
- [スケジュールされたイベントを使用した Lambda 関数の呼び出し](#)

Amazon ECS Service Connect を設定する

次のコード例は、以下の操作方法を示しています。

- VPC インフラストラクチャを作成する
- ログ記録のセットアップ
- ECS クラスターを作成する
- IAM ロールを設定する
- Service Connect を使用してサービスを作成する
- デプロイメントを確認する
- リソースをクリーンアップする

Bash

AWS CLI Bash スクリプトを使用する

Note

GitHub には、その他のリソースもあります。[デベロッパーチュートリアル](#)のサンプルレポジトリで用例一覧を検索し、設定と実行の方法を確認してください。

```
#!/bin/bash

# ECS Service Connect Tutorial Script v4 - Modified to use Default VPC
# This script creates an ECS cluster with Service Connect and deploys an nginx
# service
# Uses the default VPC to avoid VPC limits

set -e # Exit on any error

# Configuration
SCRIPT_NAME="ECS Service Connect Tutorial"
LOG_FILE="ecs-service-connect-tutorial-v4-default-vpc.log"
```

```
REGION=${AWS_DEFAULT_REGION:-${AWS_REGION:-$(aws configure get region 2>/dev/
null)}}}
if [ -z "$REGION" ]; then
    echo "ERROR: No AWS region configured."
    echo "Set one with: aws configure set region us-east-1"
    exit 1
fi
ENV_PREFIX="tutorial"
CLUSTER_NAME="${ENV_PREFIX}-cluster"
NAMESPACE_NAME="service-connect"

# Generate random suffix for unique resource names
RANDOM_SUFFIX=$(openssl rand -hex 6)

# Arrays to track created resources for cleanup
declare -a CREATED_RESOURCES=()

# Logging function
log() {
    echo "[$(date '+%Y-%m-%d %H:%M:%S')] $1" | tee -a "$LOG_FILE"
}

# Error handling function
handle_error() {
    log "ERROR: Script failed at line $1"
    log "Attempting to clean up resources..."
    cleanup_resources
    exit 1
}

# Set up error handling
trap 'handle_error $LINENO' ERR

# Function to add resource to tracking array
track_resource() {
    CREATED_RESOURCES+=("$1")
    log "Tracking resource: $1"
}

# Function to check if command output contains actual errors
check_for_errors() {
    local output="$1"
    local command_name="$2"
```

```
# Check for specific AWS CLI error patterns, not just any occurrence of
"error"
    if echo "$output" | grep -qi "An error occurred\|InvalidParameterException\|
AccessDenied\|ValidationException\|ResourceNotFoundException"; then
        log "ERROR in $command_name: $output"
        return 1
    fi
    return 0
}

# Function to get AWS account ID
get_account_id() {
    ACCOUNT_ID=$(aws sts get-caller-identity --query Account --output text)
    log "Using AWS Account ID: $ACCOUNT_ID"
}

# Function to wait for resources to be ready
wait_for_resource() {
    local resource_type="$1"
    local resource_id="$2"

    case "$resource_type" in
        "cluster")
            log "Waiting for cluster $resource_id to be active..."
            local attempt=1
            local max_attempts=30
            while [ $attempt -le $max_attempts ]; do
                local status=$(aws ecs describe-clusters --clusters
"$resource_id" --query 'clusters[0].status' --output text)
                if [ "$status" = "ACTIVE" ]; then
                    log "Cluster is now active"
                    return 0
                fi
                log "Cluster status: $status (attempt $attempt/$max_attempts)"
                sleep 10
                ((attempt++))
            done
            log "ERROR: Cluster did not become active within expected time"
            return 1
            ;;
        "service")
            log "Waiting for service $resource_id to be stable..."
            aws ecs wait services-stable --cluster "$CLUSTER_NAME" --services
"$resource_id"
```



```

        ;;
        "nat-gateway")
            log "Waiting for NAT Gateway $resource_id to be available..."
            aws ec2 wait nat-gateway-available --nat-gateway-ids "$resource_id"
            ;;
    esac
}

# Function to use default VPC infrastructure
setup_default_vpc_infrastructure() {
    log "Using default VPC infrastructure..."

    # Get default VPC
    VPC_ID=$(aws ec2 describe-vpcs --filters "Name=isDefault,Values=true" --query
'Vpcs[0].VpcId' --output text)
    if [[ "$VPC_ID" == "None" || -z "$VPC_ID" ]]; then
        log "ERROR: No default VPC found. Please create a default VPC first."
        exit 1
    fi
    log "Using default VPC: $VPC_ID"

    # Get default subnets
    SUBNETS=$(aws ec2 describe-subnets --filters "Name=vpc-id,Values=$VPC_ID"
"Name=default-for-az,Values=true" --query 'Subnets[].SubnetId' --output text)
    SUBNET_ARRAY=($SUBNETS)

    if [ ${#SUBNET_ARRAY[@]} -lt 2 ]; then
        log "ERROR: Need at least 2 subnets for ECS Service Connect. Found:
${#SUBNET_ARRAY[@]}"
        exit 1
    fi

    PUBLIC_SUBNET1=${SUBNET_ARRAY[0]}
    PUBLIC_SUBNET2=${SUBNET_ARRAY[1]}

    log "Using subnets: $PUBLIC_SUBNET1, $PUBLIC_SUBNET2"

    # Create security group for ECS tasks
    SG_OUTPUT=$(aws ec2 create-security-group \
        --group-name "${ENV_PREFIX}-ecs-sg-${RANDOM_SUFFIX}" \
        --description "Security group for ECS Service Connect tutorial" \
        --vpc-id "$VPC_ID" \

```

```

    --tag-specifications 'ResourceType=security-
group,Tags=[{Key=project,Value=doc-smith},{Key=tutorial,Value=amazon-ecs-service-
connect}]' 2>&1)
    check_for_errors "$SG_OUTPUT" "create-security-group"
    SECURITY_GROUP_ID=$(echo "$SG_OUTPUT" | grep -o '"GroupId": "[^"]*"' | cut -
d'"' -f4)
    track_resource "SG:$SECURITY_GROUP_ID"
    log "Created security group: $SECURITY_GROUP_ID"

    # Add inbound rules to security group
    aws ec2 authorize-security-group-ingress \
        --group-id "$SECURITY_GROUP_ID" \
        --protocol tcp \
        --port 80 \
        --cidr 0.0.0.0/0 >/dev/null 2>&1 || true

    aws ec2 authorize-security-group-ingress \
        --group-id "$SECURITY_GROUP_ID" \
        --protocol tcp \
        --port 443 \
        --cidr 0.0.0.0/0 >/dev/null 2>&1 || true

    log "Default VPC infrastructure setup completed"
}

# Function to create CloudWatch log groups
create_log_groups() {
    log "Creating CloudWatch log groups..."

    # Create log group for nginx container
    aws logs create-log-group --log-group-name "/ecs/service-connect-nginx"
    --tags project=doc-smith,tutorial=amazon-ecs-service-connect 2>&1 | grep -v
"ResourceAlreadyExistsException" || {
        if [ ${PIPESTATUS[0]} -eq 0 ]; then
            log "Log group /ecs/service-connect-nginx created"
            track_resource "LOG_GROUP:/ecs/service-connect-nginx"
        else
            log "Log group /ecs/service-connect-nginx already exists"
        fi
    }

    # Create log group for service connect proxy

```

```

    aws logs create-log-group --log-group-name "/ecs/service-connect-proxy"
    --tags project=doc-smith,tutorial=amazon-ecs-service-connect 2>&1 | grep -v
    "ResourceAlreadyExistsException" || {
        if [ ${PIPESTATUS[0]} -eq 0 ]; then
            log "Log group /ecs/service-connect-proxy created"
            track_resource "LOG_GROUP:/ecs/service-connect-proxy"
        else
            log "Log group /ecs/service-connect-proxy already exists"
        fi
    }
}

# Function to create ECS cluster with Service Connect
create_ecs_cluster() {
    log "Creating ECS cluster with Service Connect..."

    CLUSTER_OUTPUT=$(aws ecs create-cluster \
        --cluster-name "$CLUSTER_NAME" \
        --service-connect-defaults namespace="$NAMESPACE_NAME" \
        --tags key=Environment,value=tutorial key=project,value=doc-smith
key=tutorial,value=amazon-ecs-service-connect 2>&1)
    check_for_errors "$CLUSTER_OUTPUT" "create-cluster"

    track_resource "CLUSTER:$CLUSTER_NAME"
    log "Created ECS cluster: $CLUSTER_NAME"

    wait_for_resource "cluster" "$CLUSTER_NAME"

    # Track the Service Connect namespace that gets created
    # Wait a moment for the namespace to be created
    sleep 5
    NAMESPACE_ID=$(aws servicediscovery list-namespaces \
        --filters Name=TYPE,Values=HTTP \
        --query "Namespaces[?Name=='$NAMESPACE_NAME'].Id" --output text 2>/dev/
null || echo "")

    if [[ -n "$NAMESPACE_ID" && "$NAMESPACE_ID" != "None" ]]; then
        track_resource "NAMESPACE:$NAMESPACE_ID"
        log "Service Connect namespace created: $NAMESPACE_ID"
    fi
}

# Function to create IAM roles
create_iam_roles() {

```

```

log "Creating IAM roles..."

# Check if ecsTaskExecutionRole exists
if aws iam get-role --role-name ecsTaskExecutionRole >/dev/null 2>&1; then
    log "IAM role ecsTaskExecutionRole exists"
else
    log "Creating ecsTaskExecutionRole..."
    aws iam create-role \
        --role-name ecsTaskExecutionRole \
        --assume-role-policy-document '{
            "Version": "2012-10-17",
            "Statement": [{
                "Effect": "Allow",
                "Principal": {"Service": "ecs-tasks.amazonaws.com"},
                "Action": "sts:AssumeRole"
            }]
        }' >/dev/null 2>&1
    aws iam attach-role-policy \
        --role-name ecsTaskExecutionRole \
        --policy-arn arn:aws:iam::aws:policy/service-role/
AmazonECSTaskExecutionRolePolicy >/dev/null 2>&1
    track_resource "ROLE:ecsTaskExecutionRole"
    aws iam tag-role --role-name ecsTaskExecutionRole --tags
Key=project,Value=doc-smith Key=tutorial,Value=amazon-ecs-service-connect
    log "Created ecsTaskExecutionRole"
    sleep 10
fi

# Check if ecsTaskRole exists, create if not
if aws iam get-role --role-name ecsTaskRole >/dev/null 2>&1; then
    log "IAM role ecsTaskRole exists"
else
    log "IAM role ecsTaskRole does not exist, will create it"

    # Create trust policy for ECS tasks
    cat > /tmp/ecs-task-trust-policy.json << EOF
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "ecs-tasks.amazonaws.com"
            },

```

```

        "Action": "sts:AssumeRole"
    }
]
}
EOF

aws iam create-role \
    --role-name ecsTaskRole \
    --assume-role-policy-document file:///tmp/ecs-task-trust-policy.json
>/dev/null

    track_resource "IAM_ROLE:ecsTaskRole"
    aws iam tag-role --role-name ecsTaskRole --tags Key=project,Value=doc-
smith Key=tutorial,Value=amazon-ecs-service-connect
    log "Created ecsTaskRole"

    # Wait for role to be available
    sleep 10
fi
}

# Function to create task definition
create_task_definition() {
    log "Creating task definition..."

    # Create task definition JSON
    cat > /tmp/task-definition.json << EOF
{
    "family": "service-connect-nginx",
    "networkMode": "awsvpc",
    "requiresCompatibilities": ["FARGATE"],
    "cpu": "256",
    "memory": "512",
    "executionRoleArn": "arn:aws:iam:${ACCOUNT_ID}:role/ecsTaskExecutionRole",
    "taskRoleArn": "arn:aws:iam:${ACCOUNT_ID}:role/ecsTaskRole",
    "containerDefinitions": [
        {
            "name": "nginx",
            "image": "public.ecr.aws/docker/library/nginx:latest",
            "portMappings": [
                {
                    "containerPort": 80,
                    "protocol": "tcp",
                    "name": "nginx-port"
                }
            ]
        }
    ]
}
    EOF
}

```

```

        }
    ],
    "essential": true,
    "logConfiguration": {
        "logDriver": "awslogs",
        "options": {
            "awslogs-group": "/ecs/service-connect-nginx",
            "awslogs-region": "${REGION}",
            "awslogs-stream-prefix": "ecs"
        }
    }
}
]
}
EOF

TASK_DEF_OUTPUT=$(aws ecs register-task-definition --cli-input-json file:///
tmp/task-definition.json 2>&1)
check_for_errors "$TASK_DEF_OUTPUT" "register-task-definition"

TASK_DEF_ARN=$(echo "$TASK_DEF_OUTPUT" | grep -o '"taskDefinitionArn":
"[^"]*"' | cut -d'"' -f4)
track_resource "TASK_DEF:service-connect-nginx"
log "Created task definition: $TASK_DEF_ARN"

# Clean up temporary file
rm -f /tmp/task-definition.json
}

# Function to create ECS service with Service Connect
create_ecs_service() {
    log "Creating ECS service with Service Connect..."

    # Create service definition JSON
    cat > /tmp/service-definition.json << EOF
{
    "serviceName": "service-connect-nginx-service",
    "cluster": "${CLUSTER_NAME}",
    "taskDefinition": "service-connect-nginx",
    "desiredCount": 1,
    "launchType": "FARGATE",
    "networkConfiguration": {
        "awsvpcConfiguration": {
            "subnets": ["${PUBLIC_SUBNET1}", "${PUBLIC_SUBNET2}"],

```

```
        "securityGroups": ["${SECURITY_GROUP_ID}"],
        "assignPublicIp": "ENABLED"
    },
    "serviceConnectConfiguration": {
        "enabled": true,
        "namespace": "${NAMESPACE_NAME}",
        "services": [
            {
                "portName": "nginx-port",
                "discoveryName": "nginx",
                "clientAliases": [
                    {
                        "port": 80,
                        "dnsName": "nginx"
                    }
                ]
            }
        ],
        "logConfiguration": {
            "logDriver": "awslogs",
            "options": {
                "awslogs-group": "/ecs/service-connect-proxy",
                "awslogs-region": "${REGION}",
                "awslogs-stream-prefix": "ecs-service-connect"
            }
        }
    },
    "tags": [
        {
            "key": "Environment",
            "value": "tutorial"
        },
        {
            "key": "project",
            "value": "doc-smith"
        },
        {
            "key": "tutorial",
            "value": "amazon-ecs-service-connect"
        }
    ]
}
EOF
```

```
    SERVICE_OUTPUT=$(aws ecs create-service --cli-input-json file:///tmp/service-
definition.json 2>&1)
    check_for_errors "$SERVICE_OUTPUT" "create-service"

    track_resource "SERVICE:service-connect-nginx-service"
    log "Created ECS service: service-connect-nginx-service"

    wait_for_resource "service" "service-connect-nginx-service"

    # Clean up temporary file
    rm -f /tmp/service-definition.json
}

# Function to verify deployment
verify_deployment() {
    log "Verifying deployment..."

    # Check service status
    SERVICE_STATUS=$(aws ecs describe-services \
        --cluster "$CLUSTER_NAME" \
        --services "service-connect-nginx-service" \
        --query 'services[0].status' --output text)
    log "Service status: $SERVICE_STATUS"

    # Check running tasks
    RUNNING_COUNT=$(aws ecs describe-services \
        --cluster "$CLUSTER_NAME" \
        --services "service-connect-nginx-service" \
        --query 'services[0].runningCount' --output text)
    log "Running tasks: $RUNNING_COUNT"

    # Get task ARN
    TASK_ARN=$(aws ecs list-tasks \
        --cluster "$CLUSTER_NAME" \
        --service-name "service-connect-nginx-service" \
        --query 'taskArns[0]' --output text)

    if [[ "$TASK_ARN" != "None" && -n "$TASK_ARN" ]]; then
        log "Task ARN: $TASK_ARN"

        # Try to get task IP address
        TASK_IP=$(aws ecs describe-tasks \
            --cluster "$CLUSTER_NAME" \
```



```

        --tasks "$TASK_ARN" \
        --query 'tasks[0].attachments[0].details[?
name=='`privateIPv4Address`'].value' \
        --output text 2>/dev/null || echo "")

    if [[ -n "$TASK_IP" && "$TASK_IP" != "None" ]]; then
        log "Task IP address: $TASK_IP"
    else
        log "Could not retrieve task IP address"
    fi
fi

# Check Service Connect namespace
NAMESPACE_STATUS=$(aws servicediscovery list-namespaces \
    --filters Name=TYPE,Values=HTTP \
    --query "Namespaces[?Name=='$NAMESPACE_NAME'].Id" --output text 2>/dev/
null || echo "")

if [[ -n "$NAMESPACE_STATUS" && "$NAMESPACE_STATUS" != "None" ]]; then
    log "Service Connect namespace '$NAMESPACE_NAME' is active"
else
    log "Service Connect namespace '$NAMESPACE_NAME' not found or not active"
fi

# Display Service Connect configuration
log "Service Connect configuration:"
aws ecs describe-services \
    --cluster "$CLUSTER_NAME" \
    --services "service-connect-nginx-service" \
    --query 'services[0].serviceConnectConfiguration' 2>/dev/null || true
}

# Function to display created resources
display_resources() {
    echo ""
    echo "======"
    echo "CREATED RESOURCES"
    echo "======"
    for resource in "${CREATED_RESOURCES[@]}; do
        echo "- $resource"
    done
    echo "======"
    echo ""
}

```

```

# Function to cleanup resources
cleanup_resources() {
    log "Starting cleanup process..."

    # Delete resources in reverse order of creation
    for ((i=${#CREATED_RESOURCES[@]}-1; i>=0; i--)); do
        resource="${CREATED_RESOURCES[i]}"
        resource_type=$(echo "$resource" | cut -d':' -f1)
        resource_id=$(echo "$resource" | cut -d':' -f2)

        log "Cleaning up $resource_type: $resource_id"

        case "$resource_type" in
            "SERVICE")
                aws ecs update-service --cluster "$CLUSTER_NAME" --service
"$resource_id" --desired-count 0 2>&1 | grep -qi "error" && log "Warning: Failed
to scale down service $resource_id"
                aws ecs wait services-stable --cluster "$CLUSTER_NAME" --services
"$resource_id" 2>/dev/null || true
                aws ecs delete-service --cluster "$CLUSTER_NAME" --service
"$resource_id" --force 2>&1 | grep -qi "error" && log "Warning: Failed to delete
service $resource_id"
                ;;
            "TASK_DEF")
                TASK_DEF_ARNS=$(aws ecs list-task-definitions --family-prefix
"$resource_id" --query 'taskDefinitionArns' --output text 2>/dev/null)
                for arn in $TASK_DEF_ARNS; do
                    aws ecs deregister-task-definition --task-definition "$arn"
>/dev/null 2>&1 || true
                done
                ;;
            "ROLE")
                aws iam detach-role-policy --role-name "$resource_id" --policy-
arn "arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy" 2>/
dev/null || true
                aws iam delete-role --role-name "$resource_id" 2>&1 | grep -qi
"error" && log "Warning: Failed to delete role $resource_id"
                ;;
            "IAM_ROLE")
                aws iam detach-role-policy --role-name "$resource_id" --policy-
arn "arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy" 2>/
dev/null || true

```

```

        aws iam delete-role --role-name "$resource_id" 2>&1 | grep -qi
"error" && log "Warning: Failed to delete role $resource_id"
        ;;
    "CLUSTER")
        aws ecs delete-cluster --cluster "$resource_id" 2>&1 | grep -qi
"error" && log "Warning: Failed to delete cluster $resource_id"
        ;;
    "SG")
        for attempt in 1 2 3 4 5; do
            if aws ec2 delete-security-group --group-id "$resource_id"
2>/dev/null; then
                break
            fi
            log "Security group $resource_id still has dependencies,
retrying in 30s ($attempt/5)..."
            sleep 30
        done
        ;;
    "LOG_GROUP")
        aws logs delete-log-group --log-group-name "$resource_id" 2>&1 |
grep -qi "error" && log "Warning: Failed to delete log group $resource_id"
        ;;
    "NAMESPACE")
        # First, delete any services in the namespace
        NAMESPACE_SERVICES=$(aws servicediscovery list-services \
            --filters Name=NAMESPACE_ID,Values="$resource_id" \
            --query 'Services[].Id' --output text 2>/dev/null || echo "")

        if [[ -n "$NAMESPACE_SERVICES" && "$NAMESPACE_SERVICES" !=
"None" ]]; then
            for service_id in $NAMESPACE_SERVICES; do
                aws servicediscovery delete-service --id "$service_id" >/
dev/null 2>&1 || true
                sleep 2
            done
        fi

        # Then delete the namespace
        aws servicediscovery delete-namespace --id "$resource_id" >/dev/
null 2>&1 || true
        ;;
    esac

    sleep 2 # Brief pause between deletions

```

```
done

# Clean up temporary files
rm -f /tmp/ecs-task-trust-policy.json
rm -f /tmp/task-definition.json
rm -f /tmp/service-definition.json

log "Cleanup completed"
}

# Main execution
main() {
    log "Starting $SCRIPT_NAME v4 (Default VPC)"
    log "Region: $REGION"
    log "Log file: $LOG_FILE"

    # Get AWS account ID
    get_account_id

    # Setup infrastructure using default VPC
    setup_default_vpc_infrastructure

    # Create CloudWatch log groups
    create_log_groups

    # Create ECS cluster
    create_ecs_cluster

    # Create IAM roles
    create_iam_roles

    # Create task definition
    create_task_definition

    # Create ECS service
    create_ecs_service

    # Verify deployment
    verify_deployment

    log "Tutorial completed successfully!"

    # Display created resources
    display_resources
```

```
# Ask user if they want to clean up
echo ""
echo "=====
echo "CLEANUP CONFIRMATION"
echo "=====
echo "Do you want to clean up all created resources? (y/n): "
CLEANUP_CHOICE="y"

if [[ "$CLEANUP_CHOICE" =~ ^[Yy]$ ]]; then
    cleanup_resources
    log "All resources have been cleaned up"
else
    log "Resources left intact. You can clean them up later by running the
cleanup function."
    echo ""
    echo "To clean up resources later, you can use the AWS CLI commands or
the AWS Management Console."
    echo "Remember to delete resources in the correct order to avoid
dependency issues."
fi
}

# Make script executable and run
chmod +x "$0"
main "$@"
```

- APIの詳細については、AWS CLI コマンドリファレンスの以下のトピックを参照してください。
 - [AttachRolePolicy](#)
 - [AuthorizeSecurityGroupIngress](#)
 - [CreateCluster](#)
 - [CreateLogGroup](#)
 - [CreateRole](#)
 - [CreateSecurityGroup](#)
 - [CreateService](#)
 - [DeleteCluster](#)
 - [DeleteLogGroup](#)

- [DeleteNamespace](#)
- [DeleteRole](#)
- [DeleteSecurityGroup](#)
- [DeleteService](#)
- [DeregisterTaskDefinition](#)
- [DescribeClusters](#)
- [DescribeServices](#)
- [DescribeSubnets](#)
- [DescribeTasks](#)
- [DescribeVpcs](#)
- [DetachRolePolicy](#)
- [GetCallerIdentity](#)
- [GetRole](#)
- [ListNamespaces](#)
- [ListServices](#)
- [ListTaskDefinitions](#)
- [ListTasks](#)
- [RegisterTaskDefinition](#)
- [UpdateService](#)
- [待機](#)

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK での CloudWatch Logs の使用](#)。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

最初の Lambda 関数を作成する

次のコード例は、以下の操作方法を示しています。

- Lambda 用に IAM ロールを作成する
- 関数コードを作成する

- Lambda 関数をテストする
- リソースをクリーンアップする

Bash

AWS CLI Bash スクリプトを使用する

Note

GitHub には、その他のリソースもあります。[デベロッパーチュートリアル](#)のサンプルレポジトリで用例一覧を検索し、設定と実行の方法を確認してください。

```
#!/bin/bash
# AWS Lambda - Create Your First Function
# This script creates a Lambda function, invokes it with a test event,
# views CloudWatch logs, and cleans up all resources.
#
# Source: https://docs.aws.amazon.com/lambda/latest/dg/getting-started.html
#
# Resources created:
#   - IAM role (Lambda execution role with basic logging permissions)
#   - Lambda function (Python 3.13 or Node.js 22.x runtime)
#   - CloudWatch log group (created automatically by Lambda on invocation)

set -eE -o pipefail

#####
# Setup
#####

UNIQUE_ID=$(head -c 8 /dev/urandom | od -An -tx1 | tr -d ' ')
FUNCTION_NAME="my-lambda-function-${UNIQUE_ID}"
ROLE_NAME="lambda-execution-role-${UNIQUE_ID}"
LOG_GROUP_NAME="/aws/lambda/${FUNCTION_NAME}"

TEMP_DIR=$(mktemp -d)
readonly TEMP_DIR
LOG_FILE="${TEMP_DIR}/lambda-gettingstarted.log"

exec > >(tee -a "$LOG_FILE") 2>&1
```

```

declare -a CREATED_RESOURCES

#####
# Helper functions
#####

cleanup_resources() {
    # Disable error trap to prevent recursion during cleanup
    trap - ERR
    set +eE

    echo ""
    echo "Cleaning up resources..."
    echo ""

    for ((i=${#CREATED_RESOURCES[@]}-1; i>=0; i--)); do
        local RESOURCE="${CREATED_RESOURCES[$i]}"
        local TYPE="${RESOURCE%%:*}"
        local NAME="${RESOURCE#*:}"

        case "$TYPE" in
            log-group)
                echo "Deleting CloudWatch log group: ${NAME}"
                aws logs delete-log-group \
                    --log-group-name "$NAME" 2>&1 || echo " WARNING: Could not
delete log group ${NAME}."
                ;;
            lambda-function)
                echo "Deleting Lambda function: ${NAME}"
                aws lambda delete-function \
                    --function-name "$NAME" 2>&1 || echo " WARNING: Could not
delete Lambda function ${NAME}."
                echo " Waiting for function deletion to complete..."
                local DELETE_WAIT=0
                while aws lambda get-function --function-name "$NAME" > /dev/null
2>&1; do
                    sleep 2
                    DELETE_WAIT=$((DELETE_WAIT + 2))
                    if [ "$DELETE_WAIT" -ge 60 ]; then
                        echo " WARNING: Timed out waiting for function
deletion."
                        break
                    fi
                done
            *)
                ;;
        esac
    done
}

```



```

        done
        ;;
    iam-role-policy)
        local ROLE_PART="${NAME%|*}"
        local POLICY_PART="${NAME#*|}"
        echo "Detaching policy from role: ${ROLE_PART}"
        aws iam detach-role-policy \
            --role-name "$ROLE_PART" \
            --policy-arn "$POLICY_PART" 2>&1 || echo " WARNING: Could
not detach policy from role ${ROLE_PART}."
        ;;
    iam-role)
        echo "Deleting IAM role: ${NAME}"
        aws iam delete-role \
            --role-name "$NAME" 2>&1 || echo " WARNING: Could not delete
IAM role ${NAME}."
        ;;
    esac
done

if [ -d "$TEMP_DIR" ]; then
    rm -rf "$TEMP_DIR"
fi

echo ""
echo "Cleanup complete."
}

handle_error() {
    echo ""
    echo "======"
    echo "ERROR: Script failed at $1"
    echo "======"
    echo ""
    if [ ${#CREATED_RESOURCES[@]} -gt 0 ]; then
        echo "Attempting to clean up ${#CREATED_RESOURCES[@]} resource(s)..."
        cleanup_resources
    fi
    exit 1
}

trap 'handle_error "line $LINENO"' ERR

wait_for_resource() {

```

```

    local DESCRIPTION="$1"
    local COMMAND="$2"
    local TARGET_VALUE="$3"
    local TIMEOUT=300
    local ELAPSED=0
    local INTERVAL=5

    echo "Waiting for ${DESCRIPTION}..."
    while true; do
        local RESULT
        RESULT=$(eval "$COMMAND" 2>&1) || true
        if echo "$RESULT" | grep -q "$TARGET_VALUE"; then
            echo "  ${DESCRIPTION} is ready."
            return 0
        fi
        if [ "$ELAPSED" -ge "$TIMEOUT" ]; then
            echo "ERROR: Timed out waiting for ${DESCRIPTION} after ${TIMEOUT}
seconds."
            return 1
        fi
        sleep "$INTERVAL"
        ELAPSED=$((ELAPSED + INTERVAL))
    done
}

validate_input() {
    local input="$1"
    local pattern="$2"
    if ! [[ "$input" =~ $pattern ]]; then
        echo "ERROR: Invalid input: $input"
        return 1
    fi
    return 0
}

#####
# Region pre-check
#####

CONFIGURED_REGION=$(aws configure get region 2>/dev/null || true)
if [ -z "$CONFIGURED_REGION" ] && [ -z "$AWS_DEFAULT_REGION" ] && [ -z
"$AWS_REGION" ]; then
    echo "ERROR: No AWS region configured."
    echo "Run 'aws configure set region <region>' or export AWS_DEFAULT_REGION."

```

```

        exit 1
    fi

#####
# Runtime selection
#####

echo ""
echo "=====
echo "AWS Lambda - Create Your First Function"
echo "=====
echo ""
echo "Select a runtime for your Lambda function:"
echo "  1) Python 3.13"
echo "  2) Node.js 22.x"
echo ""
echo "Using default: Python 3.13"
RUNTIME_CHOICE="1"

case "$RUNTIME_CHOICE" in
    1)
        RUNTIME="python3.13"
        HANDLER="lambda_function.lambda_handler"
        CODE_FILE="lambda_function.py"
        cat > "${TEMP_DIR}/${CODE_FILE}" << 'PYTHON_EOF'
import json
import logging
logger = logging.getLogger()
logger.setLevel(logging.INFO)
def lambda_handler(event, context):
    if not isinstance(event, dict) or 'length' not in event or 'width' not in event:
        raise ValueError('Event must contain length and width')
    try:
        length = float(event['length'])
        width = float(event['width'])
        if length < 0 or width < 0:
            raise ValueError('Length and width must be non-negative')
        area = calculate_area(length, width)
        print(f'The area is {area}')
        logger.info(f'CloudWatch logs group: {context.log_group_name}')
        return json.dumps({'area': area})
    except (TypeError, ValueError) as e:
        logger.error(f'Error processing input: {str(e)}')

```

```

        raise
def calculate_area(length, width):
    return length * width
PYTHON_EOF
    echo "Selected runtime: Python 3.13"
    ;;
2)
    RUNTIME="nodejs22.x"
    HANDLER="index.handler"
    CODE_FILE="index.mjs"
    cat > "${TEMP_DIR}/${CODE_FILE}" << 'NODEJS_EOF'
export const handler = async (event, context) => {
    if (!event || typeof event.length !== 'number' || typeof event.width !==
'number') {
        throw new Error('Event must contain numeric length and width');
    }
    if (event.length < 0 || event.width < 0) {
        throw new Error('Length and width must be non-negative');
    }
    const area = event.length * event.width;
    console.log(`The area is ${area}`);
    console.log('CloudWatch log group: ', context.logGroupName);
    return JSON.stringify({area});
};
NODEJS_EOF
    echo "Selected runtime: Node.js 22.x"
    ;;
*)
    echo "ERROR: Invalid choice. Please enter 1 or 2."
    exit 1
    ;;
esac

#####
# Step 1: Create IAM execution role
#####

echo ""
echo "=====
echo "Step 1: Create IAM execution role"
echo "=====
echo ""

TRUST_POLICY='{

```

```

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

echo "Creating IAM role: ${ROLE_NAME}"
ROLE_OUTPUT=$(aws iam create-role \
    --role-name "$ROLE_NAME" \
    --assume-role-policy-document "$TRUST_POLICY" \
    --query 'Role.Arn' \
    --output text 2>&1)

if ! validate_input "$ROLE_OUTPUT" "^arn:aws:iam::[0-9]+:role/"; then
    echo "ERROR: Failed to create IAM role"
    exit 1
fi

echo "$ROLE_OUTPUT"
ROLE_ARN="$ROLE_OUTPUT"
CREATED_RESOURCES+=("iam-role:${ROLE_NAME}")
echo "Role ARN: ${ROLE_ARN}"

aws iam tag-role \
    --role-name "$ROLE_NAME" \
    --tags Key=project,Value=doc-smith Key=tutorial,Value=lambda-gettingstarted

echo ""
echo "Attaching AWSLambdaBasicExecutionRole policy..."
aws iam attach-role-policy \
    --role-name "$ROLE_NAME" \
    --policy-arn "arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole" 2>&1
CREATED_RESOURCES+=("iam-role-policy:${ROLE_NAME}|arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole")
echo "Policy attached."

# IAM roles can take a few seconds to propagate

```

```
echo "Waiting for IAM role to propagate..."
sleep 10

#####
# Step 2: Create Lambda function
#####

echo ""
echo "=====
echo "Step 2: Create Lambda function"
echo "=====
echo ""

echo "Creating deployment package..."
ORIGINAL_DIR=$(pwd)
cd "$TEMP_DIR" || exit 1
zip -j function.zip "$CODE_FILE" > /dev/null 2>&1 || {
    echo "ERROR: Failed to create deployment package"
    exit 1
}
cd "$ORIGINAL_DIR" || exit 1

if [ ! -f "${TEMP_DIR}/function.zip" ]; then
    echo "ERROR: Deployment package creation failed"
    exit 1
fi

echo "Creating Lambda function: ${FUNCTION_NAME}"
echo "  Runtime: ${RUNTIME}"
echo "  Handler: ${HANDLER}"
echo ""

CREATE_OUTPUT=$(aws lambda create-function \
    --function-name "$FUNCTION_NAME" \
    --runtime "$RUNTIME" \
    --role "$ROLE_ARN" \
    --handler "$HANDLER" \
    --architectures x86_64 \
    --zip-file "fileb://${TEMP_DIR}/function.zip" \
    --tags project=doc-smith,tutorial=lambda-gettingstarted \
    --query '[FunctionName, FunctionArn, Runtime, State]' \
    --output text 2>&1)

if [ -z "$CREATE_OUTPUT" ]; then
```

```

    echo "ERROR: Failed to create Lambda function"
    exit 1
fi

echo "$CREATE_OUTPUT"
CREATED_RESOURCES+=("lambda-function:${FUNCTION_NAME}")

wait_for_resource "Lambda function to become Active" \
    "aws lambda get-function-configuration --function-name ${FUNCTION_NAME} --
    query State --output text" \
    "Active"

#####
# Step 3: Invoke the function
#####

echo ""
echo "====="
echo "Step 3: Invoke the function"
echo "====="
echo ""

TEST_EVENT='{"length": 6, "width": 7}'
echo "Invoking function with test event: ${TEST_EVENT}"
echo ""

echo "$TEST_EVENT" > "${TEMP_DIR}/test-event.json"

if ! validate_input "$TEST_EVENT" '"length": [0-9]+, "width": [0-9]+'; then
    echo "ERROR: Invalid test event format"
    exit 1
fi

INVOKE_OUTPUT=$(aws lambda invoke \
    --function-name "$FUNCTION_NAME" \
    --payload "fileb://${TEMP_DIR}/test-event.json" \
    --cli-read-timeout 30 \
    "${TEMP_DIR}/response.json" 2>&1)
echo "$INVOKE_OUTPUT"

if [ ! -f "${TEMP_DIR}/response.json" ]; then
    echo "ERROR: No response file generated"
    exit 1
fi

```

```

RESPONSE=$(cat "${TEMP_DIR}/response.json")
echo ""
echo "Function response: ${RESPONSE}"
echo ""

if echo "$INVOKE_OUTPUT" | grep -qi "functionerror"; then
    echo "WARNING: Function returned an error."
fi

#####
# Step 4: View CloudWatch logs
#####

echo ""
echo "=====
echo "Step 4: View CloudWatch Logs"
echo "=====
echo ""

echo "Log group: ${LOG_GROUP_NAME}"
echo ""

echo "Waiting for CloudWatch logs to be available..."

LOG_STREAMS=""
for i in $(seq 1 6); do
    LOG_STREAMS=$(aws logs describe-log-streams \
        --log-group-name "$LOG_GROUP_NAME" \
        --order-by LastEventTime \
        --descending \
        --query 'logStreams[0].logStreamName' \
        --output text 2>/dev/null) || true
    if [ -n "$LOG_STREAMS" ] && [ "$LOG_STREAMS" != "None" ]; then
        break
    fi
    LOG_STREAMS=""
    sleep 5
done

if [ -n "$LOG_STREAMS" ] && [ "$LOG_STREAMS" != "None" ]; then
    echo "Latest log stream: ${LOG_STREAMS}"
    echo ""
    echo "--- Log events ---"

```



```

LOG_EVENTS=$(aws logs get-log-events \
    --log-group-name "$LOG_GROUP_NAME" \
    --log-stream-name "$LOG_STREAMS" \
    --query 'events[].message' \
    --output text 2>&1) || true
echo "$LOG_EVENTS"
echo "--- End of log events ---"
else
    echo "No log streams found yet. Logs may take a moment to appear."
    echo "You can view them in the CloudWatch console:"
    echo "  Log group: ${LOG_GROUP_NAME}"
fi

CREATED_RESOURCES+=("log-group:${LOG_GROUP_NAME}")

aws logs tag-log-group \
    --log-group-name "$LOG_GROUP_NAME" \
    --tags project=doc-smith,tutorial=lambda-gettingstarted

#####
# Summary and cleanup
#####

echo ""
echo "=====
echo "SUMMARY"
echo "=====
echo ""
echo "Resources created:"
echo "  IAM role:          ${ROLE_NAME}"
echo "  Lambda function:   ${FUNCTION_NAME}"
echo "  CloudWatch logs:   ${LOG_GROUP_NAME}"
echo ""
echo "=====
echo "CLEANUP"
echo "=====
echo ""
echo "Cleaning up all created resources..."
cleanup_resources

echo ""
echo "Done."

```

- API の詳細については、AWS CLI コマンドリファレンスの以下のトピックを参照してください。
 - [AttachRolePolicy](#)
 - [CreateFunction](#)
 - [CreateRole](#)
 - [DeleteFunction](#)
 - [DeleteLogGroup](#)
 - [DeleteRole](#)
 - [DescribeLogStreams](#)
 - [DetachRolePolicy](#)
 - [GetFunction](#)
 - [GetFunctionConfiguration](#)
 - [GetLogEvents](#)
 - [Invoke](#)

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK での CloudWatch Logs の使用](#)。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

CloudWatch Logs を使用して大規模なクエリを実行する

次のコード例は、CloudWatch Logs を使用して 10,000 を超えるレコードをクエリする方法を示しています。

.NET

SDK for .NET (v4)

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
using System.Diagnostics;
using System.Text.RegularExpressions;
using Amazon.CloudFormation;
using Amazon.CloudFormation.Model;
using Amazon.CloudWatchLogs;
using Amazon.CloudWatchLogs.Model;
using CloudWatchLogsActions;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;

namespace CloudWatchLogsScenario;

public class LargeQueryWorkflow
{
    /*
        Before running this .NET code example, set up your development environment,
        including your credentials.

        This .NET code example performs the following tasks for the CloudWatch Logs
        Large Query workflow:

        1. Prepare the Application:
            - Prompt the user to deploy CloudFormation stack and generate sample logs.
            - Deploy the CloudFormation template for resource creation.
            - Generate 50,000 sample log entries using CloudWatch Logs API.
            - Wait 5 minutes for logs to be fully ingested.

        2. Execute Large Query:
            - Perform recursive queries to retrieve all logs using binary search.
            - Display progress for each query executed.
            - Show total execution time and logs found.

        3. Clean up:
            - Prompt the user to delete the CloudFormation stack and all resources.
            - Destroy the CloudFormation stack and wait until removed.
    */

    public static ILogger<LargeQueryWorkflow> _logger = null!;
    public static CloudWatchLogsWrapper _wrapper = null!;
    public static IAmazonCloudFormation _amazonCloudFormation = null!;

    private static string _logGroupName = "/workflows/cloudwatch-logs/large-
query";
```

```

private static string _logStreamName = "stream1";
private static long _queryStartDate;
private static long _queryEndDate;

public static bool _interactive = true;
public static string _stackName = "CloudWatchLargeQueryStack";
private static string _stackResourcePath = "../..../..../..../scenarios/
features/cloudwatch_logs_large_query/resources/stack.yaml";

public static async Task Main(string[] args)
{
    using var host = Host.CreateDefaultBuilder(args)
        .ConfigureLogging(logging =>
            logging.AddFilter("System", LogLevel.Debug)
                .AddFilter("Microsoft", LogLevel.Information))
        .ConfigureServices((_, services) =>
            services.AddAWSService<IAmazonCloudWatchLogs>()
                .AddAWSService<IAmazonCloudFormation>()
                .AddTransient<CloudWatchLogsWrapper>()
            )
        .Build();

    if (_interactive)
    {
        _logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<LargeQueryWorkflow>();

        _wrapper = host.Services.GetRequiredService<CloudWatchLogsWrapper>();
        _amazonCloudFormation =
host.Services.GetRequiredService<IAmazonCloudFormation>();
    }

    Console.WriteLine(new string('-', 80));
    Console.WriteLine("Welcome to the CloudWatch Logs Large Query
Scenario.");
    Console.WriteLine(new string('-', 80));
    Console.WriteLine("This scenario demonstrates how to perform large-scale
queries on");
    Console.WriteLine("CloudWatch Logs using recursive binary search to
retrieve more than");
    Console.WriteLine("the 10,000 result limit.");
    Console.WriteLine();

    try

```

```
{
    Console.WriteLine(new string('-', 80));
    var prepareSuccess = await PrepareApplication();
    Console.WriteLine(new string('-', 80));

    if (prepareSuccess)
    {
        Console.WriteLine(new string('-', 80));
        await ExecuteLargeQuery();
        Console.WriteLine(new string('-', 80));
    }

    Console.WriteLine(new string('-', 80));
    await Cleanup();
    Console.WriteLine(new string('-', 80));
}
catch (Exception ex)
{
    _logger.LogError(ex, "There was a problem with the scenario,
initiating cleanup...");
    _interactive = false;
    await Cleanup();
}

Console.WriteLine("CloudWatch Logs Large Query scenario completed.");
}

/// <summary>
/// Runs the scenario workflow. Used for testing.
/// </summary>
public static async Task RunScenario()
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine("Welcome to the CloudWatch Logs Large Query
Scenario.");
    Console.WriteLine(new string('-', 80));
    Console.WriteLine("This scenario demonstrates how to perform large-scale
queries on");
    Console.WriteLine("CloudWatch Logs using recursive binary search to
retrieve more than");
    Console.WriteLine("the 10,000 result limit.");
    Console.WriteLine();

    try
```

```
{
    Console.WriteLine(new string('-', 80));
    var prepareSuccess = await PrepareApplication();
    Console.WriteLine(new string('-', 80));

    if (prepareSuccess)
    {
        Console.WriteLine(new string('-', 80));
        await ExecuteLargeQuery();
        Console.WriteLine(new string('-', 80));
    }

    Console.WriteLine(new string('-', 80));
    await Cleanup();
    Console.WriteLine(new string('-', 80));
}
catch (Exception ex)
{
    _logger.LogError(ex, "There was a problem with the scenario,
initiating cleanup...");
    _interactive = false;
    await Cleanup();
}

Console.WriteLine("CloudWatch Logs Large Query scenario completed.");
}

/// <summary>
/// Prepares the application by creating the necessary resources.
/// </summary>
/// <returns>True if the application was prepared successfully.</returns>
public static async Task<bool> PrepareApplication()
{
    Console.WriteLine("Preparing the application...");
    Console.WriteLine();

    try
    {
        var deployStack = !_interactive || GetYesNoResponse(
            "Would you like to deploy the CloudFormation stack and generate
sample logs? (y/n) ");

        if (deployStack)
        {
```

```
        if (_interactive)
        {
            Console.Write(
                $"Enter a path for the CloudFormation stack
resource .yaml file (or press Enter for default '{_stackResourcePath}'): ");
            string? inputPath = Console.ReadLine();
            if (!string.IsNullOrEmpty(inputPath))
            {
                _stackResourcePath = inputPath;
            }
        }

        _stackName = PromptUserForStackName();

        var deploySuccess = await DeployCloudFormationStack(_stackName);

        if (deploySuccess)
        {
            Console.WriteLine();
            Console.WriteLine("Generating 50,000 sample log entries...");
            var generateSuccess = await GenerateSampleLogs();

            if (generateSuccess)
            {
                Console.WriteLine();
                Console.WriteLine("Sample logs created. Waiting 5 minutes
for logs to be fully ingested...");
                await WaitWithCountdown(300);

                Console.WriteLine("Application preparation complete.");
                return true;
            }
        }
    }
    else
    {
        _logGroupName = PromptUserForInput("Enter the log group name ",
_logGroupName);
        _logStreamName = PromptUserForInput("Enter the log stream name ",
_logStreamName);

        var startDateMs = PromptUserForLong("Enter the query start date
(milliseconds since epoch): ");
```

```

        var endDateMs = PromptUserForLong("Enter the query end date
(milliseonds since epoch): ");

        _queryStartDate = startDateMs / 1000;
        _queryEndDate = endDateMs / 1000;

        Console.WriteLine("Application preparation complete.");
        return true;
    }
}
catch (Exception ex)
{
    _logger.LogError(ex, "An error occurred while preparing the
application.");
}

Console.WriteLine("Application preparation failed.");
return false;
}

/// <summary>
/// Deploys the CloudFormation stack with the necessary resources.
/// </summary>
/// <param name="stackName">The name of the CloudFormation stack.</param>
/// <returns>True if the stack was deployed successfully.</returns>
private static async Task<bool> DeployCloudFormationStack(string stackName)
{
    Console.WriteLine($"\\nDeploying CloudFormation stack: {stackName}");

    try
    {
        var request = new CreateStackRequest
        {
            StackName = stackName,
            TemplateBody = await File.ReadAllTextAsync(_stackResourcePath)
        };

        var response = await _amazonCloudFormation.CreateStackAsync(request);

        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine($"CloudFormation stack creation started:
{stackName}");

```



```

        bool stackCreated = await
WaitForStackCompletion(response.StackId);

        if (stackCreated)
        {
            Console.WriteLine("CloudFormation stack created
successfully.");
            return true;
        }
        else
        {
            _logger.LogError($"CloudFormation stack creation failed:
{stackName}");
            return false;
        }
    }
    else
    {
        _logger.LogError($"Failed to create CloudFormation stack:
{stackName}");
        return false;
    }
}
catch (AlreadyExistsException)
{
    _logger.LogWarning($"CloudFormation stack '{stackName}' already
exists. Please provide a unique name.");
    var newStackName = PromptUserForStackName();
    return await DeployCloudFormationStack(newStackName);
}
catch (Exception ex)
{
    _logger.LogError(ex, $"An error occurred while deploying the
CloudFormation stack: {stackName}");
    return false;
}
}

/// <summary>
/// Waits for the CloudFormation stack to be in the CREATE_COMPLETE state.
/// </summary>
/// <param name="stackId">The ID of the CloudFormation stack.</param>
/// <returns>True if the stack was created successfully.</returns>
private static async Task<bool> WaitForStackCompletion(string stackId)

```

```
{
    int retryCount = 0;
    const int maxRetries = 30;
    const int retryDelay = 10000;

    while (retryCount < maxRetries)
    {
        var describeStacksRequest = new DescribeStacksRequest
        {
            StackName = stackId
        };

        var describeStacksResponse = await
        _amazonCloudFormation.DescribeStacksAsync(describeStacksRequest);

        if (describeStacksResponse.Stacks.Count > 0)
        {
            if (describeStacksResponse.Stacks[0].StackStatus ==
            StackStatus.CREATE_COMPLETE)
            {
                return true;
            }
            if (describeStacksResponse.Stacks[0].StackStatus ==
            StackStatus.CREATE_FAILED ||
                describeStacksResponse.Stacks[0].StackStatus ==
            StackStatus.ROLLBACK_COMPLETE)
            {
                return false;
            }
        }

        Console.WriteLine("Waiting for CloudFormation stack creation to
        complete...");
        await Task.Delay(retryDelay);
        retryCount++;
    }

    _logger.LogError("Timed out waiting for CloudFormation stack creation to
    complete.");
    return false;
}

/// <summary>
/// Generates sample logs directly using CloudWatch Logs API.
```

```
/// Creates 50,000 log entries spanning 5 minutes.
/// </summary>
/// <returns>True if logs were generated successfully.</returns>
private static async Task<bool> GenerateSampleLogs()
{
    const int totalEntries = 50000;
    const int entriesPerBatch = 10000;
    const int fiveMinutesMs = 5 * 60 * 1000;

    try
    {
        // Calculate timestamps
        var startTimeMs = DateTimeOffset.UtcNow.ToUnixTimeMilliseconds();
        var timestampIncrement = fiveMinutesMs / totalEntries;

        Console.WriteLine($"Generating {totalEntries} log entries...");

        var entryCount = 0;
        var currentTimestamp = startTimeMs;
        var numBatches = totalEntries / entriesPerBatch;

        // Generate and upload logs in batches
        for (int batchNum = 0; batchNum < numBatches; batchNum++)
        {
            var logEvents = new List<InputLogEvent>();

            for (int i = 0; i < entriesPerBatch; i++)
            {
                logEvents.Add(new InputLogEvent
                {
                    Timestamp =
                        DateTimeOffset.FromUnixTimeMilliseconds(currentTimestamp).UtcDateTime,
                    Message = $"Entry {entryCount}"
                });

                entryCount++;
                currentTimestamp += timestampIncrement;
            }

            // Upload batch
            var success = await _wrapper.PutLogEventsAsync(_logGroupName,
                _logStreamName, logEvents);
            if (!success)
            {

```

```

        _logger.LogError($"Failed to upload batch {batchNum + 1}/{
numBatches}");
        return false;
    }

    Console.WriteLine($"Uploaded batch {batchNum + 1}/{numBatches}");
}

// Set query date range (convert milliseconds to seconds for query
API)
_queryStartDate = startTimeMs / 1000;
_queryEndDate = (currentTimestamp - timestampIncrement) / 1000;

Console.WriteLine($"Query start date:
{DateTimeOffset.FromUnixTimeSeconds(_queryStartDate):yyyy-MM-
ddTHH:mm:ss.fffZ}");
Console.WriteLine($"Query end date:
{DateTimeOffset.FromUnixTimeSeconds(_queryEndDate):yyyy-MM-ddTHH:mm:ss.fffZ}");
Console.WriteLine($"Successfully uploaded {totalEntries} log
entries");

    return true;
}
catch (Exception ex)
{
    _logger.LogError(ex, "An error occurred while generating sample
logs.");
    return false;
}
}

/// <summary>
/// Executes the large query workflow.
/// </summary>
public static async Task ExecuteLargeQuery()
{
    Console.WriteLine("Starting recursive query to retrieve all logs...");
    Console.WriteLine();

    var queryLimit = PromptUserForInteger("Enter the query limit (max 10000)
", 10000);
    if (queryLimit > 10000) queryLimit = 10000;

    var queryString = "fields @timestamp, @message | sort @timestamp asc";

```

```
var stopwatch = Stopwatch.StartNew();
var allResults = await PerformLargeQuery(_logGroupName, queryString,
_queryStartDate, _queryEndDate, queryLimit);
stopwatch.Stop();

Console.WriteLine();
Console.WriteLine($"Queries finished in
{stopwatch.Elapsed.TotalSeconds:F3} seconds.");
Console.WriteLine($"Total logs found: {allResults.Count}");

// Check for duplicates
Console.WriteLine();
Console.WriteLine("Checking for duplicate logs...");
var duplicates = FindDuplicateLogs(allResults);
if (duplicates.Count > 0)
{
    Console.WriteLine($"WARNING: Found {duplicates.Count} duplicate log
entries!");
    Console.WriteLine("Duplicate entries (showing first 10):");
    foreach (var dup in duplicates.Take(10))
    {
        Console.WriteLine($" [{dup.Timestamp}] {dup.Message} (appears
{dup.Count} times)");
    }

    var uniqueCount = allResults.Count - duplicates.Sum(d => d.Count -
1);
    Console.WriteLine($"Unique logs: {uniqueCount}");
}
else
{
    Console.WriteLine("No duplicates found. All logs are unique.");
}
Console.WriteLine();

var viewSample = !_interactive || GetYesNoResponse("Would you like to see
a sample of the logs? (y/n) ");
if (viewSample)
{
    Console.WriteLine();
    Console.WriteLine($"Sample logs (first 10 of {allResults.Count}):");
    for (int i = 0; i < Math.Min(10, allResults.Count); i++)
    {
```

```
        var timestamp = allResults[i].Find(f => f.Field ==
"@timestamp")?.Value ?? "N/A";
        var message = allResults[i].Find(f => f.Field ==
"@message")?.Value ?? "N/A";
        Console.WriteLine($"[{timestamp}] {message}");
    }
}

/// <summary>
/// Performs a large query using recursive binary search.
/// </summary>
private static async Task<List<List<ResultField>>> PerformLargeQuery(
    string logGroupName,
    string queryString,
    long startTime,
    long endTime,
    int limit)
{
    var queryId = await _wrapper.StartQueryAsync(logGroupName, queryString,
startTime, endTime, limit);
    if (queryId == null)
    {
        return new List<List<ResultField>>>();
    }

    var results = await PollQueryResults(queryId);
    if (results == null || results.Count == 0)
    {
        return new List<List<ResultField>>>();
    }

    var startDate =
DateTimeOffset.FromUnixTimeSeconds(startTime).ToString("yyyy-MM-
ddTHH:mm:ss.fffZ");
    var endDate = DateTimeOffset.FromUnixTimeSeconds(endTime).ToString("yyyy-
MM-ddTHH:mm:ss.fffZ");
    Console.WriteLine($"Query date range: {startDate} ({startTime}s) to
{endDate} ({endTime}s). Found {results.Count} logs.");

    if (results.Count < limit)
    {
        Console.WriteLine($" -> Returning {results.Count} logs (less than
limit of {limit})");
    }
}
```

```
        return results;
    }

    Console.WriteLine($" -> Hit limit of {limit}. Need to split and
recurse.");

    // Get the timestamp of the last log (sorted to find the actual last one)
    var lastLogTimestamp = GetLastLogTimestamp(results);
    if (lastLogTimestamp == null)
    {
        Console.WriteLine($" -> No timestamp found in results. Returning
{results.Count} logs.");
        return results;
    }

    Console.WriteLine($" -> Last log timestamp: {lastLogTimestamp}");

    // Parse the timestamp and add 1 millisecond to avoid querying the same
log again
    var lastLogDate = DateTimeOffset.Parse(lastLogTimestamp + " +0000");
    Console.WriteLine($" -> Last log as DateTimeOffset: {lastLogDate:yyyy-
MM-ddTHH:mm:ss.fffZ} ({lastLogDate.ToUnixTimeSeconds()}s)");

    var offsetLastLogDate = lastLogDate.AddMilliseconds(1);
    Console.WriteLine($" -> Offset timestamp (last
+ 1ms): {offsetLastLogDate:yyyy-MM-ddTHH:mm:ss.fffZ}
({offsetLastLogDate.ToUnixTimeSeconds()}s)");

    // Convert to seconds, but round UP to the next second to avoid
overlapping with logs in the same second
    // This ensures we don't re-query logs that share the same second as the
last log
    var offsetLastLogTime = offsetLastLogDate.ToUnixTimeSeconds();
    if (offsetLastLogDate.Millisecond > 0)
    {
        offsetLastLogTime++; // Move to the next full second
        Console.WriteLine($" -> Adjusted to next full second:
{offsetLastLogTime}s
({DateTimeOffset.FromUnixTimeSeconds(offsetLastLogTime):yyyy-MM-
ddTHH:mm:ss.fffZ})");
    }

    Console.WriteLine($" -> Comparing:
offsetLastLogTime={offsetLastLogTime}s vs endTime={endTime}s");
```

```
        Console.WriteLine($" -> End time as date:
{DateTimeOffset.FromUnixTimeSeconds(endTime):yyyy-MM-ddTHH:mm:ss.fffZ}");

        // Check if there's any time range left to query
        if (offsetLastLogTime >= endTime)
        {
            Console.WriteLine($" -> No time range left to query. Offset time
({offsetLastLogTime}s) >= end time ({endTime}s)");
            return results;
        }

        // Split the remaining date range in half
        var (range1Start, range1End, range2Start, range2End) =
SplitDateRange(offsetLastLogTime, endTime);

        var range1StartDate =
DateTimeOffset.FromUnixTimeSeconds(range1Start).ToString("yyyy-MM-
ddTHH:mm:ss.fffZ");
        var range1EndDate =
DateTimeOffset.FromUnixTimeSeconds(range1End).ToString("yyyy-MM-
ddTHH:mm:ss.fffZ");
        var range2StartDate =
DateTimeOffset.FromUnixTimeSeconds(range2Start).ToString("yyyy-MM-
ddTHH:mm:ss.fffZ");
        var range2EndDate =
DateTimeOffset.FromUnixTimeSeconds(range2End).ToString("yyyy-MM-
ddTHH:mm:ss.fffZ");

        Console.WriteLine($" -> Splitting remaining range:");
        Console.WriteLine($"      Range 1: {range1StartDate} ({range1Start}s) to
{range1EndDate} ({range1End}s)");
        Console.WriteLine($"      Range 2: {range2StartDate} ({range2Start}s) to
{range2EndDate} ({range2End}s)");

        // Query both halves recursively
        Console.WriteLine($" -> Querying range 1...");
        var results1 = await PerformLargeQuery(logGroupName, queryString,
range1Start, range1End, limit);
        Console.WriteLine($" -> Range 1 returned {results1.Count} logs");

        Console.WriteLine($" -> Querying range 2...");
        var results2 = await PerformLargeQuery(logGroupName, queryString,
range2Start, range2End, limit);
        Console.WriteLine($" -> Range 2 returned {results2.Count} logs");
```



```
// Combine all results
var allResults = new List<List<ResultField>>(results);
allResults.AddRange(results1);
allResults.AddRange(results2);

Console.WriteLine($" -> Combined total: {allResults.Count} logs
({results.Count} + {results1.Count} + {results2.Count})");

return allResults;
}

/// <summary>
/// Gets the timestamp string of the most recent log from a list of logs.
/// Sorts timestamps to find the actual last one.
/// </summary>
private static string? GetLastLogTimestamp(List<List<ResultField>> logs)
{
    var timestamps = logs
        .Select(log => log.Find(f => f.Field == "@timestamp")?.Value)
        .Where(t => !string.IsNullOrEmpty(t))
        .OrderBy(t => t)
        .ToList();

    if (timestamps.Count == 0)
    {
        return null;
    }

    return timestamps[timestamps.Count - 1];
}

/// <summary>
/// Splits a date range in half.
/// Range 2 starts at midpoint + 1 second to avoid overlap.
/// </summary>
private static (long range1Start, long range1End, long range2Start, long
range2End) SplitDateRange(long startTime, long endTime)
{
    var midpoint = startTime + (endTime - startTime) / 2;
    // Range 2 starts at midpoint + 1 to avoid querying the same second twice
    return (startTime, midpoint, midpoint + 1, endTime);
}
```

```
/// <summary>
/// Polls for query results until complete.
/// </summary>
private static async Task<List<List<ResultField>>>?> PollQueryResults(string
queryId)
{
    int retryCount = 0;
    const int maxRetries = 60;
    const int retryDelay = 1000;

    while (retryCount < maxRetries)
    {
        var response = await _wrapper.GetQueryResultsAsync(queryId);
        if (response == null)
        {
            return null;
        }

        if (response.Status == QueryStatus.Complete)
        {
            return response.Results;
        }

        if (response.Status == QueryStatus.Failed ||
            response.Status == QueryStatus.Cancelled ||
            response.Status == QueryStatus.Timeout ||
            response.Status == QueryStatus.Unknown)
        {
            _logger.LogError($"Query failed with status: {response.Status}");
            return null;
        }

        await Task.Delay(retryDelay);
        retryCount++;
    }

    _logger.LogError("Timed out waiting for query results.");
    return null;
}

/// <summary>
/// Cleans up the resources created during the scenario.
/// </summary>
public static async Task<bool> Cleanup()
```

```
{
    var cleanup = !_interactive || GetYesNoResponse(
        "Do you want to delete the CloudFormation stack and all resources?
(y/n) ");

    if (cleanup)
    {
        try
        {
            var stackDeleteSuccess = await
DeleteCloudFormationStack(_stackName, false);
            return stackDeleteSuccess;
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, "An error occurred while cleaning up the
resources.");
            return false;
        }
    }

    Console.WriteLine($"Resources will remain. Stack name: {_stackName}, Log
group: {_logGroupName}");
    _logger.LogInformation("CloudWatch Logs Large Query scenario is
complete.");
    return true;
}

/// <summary>
/// Deletes the CloudFormation stack and waits for confirmation.
/// </summary>
private static async Task<bool> DeleteCloudFormationStack(string stackName,
bool forceDelete)
{
    var request = new DeleteStackRequest
    {
        StackName = stackName,
    };

    if (forceDelete)
    {
        request.DeletionMode = DeletionMode.FORCE_DELETE_STACK;
    }
}
```

```
        await _amazonCloudFormation.DeleteStackAsync(request);
        Console.WriteLine($"CloudFormation stack '{stackName}' is being deleted.
This may take a few minutes.");

        bool stackDeleted = await WaitForStackDeletion(stackName, forceDelete);

        if (stackDeleted)
        {
            Console.WriteLine($"CloudFormation stack '{stackName}' has been
deleted.");
            return true;
        }
        else
        {
            _logger.LogError($"Failed to delete CloudFormation stack
'{stackName}'.");
            return false;
        }
    }

    /// <summary>
    /// Waits for the stack to be deleted.
    /// </summary>
    private static async Task<bool> WaitForStackDeletion(string stackName, bool
forceDelete)
    {
        int retryCount = 0;
        const int maxRetries = 30;
        const int retryDelay = 10000;

        while (retryCount < maxRetries)
        {
            var describeStacksRequest = new DescribeStacksRequest
            {
                StackName = stackName
            };

            try
            {
                var describeStacksResponse = await
                _amazonCloudFormation.DescribeStacksAsync(describeStacksRequest);

                if (describeStacksResponse.Stacks.Count == 0 ||
```

```

        describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.DELETE_COMPLETE)
    {
        return true;
    }

    if (!forceDelete && describeStacksResponse.Stacks[0].StackStatus
== StackStatus.DELETE_FAILED)
    {
        return await DeleteCloudFormationStack(stackName, true);
    }
}
catch (AmazonCloudFormationException ex) when (ex.ErrorCode ==
"ValidationError")
{
    return true;
}

Console.WriteLine($"Waiting for CloudFormation stack '{stackName}' to
be deleted...");
await Task.Delay(retryDelay);
retryCount++;
}

_logger.LogError($"Timed out waiting for CloudFormation stack
'{stackName}' to be deleted.");
return false;
}

/// <summary>
/// Waits with a countdown display.
/// </summary>
private static async Task WaitWithCountdown(int seconds)
{
    for (int i = seconds; i > 0; i--)
    {
        Console.Write($"\\rWaiting: {i} seconds remaining... ");
        await Task.Delay(1000);
    }
    Console.WriteLine($"\\rWait complete.                ");
}

/// <summary>
/// Helper method to get a yes or no response from the user.

```

```
/// </summary>
private static bool GetYesNoResponse(string question)
{
    Console.WriteLine(question);
    var ynResponse = Console.ReadLine();
    var response = ynResponse != null && ynResponse.Equals("y",
StringComparison.InvariantCultureIgnoreCase);
    return response;
}

/// <summary>
/// Prompts the user for a stack name.
/// </summary>
private static string PromptUserForStackName()
{
    if (_interactive)
    {
        Console.Write($"Enter a name for the CloudFormation stack (press
Enter for default '{_stackName}'): ");
        string? input = Console.ReadLine();
        if (!string.IsNullOrEmpty(input))
        {
            var regex = "[a-zA-Z][-a-zA-Z0-9]*";
            if (!Regex.IsMatch(input, regex))
            {
                Console.WriteLine($"Invalid stack name. Using default:
{_stackName}");
                return _stackName;
            }
            return input;
        }
    }
    return _stackName;
}

/// <summary>
/// Prompts the user for input with a default value.
/// </summary>
private static string PromptUserForInput(string prompt, string defaultValue)
{
    if (_interactive)
    {
        Console.Write($"{prompt}(press Enter for default '{defaultValue}'):
");
```

```
        string? input = Console.ReadLine();
        return string.IsNullOrEmpty(input) ? defaultValue : input;
    }
    return defaultValue;
}

/// <summary>
/// Prompts the user for an integer value.
/// </summary>
private static int PromptUserForInteger(string prompt, int defaultValue)
{
    if (_interactive)
    {
        Console.Write($"{prompt}(press Enter for default '{defaultValue}'):");

        string? input = Console.ReadLine();
        if (string.IsNullOrEmpty(input) || !int.TryParse(input, out var result))
        {
            return defaultValue;
        }
        return result;
    }
    return defaultValue;
}

/// <summary>
/// Prompts the user for a long value.
/// </summary>
private static long PromptUserForLong(string prompt)
{
    if (_interactive)
    {
        Console.Write(prompt);
        string? input = Console.ReadLine();
        if (long.TryParse(input, out var result))
        {
            return result;
        }
    }
    return 0;
}

/// <summary>
```

```
/// Finds duplicate log entries based on timestamp and message.
/// </summary>
private static List<(string Timestamp, string Message, int Count)>
FindDuplicateLogs(List<List<ResultField>> logs)
{
    var logSignatures = new Dictionary<string, int>();

    foreach (var log in logs)
    {
        var timestamp = log.Find(f => f.Field == "@timestamp")?.Value ?? "";
        var message = log.Find(f => f.Field == "@message")?.Value ?? "";
        var signature = $"{timestamp}|{message}";

        if (logSignatures.ContainsKey(signature))
        {
            logSignatures[signature]++;
        }
        else
        {
            logSignatures[signature] = 1;
        }
    }

    return logSignatures
        .Where(kvp => kvp.Value > 1)
        .Select(kvp =>
        {
            var parts = kvp.Key.Split('|');
            return (Timestamp: parts[0], Message: parts[1], Count:
kvp.Value);
        })
        .OrderByDescending(x => x.Count)
        .ToList();
}
}
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の以下のトピックを参照してください。
 - [GetQueryResults](#)
 - [StartQuery](#)

JavaScript

SDK for JavaScript (v3)

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

これはエントリポイントです。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { CloudWatchLogsClient } from "@aws-sdk/client-cloudwatch-logs";
import { CloudWatchQuery } from "./cloud-watch-query.js";

console.log("Starting a recursive query...");

if (!process.env.QUERY_START_DATE || !process.env.QUERY_END_DATE) {
  throw new Error(
    "QUERY_START_DATE and QUERY_END_DATE environment variables are required.",
  );
}

const cloudWatchQuery = new CloudWatchQuery(new CloudWatchLogsClient({}), {
  logGroupNames: ["/workflows/cloudwatch-logs/large-query"],
  dateRange: [
    new Date(Number.parseInt(process.env.QUERY_START_DATE)),
    new Date(Number.parseInt(process.env.QUERY_END_DATE)),
  ],
});

await cloudWatchQuery.run();

console.log(
  `Queries finished in ${cloudWatchQuery.secondsElapsed} seconds.\nTotal logs found: ${cloudWatchQuery.results.length}`,
);
```

これは、必要に応じてクエリを複数のステップに分割するクラスです。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
    StartQueryCommand,
    GetQueryResultsCommand,
} from "@aws-sdk/client-cloudwatch-logs";
import { splitDateRange } from "@aws-doc-sdk-examples/lib/utils/util-date.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

class DateOutOfBoundsError extends Error {}

export class CloudWatchQuery {
    /**
     * Run a query for all CloudWatch Logs within a certain date range.
     * CloudWatch logs return a max of 10,000 results. This class
     * performs a binary search across all of the logs in the provided
     * date range if a query returns the maximum number of results.
     *
     * @param {import('@aws-sdk/client-cloudwatch-logs').CloudWatchLogsClient}
    client
     * @param {{ logGroupNames: string[], dateRange: [Date, Date], queryConfig:
    { limit: number } }} config
     */
    constructor(client, { logGroupNames, dateRange, queryConfig }) {
        this.client = client;

        /**
         * All log groups are queried.
         */
        this.logGroupNames = logGroupNames;

        /**
         * The inclusive date range that is queried.
         */
        this.dateRange = dateRange;

        /**
         * CloudWatch Logs never returns more than 10,000 logs.
         */
        this.limit = queryConfig?.limit ?? 10000;

        /**
         * @type {import("@aws-sdk/client-cloudwatch-logs").ResultField[][]}
         */
    }
}
```

```
    this.results = [];
  }

  /**
   * Run the query.
   */
  async run() {
    this.secondsElapsed = 0;
    const start = new Date();
    this.results = await this._largeQuery(this.dateRange);
    const end = new Date();
    this.secondsElapsed = (end - start) / 1000;
    return this.results;
  }

  /**
   * Recursively query for logs.
   * @param {[Date, Date]} dateRange
   * @returns {Promise<import("@aws-sdk/client-cloudwatch-logs").ResultField[]>}
   */
  async _largeQuery(dateRange) {
    const logs = await this._query(dateRange, this.limit);

    console.log(
      `Query date range: ${dateRange
        .map((d) => d.toISOString())
        .join(" to ")}. Found ${logs.length} logs.`
    );

    if (logs.length < this.limit) {
      return logs;
    }

    const lastLogDate = this._getLastLogDate(logs);
    const offsetLastLogDate = new Date(lastLogDate);
    offsetLastLogDate.setMilliseconds(lastLogDate.getMilliseconds() + 1);
    const subDateRange = [offsetLastLogDate, dateRange[1]];
    const [r1, r2] = splitDateRange(subDateRange);
    const results = await Promise.all([
      this._largeQuery(r1),
      this._largeQuery(r2),
    ]);
    return [logs, ...results].flat();
  }
}
```

```

}

/**
 * Find the most recent log in a list of logs.
 * @param {import("@aws-sdk/client-cloudwatch-logs").ResultField[][]} logs
 */
_getLastLogDate(logs) {
  const timestamps = logs
    .map(
      (log) =>
        log.find((fieldMeta) => fieldMeta.field === "@timestamp")?.value,
    )
    .filter((t) => !!t)
    .map((t) => `${t}Z`)
    .sort();

  if (!timestamps.length) {
    throw new Error("No timestamp found in logs.");
  }

  return new Date(timestamps[timestamps.length - 1]);
}

/**
 * Simple wrapper for the GetQueryResultsCommand.
 * @param {string} queryId
 */
_getQueryResults(queryId) {
  return this.client.send(new GetQueryResultsCommand({ queryId }));
}

/**
 * Starts a query and waits for it to complete.
 * @param {[Date, Date]} dateRange
 * @param {number} maxLogs
 */
async _query(dateRange, maxLogs) {
  try {
    const { queryId } = await this._startQuery(dateRange, maxLogs);
    const { results } = await this._waitUntilQueryDone(queryId);
    return results ?? [];
  } catch (err) {
    /**
     * This error is thrown when StartQuery returns an error indicating

```

```

        * that the query's start or end date occur before the log group was
        * created.
        */
        if (err instanceof DateOutOfBoundsError) {
            return [];
        }
        throw err;
    }
}

/**
 * Wrapper for the StartQueryCommand. Uses a static query string
 * for consistency.
 * @param {[Date, Date]} dateRange
 * @param {number} maxLogs
 * @returns {Promise<{ queryId: string }>}
 */
async _startQuery([startDate, endDate], maxLogs = 10000) {
    try {
        return await this.client.send(
            new StartQueryCommand({
                logGroupNames: this.logGroupNames,
                queryString: "fields @timestamp, @message | sort @timestamp asc",
                startTime: startDate.valueOf(),
                endTime: endDate.valueOf(),
                limit: maxLogs,
            }),
        );
    } catch (err) {
        /** @type {string} */
        const message = err.message;
        if (message.startsWith("Query's end date and time")) {
            // This error indicates that the query's start or end date occur
            // before the log group was created.
            throw new DateOutOfBoundsError(message);
        }

        throw err;
    }
}

/**
 * Call GetQueryResultsCommand until the query is done.
 * @param {string} queryId

```

```
*/
_waitUntilQueryDone(queryId) {
  const getResults = async () => {
    const results = await this._getQueryResults(queryId);
    const queryDone = [
      "Complete",
      "Failed",
      "Cancelled",
      "Timeout",
      "Unknown",
    ].includes(results.status);

    return { queryDone, results };
  };

  return retry(
    { intervalInMs: 1000, maxRetries: 60, quiet: true },
    async () => {
      const { queryDone, results } = await getResults();
      if (!queryDone) {
        throw new Error("Query not done.");
      }

      return results;
    },
  );
}
```

- APIの詳細については、「AWS SDK for JavaScript API リファレンス」の以下のトピックを参照してください。
 - [GetQueryResults](#)
 - [StartQuery](#)

Python

SDK for Python (Boto3)

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

このファイルは、10,000 を超える結果の CloudWatch クエリを管理するためのサンプルモジュールを呼び出します。

```
import logging
import os
import sys

import boto3
from botocore.config import Config

from cloudwatch_query import CloudWatchQuery
from date_utilities import DateUtilities

# Configure logging at the module level.
logging.basicConfig(
    level=logging.INFO,
    format="%(asctime)s - %(levelname)s - %(filename)s:%(lineno)d - %(message)s",
)

DEFAULT_QUERY_LOG_GROUP = "/workflows/cloudwatch-logs/large-query"

class CloudWatchLogsQueryRunner:
    def __init__(self):
        """
        Initializes the CloudWatchLogsQueryRunner class by setting up date
        utilities
        and creating a CloudWatch Logs client with retry configuration.
        """
        self.date_utilities = DateUtilities()
        self.cloudwatch_logs_client = self.create_cloudwatch_logs_client()
```

```

def create_cloudwatch_logs_client(self):
    """
    Creates and returns a CloudWatch Logs client with a specified retry
    configuration.

    :return: A CloudWatch Logs client instance.
    :rtype: boto3.client
    """
    try:
        return boto3.client("logs", config=Config(retries={"max_attempts":
10}))
    except Exception as e:
        logging.error(f"Failed to create CloudWatch Logs client: {e}")
        sys.exit(1)

def fetch_environment_variables(self):
    """
    Fetches and validates required environment variables for query start and
    end dates.
    Fetches the environment variable for log group, returning the default
    value if it
    does not exist.

    :return: Tuple of query start date and end date as integers and the log
    group.
    :rtype: tuple
    :raises SystemExit: If required environment variables are missing or
    invalid.
    """
    try:
        query_start_date = int(os.environ["QUERY_START_DATE"])
        query_end_date = int(os.environ["QUERY_END_DATE"])
    except KeyError:
        logging.error(
            "Both QUERY_START_DATE and QUERY_END_DATE environment variables
are required."
        )
        sys.exit(1)
    except ValueError as e:
        logging.error(f"Error parsing date environment variables: {e}")
        sys.exit(1)

    try:
        log_group = os.environ["QUERY_LOG_GROUP"]

```



```

        except KeyError:
            logging.warning("No QUERY_LOG_GROUP environment variable, using
default value")
            log_group = DEFAULT_QUERY_LOG_GROUP

        return query_start_date, query_end_date, log_group

def convert_dates_to_iso8601(self, start_date, end_date):
    """
    Converts UNIX timestamp dates to ISO 8601 format using DateUtilities.

    :param start_date: The start date in UNIX timestamp.
    :type start_date: int
    :param end_date: The end date in UNIX timestamp.
    :type end_date: int
    :return: Start and end dates in ISO 8601 format.
    :rtype: tuple
    """
    start_date_iso8601 =
self.date_utilities.convert_unix_timestamp_to_iso8601(
    start_date
)
    end_date_iso8601 = self.date_utilities.convert_unix_timestamp_to_iso8601(
    end_date
)
    return start_date_iso8601, end_date_iso8601

def execute_query(
    self,
    start_date_iso8601,
    end_date_iso8601,
    log_group="/workflows/cloudwatch-logs/large-query",
    query="fields @timestamp, @message | sort @timestamp asc"
):
    """
    Creates a CloudWatchQuery instance and executes the query with provided
date range.

    :param start_date_iso8601: The start date in ISO 8601 format.
    :type start_date_iso8601: str
    :param end_date_iso8601: The end date in ISO 8601 format.
    :type end_date_iso8601: str
    :param log_group: Log group to search: "/workflows/cloudwatch-logs/large-
query"

```

```

        :type log_group: str
        :param query: Query string to pass to the CloudWatchQuery instance
        :type query: str
        """
        cloudwatch_query = CloudWatchQuery(
            log_group=log_group,
            query_string=query
        )
        cloudwatch_query.query_logs((start_date_iso8601, end_date_iso8601))
        logging.info("Query executed successfully.")
        logging.info(
            f"Queries completed in {cloudwatch_query.query_duration} seconds.
            Total logs found: {len(cloudwatch_query.query_results)}"
        )

def main():
    """
    Main function to start a recursive CloudWatch logs query.
    Fetches required environment variables, converts dates, and executes the
    query.
    """
    logging.info("Starting a recursive CloudWatch logs query...")
    runner = CloudWatchLogsQueryRunner()
    query_start_date, query_end_date, log_group =
runner.fetch_environment_variables()
    start_date_iso8601 = DateUtilities.convert_unix_timestamp_to_iso8601(
        query_start_date
    )
    end_date_iso8601 =
DateUtilities.convert_unix_timestamp_to_iso8601(query_end_date)
    runner.execute_query(start_date_iso8601, end_date_iso8601,
log_group=log_group)

if __name__ == "__main__":
    main()

```

このモジュールは、10,000 を超える結果の CloudWatch クエリを処理します。

```

import logging
import time

```

```

from datetime import datetime
import threading
import boto3

from date_utilities import DateUtilities

DEFAULT_QUERY = "fields @timestamp, @message | sort @timestamp asc"
DEFAULT_LOG_GROUP = "/workflows/cloudwatch-logs/large-query"

class DateOutOfBoundsError(Exception):
    """Exception raised when the date range for a query is out of bounds."""

    pass

class CloudWatchQuery:
    """
    A class to query AWS CloudWatch logs within a specified date range.

    :vartype date_range: tuple
    :ivar limit: Maximum number of log entries to return.
    :vartype limit: int
    :log_group str: Name of the log group to query
    :query_string str: query
    """

    def __init__(self, log_group: str = DEFAULT_LOG_GROUP, query_string:
str=DEFAULT_QUERY) -> None:
        self.lock = threading.Lock()
        self.log_group = log_group
        self.query_string = query_string
        self.query_results = []
        self.query_duration = None
        self.datetime_format = "%Y-%m-%d %H:%M:%S.%f"
        self.date_utilities = DateUtilities()
        self.limit = 10000

    def query_logs(self, date_range):
        """
        Executes a CloudWatch logs query for a specified date range and
        calculates the execution time of the query.

        :return: A batch of logs retrieved from the CloudWatch logs query.
        :rtype: list

```

```

"""
start_time = datetime.now()

start_date, end_date = self.date_utilities.normalize_date_range_format(
    date_range, from_format="unix_timestamp", to_format="datetime"
)

logging.info(
    f"Original query:"
    f"\n      START:      {start_date}"
    f"\n      END:        {end_date}"
    f"\n      LOG GROUP: {self.log_group}"
)
self.recursive_query((start_date, end_date))
end_time = datetime.now()
self.query_duration = (end_time - start_time).total_seconds()

def recursive_query(self, date_range):
    """
    Processes logs within a given date range, fetching batches of logs
    recursively if necessary.

    :param date_range: The date range to fetch logs for, specified as a tuple
    (start_timestamp, end_timestamp).
    :type date_range: tuple
    :return: None if the recursive fetching is continued or stops when the
    final batch of logs is processed.
    Although it doesn't explicitly return the query results, this
    method accumulates all fetched logs
    in the `self.query_results` attribute.
    :rtype: None
    """
    batch_of_logs = self.perform_query(date_range)
    # Add the batch to the accumulated logs
    with self.lock:
        self.query_results.extend(batch_of_logs)
    if len(batch_of_logs) == self.limit:
        logging.info(f"Fetched {self.limit}, checking for more...")
        most_recent_log = self.find_most_recent_log(batch_of_logs)
        most_recent_log_timestamp = next(
            item["value"]
            for item in most_recent_log
            if item["field"] == "@timestamp"
        )

```

```

        new_range = (most_recent_log_timestamp, date_range[1])
        midpoint = self.date_utilities.find_middle_time(new_range)

        first_half_thread = threading.Thread(
            target=self.recursive_query,
            args=((most_recent_log_timestamp, midpoint)),
        )
        second_half_thread = threading.Thread(
            target=self.recursive_query, args=((midpoint, date_range[1])),
        )

        first_half_thread.start()
        second_half_thread.start()

        first_half_thread.join()
        second_half_thread.join()

def find_most_recent_log(self, logs):
    """
    Search a list of log items and return most recent log entry.
    :param logs: A list of logs to analyze.
    :return: log
    :type :return List containing log item details
    """
    most_recent_log = None
    most_recent_date = "1970-01-01 00:00:00.000"

    for log in logs:
        for item in log:
            if item["field"] == "@timestamp":
                logging.debug(f"Compared: {item['value']} to
{most_recent_date}")
                if (
                    self.date_utilities.compare_dates(
                        item["value"], most_recent_date
                    )
                    == item["value"]
                ):
                    logging.debug(f"New most recent: {item['value']}")
                    most_recent_date = item["value"]
                    most_recent_log = log
    logging.info(f"Most recent log date of batch: {most_recent_date}")
    return most_recent_log

```

```
def perform_query(self, date_range):
    """
    Performs the actual CloudWatch log query.

    :param date_range: A tuple representing the start and end datetime for
the query.
    :type date_range: tuple
    :return: A list containing the query results.
    :rtype: list
    """
    client = boto3.client("logs")
    try:
        try:
            start_time = round(
self.date_utilities.convert_iso8601_to_unix_timestamp(date_range[0])
            )
            end_time = round(
self.date_utilities.convert_iso8601_to_unix_timestamp(date_range[1])
            )
            response = client.start_query(
                logGroupName=self.log_group,
                startTime=start_time,
                endTime=end_time,
                queryString=self.query_string,
                limit=self.limit,
            )
            query_id = response["queryId"]
        except client.exceptions.ResourceNotFoundException as e:
            raise DateOutOfBoundsError(f"Resource not found: {e}")
        while True:
            time.sleep(1)
            results = client.get_query_results(queryId=query_id)
            if results["status"] in [
                "Complete",
                "Failed",
                "Cancelled",
                "Timeout",
                "Unknown",
            ]:
                return results.get("results", [])
        except DateOutOfBoundsError:
            return []
```

```

def _initiate_query(self, client, date_range, max_logs):
    """
    Initiates the CloudWatch logs query.

    :param date_range: A tuple representing the start and end datetime for
    the query.
    :type date_range: tuple
    :param max_logs: The maximum number of logs to retrieve.
    :type max_logs: int
    :return: The query ID as a string.
    :rtype: str
    """
    try:
        start_time = round(

self.date_utilities.convert_iso8601_to_unix_timestamp(date_range[0])
        )
        end_time = round(

self.date_utilities.convert_iso8601_to_unix_timestamp(date_range[1])
        )
        response = client.start_query(
            logGroupName=self.log_group,
            startTime=start_time,
            endTime=end_time,
            queryString=self.query_string,
            limit=max_logs,
        )
        return response["queryId"]
    except client.exceptions.ResourceNotFoundException as e:
        raise DateOutOfBoundsError(f"Resource not found: {e}")


def _wait_for_query_results(self, client, query_id):
    """
    Waits for the query to complete and retrieves the results.

    :param query_id: The ID of the initiated query.
    :type query_id: str
    :return: A list containing the results of the query.
    :rtype: list
    """
    while True:

```

```
time.sleep(1)
results = client.get_query_results(queryId=query_id)
if results["status"] in [
    "Complete",
    "Failed",
    "Cancelled",
    "Timeout",
    "Unknown",
]:
    return results.get("results", [])
```

- API の詳細については、『AWS SDK for Python (Boto3) API リファレンス』の以下のトピックを参照してください。
 - [GetQueryResults](#)
 - [StartQuery](#)

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK での CloudWatch Logs の使用](#)。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

スケジュールされたイベントを使用した Lambda 関数の呼び出し

次のコード例は、Amazon EventBridge スケジュールされたイベントによって呼び出される AWS Lambda 関数を作成する方法を示しています。

Java

SDK for Java 2.x

AWS Lambda 関数を呼び出す Amazon EventBridge スケジュールされたイベントを作成する方法を示します。cron 式を使用して Lambda 関数が呼び出されるタイミングをスケジュールするように EventBridge を設定します。この例では、Lambda Java ランタイム API を使用して Lambda 関数を作成します。この例では、さまざまな AWS サービスを呼び出して、特定のユースケースを実行します。この例では、年間の記念日に従業員を祝福するモバイルテキストメッセージを従業員に送信するアプリを作成する方法を示します。

完全なソースコードとセットアップおよび実行の手順については、[GitHub](#) で完全な例を参照してください。

この例で使用されているサービス

- CloudWatch Logs
- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

JavaScript

SDK for JavaScript (v3)

AWS Lambda 関数を呼び出す Amazon EventBridge スケジュールされたイベントを作成する方法を示します。cron 式を使用して Lambda 関数が呼び出されるタイミングをスケジュールするように EventBridge を設定します。この例では、Lambda JavaScript ランタイム API を使用して Lambda 関数を作成します。この例では、さまざまな AWS サービスを呼び出して、特定のユースケースを実行します。この例では、年間の記念日に従業員を祝福するモバイルテキストメッセージを従業員に送信するアプリを作成する方法を示します。

完全なソースコードとセットアップおよび実行の手順については、[GitHub](#) で完全な例を参照してください。

この例は、[AWS SDK for JavaScript v3 デベロッパーガイド](#)でも使用できます。

この例で使用されているサービス

- CloudWatch Logs
- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

Python

SDK for Python (Boto3)

この例では、スケジュールされた Amazon EventBridge イベントのターゲットとして AWS Lambda 関数を登録する方法を示します。Lambda ハンドラーは、後で取得するために

Amazon CloudWatch Logs にわかりやすいメッセージと完全なイベントデータを書き込みます。

- Lambda 関数をデプロイします。
- EventBridge スケジュールイベントを作成し、Lambda 関数をターゲットにします。
- EventBridge に Lambda 関数を呼び出す許可を付与します
- CloudWatch Logs から最新のデータを出力して、スケジュールされた呼び出しの結果を表示しています。
- デモ中に作成されたすべてのリソースをクリーンアップします。

この例は GitHub で最もよく確認できます。完全なソースコードとセットアップおよび実行の手順については、[GitHub](#) で完全な例を参照してください。

この例で使用されているサービス

- CloudWatch Logs
- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK での CloudWatch Logs の使用](#)。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

を使用して CloudWatch Logs のルックアップテーブルを暗号化する AWS Key Management Service

ルックアップテーブルデータは常に CloudWatch Logs で暗号化されます。デフォルトでは、CloudWatch Logs は 256 ビット Advanced Encryption Standard Galois/Counter Mode (AES-GCM) によるサーバー側の暗号化を使用して、保管中のルックアップテーブルデータを暗号化します。別の方法として、この暗号化には AWS Key Management Service を使用できます。その場合、暗号化は AWS KMS キーを使用して行われます。を使用した暗号化 AWS KMS は、ルックアップテーブルの作成時または更新時に、KMS キーをルックアップテーブルに関連付けることで、ルックアップテーブルレベルで有効になります。

Important

CloudWatch Logs は、対称 KMS キーのみをサポートします。非対称キーを使用してルックアップテーブル内のデータを暗号化しないでください。詳細については、「[対称キーと非対称キーの使用](#)」を参照してください。

KMS キーをルックアップテーブルに関連付けると、ルックアップテーブルに保存されているすべてのデータがこのキーを使用して暗号化されます。CloudWatch Logs は、リクエストがあればいつでもこのデータを復号化します。暗号化されたデータがリクエストされた場合、CloudWatch Logs に KMS キーのアクセス許可が必要です。

後でルックアップテーブルから KMS キーの関連付けを解除すると、CloudWatch Logs は CloudWatch Logs のデフォルトの暗号化方法を使用してデータを暗号化します。ただし、関連付けを解除する前にキーが無効または削除されている場合、CloudWatch Logs はそのキーで暗号化されたデータを読み取ることができません。

CloudWatch Logs が AWS KMS を使用してログデータを暗号化する方法の一般的な情報については、「」を参照してください [を使用して CloudWatch Logs のログデータを暗号化する AWS Key Management Service](#)。

CloudWatch Logs がルックアップテーブル AWS KMS に使用する 方法

CloudWatch Logs は、エン AWS KMS ベロープ暗号化を使用してルックアップテーブルデータを保護します。KMS キーをルックアップテーブルに関連付けると、CloudWatch Logs はGenerateDataKeyリクエストを に送信します AWS KMS。 は一意のデータ暗号化キー (DEK) AWS KMS を生成し、プレーンテキストコピーと DEK の暗号化されたコピーの両方を返します。CloudWatch Logs は、プレーンテキスト DEK を使用してルックアップテーブルデータを暗号化し、暗号化された DEK を暗号化されたデータとともに保存します。プレーンテキスト DEK は保存されず、使用後にメモリから破棄されます。

CloudWatch Logs がルックアップテーブルデータを読み取る必要がある場合、暗号化された DEK AWS KMS を使用して にDecryptリクエストを送信します。 は DEK を AWS KMS 復号し、プレーンテキストの DEK を CloudWatch Logs に返し、これを使用してルックアップテーブルデータを復号します。

CloudWatch Logs は、次の暗号化コンテキストを使用してリクエストを行います AWS KMS。

```
{
  "aws:logs:arn": "arn:aws:logs:region:account-id:lookup-table:lookup-table-name"
}
```

この暗号化コンテキストを IAM ポリシーと AWS KMS キーポリシーで使用して、KMS キーへのアクセスを制御できます。詳細については、「[AWS KMS キーと暗号化コンテキスト](#)」を参照してください。

必要な アクセス許可

ルックアップテーブルで AWS KMS 暗号化を使用するには、IAM プリンシパルに KMS キーに対する次の AWS KMS アクセス許可が必要です。

- kms:Decrypt
- kms:GenerateDataKey

アクセスkms:Decrypt許可は、CloudWatch Logs がユーザーに代わってデータを復号化できるように、KMS キーで暗号化されたルックアップテーブルGetLookupTableで を呼び出すときに必要

です。暗号化されたルックアップテーブルで `lookup` コマンドを使用するクエリ `StartQuery` を呼び出す場合、キー (ルックアップテーブルの暗号化に使用される KMS キー) にも アクセス `kms:Decrypt` 許可が必要です。アクセス `kms:GenerateDataKey` 許可は、CloudWatch Logs がルックアップテーブルデータを暗号化するためのデータ暗号化キーを生成できるように、KMS キー `UpdateLookupTable` を使用して `CreateLookupTable` または `UpdateLookupTable` を呼び出すときに必要です。

さらに、CloudWatch Logs サービスには KMS キーを使用するアクセス許可が必要です。次のセクションで説明するように、KMS キーポリシーにポリシーステートメントを追加して、これらのアクセス許可を付与します。

ステップ 1: AWS KMS キーを作成する

対称 KMS キーを作成するには、次の [create-key](#) コマンドを使用します。

```
aws kms create-key
```

出力には、キーのキー ID と Amazon リソースネーム (ARN) が含まれます。出力例を次に示します。

```
{
  "KeyMetadata": {
    "Origin": "AWS_KMS",
    "KeyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
    "Description": "",
    "KeyManager": "CUSTOMER",
    "Enabled": true,
    "CustomerMasterKeySpec": "SYMMETRIC_DEFAULT",
    "KeyUsage": "ENCRYPT_DECRYPT",
    "KeyState": "Enabled",
    "CreationDate": 1478910250.94,
    "Arn": "arn:aws:kms:us-west-2:123456789012:key/6f815f63-e628-448c-8251-e40cb0d29f59",
    "AWSAccountId": "123456789012",
    "EncryptionAlgorithms": [
      "SYMMETRIC_DEFAULT"
    ]
  }
}
```

キー ARN を記録します。次のステップで必要になります。

ステップ 2: KMS キーでアクセス許可を設定する

デフォルトでは、すべての AWS KMS キーはプライベートです。リソースの所有者のみがその CMK を使用してデータを暗号化および復号できます。キーを使用するためのアクセス許可を CloudWatch Logs サービスプリンシパルに付与し、キーを使用するためのアクセス許可を呼び出し元ロールに付与する必要があります。

まず、[get-key-policy](#) コマンドを使用して、KMS キーのデフォルトポリシーを `policy.json` として保存します。

```
aws kms get-key-policy --key-id key-id --policy-name default --output text > ./policy.json
```

テキストエディタで `policy.json` ファイルを開き、次のステートメントを追加して、CloudWatch Logs サービスプリンシパルにキーを使用するアクセス許可を付与します。この例では、暗号化コンテキストに一致する Condition セクションを使用して、キーを特定のルックアップテーブルに制限します。

```
{
  "Effect": "Allow",
  "Principal": {
    "Service": "logs.region.amazonaws.com"
  },
  "Action": [
    "kms:Decrypt",
    "kms:GenerateDataKey"
  ],
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "kms:EncryptionContext:aws:logs:arn": "arn:aws:logs:region:account-id:lookup-table:lookup-table-name",
      "aws:SourceAccount": "account-id",
      "aws:SourceArn": "arn:aws:logs:region:account-id:lookup-table:lookup-table-name"
    }
  }
}
```

次に、CloudWatch Logs CreateLookupTableまたは UpdateLookupTable APIs を呼び出すロールにアクセス許可を追加します。CloudWatch Logs は を使用してkms:ViaService、お客様に代わって を AWS KMS 呼び出します。詳細については、「[kms:ViaService](#)」を参照してください。

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::account-id:role/role-name"
  },
  "Action": [
    "kms:Decrypt",
    "kms:GenerateDataKey"
  ],
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "kms:ViaService": [
        "logs.region.amazonaws.com"
      ]
    }
  }
}
```

最後に、次の [put-key-policy](#) コマンドを使用して更新されたポリシーを追加します。

```
aws kms put-key-policy --key-id key-id --policy-name default --policy file://
policy.json
```

ステップ 3: KMS キーをルックアップテーブルに関連付ける

API を使用して KMS キーを作成するとき、または CreateLookupTable API を使用して既存のルックアップテーブルを更新するときに、KMS キーをルックアップテーブルに関連付けることができますUpdateLookupTable。どちらの APIs AWSLogsConfigService の一部です。

作成時に KMS キーをルックアップテーブルに関連付けるには

CreateLookupTable API を使用して、KMS キーの ARN で kmsKeyArnパラメータを指定します。

```
aws logs create-lookup-table \
```

```
--lookup-table-name my-lookup-table \  
--kms-key-arn "arn:aws:kms:region:account-id:key/key-id"
```

KMS キーを既存のルックアップテーブルに関連付けるには

UpdateLookupTable API を使用して、KMS キーの ARN で kmsKeyArn パラメータを指定します。

```
aws logs update-lookup-table \  
--lookup-table-name my-lookup-table \  
--kms-key-arn "arn:aws:kms:region:account-id:key/key-id"
```

考慮事項

- ルックアップテーブルから KMS キーに関連付けるか関連付けを解除すると、オペレーションが有効になるまでに最大 5 分かかることがあります。
- 関連付けられたキーへの CloudWatch Logs アクセスを取り消すか、関連付けられた KMS キーを削除すると、CloudWatch Logs の暗号化されたルックアップテーブルデータを取得できなくなります。
- このトピックの手順を実行するには、`kms:CreateKey`、`kms:GetKeyPolicy`、`kms:PutKeyPolicy`、および `CreateLookupTable` または `UpdateLookupTable` を呼び出すための適切な CloudWatch Logs アクセス許可が必要です。

Amazon CloudWatch Logs のセキュリティ

のクラウドセキュリティが最優先事項 AWS です。お客様は AWS、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。

セキュリティは、AWS とお客様の間の責任共有です。[責任共有モデル](#)ではこれをクラウドのセキュリティおよびクラウド内のセキュリティと説明しています。

- クラウドのセキュリティ – AWS 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者は、[AWS コンプライアンスプログラム](#)コンプライアンスプログラムの一環として、当社のセキュリティの有効性を定期的にテストおよび検証。WorkSpaces に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、「コンプライアンスプログラム[AWS による対象範囲内のサービスコンプライアンスプログラム](#)」を参照してください。
- クラウドのセキュリティ – お客様の責任は、使用する AWS サービスによって決まります。また、お客様は、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。

このドキュメントは、Amazon CloudWatch Logs を使用する際の責任共有モデルの適用方法を理解するのに役立ちます。ここでは、セキュリティとコンプライアンスの目標を満たすように Amazon CloudWatch Logs を設定する方法について説明します。また、CloudWatch Logs リソースのモニタリングや保護に役立つ他の AWS サービスの使用方法についても説明します。

内容

- [Amazon CloudWatch Logs におけるデータ保護](#)
- [Amazon CloudWatch Logs の Identity and Access Management](#)
- [Amazon CloudWatch Logs のコンプライアンス検証](#)
- [Amazon CloudWatch Logs の耐障害性](#)
- [Amazon CloudWatch Logs のインフラストラクチャセキュリティ](#)
- [インターフェイス VPC エンドポイントでの CloudWatch Logs の使用](#)

Amazon CloudWatch Logs におけるデータ保護

Note

CloudWatch Logs では、での一般的なデータ保護に関する以下の情報に加えて AWS、ログイベントの機密データをマスキングして保護することもできます。詳細については、「[機密性の高いログデータをマスキングで保護する](#)」を参照してください。

責任 AWS [共有モデル](#)、Amazon CloudWatch Logs でのデータ保護に適用されます。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。ユーザーは、このインフラストラクチャでホストされるコンテンツに対する管理を維持する責任があります。また、使用する「AWS のサービス」のセキュリティ設定と管理タスクもユーザーの責任となります。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に関するよくある質問](#)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、認証情報を保護し AWS アカウント、AWS IAM アイデンティティセンターまたは AWS Identity and Access Management (IAM) を使用して個々のユーザー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権限のみが各ユーザーに付与されます。また、次の方法でデータを保護することもお勧めします:

- 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。TLS 1.2 は必須ですが、TLS 1.3 を推奨します。
- 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。CloudTrail 証跡を使用して AWS アクティビティをキャプチャする方法については、「AWS CloudTrail ユーザーガイド」の[CloudTrail 証跡の使用](#)を参照してください。
- AWS 暗号化ソリューションと、その中のすべてのデフォルトのセキュリティコントロールを使用します AWS のサービス。
- Amazon Macie などの高度な管理されたセキュリティサービスを使用します。これらは、Amazon S3 に保存されている機密データの検出と保護を支援します。
- 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 140-3 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。利用可能な FIPS エンドポイントの詳細については、「[連邦情報処理規格 \(FIPS\) 140-3](#)」を参照してください。

お客様の E メールアドレスなどの極秘または機密情報を、タグ、または [名前] フィールドなどの自由形式のテキストフィールドに含めないことを強くお勧めします。これは、コンソール、API、または SDK を使用して CloudWatch Logs AWS CLI または他の AWS のサービスを使用する場合も同様です。AWS SDKs タグ、または名前に使用される自由記述のテキストフィールドに入力したデータは、請求または診断ログに使用される場合があります。外部サーバーに URL を提供する場合、そのサーバーへのリクエストを検証できるように、認証情報を URL に含めないことを強くお勧めします。

保管中の暗号化

CloudWatch Logs は、暗号化を使用して保管時のデータを保護します。すべてのロググループは暗号化されます。デフォルトでは、CloudWatch Logs サービスはサーバー側の暗号化に対応し、256 ビットの Advanced Encryption Standard Galois/Counter Mode (AES-GCM) によるサーバー側の暗号化を使用して、保管中のログデータを暗号化します。

ログの暗号化と復号に使用するキーを管理する場合は、AWS KMS キーを使用します。詳細については、「[を使用して CloudWatch Logs のログデータを暗号化する AWS Key Management Service](#)」を参照してください。

転送中の暗号化

CloudWatch Logs は、転送中のデータのエンドツーエンドの暗号化を使用します。CloudWatch Logs サービスはサーバー側の暗号化キーを管理します。

Amazon CloudWatch Logs の Identity and Access Management

Amazon CloudWatch Logs にアクセスするには AWS 、 がリクエストの認証に使用できる認証情報が必要です。これらの認証情報には、クラウド AWS リソースに関する CloudWatch Logs データを取得するなど、リソースにアクセスするためのアクセス許可が必要です。次のセクションでは、[AWS Identity and Access Management \(IAM\)](#) と CloudWatch Logs を使用して、リソースにアクセスできるユーザーを制御することで、リソースをセキュリティで保護する方法について詳しく説明します。

- [認証](#)
- [アクセスコントロール](#)

認証

アクセスを提供するには、ユーザー、グループ、またはロールにアクセス許可を追加します。

- のユーザーとグループ AWS IAM アイデンティティセンター:

アクセス許可セットを作成します。「AWS IAM アイデンティティセンター ユーザーガイド」の「[アクセス許可セットを作成する](#)」の手順に従ってください。

- IAM 内で、ID プロバイダーによって管理されているユーザー:

ID フェデレーションのロールを作成します。詳細については IAM ユーザーガイド の [サードパーティー ID プロバイダー \(フェデレーション\) 用のロールを作成する](#) を参照してください。

- IAM ユーザー:

- ユーザーが担当できるロールを作成します。手順については IAM ユーザーガイド の [IAM ユーザーのロールの作成](#) を参照してください。

- (お奨めできない方法) ポリシーをユーザーに直接アタッチするか、ユーザーをユーザーグループに追加します。IAM ユーザーガイド の [ユーザー \(コンソール\) へのアクセス許可の追加](#) の指示に従います。

アクセスコントロール

有効な認証情報があればリクエストを認証できますが、アクセス許可がなければ CloudWatch Logs リソースの作成やアクセスはできません。たとえば、ログストリーム、ロググループなどを作成する許可が必要となります。

以下のセクションでは、CloudWatch Logs のアクセス許可を管理する方法について説明します。最初に概要のセクションを読むことをお勧めします。

- [CloudWatch Logs リソースへの許可の管理の概要](#)
- [CloudWatch Logs でのアイデンティティベースのポリシー \(IAM ポリシー\) の使用](#)
- [CloudWatch Logs の許可リファレンス](#)

CloudWatch Logs リソースへの許可の管理の概要

アクセスを提供するには、ユーザー、グループ、またはロールにアクセス許可を追加します。

- のユーザーとグループ AWS IAM アイデンティティセンター:

アクセス許可セットを作成します。「AWS IAM アイデンティティセンター ユーザーガイド」の「[アクセス許可セットを作成する](#)」の手順に従ってください。

- IAM 内で、ID プロバイダーによって管理されているユーザー:

ID フェデレーションのロールを作成します。詳細については IAM ユーザーガイド の [サードパーティー ID プロバイダー \(フェデレーション\) 用のロールを作成する](#) を参照してください。

- IAM ユーザー:
 - ユーザーが担当できるロールを作成します。手順については IAM ユーザーガイド の [IAM ユーザーのロールの作成](#) を参照してください。
 - (お奨めできない方法) ポリシーをユーザーに直接アタッチするか、ユーザーをユーザーグループに追加します。IAM ユーザーガイド の [ユーザー \(コンソール\) へのアクセス許可の追加](#) の指示に従います。

トピック

- [CloudWatch Logs がリソースとオペレーションをログに記録する](#)
- [リソース所有権についての理解](#)
- [リソースへのアクセスの管理](#)
- [ポリシー要素 \(アクション、効果、プリンシパル\) の指定](#)
- [ポリシーでの条件を指定する](#)

CloudWatch Logs がリソースとオペレーションをログに記録する

CloudWatch Logs では、プライマリリソースはロググループ、ログストリーム、送信先です。CloudWatch Logs はサブリソース (プライマリリソースと使用する他のリソース) をサポートしていません。

これらのリソースとサブリソースには、次の表に示すとおり、一意の Amazon リソースネーム (ARN) が関連付けられています。

リソースタイプ	ARN 形式
ロググループ	次の 2 つの形式が使用されます。末尾に :* が付いている 2 つ目の方は、describe-log-

リソースタイプ	ARN 形式
	groups CLI コマンドと [DescribeLogGroups] API によって返される形式です。 arn:aws:logs:<i>region</i>:<i>account-id</i> :log-group:<i>log_group_name</i> arn:aws:logs:<i>region</i>:<i>account-id</i> :log-group:<i>log_group_name</i> :* 次の状況では、末尾に :* がない最初のバージョンを使用します。 • CloudWatch Logs APIs logGroupIdentifier の入力フィールド。 • タグ付け API の resourceArn フィールド • IAM ポリシーで、TagResource、UntagResource、および ListTagsForResource のアクセス許可を指定する場合。 他のすべての API アクションの IAM ポリシーでアクセス許可を指定するときは、末尾に :* がある 2 番目のバージョンを使用して ARN を参照します。
ログストリーム	arn:aws:logs: <i>region</i> : <i>account-id</i> :log-group: <i>log_group_name</i> :log-stream: <i>log-stream-name</i>
送信先	arn:aws:logs: <i>region</i> : <i>account-id</i> :destination: <i>destination_name</i>

ARN の詳細については、IAM ユーザーガイドの「[ARN](#)」を参照してください。CloudWatch Logs ARN の詳細については、「Amazon Web Services 全般のリファレンス」の「[Amazon リソースネーム \(ARN\)](#)」を参照してください。CloudWatch Logs を対象とするポリシーの例については、[CloudWatch Logs でのアイデンティティベースのポリシー \(IAM ポリシー\) の使用](#) を参照してください。

CloudWatch Logs には、CloudWatch Logs リソースを操作するための一連のオペレーションが用意されています。使用可能なオペレーションのリストについては、「[CloudWatch Logs の許可リファレンス](#)」を参照してください。

リソース所有権についての理解

AWS アカウントは、リソースを作成したユーザーに関係なく、アカウントで作成されたリソースを所有します。具体的には、リソース所有者は、リソース作成リクエストを認証する[プリンシパルエンティティ](#) (ルートアカウント、ユーザー、または IAM ロール) の AWS アカウントです。次の例は、この仕組みを示しています。

- AWS アカウントのルートアカウントの認証情報を使用してロググループを作成する場合、AWS アカウントは CloudWatch Logs リソースの所有者です。
- AWS アカウントにユーザーを作成し、そのユーザーに CloudWatch Logs リソースを作成するアクセス許可を付与すると、そのユーザーは CloudWatch Logs リソースを作成できます。ただし、ユーザーが属する AWS アカウントは CloudWatch Logs リソースを所有します。
- CloudWatch Logs リソースを作成するアクセス許可を持つ AWS IAM ロールをアカウントに作成する場合、ロールを引き受けることができるすべてのユーザーが CloudWatch Logs リソースを作成できます。ロールが属する AWS アカウントは、CloudWatch Logs リソースを所有します。

リソースへのアクセスの管理

アクセス権限ポリシー では、誰が何にアクセスできるかを記述します。以下のセクションで、アクセス権限のポリシーを作成するために使用可能なオプションについて説明します。

Note

このセクションでは、CloudWatch Logs のコンテキストでの IAM の使用について説明します。これは、IAM サービスに関する詳細情報を取得できません。IAM に関する詳細なドキュメントについては、「IAM ユーザーガイド」の「[What is IAM?](#)」(IAM とは ?) を参照してく

ださい。IAM ポリシー構文の詳細と説明については、「IAM ユーザーガイド」の「[IAM ポリシーリファレンス](#)」を参照してください。

IAM ID にアタッチされたポリシーは ID ベースのポリシー (IAM ポリシー) と呼ばれ、リソースにアタッチされたポリシーはリソースベースのポリシーと呼ばれます。CloudWatch Logs はアイデンティティベースのポリシー、およびクロスアカウントのサブスクリプションを有効にするために使用する、送信先のリソースベースのポリシーをサポートします。詳細については、「[クロスアカウント、クロスリージョンのサブスクリプション](#)」を参照してください。

トピック

- [ロググループの許可と Contributor Insights](#)
- [リソースベースのポリシー](#)

ロググループの許可と Contributor Insights

Contributor Insights は、ロググループのデータを分析し、コントリビューターデータを表示する時系列を作成できる CloudWatch の機能です。トップ N コントリビューター、一意のコントリビューターの合計数、およびそれらの使用状況に関するメトリクスを確認できます。詳細については、「[Contributor Insights を使用した高カーディナリティデータの分析](#)」を参照してください。

ユーザーに `cloudwatch:PutInsightRule` と `cloudwatch:GetInsightRuleReport` アクセス許可を付与すると、そのユーザーは CloudWatch Logs でロググループを評価し、その結果を参照するルールを作成できます。結果には、これらのロググループのコントリビューターデータを含めることができます。これらのアクセス許可は、このデータを表示できるように設定したいユーザーのみに付与してください。

リソースベースのポリシー

CloudWatch Logs は、クロスアカウントのサブスクリプションを有効にするために使用する、送信先のリソースベースのポリシーをサポートします。詳細については、「[ステップ 1: 送信先を作成する](#)」を参照してください。送信先は [PutDestination](#) API を使用して作成でき、[PutDestinationPolicy](#) API を使用して、送信先にリソースポリシーを追加できます。次の例では、アカウント ID 111122223333 の他の AWS アカウントが、送信先 `arn:aws:logs:us-east-1:123456789012:destination:testDestination` にロググループをサブスクライブできるようにします。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "AWS": "111122223333"
      },
      "Action": "logs:PutSubscriptionFilter",
      "Resource": "arn:aws:logs:us-east-1:123456789012:destination:testDestination"
    }
  ]
}
```

ポリシー要素 (アクション、効果、プリンシパル) の指定

CloudWatch Logs リソースごとに、このサービスは、一連の API オペレーションを定義します。これらの API オペレーションを実行するためのアクセス許可を付与するために、CloudWatch Logs ではポリシーに一連のアクションを定義できます。一部の API オペレーションは、API オペレーションを実行するために複数のアクションに対するアクセス許可を要求できます。リソースおよび API オペレーションに関する詳細については、「[CloudWatch Logs がリソースとオペレーションをログに記録する](#)」および「[CloudWatch Logs の許可リファレンス](#)」を参照してください。

以下は、基本的なポリシーの要素です。

- リソース - Amazon リソースネーム (ARN) を使用して、ポリシーを適用するリソースを識別します。詳細については、「[CloudWatch Logs がリソースとオペレーションをログに記録する](#)」を参照してください。
- [Action] (アクション) - アクションのキーワードを使用して、許可または拒否するリソースオペレーションを識別します。たとえば、logs.DescribeLogGroups 権限は、DescribeLogGroups オペレーションの実行をユーザーに許可します。
- 効果 - ユーザーが特定のアクションをリクエストする際の効果 (許可または拒否) を指定します。リソースへのアクセスを明示的に許可していない場合、アクセスは暗黙的に拒否されます。また、

明示的にリソースへのアクセスを拒否すると、別のポリシーによってアクセスが許可されている場合でも、ユーザーはそのリソースにアクセスできなくなります。

- プリンシパル - ID ベースのポリシー (IAM ポリシー) で、ポリシーがアタッチされているユーザーが黙示的なプリンシパルとなります。リソースベースのポリシーでは、権限 (リソースベースのポリシーにのみ適用) を受け取りたいユーザー、アカウント、サービス、またはその他のエンティティを指定します。CloudWatch Logs は送信先のリソースベースのポリシーをサポートします。

IAM ポリシー構文の詳細と説明については、IAM ユーザーガイドの「[AWS IAM ポリシーリファレンス](#)」を参照してください。

すべての CloudWatch Logs API アクションとそれらが適用されるリソースの表については、「[CloudWatch Logs の許可リファレンス](#)」を参照してください。

ポリシーでの条件を指定する

アクセス権限を付与するとき、アクセスポリシー言語を使用して、ポリシーが有効になる必要がある条件を指定できます。例えば、特定の日付の後にのみ適用されるポリシーが必要になる場合があります。ポリシー言語での条件の指定の詳細については、「IAM ユーザーガイド」の「[条件](#)」を参照してください。

条件を表すには、あらかじめ定義された条件キーを使用します。各 AWS サービスでサポートされているコンテキストキーのリストと AWS ポリシー全体のキーのリストについては、「[AWS サービスのアクション、リソース、および条件キー](#)」と [AWS 「グローバル条件コンテキストキー」](#) を参照してください。

Note

ロググループや送信先などの CloudWatch Logs のリソースへのアクセスを制御するには、タグを使用します。ロググループとログストリームの間には階層的な関係があるため、ログストリームへのアクセスはロググループレベルで制御されます。リソースへのアクセスを制御するタグの使用の詳細については、[タグを使用した Amazon Web Services のリソースへのアクセスの制御](#)を参照してください。

CloudWatch Logs でのアイデンティティベースのポリシー (IAM ポリシー) の使用

このトピックでは、アカウント管理者が IAM アイデンティティ (ユーザー、グループ、ロール) へのアクセス権限ポリシーをアタッチする、アイデンティティベースのポリシーの例を示します。

Important

初めに、CloudWatch Logs リソースへのアクセスを管理するための基本概念と使用可能なオプションについて説明する概要トピックをお読みになることをお勧めします。詳細については、「[CloudWatch Logs リソースへの許可の管理の概要](#)」を参照してください。

このトピックでは次の内容について説明します。

- [CloudWatch コンソールの使用に必要な許可](#)
- [AWS CloudWatch Logs の マネージド \(事前定義\) ポリシー](#)
- [カスタマーマネージドポリシーの例](#)

以下は、アクセス権限ポリシーの例です。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams"
      ],
      "Resource": [
        "arn:aws:logs:*:*:*"
      ]
    }
  ]
}
```

```
}
```

このポリシーには、ロググループとログストリームを作成して、ログストリームにログイベントをアップロードし、ログストリームの詳細を一覧表示する権限を付与する 1 つのステートメントがあります。

このステートメントで Resource 値の末尾のワイルドカード文字 (*) は、任意のロググループに対して logs:CreateLogGroup、logs:CreateLogStream、logs:PutLogEvents、および logs:DescribeLogStreams アクションを実行するためのアクセス権限を付与することを意味します。このアクセス権限を特定のロググループに制限するには、リソース ARN 内のワイルドカード文字 (*) を特定のロググループ ARN に置き換えます。IAM ポリシーステートメント内のセクションの詳細については、IAM ユーザーガイドの「[IAM JSON ポリシーの要素のリファレンス](#)」を参照してください。すべての CloudWatch Logs アクションを示すリストについては、「[CloudWatch Logs の許可リファレンス](#)」を参照してください。

CloudWatch コンソールの使用に必要な許可

ユーザーが CloudWatch コンソールで CloudWatch Logs を使用するには、そのユーザーに AWS アカウントの他の AWS リソースを記述できる最小限のアクセス許可のセットが必要です。CloudWatch Logs コンソールで CloudWatch Logs を使用するには、次のサービスからのアクセス許可が必要になります。

- CloudWatch
- CloudWatch Logs
- OpenSearch Service
- IAM
- Kinesis
- Lambda
- Amazon S3

これらの最小限必要なアクセス権限よりも制限された IAM ポリシーを作成している場合、その IAM ポリシーを使用するユーザーに対してコンソールは意図したとおりには機能しません。CloudWatchReadOnlyAccess で説明されているとおり、ユーザーが CloudWatch コンソールを使用できること、および、[AWS CloudWatch Logs の マネージド \(事前定義\) ポリシー](#) 管理ポリシーがユーザーにアタッチされていることを確認してください。

AWS CLI または CloudWatch Logs API のみ呼び出すユーザーには、最小限のコンソールアクセス許可を付与する必要はありません。

CloudWatch コンソールを使用してログのサブスクリプションを管理していないユーザーが、コンソールを使用するために必要なフルセットのアクセス許可は、次のとおりです。

- cloudwatch:GetMetricData
- cloudwatch:ListMetrics
- logs:CancelExportTask
- logs>CreateExportTask
- logs>CreateLogGroup
- logs>CreateLogStream
- logs>DeleteLogGroup
- logs>DeleteLogStream
- logs>DeleteMetricFilter
- logs>DeleteQueryDefinition
- logs>DeleteRetentionPolicy
- logs>DeleteSubscriptionFilter
- logs:DescribeExportTasks
- logs:DescribeLogGroups
- logs:DescribeLogStreams
- logs:DescribeMetricFilters
- logs:DescribeQueryDefinitions
- logs:DescribeQueries
- logs:DescribeSubscriptionFilters
- logs:FilterLogEvents
- logs:GetLogEvents
- logs:GetLogGroupFields
- logs:GetLogRecord
- logs:GetQueryResults

- logs:PutMetricFilter
- logs:PutQueryDefinition
- logs:PutRetentionPolicy
- logs:StartQuery
- logs:StopQuery
- logs:PutSubscriptionFilter
- logs:TestMetricFilter

コンソールを使用してログのサブスクリプションを管理するユーザーには、以下のアクセス許可も必要です。

- es:DescribeElasticsearchDomain
- es:ListDomainNames
- iam:AttachRolePolicy
- iam:CreateRole
- iam:GetPolicy
- iam:GetPolicyVersion
- iam:GetRole
- iam:ListAttachedRolePolicies
- iam:ListRoles
- kinesis:DescribeStreams
- kinesis:ListStreams
- lambda:AddPermission
- lambda:CreateFunction
- lambda:GetFunctionConfiguration
- lambda:ListAliases
- lambda:ListFunctions
- lambda:ListVersionsByFunction
- lambda:RemovePermission
- s3:ListBuckets

AWS CloudWatch Logs の マネージド (事前定義) ポリシー

AWS は、によって作成および管理されるスタンドアロン IAM ポリシーを提供することで、多くの一般的なユースケースに対処します AWS。マネージドポリシーは、一般的ユースケースに必要な許可を付与することで、どの許可が必要なのかをユーザーが調査する必要をなくすることができます。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。

アカウントのユーザーとロールにアタッチできる以下の AWS 管理ポリシーは、CloudWatch Logs に固有です。

- CloudWatchLogsFullAccess – CloudWatch Logs へのフルアクセスを付与します。
- CloudWatchLogsReadOnlyAccess – CloudWatch Logs への読み取り専用アクセスを付与します。

CloudWatchLogsFullAccess

CloudWatchLogsFullAccess ポリシーは、CloudWatch Logs へのフルアクセスを付与します。このポリシーを持つユーザーが自然言語プロンプトから [CloudWatch Logs Insights](#) クエリ文字列を生成できるように、このポリシーには `cloudwatch:GenerateQuery` および `cloudwatch:GenerateQueryResultsSummary` のアクセス許可が含まれています。このポリシーのすべての内容については、「AWS マネージドポリシーリファレンスガイド」の「[CloudWatchLogsFullAccess](#)」を参照してください。

[CloudWatchLogsReadOnlyAccess]

CloudWatchLogsReadOnlyAccess ポリシーは、CloudWatch Logs への読み取り専用のアクセス権限を付与します。このポリシーを持つユーザーが自然言語プロンプトから [CloudWatch Logs Insights](#) クエリ文字列を生成できるように、このポリシーには `cloudwatch:GenerateQuery` および `cloudwatch:GenerateQueryResultsSummary` のアクセス許可が含まれています。このポリシーのすべての内容については、「AWS マネージドポリシーリファレンスガイド」の「[CloudWatchLogsReadOnlyAccess](#)」を参照してください。

CloudWatchOpenSearchDashboardsFullAccess

CloudWatchOpenSearchDashboardsFullAccess ポリシーは、OpenSearch Service との統合を作成、管理、削除し、それらの統合において提供されるログダッシュボードを作成、削除、管理するためのアクセス権限を付与します。詳細については、「[Amazon OpenSearch Service で分析する](#)」を参照してください。

このポリシーのすべての内容については、「AWS マネージドポリシーリファレンスガイド」の「[CloudWatchOpenSearchDashboardsFullAccess](#)」を参照してください。

CloudWatchOpenSearchDashboardAccess

CloudWatchOpenSearchDashboardAccess ポリシーは、Amazon OpenSearch Service 分析で作成され、提供されるログダッシュボードを表示するアクセス権を付与します。詳細については、「[Amazon OpenSearch Service で分析する](#)」を参照してください。

Important

このポリシーの付与に加えて、ロールまたはユーザーが提供されるログダッシュボードを表示できるようにするには、OpenSearch Service との統合を作成するときにそれらも指定する必要があります。詳細については、「[ステップ 1: OpenSearch Service との統合を作成する](#)」を参照してください。

このポリシーのすべての内容については、「AWS マネージドポリシーリファレンスガイド」の「[CloudWatchOpenSearchDashboardAccess](#)」を参照してください。

CloudWatchLogsCrossAccountSharingConfiguration

CloudWatchCrossAccountSharingConfiguration ポリシーは、アカウント間で CloudWatch Logs リソースを共有するための Observability Access Manager リンクを作成、管理、および表示するためのアクセス許可を付与します。詳細については、「[CloudWatch のクロスアカウントオブザーバビリティ](#)」を参照してください。

このポリシーのすべての内容については、「AWS マネージドポリシーリファレンスガイド」の「[CloudWatchLogsCrossAccountSharingConfiguration](#)」を参照してください。

CloudWatchLogsAPIKeyAccess

CloudWatchLogsAPIKeyAccess ポリシーは CloudWatch Logs API キー認証と暗号化されたログ取り込みを有効にします。このポリシーは、ベアラートークンを使用して認証し、CloudWatch Logs にログイベントを書き込むためのアクセス許可と、ログが暗号化されたときにデータキーを復号および生成するための追加の AWS KMS アクセス許可を付与します。

このポリシーは以下のアクセス権限を与えます。

- logs – プリンシパルが API キーベアラートークンを介して認証し、CloudWatch Logs ストリームにログイベントを書き込むことを許可します。
- kms – プリンシパルが AWS KMS キーメタデータの読み取り、暗号化用のデータキーの生成、データの復号を行うことができます。これらのアクセス許可は、サービスがカスタマーマネー

ジド AWS KMS キーを使用してログデータを暗号化できるようにすることで、暗号化された CloudWatch Logs をサポートします。アクセスは、CloudWatch Logs サービスを介して呼び出されるオペレーションに制限されます。

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 [CloudWatchLogsAPIKeyAccess](#)」を参照してください。

AWS マネージドポリシーに対する CloudWatch Logs の更新

このサービスがこれらの変更の追跡を開始してからの CloudWatch Logs の AWS マネージドポリシーの更新に関する詳細を表示します。このページの変更に関する自動通知を入手するには、CloudWatch Logs ドキュメントの履歴ページから、RSS フィードにサブスクライブしてください。

変更	説明	日付
CloudWatchLogsAPIKeyAccess – 新しいポリシー。	CloudWatch Logs に新しい管理ポリシー CloudWatchLogsAPIKeyAccess が追加されました。 このポリシーは、CloudWatch Logs API キー認証と暗号化されたログ取り込みを有効にし、ベアラートークンを使用して認証し、CloudWatch Logs にログイベントを書き込むアクセス許可を付与します。	2026 年 2 月 17 日
CloudWatchLogsFullAccess — 既存のポリシーへの更新	CloudWatch Logs が、CloudWatchLogsFullAccess に対するアクセス許可を追加しました。	2025 年 12 月 2 日

変更	説明	日付
	テレメトリパイプラインと S3 テーブル統合への読み取り専用アクセスを許可するため、オブザーバビリティ管理アクションのアクセス許可が追加されました。	
CloudWatchLogsReadOnlyAccess — 既存のポリシーへの更新。	CloudWatch Logs が、CloudWatchLogsReadOnlyAccess に対するアクセス許可を追加しました。 テレメトリパイプラインと S3 テーブル統合への読み取り専用アクセスを許可するため、オブザーバビリティ管理アクションのアクセス許可が追加されました。	2025 年 12 月 2 日
CloudWatchLogsFullAccess — 既存のポリシーへの更新	CloudWatch Logs が、CloudWatchLogsFullAccess に対するアクセス許可を追加しました。 cloudwatch:GenerateQueryResultsSummary のアクセス許可が追加され、クエリ結果の自然言語の概要を生成できるようになりました。	2025 年 5 月 20 日

変更	説明	日付
CloudWatchLogsReadOnlyAccess — 既存のポリシーへの更新。	CloudWatch Logs が、CloudWatchLogsReadOnlyAccess に対するアクセス許可を追加しました。 cloudwatch:GenerateQueryResultsSummary のアクセス許可が追加され、クエリ結果の自然言語の概要を生成できるようになりました。	2025 年 5 月 20 日
CloudWatchLogsFullAccess — 既存のポリシーへの更新	CloudWatch Logs が、CloudWatchLogsFullAccess に対するアクセス許可を追加しました。 一部の機能で CloudWatch Logs と OpenSearch Service の統合を有効にするために、Amazon OpenSearch Service および IAM のアクセス許可が追加されました。	2024 年 12 月 1 日

変更	説明	日付
CloudWatchOpenSearchDashboardsFullAccess – 新しい IAM ポリシー。	CloudWatch Logs は新しい IAM ポリシーである CloudWatchOpenSearchDashboardsFullAccess を追加しました。 - このポリシーは、OpenSearch Service との統合を作成、管理、削除し、それらの統合において提供されるログダッシュボードを作成、管理、削除するためのアクセス権限を付与します。詳細については、「 Amazon OpenSearch Service で分析する 」を参照してください。	2024 年 12 月 1 日
CloudWatchOpenSearchDashboardAccess – 新しい IAM ポリシー。	CloudWatch Logs は新しい IAM ポリシーである CloudWatchOpenSearchDashboardAccess を追加しました。 - このポリシーは、Amazon OpenSearch Serviceを利用して提供されるログダッシュボードを表示するためのアクセス権限を付与します。詳細については、「 Amazon OpenSearch Service で分析する 」を参照してください。	2024 年 12 月 1 日

変更	説明	日付
CloudWatchLogsFullAccess — 既存のポリシーへの更新	CloudWatch Logs が、CloudWatchLogsFullAccess にアクセス許可を追加しました。 このポリシーを持つユーザーが自然言語プロンプトから CloudWatch Logs Insights クエリ文字列を生成できるように、cloudwatch:GenerateQuery アクセス許可を追加しました。	2023 年 11 月 27 日
CloudWatchLogsReadOnlyAccess — 既存のポリシーへの更新。	CloudWatch が、CloudWatchLogsReadOnlyAccess にアクセス許可を追加しました。 このポリシーを持つユーザーが自然言語プロンプトから CloudWatch Logs Insights クエリ文字列を生成できるように、cloudwatch:GenerateQuery アクセス許可を追加しました。	2023 年 11 月 27 日

変更	説明	日付
CloudWatchLogsReadOnlyAccess — 既存のポリシーへの更新	CloudWatch Logs が、CloudWatchLogsReadOnlyAccess に対するアクセス許可を追加しました。 logs:StartLiveTail および logs:StopLiveTail のアクセス権限が追加されたため、このポリシーを持つユーザーは、コンソールを使用して CloudWatch Logs の Live Tail セッションを開始および停止できます。詳細については、「ライブテールを使用してほぼリアルタイムでログを表示する」を参照してください。	2023 年 6 月 6 日
CloudWatchCrossAccountSharingConfiguration – 新しいポリシー	CloudWatch Logs が、CloudWatch Logs ロググループを共有する CloudWatch クロスアカウントオブザーバビリティのリンクを管理できるようにする新しいポリシーを追加しました。 詳細については、「CloudWatch のクロスアカウントオブザーバビリティ」を参照してください。	2022 年 11 月 27 日

変更	説明	日付
CloudWatchLogsReadOnlyAccess — 既存のポリシーへの更新	CloudWatch Logs が、CloudWatchLogsReadOnlyAccess に対するアクセス許可を追加しました。 このポリシーを持つユーザーがコンソールを使用して、CloudWatch のクロスアカウントオブザーバビリティでソースアカウントから共有されたデータを表示できるようにするための <code>oam:ListSinks</code> および <code>oam:ListAttachedLinks</code> 許可が追加されました。	2022 年 11 月 27 日

カスタマーマネージドポリシーの例

独自のカスタム IAM ポリシーを作成して、CloudWatch Logs アクションとリソースのためのアクセス権限を許可することができます。こうしたカスタムポリシーは、該当するアクセス許可が必要なユーザーまたはグループにアタッチできます。

このセクションでは、さまざまな CloudWatch Logs アクションのアクセス許可を付与するユーザーポリシー例を示しています。これらのポリシーは、CloudWatch Logs API、AWS SDK、または AWS CLIを使用しているときに機能します。

例

- [例 1: CloudWatch Logs へのフルアクセスを許可する](#)
- [例 2: CloudWatch Logs への読み取り専用アクセスを許可する](#)
- [例 3: 1 つのロググループへのアクセスを許可する](#)

例 1: CloudWatch Logs へのフルアクセスを許可する

以下のポリシーでは、ユーザーにすべての CloudWatch Logs アクションへのアクセスを許可します。

JSON

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

例 2: CloudWatch Logs への読み取り専用アクセスを許可する

AWS は、CloudWatch Logs データへの読み取り専用アクセスを有効にする

CloudWatchLogsReadOnlyAccess ポリシーを提供します。CloudWatch 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。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "logs:Describe*",
        "logs:Get*",
        "logs:List*",
        "logs:StartQuery",
        "logs:StopQuery",
        "logs:TestMetricFilter",
        "logs:FilterLogEvents",

```



```
        "logs:StartLiveTail",
        "logs:StopLiveTail",
        "cloudwatch:GenerateQuery"
    ],
    "Effect": "Allow",
    "Resource": "*"
  }
}
```

例 3: 1 つのロググループへのアクセスを許可する

次のポリシーでは、指定した 1 つのロググループのログイベントの読み取りと書き込みをユーザーに許可します。

Important

Resource 行のロググループ名の末尾にある `:*` は、ポリシーがこのロググループのすべてのログストリームに適用されることを示すために必要です。`:*` を省略すると、ポリシーは適用されません。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "logs:CreateLogStream",
        "logs:DescribeLogStreams",
        "logs:PutLogEvents",
        "logs:GetLogEvents"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:logs:us-west-2:123456789012:log-group:SampleLogGroupName:*"
    }
  ]
}
```

ロググループレベルでのコントロールのためにタグ付けと IAM ポリシーを使用する

他のロググループへのアクセスを防止しながら、特定のロググループへのアクセスをユーザーに許可することができます。これを行うには、ロググループにタグを付け、IAM ポリシーを使用してそれらのタグを参照します。タグをロググループに適用するには、logs:TagResource または logs:TagLogGroup のアクセス許可が必要です。これは、作成時にロググループにタグを割り当てる場合と、後で割り当てる場合の両方に当てはまります。

ロググループのタグ付けの詳細については、「[Amazon CloudWatch Logs のロググループにタグを付ける](#)」を参照してください。

ロググループにタグを付けるときは、特定のタグを持つロググループのみにアクセスを許可する IAM ポリシーをユーザーに付与できます。たとえば、以下のポリシーステートメントでは、タグキー Green の値が Team のロググループにのみアクセス権が付与されます。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "logs:*"
      ],
      "Effect": "Allow",
      "Resource": "*",
      "Condition": {
        "StringLike": {
          "aws:ResourceTag/Team": "Green"
        }
      }
    }
  ]
}
```

StopQuery および StopLiveTail API オペレーションは、従来の意味では AWS リソースとやり取りしません。何らかの方法でデータを返したり、データを入力したり、リソースを変更したりすることはありません。代わりに、特定のライブテールセッションまたは特定の CloudWatch Logs Insights クエリなど、リソースとして分類されていないものでのみ動作します。そのため、これらの操作の

IAM ポリシーで Resource フィールドを指定するとき、次の例のように、Resource フィールドの値を * として設定する必要があります。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:StopQuery",
        "logs:StopLiveTail"
      ],
      "Resource": "*"
    }
  ]
}
```

IAM ポリシーステートメントの詳細については、『IAM ユーザーガイド』の「[ポリシーを使用したアクセス制御](#)」を参照してください。

CloudWatch Logs の許可リファレンス

[アクセスコントロール](#) をセットアップし、IAM アイデンティティ (アイデンティティベースのポリシー) にアタッチできるアクセス権限ポリシーを作成する際、以下の表をリファレンスとして使用できます。この表には、各 CloudWatch Logs API オペレーション、およびその実行のためのアクセス権限を付与できる対応するアクションを示しています。アクションは、ポリシーの Action フィールドで指定します。Resource フィールドでは、ロググループまたはログストリームの ARN を指定するか、* を指定してすべての CloudWatch Logs リソースを表すことができます。

CloudWatch Logs ポリシーで AWS 全体の条件キーを使用して、条件を表現できます。AWS 全体のキーの完全なリストについては、IAM ユーザーガイドの [AWS 「グローバルおよび IAM 条件コンテキストキー」](#) を参照してください。

Note

アクションを指定するには、API オペレーション名の前に `logs:` プレフィックスを使用します。たとえば、`logs:CreateLogGroup`、`logs:CreateLogStream`、または `logs:*` (すべての CloudWatch Logs アクションの場合)。

CloudWatch Logs API オペレーションおよびアクションに必要な許可

CloudWatch Logs API のオペレーション	必要なアクセス許可 (API アクション)
CancelExportTask	<code>logs:CancelExportTask</code> 保留中または実行中のエクスポートタスクをキャンセルするのに必要です。
CreateExportTask	<code>logs:CreateExportTask</code> ロググループから Amazon S3バケットにデータをエクスポートするのに必要です。
CreateLogGroup	<code>logs:CreateLogGroup</code> 新しいロググループを作成するのに必要です。
CreateLogStream	<code>logs:CreateLogStream</code> ロググループに新しいログストリームを作成するのに必要です。
DeleteDestination	<code>logs:DeleteDestination</code> ログ宛先を削除したり、サブスクリプションのフィルタを無効化するのに必要です。
DeleteLogGroup	<code>logs>DeleteLogGroup</code>

CloudWatch Logs API のオペレーション	必要なアクセス許可 (API アクション)
	ロググループや関連したアーカイブログイベントを削除するのに必要です。
DeleteLogStream	logs:DeleteLogStream ログストリームや関連したアーカイブログイベントを削除するのに必要です。
DeleteMetricFilter	logs:DeleteMetricFilter ロググループに関連したメトリクスフィルタを削除するのに必要です。
DeleteQueryDefinition	logs:DeleteQueryDefinition CloudWatch Logs Insights で保存されたクエリ定義を削除するのに必要です。
DeleteResourcePolicy	logs:DeleteResourcePolicy CloudWatch Logs リソースポリシーを削除するために必要です。
DeleteRetentionPolicy	logs:DeleteRetentionPolicy ロググループの保持ポリシーを削除するのに必要です。
DeleteSubscriptionFilter	logs:DeleteSubscriptionFilter ロググループに関連したサブスクリプションのフィルタを削除するのに必要です。

CloudWatch Logs API のオペレーション	必要なアクセス許可 (API アクション)
DescribeDestinations	<code>logs:DescribeDestinations</code> アカウントに関連したすべての送信先を表示するのに必要です。
DescribeExportTasks	<code>logs:DescribeExportTasks</code> アカウントに関連したすべてのエクスポートタスクを表示するのに必要です。
DescribeLogGroups	<code>logs:DescribeLogGroups</code> アカウントに関連したすべてのロググループを表示するのに必要です。
DescribeLogStreams	<code>logs:DescribeLogStreams</code> ロググループに関連したすべてのログストリームを表示するのに必要です。
DescribeMetricFilters	<code>logs:DescribeMetricFilters</code> ロググループに関連したすべてのメトリクスを表示するのに必要です。
DescribeQueryDefinitions	<code>logs:DescribeQueryDefinitions</code> CloudWatch Logs Insights で保存されたクエリ定義のリストを表示するために必要です。

CloudWatch Logs API のオペレーション	必要なアクセス許可 (API アクション)
DescribeQueries	<code>logs:DescribeQueries</code> スケジュールされた、実行された、または最近実行された CloudWatch Logs Insights クエリのリストを表示するために必要です。
DescribeResourcePolicies	<code>logs:DescribeResourcePolicies</code> CloudWatch Logs リソースポリシーのリストを表示するために必要です。
DescribeSubscriptionFilters	<code>logs:DescribeSubscriptionFilters</code> ロググループに関連したすべてのサブスクリプションフィルタを表示するのに必要です。
FilterLogEvents	<code>logs:FilterLogEvents</code> ロググループのフィルタパターンでログイベントをソートするのに必要です。
GetLogEvents	<code>logs:GetLogEvents</code> ログストリームからログイベントを取得するのに必要です。
GetLogGroupFields	<code>logs:GetLogGroupFields</code> ロググループのログイベントに含まれるフィールドのリストを取得するために必要です。

CloudWatch Logs API のオペレーション	必要なアクセス許可 (API アクション)
GetLogRecord	<code>logs:GetLogRecord</code> 1 つのログイベントから詳細を取得するために必要です。
GetLogObject	<code>logs:GetLogRecord</code> <code>PutOpenTelemetryLogs</code> API によって取り込まれたログイベントの大部分のコンテンツを取得するために必要です。
GetQueryResults	<code>logs:GetQueryResults</code> CloudWatch Logs Insights クエリの結果を取得するために必要です。
<code>ListEntitiesForLogGroup</code> (CloudWatch コンソールでのみアクセス許可)	<code>logs:ListEntitiesForLogGroup</code> ロググループに関連付けられたエンティティを検索するために必要です。CloudWatch コンソール内の関連するログを調べるために必要です。
<code>ListLogGroupsForEntity</code> (CloudWatch コンソールでのみアクセス許可)	<code>logs:ListLogGroupsForEntity</code> エンティティに関連付けられたロググループを検索するために必要です。CloudWatch コンソール内の関連するログを調べるために必要です。
ListTagsLogGroup	<code>logs:ListTagsLogGroup</code> ロググループに関連したタグをリストするのに必要です。

CloudWatch Logs API のオペレーション	必要なアクセス許可 (API アクション)
ListLogGroups	<code>logs:DescribeLogGroups</code> アカウントに関連したすべてのロググループを表示するのに必要です。
PutDestination	<code>logs:PutDestination</code> 宛先ログストリーム (Kinesis ストリームなど) の作成や更新に必要です。
PutDestinationPolicy	<code>logs:PutDestinationPolicy</code> 既存のログ宛先に関連するアクセスポリシーの作成や更新に必要です。
PutLogEvents	<code>logs:PutLogEvents</code> 一連のログイベントをログストリームに更新するのに必要です。
PutMetricFilter	<code>logs:PutMetricFilter</code> メトリクスフィルタの作成や更新、またはそれをロググループに関連付けるのに必要です。
PutQueryDefinition	<code>logs:PutQueryDefinition</code> パラメータを含む保存されたクエリなど、CloudWatch Logs Insights にクエリを保存するために必要です。

CloudWatch Logs API のオペレーション	必要なアクセス許可 (API アクション)
PutResourcePolicy	<code>logs:PutResourcePolicy</code> CloudWatch Logs リソースポリシーを作成するために必要です。
PutRetentionPolicy	<code>logs:PutRetentionPolicy</code> ロググループでログイベント (保持) を維持する日数を設定するのに必要です。
PutSubscriptionFilter	<code>logs:PutSubscriptionFilter</code> サブスクリプションフィルタの作成や更新、またはそれをロググループに関連付けるのに必要です。
StartQuery	<code>logs:StartQuery</code> CloudWatch Logs Insights クエリを開始するために必要です。パラメータを使用して保存されたクエリを実行するには、 も必要です <code>logs:DescribeQueryDefinitions</code> 。
StopQuery	<code>logs:StopQuery</code> 進行中の CloudWatch Logs Insights クエリを停止するために必要です。
TagLogGroup	<code>logs:TagLogGroup</code> ロググループのタグを追加または更新するのに必要です。

CloudWatch Logs API のオペレーション	必要なアクセス許可 (API アクション)
TestMetricFilter	<code>logs:TestMetricFilter</code> ログイベントメッセージのサンプリングに対してフィルタパターンをテストするのに必要です。

CloudWatch Logs のサービスにリンクされたロールの使用

Amazon CloudWatch Logs は AWS Identity and Access Management、(IAM) [サービスにリンクされたロール](#)を使用します。サービスにリンクされたロールは、CloudWatch Logs に直接リンクされた一意のタイプの IAM ロールです。サービスにリンクされたロールは CloudWatch Logs によって事前定義されており、サービスがユーザーに代わって他の AWS サービスを呼び出すために必要なすべてのアクセス許可が含まれています。

サービスにリンクされたロールでは、必要なアクセス許可を手動で追加する必要がないため、CloudWatch Logs の設定が効率的になります。CloudWatch Logs は、サービスにリンクされたロールのアクセス許可を定義します。特に定義されていない限り、CloudWatch Logs のみがこれらのロールを引き受けることができます。定義されたアクセス権限には、信頼ポリシーとアクセス権限ポリシーが含まれます。アクセス権限ポリシーを他の IAM エンティティにアタッチすることはできません。

サービスにリンクされたロールをサポートしているその他のサービスの詳細については、「[IAM と連携するAWS のサービス](#)」を参照してください。サービスにリンクされたロール列が「はい」になっているサービスを見つけます。サービスにリンクされたロールに関するドキュメントをサービスで表示するには、[はい] リンクを選択します。

CloudWatch Logs のサービスにリンクされたロールの許可

CloudWatch Logs は `AWSServiceRoleForLogDelivery` というサービスにリンクされたロールを使用します。CloudWatch Logs は、このサービスにリンクされたロールを使用して Firehose に直接ログを書き込みます。詳細については、「[AWS サービスからのログ記録を有効にする](#)」を参照してください。

`AWSServiceRoleForLogDelivery` サービスにリンクされたロールは、ロールの引き受けについて以下のサービスを信頼します。

- `logs.amazonaws.com`

ロールのアクセス許可ポリシーは、指定したリソースに対して以下のアクションを完了することを CloudWatch Logs に許可します。

- アクション: 値が `True` の `LogDeliveryEnabled` キーを持つタグが付いたすべての Firehose ストリーム上で `firehose:PutRecordBatch` および `firehose:PutRecord`。このタグは、ログを Firehose に配信するサブスクリプションを作成すると、Firehose ストリームに自動的にアタッチされます。

IAM エンティティがサービスにリンクされたロールを作成、編集、削除できるようにするには、アクセス許可を設定する必要があります。このエンティティは、ユーザー、グループ、またはロールです。詳細については、IAM ユーザーガイドの「[サービスにリンクされたロールのアクセス許可](#)」を参照してください。

CloudWatch Logs のサービスにリンクされたロールの作成

サービスにリンクされたロールを手動で作成する必要はありません。AWS マネジメントコンソール、AWS CLI または AWS API で Firehose ストリームに直接送信されるログを設定すると、CloudWatch Logs によってサービスにリンクされたロールが作成されます。

このサービスリンクロールを削除した後で再度作成する必要がある場合は同じ方法でアカウントにロールを再作成できます。Firehose ストリームに直接送信するように再度ログを設定すると、CloudWatch Logs によってサービスにリンクされたロールが再び作成されます。

CloudWatch Logs のサービスにリンクされたロールの編集

CloudWatch Logs では、ロールを作成した後に、`AWSServiceRoleForLogDelivery` または他のサービスにリンクされたロールを編集することはできません。さまざまなエンティティがロールを参照する可能性があるため、ロール名を変更することはできません。ただし、IAM を使用したロールの説明の編集はできます。詳細については、IAM ユーザーガイドの「[サービスにリンクされたロールの編集](#)」を参照してください。

CloudWatch Logs のサービスにリンクされたロールの削除

サービスリンクロールを必要とする機能またはサービスが不要になった場合には、そのロールを削除することをお勧めします。そうすることで、積極的にモニタリングまたは保守されていない未使用のエンティティを排除できます。ただし、手動で削除する前に、サービスにリンクされたロールのリソースをクリーンアップする必要があります。

Note

リソースを削除する際に、CloudWatch Logs サービスでそのロールが使用されている場合、削除は失敗することがあります。失敗した場合は数分待ってから操作を再試行してください。

AWSServiceRoleForLogDelivery サービスにリンクされたロールによって使用されている CloudWatch Logs リソースを削除するには

- Firehose ストリームへのログの直接送信を停止します。

サービスリンクロールを IAM で手動削除するには

IAM コンソール、AWS CLI、または AWS API を使用して、AWSServiceRoleForLogDelivery サービスにリンクされたロールを削除します。詳細については、「[サービスにリンクされたロールの削除](#)」を参照してください。

CloudWatch Logs のサービスにリンクされたロールでサポートされるリージョン

CloudWatch Logs は、サービスが利用可能なすべての AWS リージョンでサービスにリンクされたロールの使用をサポートしています。詳細については、「[CloudWatch Logs リージョンとエンドポイント](#)」を参照してください。

CloudWatch Logs AWS のサービスにリンクされたロールの更新

この AWS サービスがこれらの変更の追跡を開始してからの CloudWatch Logs のサービスにリンクされたロールの更新に関する詳細を表示します。このページの変更に関する自動通知を入手するには、CloudWatch Logs ドキュメントの履歴ページから、RSS フィードにサブスクライブしてください。

変更	説明	日付
AWSServiceRoleForLogDelivery サービスにリンクされたロールポリシー – 既存のポリシーに更新	CloudWatch Logs で AWSServiceRoleForLogDelivery サービスにリンク	2021 年 7 月 15 日

変更	説明	日付
	されたロールに関連付けられた IAM ポリシーの許可が変更されました。以下の変更が行われました。 • <code>firehose:ResourceTag/LogDeliveryEnabled</code>: <code>"true"</code> 条件キーが <code>aws:ResourceTag/LogDeliveryEnabled</code>: <code>"true"</code> に変更されました。	
CloudWatch Logs が変更の追跡を開始しました	CloudWatch Logs は AWS 、管理ポリシーの変更の追跡を開始しました。	2021 年 6 月 10 日

Amazon CloudWatch Logs のコンプライアンス検証

AWS のサービス が特定のコンプライアンスプログラムの範囲内にあるかどうかを確認するには、「コンプライアンス [AWS のサービス プログラムによるスコープ](#)」の「コンプライアンス」を参照して、関心のあるコンプライアンスプログラムを選択します。一般的な情報については、[AWS 「コンプライアンスプログラム」](#)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。

を使用する際のお客様のコンプライアンス責任 AWS のサービス 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。を使用する際のコンプライアンス責任の詳細については AWS のサービス、[AWS 「セキュリティドキュメント」](#)を参照してください。

Amazon CloudWatch Logs の耐障害性

AWS グローバルインフラストラクチャは、AWS リージョンとアベイラビリティゾーンを中心に構築されています。リージョンには、低レイテンシー、高いスループット、そして高度の冗長ネットワークで接続されている複数の物理的に独立および隔離されたアベイラビリティゾーンがあります。アベイラビリティゾーンでは、ゾーン間で中断することなく自動的にフェイ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可用性、フォールトトレランス、および拡張性が優れています。

AWS リージョンとアベイラビリティ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。

Amazon CloudWatch Logs のインフラストラクチャセキュリティ

マネージドサービスである Amazon CloudWatch Logs は、AWS グローバルネットワークセキュリティで保護されています。AWS セキュリティサービスと [AWS 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環境を AWS 設計するには、「Security Pillar AWS Well-Architected Framework」の [「Infrastructure Protection」](#) を参照してください。

AWS 公開された API コールを使用して、ネットワーク経由で CloudWatch Logs にアクセスします。クライアントは以下をサポートする必要があります。

- Transport Layer Security (TLS)。TLS 1.2 が必須で、TLS 1.3 をお勧めします。
- DHE (楕円ディフィー・ヘルマン鍵共有) や ECDHE (楕円曲線ディフィー・ヘルマン鍵共有) などの完全前方秘匿性 (PFS) による暗号スイート。これらのモードは、Java 7 以降など、最近のほとんどのシステムでサポートされています。

インターフェイス VPC エンドポイントでの CloudWatch Logs の使用

Amazon Virtual Private Cloud (Amazon VPC) を使用して AWS リソースをホストする場合、VPC と CloudWatch Logs の間にプライベート接続を確立できます。この接続を使用して、インターネットを経由して送信せずに CloudWatch Logs にログを送信できます。CloudWatch Logs は、すべてのリージョンで IPv4 VPC エンドポイントをサポートし、すべてのリージョンで IPv6 エンドポイントをサポートします。

Amazon VPC は、定義した仮想ネットワークで AWS リソースを起動するために使用できる AWS サービスです。VPC を使用することで、IP アドレス範囲、サブネット、ルートテーブル、ネットワークゲートウェイなどのネットワーク設定を制御できます。VPC を CloudWatch Logs に接続するには、CloudWatch Logs のインターフェイス VPC エンドポイントを定義します。このタイプのエンドポイントにより、VPC を AWS のサービスに接続できるようになります。このエンドポイントは、インターネットゲートウェイ、ネットワークアドレス変換 (NAT) インスタンス、または VPN 接続を必要とせず、信頼性が高くスケーラブルな CloudWatch Logs への接続を提供します。詳細については、「Amazon VPC ユーザーガイド」の「[Amazon VPC とは](#)」を参照してください。

インターフェイス VPC エンドポイントは、プライベート IP アドレスを持つ Elastic Network Interface を使用して AWS サービス間のプライベート通信を可能にする AWS テクノロジーである AWS PrivateLink を利用しています。詳細については、「[New – AWS PrivateLink for AWS Services](#)」を参照してください。

以下の手順は、Amazon VPC のユーザー向けです。詳細については、『Amazon VPC ユーザーガイド』の「[開始方法](#)」を参照してください。

利用可能な状況

CloudWatch Logs は現在、AWS リージョンを含むすべての AWS GovCloud (US) リージョンで VPC エンドポイントをサポートしています。

CloudWatch Logs 用の VPC エンドポイントの作成

VPC で CloudWatch Logs の使用を開始するには、CloudWatch Logs のインターフェイス VPC エンドポイントを作成します。選択するサービスは、[com.amazonaws.**Region**.logs] です。FIPS エンドポイントに接続する場合、選択するサービスは `com.amazonaws.Region.logs-fips` です。CloudWatch Logs のあらゆる設定を変更する必要はありません。詳細については、『Amazon VPC ユーザーガイド』の「[インターフェイスエンドポイントの作成](#)」を参照してください。

StartLiveTail や GetLogObject などの一部の CloudWatch Logs APIs は、別のエンドポイントと VPC エンドポイント でホストされます `stream-logs.Region.amazonaws.com`。これらの APIs のインターフェイス VPC エンドポイントを作成するには、`com.amazonaws.Region.stream-logs` を選択します。FIPS エンドポイントに接続する場合、選択するサービスは `com.amazonaws.Region.stream-logs-fips` です。

VPC と CloudWatch Logs との間の接続のテスト

エンドポイントの作成が完了したら、接続をテストできます。

VPC と CloudWatch Logs のエンドポイント間の接続をテストするには

1. VPC にある Amazon EC2 インスタンスに接続します。接続の詳細については、Amazon EC2 ドキュメントの [Linux インスタンスへの接続](#) または [Windows インスタンスへの接続](#) を参照してください。
2. インスタンスから、 を使用して AWS CLI、既存のロググループの 1 つにログエントリを作成します。

まず、ログイベントを持つ JSON ファイルを作成します。タイムスタンプは、1970 年 1 月 1 日 00:00:00 UTC からの経過ミリ秒数で指定する必要があります。

```
[
  {
    "timestamp": 1533854071310,
    "message": "VPC Connection Test"
  }
]
```

次に、put-log-events コマンドを使用してログエントリを作成します。

```
aws logs put-log-events --log-group-name LogGroupName --log-stream-  
name LogStreamName --log-events file://JSONFileName
```

コマンドに対する応答に nextSequenceToken が含まれていた場合、コマンドは成功しており VPC エンドポイントが機能しています。

CloudWatch Logs の VPC エンドポイントへのアクセスの制御

VPC エンドポイントポリシーは、エンドポイントの作成時または変更時にエンドポイントに加える IAM リソースポリシーです。エンドポイントの作成時にポリシーをアタッチしない場合、サービスへのフルアクセスを許可するデフォルトのポリシーがアタッチされます。エンドポイントポリシーは、IAM ポリシーやサービス固有のポリシーを上書き、または置き換えたりするものではありません。これは、評価項目から指定されたサービスへのアクセスを制御するための別のポリシーです。

評価項目のポリシーは、JSON形式で記載する必要があります。

詳細については、「Amazon VPCユーザーガイド」の「[VPC評価項目によるサービスのアクセス制御](#)」を参照してください。

CloudWatch Logs のエンドポイントポリシーの例を次に示します。このポリシーでは、VPC を介して CloudWatch Logs に接続するユーザーはログストリームを作成してログを CloudWatch Logs に送信できますが、他の CloudWatch Logs アクションは実行できません。

```
{
  "Statement": [
    {
      "Sid": "PutOnly",
      "Principal": "*",
      "Action": [
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

CloudWatch Logs の VPC エンドポイントポリシーを変更するには

1. Amazon VPC コンソールの <https://console.aws.amazon.com/vpc/> を開いてください。
2. ナビゲーションペインで、[Endpoints] (エンドポイント) を選択します。
3. CloudWatch Logs のエンドポイントをまだ作成していない場合は、[Create Endpoint (エンドポイントの作成)] を選択します。次に、[com.amazonaws.**Region**.logs] を選択し、[エンドポイントの作成] を選択します。
4. [com.amazonaws.**Region**.logs] エンドポイントを選択し、画面の下部で [ポリシー] タブを選択します。
5. [ポリシーの編集] を選択してポリシーを変更します。

VPC コンテキストキーのサポート

CloudWatch Logs では、特定の VPC または特定の VPC エンドポイントへのアクセスを制限できる `aws:SourceVpc` および `aws:SourceVpce` コンテキストキーをサポートしています。これらのキーは、ユーザーが VPC エンドポイントを使用している場合にのみ使用できます。詳細については、「IAM ユーザーガイド」の「[一部のサービスに使用可能なキー](#)」を参照してください。

での CloudWatch Logs API およびコンソールオペレーションのログ記録 AWS CloudTrail

Amazon CloudWatch Logs は、ユーザー、ロール、または AWS のサービスが実行したアクションの記録を提供するサービスである [AWS CloudTrail](#) と統合されています。CloudTrail は、CloudWatch Logs の API コールをイベントとしてキャプチャします。キャプチャされたコールには、CloudWatch Logs コンソールからの呼び出しと、CloudWatch Logs API オペレーションへのコード呼び出しが含まれます。CloudTrail で収集した情報を使用して、CloudWatch Logs に対するリクエスト、リクエスト元の IP アドレス、リクエストの作成日時、その他の詳細を確認できます。

各イベントまたはログエントリには、誰がリクエストを生成したかという情報が含まれます。アイデンティティ情報は、以下を判別するのに役立ちます。

- ルートユーザーまたはユーザー認証情報のどちらを使用してリクエストが送信されたか。
- リクエストが IAM Identity Center ユーザーに代わって行われたかどうか。
- リクエストがロールまたはフェデレーションユーザーのテンポラリなセキュリティ認証情報を使用して行われたかどうか。
- リクエストが、別の AWS のサービスによって送信されたかどうか。

CloudTrail は、アカウントを作成する AWS アカウント と アクティブになり、CloudTrail イベント履歴に自動的にアクセスできます。CloudTrail の [イベント履歴] では、AWS リージョンで過去 90 日間に記録された管理イベントの表示、検索、およびダウンロードが可能で、変更不可能な記録を確認できます。詳細については、「AWS CloudTrail ユーザーガイド」の「[CloudTrail イベント履歴の使用](#)」を参照してください。[イベント履歴] の閲覧には CloudTrail の料金はかかりません。

AWS アカウント 過去 90 日間のイベントの継続的な記録については、証跡または [CloudTrail Lake](#) イベントデータストアを作成します。

CloudTrail 証跡

証跡により、CloudTrail はログファイルを Amazon S3 バケットに配信できます。を使用して作成されたすべての証跡 AWS マネジメントコンソール はマルチリージョンです。AWS CLIを使用する際は、単一リージョンまたは複数リージョンの証跡を作成できます。アカウント AWS リージョン 内のすべての でアクティビティをキャプチャするため、マルチリージョン証跡を作成することをお勧めします。単一リージョンの証跡を作成する場合、証跡の AWS リージョンに記録

されたイベントのみを表示できます。証跡の詳細については、「AWS CloudTrail ユーザーガイド」の「[AWS アカウントの証跡の作成](#)」および「[組織の証跡の作成](#)」を参照してください。

証跡を作成すると、進行中の管理イベントのコピーを 1 つ無料で CloudTrail から Amazon S3 バケットに配信できますが、Amazon S3 ストレージには料金がかかります。CloudTrail の料金の詳細については、「[AWS CloudTrail の料金](#)」を参照してください。Amazon S3 の料金に関する詳細については、「[Amazon S3 の料金](#)」を参照してください。

CloudTrail Lake イベントデータストア

[CloudTrail Lake] を使用すると、イベントに対して SQL ベースのクエリを実行できます。CloudTrail Lake は、行ベースの JSON 形式の既存のイベントを [Apache ORC](#) 形式に変換します。ORC は、データを高速に取得するために最適化された単票ストレージ形式です。イベントは、イベントデータストアに集約されます。イベントデータストアは、[高度なイベントセレクト](#)を適用することによって選択する条件に基づいた、イベントのイミュータブルなコレクションです。どのイベントが存続し、クエリに使用できるかは、イベントデータストアに適用するセレクトが制御します。CloudTrail Lake の詳細については、AWS CloudTrail ユーザーガイドの[AWS CloudTrail 「Lake の使用」](#)を参照してください。

CloudTrail Lake のイベントデータストアとクエリにはコストがかかります。イベントデータストアを作成する際に、イベントデータストアに使用する[料金オプション](#)を選択します。料金オプションによって、イベントの取り込みと保存にかかる料金、および、そのイベントデータストアのデフォルトと最長の保持期間が決まります。CloudTrail の料金の詳細については、「[AWS CloudTrail の料金](#)」を参照してください。

CloudWatch Logs は、CloudTrail ログファイルのイベントとして以下のアクションを記録します。

- [AssociateKmsKey](#)
- [CancelExportTask](#)
- [CreateDelivery](#)
- [CreateExportTask](#)
- [CreateLogAnomalyDetector](#)
- [CreateLogGroup](#)
- [CreateLogStream](#)
- [DeleteAccountPolicy](#)
- [DeleteDataProtectionPolicy](#)

- [DeleteDelivery](#)
- [DeleteDeliveryDestination](#)
- [DeleteDeliveryDestinationPolicy](#)
- [DeleteDeliverySource](#)
- [DeleteDestination](#)
- [DeleteIndexPolicy](#)
- [DeleteIntegration](#)
- [DeleteLogAnomalyDetector](#)
- [DeleteLogGroup](#)
- [DeleteLogStream](#)
- [DeleteMetricFilter](#)
- [DeleteQueryDefinition](#)
- [DeleteResourcePolicy](#)
- [DeleteRetentionPolicy](#)
- [DeleteSubscriptionFilter](#)
- [DeleteTransformer](#)
- [DescribeAccountPolicies](#)
- [DescribeConfigurationTemplates](#)
- [DescribeDeliveries](#)
- [DescribeDeliveryDestinations](#)
- [DescribeDeliverySources](#)
- [DescribeDestinations](#)
- [DescribeExportTasks](#)
- [DescribeFieldIndexes](#)
- [DescribeIndexPolicies](#)
- [DescribeLogGroups](#)
- [DescribeLogStreams](#)
- [DescribeMetricFilters](#)

- [DescribeQueries](#)
- [DescribeQueryDefinitions](#)
- [DescribeResourcePolicies](#)
- [DescribeSubscriptionFilters](#)
- [DisassociateKmsKey](#)
- [FilterLogEvents](#)
- [GetDataProtectionPolicy](#)
- [GetDelivery](#)
- [GetDeliveryDestination](#)
- [GetDeliveryDestinationPolicy](#)
- [GetDeliverySource](#)
- [GetIntegration](#)
- [GetLogAnomalyDetector](#)
- [GetLogEvents](#)
- [GetLogGroupFields](#)
- [GetLogRecord](#)
- [GetQueryResults](#)
- [GetTransformer](#)
- [ListAnomalies](#)
- [ListIntegrations](#)
- [ListLogAnomalyDetectors](#)
- [ListLogGroups](#)
- [ListLogGroupsForQuery](#)
- [ListTagsForResource](#)
- [ListTagsLogGroup](#)
- [PutAccountPolicy](#)
- [PutDataProtectionPolicy](#)
- [PutDeliveryDestination](#)

- [PutDeliveryDestinationPolicy](#)
- [PutDeliverySource](#)
- [PutDestination](#)
- [PutDestinationPolicy](#)
- [PutIndexPolicy](#)
- [PutIntegration](#)
- [PutMetricFilter](#)
- [PutQueryDefinition](#)
- [PutResourcePolicy](#)
- [PutRetentionPolicy](#)
- [PutSubscriptionFilter](#)
- [PutTransformer](#)
- [StartLiveTail](#)
- [StartQuery](#)
- [StopQuery](#)
- [TagResource](#)
- [TestMetricFilter](#)
- [TestTransformer](#)
- [UntagResource](#)
- [UpdateAnomaly](#)
- [UpdateDeliveryConfiguration](#)
- [UpdateLogAnomalyDetector](#)

各イベントまたはログエントリには、誰がリクエストを生成したかという情報が含まれます。同一性情報は次の判断に役立ちます。

- リクエストが、ルートと IAM ユーザー認証情報のどちらを使用して送信されたか。
- リクエストがロールまたはフェデレーションユーザーの一時的なセキュリティ認証情報を使用して行われたかどうか。
- リクエストが別の AWS サービスによって行われたかどうか。

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

CloudTrail でのクエリ生成情報

クエリジェネレーターコンソールイベントの CloudTrail ログ記録もサポートされています。クエリジェネレーターは現在、CloudWatch Logs Insights と CloudWatch Metric Insights でサポートされています。これらの CloudTrail イベントでは、eventSource は `monitoring.amazonaws.com` です。

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

```
{
  "eventVersion": "1.09",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "EX_PRINCIPAL_ID",
    "arn": "arn:aws:iam::123456789012:assumed-role/role_name",
    "accountId": "123456789012",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::111222333444:role/Administrator",
        "accountId": "123456789012",
        "userName": "SAMPLE_NAME"
      },
      "attributes": {
        "creationDate": "2020-04-08T21:43:24Z",
        "mfaAuthenticated": "false"
      }
    }
  },
  "eventTime": "2020-04-08T23:06:30Z",
  "eventSource": "monitoring.amazonaws.com",
  "eventName": "GenerateQuery",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "127.0.0.1",
  "userAgent": "exampleUserAgent",
  "requestParameters": {
    "query_ask": "****",
```



```
    "query_type": "LogsInsights",
    "logs_insights": {
      "fields": "****",
      "log_group_names": ["yourloggroup"]
    },
    "include_description": true
  },
  "responseElements": null,
  "requestID": "2f56318c-cfbd-4b60-9d93-1234567890",
  "eventID": "52723fd9-4a54-478c-ac55-1234567890",
  "readOnly": true,
  "eventType": "AwsApiCall",
  "managementEvent": true,
  "recipientAccountId": "111122223333",
  "eventCategory": "Management"
}
```

ログファイルエントリの理解

追跡は、指定したAmazon S3バケットにイベントをログファイルとして配信するように設定できます。CloudTrail のログファイルは、単一か複数のログエントリを含みます。イベントは、任意の出典からの単一のリクエストを表し、リクエストされたアクション、アクションの日時、リクエストパラメータなどに関する情報が含まれます。CloudTrail ログファイルは、パブリック API 呼び出しの順序付けられたスタックトレースではないため、特定の順序では表示されません。

以下のログファイルエントリは、ユーザーが CloudWatch Logs CreateExportTask アクションを呼び出したことを示します。

```
{
  "eventVersion": "1.03",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "EX_PRINCIPAL_ID",
    "arn": "arn:aws:iam::123456789012:user/someuser",
    "accountId": "123456789012",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "userName": "someuser"
  },
  "eventTime": "2016-02-08T06:35:14Z",
  "eventSource": "logs.amazonaws.com",
  "eventName": "CreateExportTask",
  "awsRegion": "us-east-1",
```

```
"sourceIPAddress": "127.0.0.1",
"userAgent": "aws-sdk-ruby2/2.0.0.rc4 ruby/1.9.3 x86_64-linux Seahorse/0.1.0",
"requestParameters": {
  "destination": "yourdestination",
  "logGroupName": "yourloggroup",
  "to": 123456789012,
  "from": 0,
  "taskName": "yourtask"
},
"responseElements": {
  "taskId": "15e5e534-9548-44ab-a221-64d9d2b27b9b"
},
"requestID": "1cd74c1c-ce2e-12e6-99a9-8dbb26bd06c9",
"eventID": "fd072859-bd7c-4865-9e76-8e364e89307c",
"eventType": "AwsApiCall",
"apiVersion": "20140328",
"recipientAccountId": "123456789012"
}
```

CloudWatch メトリクスによるモニタリング

このセクションの表を使用して、Amazon CloudWatch Logs が毎分 Amazon CloudWatch に送信するメトリクスを確認できます。

CloudWatch Logs のメトリック

AWS/Logs 名前空間には、次のメトリクスが含まれます。

メトリクス	説明
CallCount	アカウントで実行された指定された API オペレーションの数。 CallCount は CloudWatch Logs サービスの使用状況メトリクスです。詳細については、「CloudWatch Logs サービスの使用状況メトリクス」を参照してください。 有効なディメンション: クラス、リソース、サービス、タイプ 有効な統計: Sum 単位: なし
DeliveryErrors	データをサブスクリプション送信先に転送するときに CloudWatch Logs がエラーを受け取ったログイベントの数。送信先のサービスが、スロットリングの例外や再試行可能なサービス例外 (HTTP 5xx など) の再試行可能なエラーを返した場合、CloudWatch Logs は最大 24 時間配信を再試行し続けます。AccessDeniedException や ResourceNotFoundException などの再試行不可能なエラーの場合、CloudWatch Logs は再配信を試みません。 有効なディメンション: LogGroupName、DestinationType、FilterName、PolicyLevel 有効な統計: Sum 単位: なし

メトリクス	説明
DeliveryThrottling	データをサブスクリプション送信先に転送するときに CloudWatch Logs がスロットルされたロギイベントの数。 送信先のサービスが、スロットリングの例外や再試行可能なサービス例外 (HTTP 5xx など) の再試行可能なエラーを返した場合、CloudWatch Logs は最大 24 時間配信を再試行し続けます。AccessDeniedException や ResourceNotFoundException などの再試行不可能なエラーの場合、CloudWatch Logs は再配信を試みません。 有効なディメンション: LogGroupName、DestinationType、FilterName、PolicyLevel 有効な統計: Sum 単位: なし
EMFDisabledErrors	ロググループに一致する METRIC_EXTRACTION_POLICY タイプのアクティブなアカウントポリシーが原因で無視された EMF 形式のロギイベントの数。メトリクス抽出ポリシーについて詳しくは「PutAccountPolicy」を参照してください。 有効なディメンション: LogGroupName 有効な統計: Sum 単位: なし
EMFParsingErrors	埋め込みメトリクスフォーマットログの処理中に発生した解析エラーの数。このようなエラーは、埋め込みメトリクス形式として識別されたログが、適切な形式に従っていない場合に発生します。埋め込みメトリクス形式の詳細については、「仕様: 埋め込みメトリクスフォーマット」を参照してください。 有効なディメンション: LogGroupName 有効な統計: Sum 単位: なし

メトリクス	説明
EMFValidationErrors	埋め込みメトリクス形式ログの処理中に発生した検証エラーの数。これらのエラーは、埋め込みメトリクス形式ログ内のメトリクスの定義が、埋め込みメトリクス形式と <code>MetricDatum</code> の仕様に準拠していない場合に発生します。埋め込みメトリクス形式の詳細については、「仕様: 埋め込みメトリクス形式」を参照してください。データタイプ <code>MetricDatum</code> の詳細については、「Amazon CloudWatch API リファレンス」の「MetricDatum」を参照してください。  Note 特定の検証エラーにより、EMF ログ内の複数のメトリクスが公開されないことがあります。例えば、無効な名前空間で設定されたメトリクスはすべて削除されます。 有効なディメンション: <code>LogGroupName</code> 有効な統計: <code>Sum</code> 単位: なし
ErrorCount	アカウントで実行され、エラーが発生した API オペレーションの数。 <code>ErrorCount</code> は CloudWatch Logs サービスの使用状況メトリクスです。詳細については、「CloudWatch Logs サービスの使用状況メトリクス」を参照してください。 有効なディメンション: クラス、リソース、サービス、タイプ 有効な統計: <code>Sum</code> 単位: なし

メトリクス	説明
ForwardedBytes	サブスクリプション送信先に転送されたログイベントのボリューム (圧縮済みバイト数)。 有効なディメンション: LogGroupName、DestinationType、FilterName 有効な統計: Sum 単位: バイト
ForwardedLogEvents	サブスクリプション送信先に転送されたログイベントの数。 有効なディメンション: LogGroupName、DestinationType、FilterName、PolicyLevel 有効な統計: Sum 単位: なし
IncomingBytes	CloudWatch Logs にアップロードされたログイベントのボリューム (非圧縮バイト数)。LogGroupName ディメンションと同時に使用すると、ロググループにアップロードされたログイベントのボリューム (非圧縮バイト数) になります。 有効なディメンション: LogGroupName 有効な統計: Sum 単位: バイト
IncomingLogEvents	CloudWatch Logs にアップロードされたログイベントの数。LogGroupName ディメンションと同時に使用すると、ロググループにアップロードされたログイベントの数になります。 有効なディメンション: LogGroupName 有効な統計: Sum 単位: なし

メトリクス	説明
LogEvents WithFindings	CloudWatch Logs データ保護機能を使用して監査しているデータ文字列に一致したログイベントの数。詳細については、「機密性の高いログデータをマスキングで保護する」を参照してください。 有効なディメンション: なし 有効な統計: Sum 単位: なし
ThrottleCount	アカウントで実行され、使用クォータによって調整された API オペレーションの数。 ThrottleCount は CloudWatch Logs サービスの使用状況メトリクスです。詳細については、「CloudWatch Logs サービスの使用状況メトリクス」を参照してください。 有効なディメンション: クラス、リソース、サービス、タイプ 有効な統計: Sum 単位: なし

CloudWatch Logs メトリックのディメンション

ほとんどの CloudWatch Logs メトリクスで利用できるディメンションを以下のテーブルに示します。

ディメンション	説明
LogGroupName	メトリクスを表示する CloudWatch Logs ロググループの名前。
DestinationType	CloudWatch Logs データのサブスクリプション先であり、AWS Lambda、Amazon Kinesis Data Streams、または Amazon Data Firehose を使用できます。

ディメンション	説明
FilterName	ロググループから送信先にデータを転送するサブスクリプションフィルタの名前。サブスクリプションフィルタ名は CloudWatch で自動的に ASCII に変換され、サポートされていない文字は疑問符 (?) に置き換えられます。

サブスクリプションフィルタメトリクスのディメンション

アカウントレベルのサブスクリプションフィルタに関連するメトリクスのディメンションを次の表に示します。

ディメンション	説明
PolicyLevel	ポリシーが適用されるレベル。現在、このディメンションの有効な値は AccountPolicy のみです。
DestinationType	CloudWatch Logs データのサブスクリプション先であり、AWS Lambda、Amazon Kinesis Data Streams、または Amazon Data Firehose を使用できます。
FilterName	ロググループから送信先にデータを転送するサブスクリプションフィルタの名前。サブスクリプションフィルタ名は CloudWatch で自動的に ASCII に変換され、サポートされていない文字は疑問符 (?) に置き換えられます。

トランスフォーマーのメトリクスとディメンションをログに記録する

CloudWatch Logs は、以下のログトランスフォーマーメトリクスを AWS/Logs 名前空間の CloudWatch に発行します。

メトリクス	説明
TransformationErrors	ログイベントの変換中に指定されたトランスフォーマーで発生したエラーの数。

メトリクス	説明
	単位: なし 有効な統計: Sum
TransformedBytes	変換されたログイベントの出力の、非圧縮バイト単位のボリューム。 単位: バイト 有効な統計: Sum
TransformedLogEvents	変換されたログイベントの数。 単位: なし 有効な統計: Sum

以下のディメンションはトランスフォーマーメトリクスで使用されます。

ディメンション	説明
LogGroupname	このディメンションは log-group-level トランスフォーマーにのみ使用されます。
PolicyLevel	このディメンションは account-level トランスフォーマーにのみ使用されます。現在、このディメンションの有効な値は AccountPolicy のみです。

一元化メトリクスとディメンション

複数のアカウントとリージョンを一元的に監視する場合、CloudWatch Logs Centralization を使用するとログデータとメトリクスを一元的に統合できます。詳細については、「[クロスアカウントクロスリージョンログの一元化](#)」を参照してください。

CloudWatch Logs は、以下の一元化メトリクスを AWS/Logs 名前空間の CloudWatch に発行します。これらのメトリクスは、CloudWatch Logs 一元化ルールを使用するときに、ソースアカウントから送信先アカウントへのログデータのレプリケーションをモニタリングするのに役立ちます。

Note

CloudWatch は、一元化ルールの作成直後に、一元化メトリクスのレポートを開始します。メトリクスはベストエフォートベースで公開され、一元化ルールの作成後に届く新しいログイベントのみを追跡します。

メトリクス	説明	で発行
IncomingCopiedBytes	送信先アカウントにレプリケートされた非圧縮バイト単位のログデータの量。このメトリクスは、一元化ルールの作成後に到着する新しいログイベントにのみ適用されます。 有効なディメンション: SourceRegion、SourceAccount 有効な統計: Sum 単位: バイト	送信先アカウント
IncomingCopiedLogEvents	送信先アカウントにレプリケートされたログイベントの数。このメトリクスは、一元化ルールの作成後に到着する新しいログイベントにのみ適用されます。 有効なディメンション: SourceRegion、SourceAccount 有効な統計: Sum 単位: なし	送信先アカウント
OutgoingCopiedBytes	ソースアカウントから送信先アカウントに送信された非圧縮バイト単位のログデータの量。このメトリクスは、一元化ルールの作成後に送信先にコピーされた新しいログイベントにのみ適用されます。	ソースアカウント

メトリクス	説明	で発行
	有効なディメンション: DestinationRegion 有効な統計: Sum 単位: バイト	
OutgoingCopiedLogEvents	ソースアカウントから送信先アカウントに送信されたログイベントの数。このメトリクスは、一元化ルールを作成後に送信先にコピーされた新しいログイベントにのみ適用されます。 有効なディメンション: DestinationRegion 有効な統計: Sum 単位: なし	ソースアカウント
CentralizationError	ログデータのレプリケーション中に発生したエラーの数。エラーは、KMS キーのアクセス許可の問題、ロググループのクォータ制限、ログ階層の不一致など、さまざまな理由で発生する可能性があります。レプリケーションに失敗した特定のロググループまたはログストリームとその失敗理由を特定するには、ErrorType ディメンションをモニタリングします。 有効なディメンション: ErrorType 有効な統計: Sum 単位: なし	送信先アカウント

メトリクス	説明	で発行
CentralizationThrottled	一元化処理がスロットリングされた回数。スロットリングにより、一元化の処理が遅くなりますが、ログデータのレプリケートは妨げられません。 有効なディメンション: DestinationRegion 有効な統計: Sum 単位: なし	ソースアカウント

次のディメンションは、一元化メトリクスで使用されます。

ディメンション	説明
SourceRegion	ソースログデータが発信された AWS リージョン。
SourceAccount	ソースログデータが発信された AWS アカウント ID。
DestinationRegion	ログデータがレプリケートされる AWS リージョン。
ErrorType	一元化中に発生したエラーのタイプ。可能な値は以下のとおりです: - LogGroupQuotaExceeded : 送信先アカウントがロググループのクォータに達しました。 - InvalidKMS : KMS キーが欠落しているか、削除されているか、有効になっていないか、対称ではなく非対称です。 - AccessDenied : ログデータのレプリケート時にアクセスが拒否されました。これは、KMS キーの誤ったアクセス許可、IAM アクセス許可の欠落、制限的なリソースポリシーなど、アクセス許可が不十分なために発生する可能性があります。 - LogTierMismatch : 送信元ロググループと送信先ロググループのログ階層が一致しません。

ディメンション	説明
	- InvalidLogStream : ログストリーム名が最大長を超えた場合など、ログストリームを作成または書き込むときに発生する無効なパラメータ。 - InvalidLogGroup : 送信先アカウントでロググループを作成または設定するときに無効なパラメータが発生しました。 - DestinationEncryptionMismatch : 送信先ロググループは、一元化ルールで指定されているものとは異なる KMS 暗号化設定で既に存在します。

CloudWatch Logs サービスの使用状況メトリクス

CloudWatch Logs は、CloudWatch Logs API オペレーションの使用状況を追跡するメトリクスを CloudWatch に送信します。これらのメトリクスは、AWS サービスクォータに対応しています。これらのメトリクスを追跡することで、クォータを積極的に管理できます。詳細については、「[Service Quotas の統合と使用状況メトリクス](#)」を参照してください。

例えば、ThrottleCount メトリクスを追跡したり、そのメトリクスにアラームを設定したりできます。このメトリクスの値が上昇した場合は、スロットリングされた API オペレーションのためにクォータの引き上げをリクエストすることを検討してください。CloudWatch Logs のサービスクォータの詳細については、「[CloudWatch Logs クォータ](#)」を参照してください。

CloudWatch Logs は、AWS/Usage と AWS/Logs の名前空間の両方で 1 分ごとにサービスクォータ使用状況メトリクスを発行します。

次の表は、CloudWatch Logs によって発行されるサービス使用状況メトリクスを示しています。これらのメトリクスには、指定された単位がありません。これらのメトリクスの最も有用な統計は SUM です。これは、1 分間の合計オペレーション数を表します。

これらのメトリクスは、Service、Class、Type、Resource のすべてのディメンションの値とともに発行されます。また、Account Metrics と呼ばれる 1 つのディメンションで発行されます。アカウント内のすべての API オペレーションのメトリクスの合計を確認するには、Account Metrics ディメンションを使用します。特定の API のメトリクスを検索するには、他のディメンションを使用し、Resource ディメンションの API オペレーションの名前を指定します。

メトリクス

メトリクス	説明
CallCount	アカウントで実行された指定されたオペレーションの数。 CallCount は、AWS/Usage と AWS/Logs の両方の名前空間で発行されます。
ErrorCount	アカウントで実行され、エラーが発生した API オペレーションの数。 ErrorCount は AWS/Logs にのみ発行されます。
ThrottleCount	アカウントで実行され、使用クォータによって調整された API オペレーションの数。 ThrottleCount は AWS/Logs にのみ発行されます。

ディメンション

ディメンション	説明
Account metrics	このディメンションを使用して、すべての CloudWatch Logs API のメトリクスの合計を取得します。 特定の API のメトリクスを表示するには、この表に示されている他のディメンションを使用して、API 名を Resource の値として指定します。
Service	リソースを含む AWS サービスの名前。CloudWatch Logs 使用状況メトリクスの場合、このディメンションの値は Logs です。
Class	追跡されているリソースのクラス。CloudWatch Logs API 使用状況メトリクスでは、値が None のこのディメンションを使用します。
Type	追跡されるリソースのタイプ。現在、Service ディメンションが Logs である場合、Type の有効な値は API のみです。
Resource	API オペレーションの名前。有効な値には、 アクション にリストされているすべての API オペレーション名が含まれます。例えば、PutLogEvents

CloudWatch Logs クォータ

このセクションの表を使用して、Amazon CloudWatch Logs の AWS アカウントのデフォルトのサービスクォータを確認できます。これは制限とも呼ばれます。Service Quotas は、すべてではありませんが、そのほとんどが Service Quotas コンソールの Amazon CloudWatch Logs 名前空間に一覧表示されます。

Note

これらのクォータに対するクォータの引き上げをリクエストするには、このセクションの後半にある[手順](#)を参照してください。

名前	デフォルト	引き上げ可能	説明
アクティブなエクスポートタスク	サポートされている各リージョン: 1	[いいえ]	アカウントあたりのアクティブな (実行中または保留中の) エクスポートタスクの数
AssociateKmsKey スロットリング制限 (トランザクション/秒)	サポートされている各リージョン: 5/秒	いいえ	アカウント/リージョンごとの 1 秒あたりの associate-kms-key 呼び出しの最大数
1 秒あたりのトランザクションの AssociateSourceToS3TableIntegration スロットル制限	サポートされている各リージョン: 5	いいえ	アカウント/リージョンごとの 1 秒あたりの AssociateSourceToS3TableIntegration 呼び出しの最大数

名前	デフォルト	引き上げ可能	説明
バーストの 1 秒あたりのトランザクション数における AssociateSourceToS3TableIntegration スロットル制限	サポートされている各リージョン: 10	いいえ	バーストにおけるアカウント/リージョンごとの 1 秒あたりの AssociateSourceToS3TableIntegration 呼び出しの最大数
バッチサイズ	サポートされている各リージョン: 1 メガバイト	いいえ	put-log-events リクエストの最大バッチサイズ (MB)
CancelExportTask スロットリング制限 (トランザクション/秒)	サポートされている各リージョン: 5/秒	いいえ	アカウント/リージョンごとの 1 秒あたりの cancel-export-task 呼び出しの最大数
1 秒あたりのトランザクション数の CancellImportTask スロットル制限	サポートされている各リージョン: 1	[いいえ]	アカウント/リージョンあたりの 1 秒あたりの CancellImportTask 呼び出しの最大数
バーストの 1 秒あたりのトランザクション数における CancellImportTask スロットル制限	サポートされている各リージョン: 1	[いいえ]	バーストにおけるアカウント/リージョンごとの 1 秒あたりの CancellImportTask 呼び出しの最大数
CreateExportTask スロットリング制限 (トランザクション/秒)	サポートされている各リージョン: 5/秒	いいえ	アカウント/リージョンごとの 1 秒あたりの create-export-task 呼び出しの最大数

名前	デフォルト	引き上げ可能	説明
1 秒あたりのトランザクションの CreateImportTask スロットル制限	サポートされている各リージョン: 1	[いいえ]	アカウント/リージョンあたりの 1 秒あたりの CreateImportTask 呼び出しの最大数
バーストの 1 秒あたりのトランザクション数における CreateImportTask スロットル制限	サポートされている各リージョン: 1	[いいえ]	バーストにおけるアカウント/リージョンあたりの 1 秒あたりの CreateImportTask 呼び出しの最大数
CreateLogGroup スロットリング制限 (トランザクション/秒)	サポートされている各リージョン: 10/秒	あり	アカウント/リージョンごとの 1 秒あたりの create-log-group 呼び出しの最大数
CreateLogStream スロットリング制限 (トランザクション/秒)	サポートされている各リージョン: 50/秒	あり	アカウント/リージョンごとの 1 秒あたりの create-log-stream 呼び出しの最大数
データアーカイブ	サポートされている各リージョン: 5 GB	いいえ	無料データアーカイブの最大サイズ (GB)
1 秒あたりのトランザクションの DeleteDataProtectionPolicy スロットル制限	サポートされている各リージョン: 5	いいえ	アカウント/リージョンごとの 1 秒あたりの delete-data-protection-policy 呼び出しの最大数

名前	デフォルト	引き上げ可能	説明
DeleteDestination スロットリング制限 (トランザクション/秒)	サポートされている各リージョン: 5/秒	いいえ	アカウント/リージョンごとの 1 秒あたりの delete-destination 呼び出しの最大数
DeleteLogGroup スロットリング制限 (トランザクション/秒)	サポートされている各リージョン: 10/秒	あり	アカウント/リージョンごとの 1 秒あたりの delete-log-group 呼び出しの最大数
DeleteLogStream スロットリング制限 (トランザクション/秒)	サポートされている各リージョン: 15/秒	あり	アカウント/リージョンごとの 1 秒あたりの delete-log-stream 呼び出しの最大数
DeleteMetricFilter スロットリング制限 (トランザクション/秒)	サポートされている各リージョン: 5/秒	いいえ	アカウント/リージョンごとの 1 秒あたりの delete-metric-filter 呼び出しの最大数
DeleteRetentionPolicy スロットリング制限 (トランザクション/秒)	サポートされている各リージョン: 5/秒	いいえ	アカウント/リージョンごとの 1 秒あたりの delete-retention-policy 呼び出しの最大数
DeleteSubscriptionFilter スロットリング制限 (トランザクション/秒)	サポートされている各リージョン: 5/秒	いいえ	アカウント/リージョンごとの 1 秒あたりの delete-subscription-filter 呼び出しの最大数

名前	デフォルト	引き上げ可能	説明
DeleteTransformer スロットリング制限 (トランザクション/秒)	サポートされている各リージョン: 5/秒	いいえ	アカウント/リージョンごとの 1 秒あたりの delete-transformer 呼び出しの最大数
DescribeDestinations スロットリング制限 (トランザクション/秒)	サポートされている各リージョン: 5/秒	いいえ	アカウント/リージョンごとの 1 秒あたりの describe-destinations 呼び出しの最大数
DescribeExportTasks スロットリング制限 (トランザクション/秒)	サポートされている各リージョン: 5/秒	いいえ	アカウント/リージョンごとの 1 秒あたりの describe-export-tasks 呼び出しの最大数
DescribeImportTaskBatches 1 秒あたりのトランザクションのスロットリング制限	サポートされている各リージョン: 10	いいえ	アカウント/リージョンあたりの 1 秒あたりの DescribeImportTaskBatches 呼び出しの最大数
DescribeImportTaskBatches バーストの 1 秒あたりのトランザクションのスロットリング制限	サポートされている各リージョン: 10	いいえ	バーストにおけるアカウント/リージョンごとの 1 秒あたりの DescribeImportTaskBatches 呼び出しの最大数
DescribeImportTasks の 1 秒あたりのトランザクション数スロットル制限	サポートされている各リージョン: 10	いいえ	アカウント/リージョンあたりの 1 秒あたりの DescribeImportTasks 呼び出しの最大数

名前	デフォルト	引き上げ可能	説明
DescribeImportTasks バーストの 1 秒あたりのトランザクションのスロットリング制限	サポートされている各リージョン: 10	いいえ	バーストにおけるアカウント/リージョンあたりの 1 秒あたりの DescribeImportTasks 呼び出しの最大数
DescribeLogGroups スロットリング制限 (トランザクション/秒)	サポートされている各リージョン: 10/秒	あり	アカウント/リージョンごとの 1 秒あたりの describe-log-groups 呼び出しの最大数
DescribeLogStreams スロットリング制限 (トランザクション/秒)	サポートされている各リージョン: 25/秒	あり	アカウント/リージョンごとの 1 秒あたりの describe-log-streams 呼び出しの最大数
DescribeMetricFilters スロットリング制限 (トランザクション/秒)	サポートされている各リージョン: 5/秒	いいえ	アカウント/リージョンごとの 1 秒あたりの describe-metric-filters 呼び出しの最大数
DescribeSubscriptionFilters スロットリング制限 (トランザクション/秒)	サポートされている各リージョン: 5/秒	いいえ	アカウント/リージョンごとの 1 秒あたりの describe-subscription-filters 呼び出しの最大数
DisassociateSourceFromS3TableIntegration スロットリング制限の 1 秒あたりのトランザクション数	サポートされている各リージョン: 5	いいえ	アカウント/リージョンごとの 1 秒あたりの DisassociateSourceFromS3TableIntegration 呼び出しの最大数

名前	デフォルト	引き上げ可能	説明
DisassociateSourceFromS3TableIntegration バーストの 1 秒あたりのトランザクション数スロットル制限	サポートされている各リージョン: 10	はい	バーストにおけるアカウント/リージョンごとの 1 秒あたりの DisassociateSourceFromS3TableIntegration 呼び出しの最大数
イベントサイズ	サポートされている各リージョン: 1,024 キロバイト	はい	ログイベントの最大サイズ (KB)
FilterLogEvents スロットリング制限 (トランザクション/秒)	us-east-1: 25/秒 ap-northeast-3: 5/秒 ap-southeast-3: 5/秒 ca-west-1: 5/秒 eu-central-1: 5/秒 il-central-1: 5/秒 他のサポートされている各リージョン: 10/秒	はい	アカウント/リージョンごとの 1 秒あたりの filter-log-events 呼び出しの最大数

名前	デフォルト	引き上げ可能	説明
1 秒あたりのトランザクションの GetDataProtectionPolicy スロットル制限	サポートされている各リージョン : 5	はい	アカウント/リージョンごとの 1 秒あたりの get-data-protection-policy 呼び出しの最大数
GetLogEvents スロットリング制限 (トランザクション/秒)	us-west-2: 10/秒 ap-northeast-3: 10/秒 ap-southeast-3: 10/秒 ca-west-1: 10/秒 eu-central-1: 10/秒 eu-west-1: 10/秒 eu-west-3: 30/秒 il-central-1: 10/秒 他のサポートされている各リージョン: 25/秒	はい	アカウント/リージョンごとの 1 秒あたりの get-log-events 呼び出しの最大数
GetQueryResults スロットリング制限 (トランザクション/秒)	me-central-1: 6 他のサポートされている各リージョン: 10	はい	アカウント/リージョンごとの 1 秒あたりの get-query-results 呼び出しの最大数

名前	デフォルト	引き上げ可能	説明
GetTransformer スロットリング制限 (トランザクション/秒)	サポートされている各リージョン: 5/秒	はい	アカウント/リージョンごとの 1 秒あたりの get-transformer 呼び出しの最大数
ListSourcesForS3TableIntegration スロットリング制限、1 秒あたりのトランザクション数	サポートされている各リージョン: 5	はい	アカウント/リージョンごとの 1 秒あたりの ListSourcesForS3TableIntegration 呼び出しの最大数
ListSourcesForS3TableIntegration バーストの 1 秒あたりのトランザクション数スロットル制限	サポートされている各リージョン: 10	はい	バーストにおけるアカウント/リージョンごとの 1 秒あたりの ListSourcesForS3TableIntegration 呼び出しの最大数
ListTagsForResource スロットリング制限 (トランザクション/秒)	サポートされている各リージョン: 5/秒	はい	アカウント/リージョンごとの 1 秒あたりの list-tags-for-resource 呼び出しの最大数
ListTagsLogGroup スロットリング制限 (トランザクション/秒)	サポートされている各リージョン: 5/秒	はい	アカウント/リージョンごとの 1 秒あたりの list-tags-log-group 呼び出しの最大数
Live Tail の同時セッション数の上限	サポートされている各リージョン: 15	可能	アカウントあたりの同時アクティブ Live Tail セッションの最大数

名前	デフォルト	引き上げ可能	説明
ロググループ	サポートされている各リージョン: 1,000,000	あ り	アカウントが保持できるロググループの最大数
Live Tail セッションあたりのスキャンされるロググループ	サポートされている各リージョン: 10	い い え	Live Tail セッションごとにスキャンできるロググループの最大数
ロググループあたりのメトリクスフィルター	サポートされている各リージョン: 100	い い え	ロググループあたりのメトリクスフィルターの数
1 秒あたりのトランザクションの PutBearerTokenAuthentication スロットル制限	サポートされている各リージョン: 5	い い え	アカウント/リージョンあたりの 1 秒あたりの put-bearer-token-authentication 呼び出しの最大数
バーストの 1 秒あたりのトランザクションの PutBearerTokenAuthentication スロットル制限	サポートされている各リージョン: 10	い い え	バーストにおけるアカウント/リージョンごとの 1 秒あたりの put-bearer-token-authentication 呼び出しの最大数
1 秒あたりのトランザクションの PutDataProtectionPolicy スロットル制限	サポートされている各リージョン: 5	い い え	アカウント/リージョンあたりの 1 秒あたりの put-data-protection-policy 呼び出しの最大数

名前	デフォルト	引き上げ可能	説明
PutDestination スロットリング制限 (トランザクション/秒)	サポートされている各リージョン: 5/秒	いいえ	アカウント/リージョンごとの 1 秒あたりの put-destination 呼び出しの最大数
PutDestinationPolicy スロットリング制限 (トランザクション/秒)	サポートされている各リージョン: 5/秒	いいえ	アカウント/リージョンごとの 1 秒あたりの put-destination-policy 呼び出しの最大数
PutLogEvents スロットリング制限 (トランザクション/秒)	サポートされている各リージョン: 5,000/秒	あり	アカウント/リージョンごとの 1 秒あたりの put-log-events 呼び出しの最大数
PutMetricFilter スロットリング制限 (トランザクション/秒)	サポートされている各リージョン: 5/秒	いいえ	アカウント/リージョンごとの 1 秒あたりの put-metric-filter 呼び出しの最大数
PutRetentionPolicy スロットリング制限 (トランザクション/秒)	サポートされている各リージョン: 5/秒	いいえ	アカウント/リージョンごとの 1 秒あたりの put-retention-policy 呼び出しの最大数
PutSubscriptionFilter スロットリング制限 (トランザクション/秒)	サポートされている各リージョン: 5/秒	いいえ	アカウント/リージョンごとの 1 秒あたりの put-subscription-filter 呼び出しの最大数

名前	デフォルト	引き上げ可能	説明
PutTransformer スロットリング制限 (トランザクション/秒)	サポートされている各リージョン: 5/秒	はい	アカウント/リージョンごとの 1 秒あたりの put-transformer 呼び出しの最大数
リソースポリシー	サポートされている各リージョン: 10	はい	アカウント/リージョンごとのリソースポリシーの最大数
StartLiveTail スロットリング制限 (トランザクション/秒)	サポートされている各リージョン: 5	はい	アカウント/リージョンごとの 1 秒あたりの start-live-tail 呼び出しの最大数。この制限は、コンソールと API の両方に個別に適用されます
StartQuery スロットリング制限 (トランザクション/秒)	me-central-1: 6 他のサポートされている各リージョン: 10	はい	アカウント/リージョンごとの 1 秒あたりの start-query 呼び出しの最大数
ロググループあたりのサブスクリプションフィルター	サポートされている各リージョン: 2	はい	ロググループあたりのサブスクリプションフィルターの数
TagLogGroup スロットリング制限 (トランザクション/秒)	サポートされている各リージョン: 5/秒	はい	アカウント/リージョンごとの 1 秒あたりの tag-log-group 呼び出しの最大数

名前	デフォルト	引き上げ可能	説明
TagResource スロットリング制限 (トランザクション/秒)	サポートされている各リージョン: 5/秒	いいえ	アカウント/リージョンごとの 1 秒あたりの tag-resource 呼び出しの最大数
TestMetricFilter スロットリング制限 (トランザクション/秒)	サポートされている各リージョン: 5/秒	いいえ	アカウント/リージョンごとの 1 秒あたりの test-metric-filter 呼び出しの最大数
TestTransformer スロットリング制限 (トランザクション/秒)	サポートされている各リージョン: 5/秒	いいえ	アカウント/リージョンごとの 1 秒あたりの test-transformer 呼び出しの最大数
UntagLogGroup スロットリング制限 (トランザクション/秒)	サポートされている各リージョン: 5/秒	いいえ	アカウント/リージョンごとの 1 秒あたりの untag-log-group 呼び出しの最大数
UntagResource スロットリング制限 (トランザクション/秒)	サポートされている各リージョン: 5/秒	いいえ	アカウント/リージョンごとの 1 秒あたりの untag-resource 呼び出しの最大数
ログ異常ディテクター	サポートされている各リージョン: 500	あり	アクティブなログ異常ディテクターの最大数

CloudWatch Logs サービスクォータの管理

CloudWatch Logs は Service Quotas は、クォータを一元的に表示および管理できる AWS サービスです。詳細は、Service Quotas ユーザーガイドの「[Service Quotas とは](#)」を参照してください。

Service Quotas を使用すれば、CloudWatch Logs サービスクォータの値を簡単に調べることができます。

AWS マネジメントコンソール

コンソールを使用して CloudWatch Logs のサービスクォータを表示するには

1. <https://console.aws.amazon.com/servicequotas/> で Service Quotas コンソールを開きます。
2. ナビゲーションペインで、[AWS サービス] を選択します。
3. AWS のサービスのリストから、Amazon CloudWatch Logs を検索して選択します。

[Service Quotas] の一覧には、サービスクォータ名、適用された値 (使用可能な場合)、AWS デフォルトのクォータ、クォータ値が調整可能かどうかが表示されます。

4. 説明など、Service Quotas に関する追加情報を表示するには、クォータ名を選択します。
5. (オプション) クォータの引き上げをリクエストするには、[Request quota increase (クォータ引き上げリクエスト)] を選択、または必要な情報を入力または選択して、[Request (リクエスト)] を選択します。

コンソールを使用してさらにサービスクォータの操作を行うには、[Service Quotas ユーザーガイド](#)を参照してください。クォータの引き上げをリクエストするには、「Service Quotas ユーザーガイド」の「[クォータ引き上げリクエスト](#)」を参照してください。

AWS CLI

を使用して CloudWatch Logs のサービスクォータを表示するには AWS CLI

次のコマンドを実行して、デフォルトの CloudWatch Logs クォータを表示します。

```
aws service-quotas list-aws-default-service-quotas \
  --query 'Quotas[*]'.
{Adjustable:Adjustable,Name:QuotaName,Value:Value,Code:QuotaCode}' \
  --service-code logs \
  --output table
```

を使用してサービスクォータをさらに操作するには AWS CLI、[Service Quotas AWS CLI コマンドリファレンス](#)」を参照してください。クォータの引き上げをリクエストするには、[AWS CLI Command Reference](#) の [request-service-quota-increase](#) コマンドを参照してください。

ドキュメント履歴

次の表に、2018 年 6 月以降の CloudWatch Logs 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。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
CloudWatch Logs Insights のパラメータを含む保存されたクエリ	CloudWatch Logs Insights で、名前付きパラメータを使用して再利用可能なクエリテンプレートを作成できるようになりました。単一のテンプレートを定義し、複数のほぼ同一の保存済みクエリを維持するのではなく、実行時に異なる値を渡します。詳細については、 「パラメータを使用した保存済みクエリの使用」 を参照してください。	2026 年 3 月 26 日
新しい CloudWatch Logs 管理ポリシー CloudWatchLogsAPIKeyAccess	CloudWatch Logs は、CloudWatch Logs API キー認証と暗号化されたログ取り込みCloudWatchLogsAPIKeyAccess を有効にする新しい 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、 CloudWatch Logs updates to AWS managed policies 」を参照してください。	2026 年 2 月 17 日
CloudWatchLogsRead OnlyAccess ポリシーを更新しました	テレメトリパイプラインと S3 テーブル統合への読み取り専用アクセスを許可するため、オブザーバビリティ管	2025 年 12 月 2 日

理アクションのアクセス許可が [CloudWatchLogsReadOnlyAccess](#) に追加されました。

[CloudWatchLogsFullAccess ポリシーを更新しました](#)

テレメトリパイプラインと S3 テーブル統合への読み取り専用アクセスを許可するため、オブザーバビリティ管理アクションのアクセス許可が [CloudWatchLogsFullAccess](#) に追加されました。

2025 年 12 月 2 日

[CloudWatch Logs の新しいログ管理機能](#)

CloudWatch は、異なるツールやデータストア間で同じデータの複数のコピーを保存および維持することなく、ログ管理を組み込みのガバナンス機能を持つ単一のサービスに統合します。詳細については、「[ログ管理](#)」を参照してください。

2025 年 12 月 2 日

[CloudWatch Logs の新しいデータ取り込みと正規化機能](#)

CloudWatch AWS は、AWS Organizations およびサードパーティーソース用の構築済みコネクタと統合することで、アカウントと AWS リージョン間で販売されたログを自動的に収集します。詳細については、「[取り込み中のログの変換](#)」を参照してください。

2025 年 12 月 2 日

[CloudWatch Logs Insights の データソース機能に基づく新 しい分析とクエリ](#)

CloudWatch Insights では、クエリのファセットが導入されています。ファセットは、クエリを実行せずにデータをインタラクティブにフィルタリングおよびグループ化できるため、ログの分析に役立ちます。ファセットは、ロググループ間のフィルタリング、集約、分析を可能にするログ内のフィールド (ServiceName や StatusCode など) です。詳細については、[「ファセットを使用してログをグループ化および探索する」](#)を参照してください。

2025 年 12 月 2 日

[新しい S3 テーブルの統合](#)

S3 Tables Integration with CloudWatch を使用すると、Amazon Athena、Amazon Redshift、Apache Iceberg 互換ストアへの接続をサポートするサードパーティーツールなどの分析エンジンを使用して CloudWatch に取り込まれたログデータにアクセスできます。詳細については、[S3 Tables Integration によるログへのアクセス](#)」を参照してください。

2025 年 12 月 2 日

[CloudWatch Logs がクロスアカウントクロスリージョンログの一元化のサポートを追加](#)

CloudWatch Logs は、クロスアカウントクロスリージョンログ一元化のサポートを追加しました。これにより、CloudWatch Logs は AWS Organizations メンバーアカウントから中央アカウントにログデータを収集して分析できるようになります。詳細については、「[クロスアカウントクロスリージョンログの一元化](#)」を参照してください。

2025 年 9 月 17 日

[CloudWatch Logs 異常検出のサンプルポリシー更新](#)

CloudWatch Logs は KMS サンプルポリシーを更新し、aws:SourceAccount と aws:SourceArn の条件を追加することでスコープを縮小し、セキュリティを強化しました。詳細については、「[異常ディテクターとその結果を KMS AWS で暗号化する](#)」を参照してください。

2025 年 7 月 14 日

[CloudWatch Logs Insights が Amazon VPC Route Server ログのサポートを追加](#)

CloudWatch Logs Insights に Amazon VPC Route Server ログのサポートが追加されました。詳細については、「[AWS のサービスからのログ記録を有効にする](#)」を参照してください。Amazon VPC Route Server の新しいログソース EVENT_LOGS は [PutDeliverySource](#) に記載されています。

2025 年 6 月 12 日

[CloudWatch Logs Insights が AWS PCS ログのサポートを追加](#)

CloudWatch Logs Insights に AWS PCS ログのサポートが追加されました。詳細については、「[AWS のサービスからのログ記録を有効にする](#)」を参照してください。PCS_SCHEDULER_LOGS および の新しいログソース PCS_JOBCOMP_LOGS については AWS PCS、[PutDeliverySource](#) に記載されています。

2025 年 6 月 12 日

[CloudWatch Logs Insights が AWS Entity Resolution ログのサポートを追加](#)

CloudWatch Logs Insights に AWS Entity Resolution ログのサポートが追加されました。詳細については、「[AWS のサービスからのログ記録を有効にする](#)」を参照してください。の新しいログソース WORKFLOW_LOGS は AWS Entity Resolution、[PutDeliverySource](#) に記載されています。

2025 年 5 月 22 日

[自然言語の概要をサポートする CloudWatch Logs マネージドポリシーの更新](#)

cloudwatch:GenerateQueryResultsSummary のアクセス許可が CloudWatchLogsFullAccess と CloudWatchLogsReadOnlyAccess に追加され、クエリ結果の自然言語の概要を生成できるようになりました。このポリシーの内容を確認するには、「AWS マネージドポリシーリファレンスガイド」の「[CloudWatchLogsFullAccess](#)」および「[CloudWatchLogsReadOnlyAccess](#)」を参照してください。

2025 年 5 月 20 日

[CloudWatch Logs Insights が CloudWatch Logs Insights クエリ結果から自然言語の概要を生成するためのサポートを追加](#)

CloudWatch Logs Insights で自然言語の概要のサポートを追加しました。この機能は、現在米国東部 (バージニア北部) で利用可能な、人間が読めるクエリ結果の概要を生成します。詳細については、「[CloudWatch Logs Insights クエリ結果から自然言語の概要を生成する](#)」を参照してください。

2025 年 5 月 20 日

[CloudWatchOpenSearchDashboardsFullAccess と CloudWatchOpenSearchDashboardsFullAccess は新しい IAM ポリシーです](#)

CloudWatch Logs は、CloudWatchOpenSearchDashboardsFullAccess と CloudWatchOpenSearchDashboardAccess の 2 つの新しい IAM ポリシーを追加しました。CloudWatchOpenSearchDashboardsFullAccess は、OpenSearch Service との統合を作成および管理するためのアクセス許可を付与します。CloudWatchOpenSearchDashboardAccess は、これらの統合で作成され、提供されるログダッシュボードを表示するアクセス許可を付与します。詳細については、「[Amazon OpenSearch Service を搭載して提供されるログダッシュボード](#)」を参照してください。

2024 年 12 月 1 日

[CloudWatchLogsFullAccess ポリシーを更新](#)

CloudWatch Logs は、CloudWatchLogsFullAccess ポリシーに Amazon OpenSearch Service および IAM のアクセス許可を追加し、一部の機能で CloudWatch Logs と OpenSearch Service の統合を有効にしました。

2024 年 12 月 1 日

[CloudWatch Logs Insights がクエリ構文に新しい構造型を追加](#)

CloudWatch Logs Insights は、`unnest` コマンドと 2 つの JSON 関数を追加します。これにより、JSON 文字列をマップとリストとして操作できます。詳細については、「[構造型](#)」を参照してください。

2024 年 11 月 21 日

[CloudWatch Logs がログ取り込み中のログ変換をサポート](#)

取り込み時にロギイベントを変更できるログトランスフォーマーを作成できるため、ログをさまざまなソースからさまざまな形式で、一貫性のあるコンテキスト豊富な形式に正規化できます。詳細については、「[取り込み中のログの変換](#)」を参照してください。

2024 年 11 月 20 日

[CloudWatch Logs Insights がフィールドインデックス作成を追加](#)

CloudWatch Logs Insights に、ログのフィールドインデックス作成のサポートが追加されました。CloudWatch Logs Insights クエリでフィールドインデックスを使用すると、クエリはインデックス付きフィールドを含まないことがわかっているロギイベントの処理のスキップを試行します。詳細については、「[フィールドインデックスを作成してクエリパフォーマンスを改善し、スキャン量を削減する](#)」を参照してください。

2024 年 11 月 20 日

[CloudWatch Logs Insights による自然言語クエリの生成のサポートを一般向けに提供](#)

CloudWatch Logs Insights で、自然言語を使用したクエリの生成と更新がサポートされます。詳細については、「[Use natural language to generate and update CloudWatch Logs Insights queries](#)」を参照してください。

2024 年 6 月 20 日

[CloudWatchLogsRead OnlyAccess ポリシーが更新](#)

CloudWatch Logs が CloudWatchLogsRead OnlyAccess に `cloudwatch:GenerateQuery` アクセス許可を追加し、このポリシーを持つユーザーが自然言語プロンプトから [CloudWatch Logs Insights](#) クエリ文字列を生成できるようにしました。

2023 年 11 月 26 日

[CloudWatchLogsFullAccess ポリシーを更新](#)

CloudWatch Logs が CloudWatchLogsFullAccess に `cloudwatch:GenerateQuery` アクセス許可を追加し、このポリシーを持つユーザーが自然言語プロンプトから [CloudWatch Logs Insights](#) クエリ文字列を生成できるようにしました。

2023 年 11 月 26 日

[CloudWatch Logs がログパターン分析を追加](#)

CloudWatch Logs で、ユーザーが CloudWatch Logs Insights クエリを実行するたびにログイベントのパターンがスキャンされるようになりました。詳細については、「[Pattern analysis](#)」を参照してください。

2023 年 11 月 26 日

[CloudWatch Logs がログ異常検出を追加](#)

ロググループのログ異常ディテクターを作成できます。異常ディテクターは、ロググループに取り込まれたログイベントをスキャンし、ログデータ内の異常を検出します。詳細については、「[ログ異常検出](#)」を参照してください。

2023 年 11 月 26 日

[CloudWatch Logs が比較機能を追加](#)

CloudWatch Logs Insights を使用して、ログイベントの変化を経時的に比較できるようになりました。詳細については、「[Compare \(diff\) with previous time ranges](#)」を参照してください。

2023 年 11 月 26 日

[CloudWatch Logs が新しいログクラスを追加](#)

CloudWatch Logs で 2 つのクラスのロググループがサポートされます。これにより、アクセス頻度の低いログに対して費用対効果の高いオプションを使用し、リアルタイムモニタリングやその他の機能が必要とするログに対してはフル機能オプションを使用することができます。詳細については、「[ログクラス](#)」を参照してください。

2023 年 11 月 26 日

[CloudWatch Logs Insights が自然言語クエリの生成をサポート](#)

CloudWatch Logs Insights で、自然言語を使用したクエリの生成と更新がサポートされます。詳細については、「[Use natural language to generate and update CloudWatch Logs Insights queries](#)」を参照してください。

2023 年 11 月 26 日

[CloudWatch Logs にライブテール用の正規表現フィルターパターンの構文サポートを追加](#)

ライブテールフィルターパターンで柔軟な正規表現を使用して、検索と一致操作をニーズに合わせてさらにカスタマイズできるようになりました。詳細については、「Amazon CloudWatch Logs ユーザーガイド」の「[フィルターパターン構文](#)」を参照してください。

2023 年 11 月 13 日

[CloudWatch Logs にメトリクスフィルター、サブスクリプションフィルター、フィルターロギイベントの正規表現フィルターパターン構文サポートを追加](#)

フィルターパターンで柔軟な正規表現を使用して、検索と一致操作をニーズに合わせてさらにカスタマイズできるようになりました。詳細については、「Amazon CloudWatch Logs ユーザーガイド」の「[フィルターパターン構文](#)」を参照してください。

2023 年 9 月 5 日

[CloudWatch Logs Insights がパターンコマンドを追加](#)

CloudWatch Logs Insights クエリでパターンを使用して、自動的にログデータをパターンにクラスター化できるようになりました。パターンは、ログフィールド間で繰り返される共有テキスト構造です。詳細については、「Amazon CloudWatch Logs ユーザーガイド」の「[pattern](#)」を参照してください。

2023 年 7 月 17 日

[CloudWatch Logs Insights が重複排除コマンドを追加](#)

CloudWatch Logs Insights のクエリで重複排除を使用して、指定したフィールドの特定の値に基づき、重複する結果を削除できるようになりました。詳細については、「Amazon CloudWatch Logs ユーザーガイド」の「[dedup](#)」を参照してください。

2023 年 6 月 20 日

[アカウントレベルのデータ保護ポリシー](#)

データ保護ポリシーをアカウントレベルで設定できるようになりました。アカウントレベルのポリシーでは、アカウント内のすべてのロググループの、ログイベントの機密情報を監査およびマスキングできます。詳細については、「Amazon CloudWatch Logs ユーザーガイド」の「[機密性の高いログデータをマスキングで保護する](#)」を参照してください。

2023 年 6 月 8 日

[Live Tail 機能が追加](#)

CloudWatch Logs に Live Tail 機能が追加されました。ログを取り込み時にスキャンして、トラブルシューティングに役立てることができます。また、指定した用語に基づいて、表示されるログイベントのストリームをフィルタリングしたり、指定した用語を含むログイベントを強調表示したりすることもできます。詳細については、「[Use Live Tail to view logs in near real time](#)」を参照してください。

2023 年 6 月 6 日

[CloudWatchLogsRead](#)[OnlyAccess ポリシーが更新](#)

CloudWatch Logs

2023 年 6 月 6 日

が、CloudWatchLogsRead
OnlyAccess に対するア
クセス許可を追加しました。

logs:StartLiveTail

および logs:Stop

LiveTail のアクセス権限
が追加されたため、このポリ
シーを持つユーザーは、コン
ソールを使用して CloudWatch
Logs の Live Tail セッションを
開始および停止できます。詳
細については、「[ライブテ
ールを使用してほぼリアルタイ
ムでログを表示する](#)」を参照
してください。

[CloudWatch Logs Insights が
リリースされました](#)

その後、CloudWatch Logs
Insights を使用してログデータ
をインタラクティブに検索お
よび分析できます。詳細につ
いては、Amazon CloudWatc
h Logs ユーザーガイドの
「[Analyze Log Data with
CloudWatch Logs Insights](#)」を
参照してください。

2018 年 11 月 27 日

[Amazon VPC エンドポイント
のサポート](#)

これで、VPC と CloudWatch
Logs との間でプライベート接
続を確立できます。詳細につ
いては、Amazon CloudWatch
Logs ユーザーガイドの「[イン
ターフェイス VPC エンドポイ
ントでの CloudWatch Logs の
使用](#)」を参照してください。

2018 年 6 月 28 日

以下の表は Amazon CloudWatch Logs ユーザーガイドの重要な変更点をまとめたものです。

変更	説明	リリース日
インターフェイス VPC エンドポイント	一部のリージョンでは、インターフェイス VPC エンドポイントを使用して、Amazon VPC と CloudWatch Logs との間のトラフィックが Amazon ネットワークから離れないように維持できます。詳細については、「 インターフェイス VPC エンドポイントでの CloudWatch Logs の使用 」を参照してください。	2018 年 3 月 7 日
Route 53 DNS クエリログ	CloudWatch Logs を使用して Route 53 が受け取った DNS クエリについてのログを保存できます。詳細については、「 Amazon CloudWatch Logs とは 」または Amazon Route 53 デベロッパーガイドの「 パブリック DNS クエリのログ記録 」を参照してください。	2017 年 9 月 7 日
ロググループのタグ付け	タグを使用すると、ロググループを分類できます。詳細については、「 Amazon CloudWatch Logs のロググループにタグを付ける 」を参照してください。	2016 年 12 月 13 日
コンソールの改善	メトリクスグラフから関連するロググループに移動できます。詳細については、「 メトリクスからログへのピボット 」を参照してください。	2016 年 11 月 7 日
コンソールの再利用可能性の向上	検索、フィルタ、トラブルシューティングの作業が容易になりました。たとえば、日時の範囲でログデータをフィルタすることができます。詳細については、「 CloudWatch Logs に送信されたログデータを表示する 」を参照してください。	2016 年 8 月 29 日
Amazon CloudWatch Logs と新しい CloudWatch	CloudWatch Logs AWS CloudTrail のサポートが追加されました。詳細については、「 での CloudWatch Logs API およびコンソールオペレー	2016 年 3 月 10 日

変更	説明	リリース日
Logs メトリクス AWS CloudTrail のサポートを追加	シヨンのログ記録 AWS CloudTrail 」を参照してください。	
Amazon S3 へ の CloudWatch Logs のエクス ポートに対する サポートを追加 しました	CloudWatch Logs データを Amazon S3 にエクスポートするためのサポートが追加されました。詳細については、「 Amazon S3 へのログデータのエクスポート 」を参照してください。	2015 年 12 月 7 日
Amazon CloudWatch Logs で AWS CloudTrail ログ に記録されたイ ベントのサポー トを追加	CloudWatch にアラームを作成して、CloudTrail がキャプチャした特定 API アクティビティの通知を受け取り、通知をトラブルシューティングの実行に使用できます。	2014 年 11 月 10 日
Amazon CloudWatch Logs のサポート が追加されまし た	Amazon CloudWatch Logs を使用して、Amazon Elastic Compute Cloud (Amazon EC2) インスタンスやその他のソースからシステム、アプリケーション、カスタムログファイルを監視、保存、およびアクセスすることができます。その後、Amazon CloudWatch コンソール、の CloudWatch Logs コマンド、または CloudWatch Logs SDK を使用して AWS CLI、CloudWatch Logs から関連するログデータを取得できます。Amazon CloudWatch 詳細については、「 Amazon CloudWatch Logs とは 」を参照してください。	2014 年 7 月 10 日

AWS 用語集

最新の AWS 用語については、AWS の用語集 リファレンスの[AWS 用語集](#)を参照してください。

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